Contents

1 Welcome, researchers! 3
2 Welcome, students! 9
3 News 13
4 The Aalto environment 19
5 Data management 83
6 Triton 97
7 Scientific computing 251
8 Training 269
9 Metadocs 333
This contains documentation about scientific and data-intensive computing at Aalto and beyond. It is targeted towards Aalto researchers, but has some useful information for everyone. The data management section is useful even to non-computational researchers.

These docs are maintained by Aalto Science-IT with the help of the Aalto community.
Welcome, researchers!

Welcome to Aalto, researchers. Aalto has excellent resources for you, but it can be quite hard to know of them all. These pages will provide a good overview of IT services for researchers for you (focused on computation and data-intensive work, including experimental work).

These aren’t generic IT instructions - ITS has an introduction for staff somewhere (but apparently not online). There is also a handy list of research-focused services provided which compliments this.

1.1 Aalto service units

Understanding all the Aalto services can be quite confusing. Here are some of the key players:

- **Department IT**: Only a few departments (mainly in SCI) have their own IT staff. Others have people such as laboratory managers which may be able to provide some useful advice. Known links: CS, NBE, PHYS.

- **Science-IT**: Overlaps with SCI department IT groups. They run the Triton cluster and support scientific computing. Their services may be used throughout the entire university, but support is organized from the departments which fund them. The core Science-IT departments are CS, NBE, and PHYS. Science-IT runs a weekly SciComp garage, where we provide hands on support for anything related to scientific computing. *This site (scicomp.aalto.fi)* is the main home, but there is also an formal site.

- **Aalto IT Services (ITS)**: Provides central IT infrastructure. They have a “Services for Research” group, but it is less specialized than Science-IT. ITS may be the first place to contact if not in the School of Science. Their infrastructure is used in all schools including SCI, and the base on which everyone builds. Their instructions are on Aalto Inside, but most importantly the already-mentioned IT Services for Research page. Contact via servicedesk.

- **Aalto Research Services**: Administrative-type support. Provides support for grantwriting, innovation and commercialization, sponsored projects, legal services for research, and research infrastructures. (In 2019 a separate “innovation services” split from the previous “research and innovation services”)

- **CSC** is the Finnish academic computing center (and more). They provide a lot of basic infrastructure you use without knowing it, as well as computing and data services to researchers (all for free). *website <csc>_*. Also, currently Aalto has information scattered on websites everywhere:
Aalto scientific computing guide

- **aalto.fi** is the normal homepage, but doesn’t have much practical information for researchers. As of late 2018, information from inside and into is supposed to move here. The new site is well known for being hard to use (sorry, we can’t do anything about that). This site is “not designed to have a logical structure and instead, you are expected to search for information” (actual quote). Some pages get more information if you log in, and there is no indication of which ones. In general, unless you know what you are looking for, don’t expect to find anything here without extensive work.

- **it.aalto.fi** is a new location for IT instructions, and somewhat replaces the old IT instructions on inside.aalto.fi and into.aalto.fi. These generally relate to how to do a specific task - but not necessarily what is the best thing to do.

- **inside.aalto.fi** is the former typical “official” staff documentation area, but now gone since 2019 with all information moved to aalto.fi. It wasn’t in search engines, but once you found a page it wasn’t that hard to navigate.

- **into.aalto.fi** is the student official information. It used to contain a lot of duplicate information to inside but was public, so people would end up there.

- **wiki.aalto.fi** is obviously the Aalto wiki space. Anyone can make a space here, and many departments’ internal sites are here. Searching can randomly find useful information. Most sites aren’t publically searchable.

- **scicomp.aalto.fi** is where you are now. Scicomp was started by the Science-IT team from the Triton (HPC cluster) documentation, and scicomp is slowly taking over from our departments’ research IT instructions. Now it is our general guidance to researchers and the best place to find information on research and scientific computing - as opposed to general “staff computing” you find other places.

### 1.2 End-user systems

Aalto provides computers to its employees, obviously. You can choose a managed system or standalone. If it’s standalone, you are on your own. If managed, login is through your Aalto account. You can get laptop or desktop, and Linux, Mac, or Windows.

Desktops are connected directly to the wired networks and are typically preferred by researchers using serious data or computation. **Linux desktops** have fast and automatic access to all of the university data storage systems, including Triton and department storage. They also have a wide variety of scientific software already available (and somewhat similar to Triton). We have some limited instructions and pointers to the main instructions for **mac** and **windows** computers.

Managed laptops are usable in and out of the Aalto networks, and you can become “primary user” which allows you to install some software yourself (Linux at least).

### 1.3 Computing

You have two primary Aalto options: workstations and Triton. The Aalto workstations have basic scientific software installed. From the workstations, you can use the **HTCondor** distributed computing framework.

Most demanding computing at Aalto is performed on **Triton**, the Aalto high performance computing cluster. It is a fairly standard medium-sized cluster, and its main advantage is the close integration into the Aalto environment: it shares Aalto accounts, its data storage (2PB) is also available on workstations, and has local support. If you need dedicated resources, you can purchase them and they can be managed by us as part of Triton so that you get dedicated resources and can easily scale to the full power of Triton. Triton is part of the Finnish Grid and Cloud Infrastructure. Triton is the largest publically known computing cluster in Finland after the CSC clusters. Triton provides a web-based interface via **JupyterHub**.

**CSC** (the Finnish IT Center for Science) is a government-owned organization which provides a lot of services, most notably huge HPC clusters, data, and IT infrastructure services to the academic sector. All of their services are free.
to the academic community (paid directly by the state of Finland). They also coordinate the Finnish Grid and Cloud Infrastructure. They have the largest known clusters in Finland.

## 1.4 Data

Data management isn’t just storage: if data is just put somewhere, you get a massive mess and data isn’t usable in even 5 years. Funders now require “data management plans”. Thus data management is not just a hot topic, it’s an important one. We have a whole section on data, and also there are higher level guides from Aalto. If you just want to get something done, you should start with our Aalto-specific guideline for Science-IT data storage (used in CS, NBE, PHYS) - if you follow our plan, you will be doing better than most people. If you have specific questions, there is an official service email address you can use (see the Aalto pages), or you can ask the Science-IT team.

Aalto has many data storage options, most free. In general, you should put your data in some centralized location shared with your group: if you keep it only on your own systems, the data dies when you leave. We manage data by projects: a group of people with shared access and a leader. Groups provide flexibility, sharing, and long-term management (so that you don’t lose or forget about data every time someone leaves). You should request as many projects as you need depending on how fine-grained you need access control, and each can have its own members and quota. You can read about the storage locations available and storage service policy.

Triton has 2PB of non-backed up data storage on the high-performance Lustre filesystem. This is used for large active computation purposes. The Triton nodes have an incredible bandwidth to this and it is very fast and parallel. This is mounted by default at Science-IT departments, and can be by default in other departments too.

Aalto provides “work” and “teamwork” centralized filesystems which are large, backed up, snapshotted, shared: everything you may want. Within the Science-IT departments, Science-IT and department IT manages it and provides access. For other schools/departments, both are provided by Aalto ITS but you will have to figure out your school’s policies yourself. It’s possible to hook this storage into whatever else you need over the network. (In general, “work” is organized by the Aalto hierarchy, while “teamwork” is flatter. If you consider yourself mainly Aalto staff who fits in the hierarchy, work is probably better. If you consider yourself a research who collaborates with whoever, teamwork is better.) Teamwork instructions

CSC provides both high-performance Lustre filesystems (like Triton) and archive systems. CSC research portal.

In our data management section, we provide many more links to long-term data repositories, archival, and so on. The fairdata.fi project is state-supported and has a lot more information on data. They also provide some data storage focused on safety and longer-term storage (like IDA), though they are not very used at Aalto because we provide such good services locally.

Aalto provides, with Aalto accounts, Google Drive (unlimited, also Team Drives), Dropbox (unlimited), and Microsoft OneDrive (5TB). Be aware that once you leave Aalto, this data will disappear!

## 1.5 Software

Triton and Aalto Linux workstations come with a lot of scientific software installed, with in the Lmod system. Triton generally has more. If you need something, it can be worth asking us first to install it for everyone.

If you are the primary user of a workstation, you can install Ubuntu packages yourself (and if you aren’t, you should ask to be marked as primary user). If you use Triton or are in a Science-IT department, it can be worth asking Science-IT about software you need - we are experts in this and working to simplify the mess that scientific software is. Windows workstations can have things automatically installed, check the windows page.

Triton and Aalto workstations have the central software available, currently for laptops you are on your own except for some standard stuff.
On Triton and Linux workstations, type `module spider $name` to search for available software. We are working to unify the software stack available on Triton and Aalto workstations so that they have all the same stuff.

ITS has a software and licenses (FI) page, and also a full list of licenses. There is also https://download.aalto.fi/.

CSC also has a lot of software. Some is on CSC computers, some is exported to Triton.

## 1.6 Starting a project

Each time you start a project, it’s worth putting a few minutes into planning so that you create a good base (and don’t end up with chaos in a few years). We don’t mean some grant, we mean a line of work with a common theme, data, etc.

- Think about how you’ll manage data. It’s always easy to just start working, but it can be worth getting all project members on the same page about where data will be stored and what you want to happen to it in the end. Having a very short thing written will also help a lot to get newcomers started. The “practical DMP” section here can help a lot - try filling out that A4 page to consider the big sections.
- Request a data group (see above) if you don’t already have a shared storage location. This will keep all of your data together, in the same place. As people join, you can easily give them access. When people leave, their work isn’t lost.
  - If you already have a data group that is suitable (similar members), you can use that. But there’s no limit to the number of projects, so think about if it’s better to keep things apart earlier.
  - Mail your department IT support and request a group. Give the info requested at the bottom of data outline page.
  - In the same message, request the different data storage locations, e.g. scratch, project, archive. Quotas can always be increased later.

## 1.7 Training

Of course you want to get straight to research. However, we come from a wide range of backgrounds and we’ve noticed that missing basic skills (computer as a tool) can be a research bottleneck. We have constructed a multi-level training plan so that you can find the right courses for your needs. These courses are selected by researchers for researchers, so we make sure that everything is relevant to you.

Check our upcoming training page for a list of upcoming courses. If you do anything computational or code-based at all, you should consider the twice-yearly CodeRefinery workshops (announced on our page). If you have a Triton account or do high-performance computing or intensive computing or data-related tasks, you should come to the Summer (3 days) or Winter (1 day) kickstart, which teaches you the basics of Triton and HPC usage (we say it is “required” if you have a Triton account).

## 1.8 Other notes

Remember to keep the IT Services for Research page close close at hand!

Research is usually collaborative, but sometimes you can feel isolated - either because you are lost in a crowd, or far away from your colleagues. Academic courses don’t teach you everything you need to be good at scientific computing - put some effort into working together with, learning from, and teaching your colleagues and you will get much further.
There are some good cheatsheets which our team maintains. They are somewhat specialized, but useful in the right places.

It can be hard to find your way around Aalto, the official campus maps and directions are known for being confusing. Try UsefulAaltoMap instead.
Welcome, students!

Welcome to the Aalto! We are glad you are interested in scientific computing and data. This site may be useful to you, but are somewhat targeted to research usage. However, it can still serve as a good introduction to resources for scientific and data-intensive computing at Aalto if you are a student. This page is devoted to resources which are available to students.

If you are involved in a research group or doing researcher for a professor/group leader, you are a researcher! You should acquaint yourself with all information on this site, starting with Welcome, researchers! and use whatever you need.

General IT instructions can be found at https://it.aalto.fi. There used to be some on into.aalto.fi, but these are gone now. There also used to be a 2-page PDF introduction for students, but it also seems to be gone from online.

2.1 Accounts

In general, your Aalto account is identical to that which researchers have — the only difference is that you don’t have an departmental affiliation.

2.2 Getting help

As a student, the ITS servicedeskgs are the first place to go for help. The site https://it.aalto.fi is the new central site for IT instructions. Previously, some public instructions were on https://into.aalto.fi (studies focused) and https://inside.aalto.fi (staff focused) and finding information was a great challenge. Note that in 2018, all of these are being merged somehow, but the dust hasn’t settled yet.

This site, http://scicomp.aalto.fi, is intended for research scientific computing support but has a few page useful to you.

2.3 Computation

As a student, you have access to various light computational resources.
The Jupyter service at https://jupyter.cs.aalto.fi is available to everyone with an Aalto account. It provides at least basic Python and R software; we try to keep it up to date with the things people need most.

The shell servers brute and force are for light computing, and generally for students. You may find them useful, but can often be overloaded. Learn how to launch Jupyter notebook on there.

For GPU computing, the Paniikki Linux computer lab (map) has GPUs in all workstations. Software is available via module spider $name to search and module load $name to load (and the module anaconda3 has Python, tensorflow, etc.). Read the Paniikki cheatsheet here. The instructions for Aalto workstations sort of apply there as well. The software on these machines is managed by the Aalto-IT team. This is the place if you need to play with GPUs, deep learning, etc.

A new (2018) remote desktop service is available at https://vdi.aalto.fi. This provides Windows and Linux desktops and is designed to replace the need for computer classrooms with special software installed. You can access it via a web browser or the VMware Horizon client.

The use of Triton is for research purposes and students can’t get access unless you are affiliated with a research project or (in very rare cases), a course makes special arrangements.

### 2.4 Data storage

Aalto home directories have a 40GB quota, and this is suitable for small use. Note that files here are lost once you leave Aalto, so make sure you back up.

The IT Services for Research page contains some other cloud services which may be useful for data storage. Of the cloud services, note that everyone at Aalto can get an unlimited Google Drive account through the Aalto Google Apps service: instructions. Your Aalto Google account will expire once you are no longer affiliated, so your files here will become inaccessible.

### 2.5 Software

ITS has a software and licenses (FI) page, and also a full list of licenses. There is also http://download.aalto.fi/. Various scientific software can be found for your own use via the Aalto software portals.

The Lmod (module) system provides more software on brute/force and in Paniikki. For example, to access a bunch of scientific Python software, you can do module load anaconda3. The researcher-focused instructions are here, but like many things on this site you may have to adapt to the student systems.

Common software:
2.6 Other notes

It can be hard to find your way around Aalto, the official campus maps and directions are known for being confusing confusing. Try UsefulAaltoMap instead.

Do you have suggestions for this page? Please leave an issue on Github (make sure you have a good title that mentions the audience is students, so we can put the information in the right place). Better yet, send a pull request to us yourself.
3.1 News

3.1.1 Scicomp garage

The Aalto Scicomp Garage is a help session for scientific computing at Aalto organized by the Science-IT team (Triton admins). It’s the best time to talk to the people behind scientific computing at Aalto. This is a place to get stuff done, so bring your laptop and coffee/food, and come hang out.

Most of the time, we are just there to answer questions.

Sometimes, there may be a short presentation on some topic, but you can still ask questions to the other staff before, during, and after that.

Come if you want to:

- Solve problems
- Discuss bigger problems
- Network with others who are doing similar work
- Learn something new
- Give feedback

Schedule

- Days: Triton garage is every week from 13:00-14:00 on Thursdays.
- Time: We promise to be there only the first 30 minutes, so if everyone leaves before then we might leave too (usually we are there the whole time).
- Location: See below. Common locations: T4 and A106 (CS building, by entrance around back of T2), A237 (CS building, by main coffee room on 2nd floor A wing) and B121 (CS building).
We are piloting also a data & science garage at NBE every first Wednesday of the month, room F336 13:00. NBE garage will also cover issues related to working with personal data, research ethics, best practices in biomedical research.

<table>
<thead>
<tr>
<th>Date</th>
<th>Time</th>
<th>Location</th>
<th>Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.jun</td>
<td>13-14</td>
<td>A106</td>
<td></td>
</tr>
<tr>
<td>13.jun</td>
<td>13-14</td>
<td>A106</td>
<td></td>
</tr>
<tr>
<td>20.jun</td>
<td>13-14</td>
<td>A106</td>
<td>No garage, Juhannus Friday</td>
</tr>
<tr>
<td>27.jun</td>
<td>13-14</td>
<td>A106</td>
<td></td>
</tr>
<tr>
<td>4.jul</td>
<td>13-14</td>
<td>A106</td>
<td></td>
</tr>
<tr>
<td>11.jul</td>
<td>13-14</td>
<td>A106</td>
<td></td>
</tr>
<tr>
<td>18.jul</td>
<td>13-14</td>
<td>A106</td>
<td></td>
</tr>
<tr>
<td>25.jul</td>
<td>13-14</td>
<td>A106</td>
<td></td>
</tr>
<tr>
<td>1.aug</td>
<td>13-14</td>
<td>A106</td>
<td></td>
</tr>
<tr>
<td>8.aug</td>
<td>13-14</td>
<td>A106</td>
<td></td>
</tr>
<tr>
<td>15.aug</td>
<td>13-14</td>
<td>A106</td>
<td></td>
</tr>
<tr>
<td>22.aug</td>
<td>13-14</td>
<td>A106</td>
<td></td>
</tr>
<tr>
<td>29.aug</td>
<td>13-14</td>
<td>A106</td>
<td></td>
</tr>
</tbody>
</table>


**Topics**

- Triton intro: interactive jobs
- Git

**Possible special topics**

- Profiling and performance monitoring
- debugging
- open source: making software and running a project, licenses
- shell scripting and automation
- unix intro
- software testing
- building good programs
- porting python2 to python3
- R
- matlab
- GPU / deep learning computing
- molecular dynamics software

**Past events**

Scicomp Garage has existed since Spring 2017.
3.1.2 New project: Paniikki GPU machine remote utilisation for AI

TL;DR

Aalto CS-IT is initiating a new project: Paniikki GPU machine remote utilisation for AI. You will be able to use Paniikki machines for your Kaggle projects or Stanford Convolutional Neural Network course from anywhere. If you find this interesting please give us a feedback :)

Background

AI is the new sexy

Alphago is the new GO champion. Everyone is hiring data scientists. Aalto machine learning course is exploding with 600 students and everyone is teaching machine learning and deep learning. Aalto machine learning master’s program, MACADAMIA, is packed with international talents.

While the topic itself is very interesting and there are many resources for studying it, there are some challenges that people face: GPU and dependency installations. You could always use a powerful GPU workstation in Paniikki but you may be busy to go there or you may not be familiar with remote SSH.

Can Aalto help you?

YES!!! We are always glad to help you and we want you to get advantage of the luxurious workstations in Paniikki(The NVIDIA Quadro P5000 GPUs in Paniikki costed us €2K/module). We will build a desktop app that allows you to use complete scientific containers on Paniikki machines, remotely without any use of Terminal. We will also prepare containers for famous ML courses so you don’t have to struggle setting the infrastructures.

How are we going to deliver this?

We will distribute a cross-platform app. You install it on your Linux/Windows/Mac. When you start the app, click a container of your choice e.g. Tensorflow_py35. A jupyter notebook server will start on a most idle machine in Paniikki and you will be given an URL & token for the notebook. We are considering of using ElectronJS for the front-end and Singularity for the containers. We use containers because you can just take them if you want to use it somewhere else.

Who could use it?

All Aalto students and staffs.

When will it be available?

2018 spring. Hopefully April.

Last but not least: let’s make Aalto great together!

Our mission is to serve you the best to enhance the science. We work hard but we want to make sure that we are doing it right. So your feedbacks are tremendously invaluable to us. Tell us how you like our ideas and what you want and need, but please be nice :) We are humans and we have feelings like you. If you would like to collaborate with us you are more than welcome! Please pay us a visit at A243 at CS building or send us an e-mail @ guru@cs.hut.fi.
Appendix

In Paniikki (panic in Finnish), a computer room in the CS building, there are 31 monster machines waiting to rock 'n roll with you. Here is the spec:

<table>
<thead>
<tr>
<th>CPU properties</th>
<th>Spec</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td>Intel(R) Xeon(R) CPU E5-1650 v4 @ 3.60GHz</td>
</tr>
<tr>
<td>Architecture</td>
<td>x86_64</td>
</tr>
<tr>
<td>Byte Order</td>
<td>Little Endian</td>
</tr>
<tr>
<td>CPU(s)</td>
<td>12</td>
</tr>
<tr>
<td>Thread(s) per core</td>
<td>2</td>
</tr>
<tr>
<td>MHz</td>
<td>1200.796</td>
</tr>
<tr>
<td>max MHz</td>
<td>4000.0000</td>
</tr>
<tr>
<td>min MHz</td>
<td>1200.0000</td>
</tr>
<tr>
<td>Virtualization</td>
<td>VT-x</td>
</tr>
<tr>
<td>L1d cache</td>
<td>32K</td>
</tr>
<tr>
<td>L1i cache</td>
<td>32K</td>
</tr>
<tr>
<td>L2 cache</td>
<td>256K</td>
</tr>
<tr>
<td>L3 cache</td>
<td>15360K</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>GPU properties</th>
<th>Spec</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td>NVIDIA Quadro P5000</td>
</tr>
<tr>
<td>Core</td>
<td>GP104GL (Pascal-based)</td>
</tr>
<tr>
<td>Core clock</td>
<td>1607 MHz</td>
</tr>
<tr>
<td>Memory clock</td>
<td>1251 MHz</td>
</tr>
<tr>
<td>Memory size</td>
<td>16384 MiB</td>
</tr>
<tr>
<td>Memory type</td>
<td>256-bit GDDR5X</td>
</tr>
<tr>
<td>Memory bandwidth</td>
<td>320</td>
</tr>
<tr>
<td>CUDA cores</td>
<td>2560</td>
</tr>
<tr>
<td>CUDA compute capability</td>
<td>6.1</td>
</tr>
<tr>
<td>OpenGL</td>
<td>4.5</td>
</tr>
<tr>
<td>OpenCL</td>
<td>1.2</td>
</tr>
<tr>
<td>Near GeForce Model</td>
<td>GeForce GTX 1080</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Memory properties</th>
<th>Spec</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM</td>
<td>32GiB</td>
</tr>
</tbody>
</table>

3.1.3 Jupyter Workshop, 8 March 2019

Jupyter notebooks are a web-based system for interactive computing, and the Aalto Jupyterhub (https://jupyter.cs.aalto.fi) extends this with some features useful for teaching.

This workshop is to introduce the Aalto community to the Aalto JupyterHubs and how they can be used for teaching (including independent projects) and research. The workshop has the goals of introducing new users to Jupyter, providing advice on implementation in courses, and guiding future development. The main audience is teachers, but any interested people are welcome to attend, but research uses will be discussed too.

Date/time: 2019-03-08, 10:00-12:00
Location: T4, CS building.
Registration: none, just show up.
Program

The first hour will be a general introduction about Jupyter and our services (good for those determining if this is of any use and strategy) the second half will be focused on practical examples and discussion for those who want to actively use (sharing knowledge among those who are or will use it).

- 10:00-10:15: **Intro to Aalto JupyterHub**, Joakim Järvinen
  - Strategic benefits of JupyterHub and its use in Machine Learning Basic Principles 2018
- 10:15-10:20: **The Aalto A+ learning management system**, Jaakko Kantojärvi
  - What the A+ system is and how it relates (and could relate to) to JupyterHub.
- 10:20-10:35: **Aalto JupyterHub details and ecosystem**, Richard Darst
  - Background, where we are going, and primary limitations.
- 10:35-10:50: **Demos of Data Science and Kernel Methods notebooks in courses**, Petri Jehkonen and Eric Bach
- Discussion/break: 10 min
- *(end of first half, begin details half for those who want to stay)*
- 11:00-11:20: **From start to Jupyter: demo of a full course setup**, Richard Darst
  - Requesting a course, tracking files, managing students, grading, sharing files, integration with other services, etc.
- 11:20-???: **Notebook construction for autograding**, Alexander Ilin and Joakim Järvinen
  - Making notebooks for autograding is quite challenging. Learn how to do it well.
- Discussion and planning for the future

About the Aalto JupyterHubs

**Jupyter** is an interactive computing ecosystem. It is most known for the Jupyter (formerly IPython) notebooks, which combine code, documentation, and output together in a usable form. JupyterHub provides multiuser access over the web: Aalto provides two, one for all Aalto users, in particular students and with support for courses and one for research, as part of Triton.

See also

The **Jupyter4edu** book might be the best external known resource so far.
Aalto provides a wide variety of support for scientific computing. For a summary, see the IT Services for Research page. For information about data storage at Aalto, see the section on data management below.

4.1 The Aalto environment

For more services provided at the Aalto level, see the IT Services for Research page.

4.1.1 Aalto account

Extension to Aalto account and email

Aalto account expiration is bound to staff or student status. Account closes one week after the affiliation to Aalto university ends. Expiration is managed completely by Aalto IT Services, and department IT staff is not able to extend Aalto accounts.

If extension to account is needed, this may be achieved with visitor contract. The contract requires host information, so you should contact your supervisor who (if accepting your request) contacts HR with needed details to prepare the official visitor contract.

4.1.2 Aalto Linux

Aalto Linux is provided to all departments in Aalto. Department IT co-maintains this, and in some departments provides more support (specifically, CS, NBE, PHYS at least). It contains a lot of software and features to support scientific computing and data. Both laptop and desktop setups are available.

This page is mainly about the Linux flavor in CS/PHYS/NBE, co-managed by these departments and Science-IT. Most of it is relevant to all Aalto, though.
Basics

- Aalto home directory. In the Aalto Ubuntu workstations, your home directory will be your Aalto home directory. That is, the same home directory that you have in Aalto Windows machines and the Aalto Linux machines, including shell servers (kosh, talta, lyta, brute, force).

- Most installations have Ubuntu 16.04, 18.04 is coming soon.

- Some basic information from Aalto is available at Inside.

- **Login is with Aalto credentials.** Anyone can log in to any computer. Since login is tied to your Aalto account, login is tied to your contract status. Please contact HR if you need to access systems after you leave the university or your account stops working due to contract expiration.

- All systems are effectively identical, except for local Ubuntu packages installed. Thus, switching machines is a low-cost operation.

- Systems are centrally managed using puppet. Any sort of configuration group can be set up, for example to apply custom configuration to one group’s computers.

- Large scientific computing resources are provided by the Science-IT project. *The compute cluster there is named Triton.* Science-IT is a school of science collaboration, and its administrators are embedded in NBE, PHYS, CS IT.

- Workstations are on a dedicated network VLAN. The network port must be configured before it can be turned on and you can’t just assume that you can move your computer to anywhere else. You can request other network ports enabled for personal computers, just ask.

- Installation is fully automated via netboot. Once configuration is set up, you can reboot and PXE boot to get a fresh install. There is almost no local data (except the filesystem for tmp data on the hard disks which is not used for anything by default, `/l/` below), so reinstalling is a low-cost operation. The same should be true for upgrading, once the new OS is ready you reboot and netinstall. Installation takes less than two hours.

- Default user interface. The new default user interface for Aalto Linux is **Unity**. If you want to switch to the previous default interface (Gnome), before logging in please select “Gnome Flashback (Metacity)” by clicking the round ubuntu logo close to the “Login” input field.

- Personal web pages. What you put under `~/public_html` will be visible at `https://users.aalto.fi/~username`. See Data storage.

When requesting a new computer:

- Contact your department IT

When you are done with a computer:

- Ensure that data is cleaned up. Usually, disks will be wiped, but if this is important then you must explicitly confirm before you leave. There may be data if you use the workstation local disks (not the default). There is also a local cache (`$XDG_CACHE_HOME`), which stores things such as web browser cache. Unix permissions protect all data, even if the primary user changes, but it is better safe than sorry. Contact IT if you want wipes.

Laptops

- You can get laptops with Linux on it.

- Each user should log in the first time while connected to the Aalto network. This will cache the authentication information, then you can use it wherever you want.
• Home directories can be synced with the Aalto home directories. This is done using unison. TODO: not documented, what about this?

• If you travel, make sure that your primary user is set correctly before you go. The system configuration can’t be updated remotely.

• Otherwise, environment is like the workstations. You don’t have access to the module system, though.

• If the keychain password no longer works: see FAQ at the bottom.

Workstations

Most material on this page defaults to the workstation instructions.

Primary User

The workstations have a concept of the “primary user”. This user can install software from the existing software repositories and ssh remotely to the desktops.

• Primary users are implemented as a group with name $hostname-primaryuser. You can check primary user of a computer by using getent group $hostname-primaryuser or check your primary-userness with groups.

• If you have a laptop setup, make sure you have the PrimaryUser set! This can’t be set remotely.

• Make sure to let us know about primary users when you get a new computer set up or change computers. You don’t have to, but it makes it convenient for you.

• It is not currently possible to have group-based primary users (a group of users all have primary user capabilities across a whole set of computers, which would be useful in flexible office spaces). TODO: are we working on this? (however, one user can have primary user access across multiple computers, and hosts can have multiple primary users, but this does not scale well)

Data

See the general storage page for the full story (this is mainly oriented towards Linux). All of the common shared directories are available on department Linux by default.

We recommend that most data is stored in shared group directories, to provide access control and sharing. See the Aalto data page.

You can use the program unison or unison-gtk to synchronise files.

Full disk encryption (Laptops)

All new (Ubuntu 16.04) laptops come with full disk encryption by default (instructions). This is a big deal and quite secure, if you use a good password.

When the computer is first turned on, you will be asked for a disk encryption password. Enter something secure and remember it - you have only one chance. Should you want to change this password, take the computer to an Aalto ITS service desk. They can also add more passwords for alternative users for shared computers. Aalto ITS also has a backup master key. (If you have local root access, you can do this with cryptsetup, but if you mess up there’s nothing we can do).

Desktop workstations do not have full disk encryption, because data is not stored directly on them.

4.1. The Aalto environment
Software

Already available

- Python: module load anaconda3 (or anaconda2) (desktops)
- Matlab: automatically installed on desktops, Ubuntu package on laptops.

Ubuntu packages

If you have PrimaryUser privileges, you can install Ubuntu packages using one of the following commands:

- By going to the Ubuntu Software Center (Applications -> System Tools -> Administration -> Ubuntu Software Centre). Note: some software doesn’t appear here! Use the next option.
  - `aptcon --install $ubuntu_package_name` (search for stuff using `apt search`)
- By requesting IT to make a package available across all computers as part of the standard environment. Help us to create a good standard operating environment!

The module system

The command `module` provides a way to manage various installed versions of software across many computers. This is the way that we install custom software and newer versions of software, if it is not available in Ubuntu. Note that these are shell functions that alter environment variables, so this needs to be repeated in each new shell (or automated in login).

- See the Triton module docs docs for details.
- `module avail` to list all available package.
- `module spider $name` to search for a particular name.
- `module load $name` to load a module. This adjusts environment variables to bring various directories into PATH, LD_LIBRARY_PATH, etc.
- We will try to keep important modules synced across the workstations and Triton, but let us know.

Useful modules:

- `anaconda3` and `anaconda2` will always be kept up to date with the latest Python Anaconda distribution, and we’ll try to keep this in sync across Aalto Linux and Triton.

Admin rights

Most times you don’t need to be an admin on workstations. Our Linux systems are centrally managed with non-standard improvements and features, and 90% of cases can be handled using existing tools:

Do you want to:

- Install Ubuntu packages: Use `aptcon --install $package_name` as primary user.
- This website tells me to run `sudo apt-get` to install something. Don’t, use the instructions above.
- This website gives me some random instructions involving `sudo` to install their program. These are not always a good idea to run, especially since our computers are networked, centrally managed, and these instructions don’t always work. Sometimes, these things can be installed as a normal user with simple modifications. Sometimes their instructions will break our systems. In this case, try to install as normal user and then send a support
request first. *If none of these work and you have studied enough to understand the risk, you can ask us. Make sure you give details of what you want to do.*

- I need to change network or some other settings. Desktops are bound to a certain network and settings can’t be changed, users can’t be managed, etc.
- It’s a laptop: *then yes, there are slightly more cases you need this, but see above first.*
- I do low-level driver, network protocol, or related systems development. *Then this is a good reason for root, ask us.*

If you do have root and something goes wrong, our help is limited to reinstalling (wiping all data - note that most data is stored on network drives anyway).

If you do need root admin rights, you will have to fill out a form and get a new wa account, then Aalto has to approve. Contact your department IT to get the process started.

**Remote access to your workstation**

If you are primary user, you can ssh to your own workstation from certain Aalto servers, including at least taltta. See the remote access page.

**More powerful computers**

There are different options for powerful computing.

First, we have desktop Linux workstations that are more powerful than normal. If you want one of these, just ask. It includes a medium-power GPU card. You can buy a more powerful workstation if you need, but...

Beyond that, we recommend the use of Triton rather than constructing own servers which will only be used part-time. You can either use Triton as-is for free, or pay for dedicated hardware for your group. Your own hardware as part of Triton means that you can use all Triton and even CSC if you need with little extra work. You could have your own login node, or resources as part of the queues.

Triton is Aalto’s high-performance computing cluster. It is not a part of the department Linux, but is heavily used by researchers. You should see the main documentation at the [Triton user guide](#), but for convenience some is reproduced here:

- Triton is CentOS (compatible with the Finnish Grid and Cloud Infrastructure), while CS workstations are Ubuntu. So, they are not identical environments, but we are trying to minimize the differences.
  - Since it is is part of FGCI, it is easy to scale to more power if needed.
- We will try to have similar software installed in workstation and Triton module systems.
- The paths `/m/$dept/` are designed to be standard across computers
- The `project` and `archive` filesystems are not available on all Triton nodes. This is because they are NFS shares, and if someone starts a massively parallel job accessing data from here, it will kill performance for everyone. Since history shows this will eventually happen, we have not yet mounted them across all nodes.
  - These are mounted on the login nodes, certain interactive nodes, and dedicated group nodes.
  - TODO: make this actually happen.
- Triton was renewed in 2016 and late 2018.
- All info in the [triton user guide](#)
Common problems

Network shares are not accessible

If network shares do not work, there is usually two things to try:

- Permission denied related problems are usually solved by obtaining new Kerberos ticket with command ‘kinit’
- If share is not visible when listing directories, try to ‘cd’ to that directory from terminal. Shares are mounted automatically when they are accessed, and might not be visible before you try to change to the directory.

Graphical User Interface on Aalto CS Linux desktop is sluggish, unstable or does not start

- 1. Check your disk quota from terminal with command `quota`. If you are not able to log in to GUI, you can change to text console with CTRL+ALT+F1 key combo and log in from there. GUI login can be found with key combo CTRL+ALT+F7.
- 2. If you are running low on quota (blocks count is close quota), you should clean up some files and then reboot the workstation to try GUI login again.
   - You can find out what is consuming quota from terminal with command: `bash -c 'cd && du -sch .[!\.]\* \* |sort -h'`

Enter password to unlock your login keyring

You should change your Aalto password in your main Aalto workstation. If you change the password through e.g. https://password.aalto.fi, then your workstation’s password manager (keyring) does not know the new password and requests you to input the old Aalto password.

If you remember your old password, try this:

1. Start application Passwords and Keys (“seahorse”)
2. Click the “Login” folder under “Passwords” with right mouse button and select “Change password”
3. Type in your old password to the opening dialog
4. Input your current Aalto password to the “new password” dialog
5. Reboot the workstation / laptop

If changing password didn’t help, then try this:

- Then instead of selecting the “change password” from the menu behind right mouse key select “delete” and reboot the workstation. When logging in, the keyring application should use your logging key automatically.

In Linux some process is stuck and freeze the whole session

You can kill a certain (own) process via text console.

How do I use eJournals, Netmot and other Aalto library services from home?

There is a weblogin possibility at Aalto Library. After this, all library provided services are available. There are links for journals (nelli) and netmot.
Rsync complains about Quota, even though there is plenty left.

The reason usually is that default rsync -av tries to preserve the group. Thus, there is wrong group in the target. Try using rsync -rlptDxvz --chmod=Dg+s <source> <target>. This will make group setting correct on /scratch/ etc and quota should then be fine.

Quota exceeded or unable to write files to project / work / scratch / archive

Most likely this is due to wrong Linux filesystem permissions. Quota is set per group (e.g. braindata) and by default file go to the default group (domain users). If this happens under some project, scratch etc directory it will complain about “Disk quota exceeded”.

In general this is fixed by admins by setting the directory permissions such that all goes ok automatically. But sometimes this breaks down. Some programs often are responsible for this (rsync, tar for instance).

There are two easy ways to fix this

- In terminal, run the command find . -type d -exec chmod g+rwxs {} \; under your project directory. After this all should be working normally again.
- If it’s on scratch or work, see the Triton quotas page
- Contact NBE-IT and we will reset the directory permissions for the given directory

I cannot start Firefox

There are two reasons for this.

1. Your network home disk is full

```
# Go to your user dir
cd ~/..
# Check disk usage
du -sh *
```

The sum should be less than the max quota which is 20GB. If your disk is full then delete something or move it to a local directory, /l/.

2. Something went wrong with your browser profile

If you get an error like “The application did not identify itself”, following might solve the issue.

Open terminal,

```
firefox -P -no-remote
```

This will launch Firefox and ask you to choose a profile. Note that when you delete a profile you delete passwords, bookmarks and etc. So it’s better to create a new profile, migrate bookmarks and delete the old one.
4.1.3 Aalto Mac

This page describes the Aalto centrally-managed Mac computers, where login is via Aalto accounts. If you have a standalone laptop (one which does not use your Aalto account), some of this may be relevant, but for the most part you are on your own and you will access your data and Aalto resources via Remote Access.

More instructions: https://inside.aalto.fi/display/ITServices/Mac

Basics

In the Aalto installations, login is via Aalto account only.

- When you get a computer, ask to be made primary user (this should be default, but it’s always good to confirm). This will allow you to manage the computer and install software.

- The first time you login, you must be on an Aalto network (wired or aalto wifi) so that the laptop can communicate with Aalto servers and get your login information. After this point, you don’t need to be on the Aalto network anymore.

- Login is via your Aalto account. The password stays synced when you connect from an Aalto netowrk.

Full disk encryption

This must be enabled per-user, using FileVault. **You should always do this, there is no downside.** On Aalto-managed laptops, install “Enable FileVault disk encryption” (it’s a custom Aalto thing that does it for you). To do this manually, “Settings → Privacy → enable File Vault.”

Data

You can mount Aalto filesystems by using SMB. Go to Finder → File or Go (depending on OS) → Connect to Server → enter the smb:// URL from the data storage pages.

You can find more information at For generic ways of accessing, see Remote Access. For Aalto data storage locations see Data storage, and for the big picture of where and how to store data see Data: outline, requesting space, requesting access.

The program AaltoFileSync is pre-installed and can be used to synchronize files. But you basically have to set it up yourself.

Software

.dmg files

If you are the primary user, in the Software Center you can install the program “Get temporary admin rights”. This will allow you to become an administrator for 30 minutes at a time. Then, you can install .dmg files yourself. This is the recommended way of installing .dmg files.

Aalto software

There is an application called “Managed software center” pre-installed (or “Managed software update” in older versions). You can use this to install a wide variety of ready-packaged software. (ITS instructions).
Homebrew

Homebrew is a handy package manager on Macs. On Aalto Macs, you have to install Brew in your home dir. Once you install brew, you can easily install whatever you may need.

First install Xcode through Managed Software Centre (either search Xcode, or navigate through Categories -> Productivity -> Xcode).

```bash
# Go to wherever you want to have your Brew and run this
mkdir Homebrew && curl -L https://github.com/Homebrew/brew/tarball/master | tar xz -C Homebrew --strip 1

# This is a MUST!!!
echo "export PATH=$PATH:$pwd/Homebrew/bin" >> ~/.bash_profile

# Reload the profile
source ~/.bash_profile

# Check if brew is correctly installed.
which brew  # /Users/username/Homebrew/bin/brew
```

Admin rights

The “Get temporary admin rights” program described under .dmg file installation above lets you get some admin rights - but not full sudo and all.

You don’t need full admin rights to install brew.

If you need sudo rights, you need a workstation admin (wa) account. Contact your department admin for details.

CS Mac backup service

The CS department provides a full clone-backup service for Aalto-installation mac computers. Aalto-installation means the OS is installed from Aalto repository.

We use Apple Time Machine. Backup is wireless, encrypted, automatic, periodic and can be used even outside the campus using the Aalto VPN. It is “clone” because we can restore your environment in its entirety. You can think of it as a snapshot backup (though it isn’t). We provide twice the space of your SSD; your Mac has 250GB of space, you get 500GB of backup space. If you would like to enroll in the program please pay a visit to our office, T-talo A243.

Encryption

We provide two options for encryption:

1. You set your own encryption key and only you know it. The key is neither recoverable nor resettable. You lose it, you lose your backup.
2. We set it on behalf of you and only we know it.

Restore

With Time Machine you have two options for restore.

1. Partial
Aalto scientific computing guide

• You can restore file-by-file. Watch the video,

2. Complete restore
  • In case your Mac is broken, you can restore completely on a new Mac. For this, you must visit us.

Trouble-shooting

Can’t find the backup destination

This happens because either 1). you changed your Aalto password or 2). the server is down. Debug in the following manner,

```bash
# Is the server alive?
ping timemachine.cs.aalto.fi

# If alive, probably it's your keychain.
# Watch the video below.

# If dead, something's wrong with the server.
# Please contact CS-IT.
```

Corrupted backup

This is an unfortunate situation with an unknown reason. We take a snapshot of your backup. Please contact CS-IT.

Common problems

Insane CPU rampage by UserEventAgent

It is a mysterious bug which Apple hasn’t solved yet. We can reinstall your system for you.
4.1.4 Aalto Windows

This page describes the Aalto centrally-managed Windows computers, where login is via Aalto accounts. If you have a standalone laptop (login not using Aalto account), some of this may be relevant, but for the most part you will access your data and Aalto resources via Remote Access.

More instructions: https://inside.aalto.fi/display/ITServices/Windows

Basics

In the Aalto installations, login is via Aalto account only.

- You must be on the Aalto network the first time you connect.

Full disk encryption

Aalto Windows laptops come with this by default, tied to your login password. To verify encryption, find “BitLocker” from the start menu and check that it is on.

Note, that on standalone installations, you can do encryption by searching “TrueCrypt” in programs - it is already included.

Data

This section details built-in ways of accessing data storage locations. For generic ways of accessing remotely, see Remote Access. For Aalto data storage locations, see Data storage and Data: outline, requesting space, requesting access.

Your home directory is automatically synced to some degree.

You can store local data at C:\LocalUserData\User-data\<yourusername>. Note that this is not backed up or supported. For data you want to exist in a few years, use a network drive. It can be worth making a working copy here, since it can be faster.

Software

Aalto software

There is a Windows software self-service portal which can be used to install some software automatically.

Installing other software

To install most other software, you need to apply for a workstation admin (wa) account. Contact your department IT to get the process started.

Common problems

4.1.5 Data storage

This page outlines the storage options available for your data. There are many options available, some provided by CSIT, some provided by Aalto IT Services, and some provided by Science IT. For clarity, this page describes them all,
so that you can have an easy reference.

When starting a new project, please first consider the big picture of good Research Data Management: See the general data management pages here and Aalto’s page. On Aalto’s page, there are links to solutions for Opening, Collaborating and Archiving. Our department’s resources are just one part of that.

This page is currently a bit Linux-centric, because Linux is best supported.

**Other operating systems:** Windows and OSX workstations do not currently have any of these paths mounted. In the future, project and archive may be automatically mounted. You can always remote mount via sshfs or SMB. See the remote access page for Linux, Mac, and Windows instructions for home, project, and archive. In OSX, there is a shortcut in the launcher for mounting home. In Windows workstations, this is Z drive. On your own computers, you may need to use AALTO\username as your username for any of the SMB mounts.

**Laptops:** Laptops have their own filesystems, including home directories. These are not backed up automatically. Other directories can be mounted as described on the remote access page.

**Summary table**

This table lists all available options in Science-IT departments, including those not managed by departments. In general, project is for most research data that requires good backups. For big data, use scratch. Request separate projects when needed to keep things organized.
<table>
<thead>
<tr>
<th>Filesystem</th>
<th>Path (Linux)</th>
<th>Triton?</th>
<th>Quota</th>
<th>Backups?</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>home</td>
<td>/u/.../username/unix</td>
<td>no</td>
<td>40 GiB</td>
<td>yes, $HOME/.../snapshot/</td>
<td>Used for personal and non-research files</td>
</tr>
<tr>
<td>project</td>
<td>/m/$dept/project</td>
<td>some</td>
<td>per-project, up to 100s of GiB</td>
<td>Yes, hourly/daily/weekly (.snapshot)</td>
<td></td>
</tr>
<tr>
<td>archive</td>
<td>/m/$dept/archive</td>
<td>some</td>
<td>per-project, up to 100s of GiB</td>
<td>Yes, hourly/daily weekly. + off-site tape backups. (.snapshot)</td>
<td></td>
</tr>
<tr>
<td>scratch</td>
<td>/m/$dept/scratch</td>
<td>yes</td>
<td>per-project, 2 PiB available</td>
<td>RAID6, but no backups.</td>
<td>Don’t even think about leaving irreplaceable files here! Need Triton account.</td>
</tr>
<tr>
<td>work</td>
<td>/m/$dept/work</td>
<td>yes</td>
<td>200GB default</td>
<td>RAID6, but no backups.</td>
<td>same as scratch. Need Triton account.</td>
</tr>
<tr>
<td>local</td>
<td>/l/username</td>
<td>yes</td>
<td>usually a few 100s GiB available</td>
<td>No, and destroyed if computer reinstalled.</td>
<td>Directory needs to be created and permissions should be made reasonable (quite likely ‘chmod 700 /l/USER’, by default has read access for everyone!) Space usage: ‘du -sh /l/’. Not shared among computers.</td>
</tr>
<tr>
<td>tmpfs</td>
<td>/run/user/$uid</td>
<td>yes</td>
<td>local memory</td>
<td>No</td>
<td>Not shared.</td>
</tr>
<tr>
<td>web-home</td>
<td>$HOME/public_html</td>
<td></td>
<td>1 GiB</td>
<td><a href="https://users.aalto.fi/~USER/">https://users.aalto.fi/~USER/</a></td>
<td></td>
</tr>
<tr>
<td></td>
<td>/m/webhome/...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>custom</td>
<td>solutions</td>
<td></td>
<td></td>
<td>Contact us for special needs, like sensitive data, etc.</td>
<td></td>
</tr>
</tbody>
</table>

**General notes**

- The table below details the types of filesystems available.
- The path `/m/$dept/` is designed to be a standard location for mounts. In particular, this is shared with Triton.
- The server `magi` is `magi.TODO` and is for the CS department. Home directory is mounted here without kerberos protection but directories under `/m/` need active kerberos ticket (that can be acquired with `kinit` command). `.talta` is `.talta.aalto.fi` and is for all Aalto staff. Both use normal Aalto credentials.
- **Common problem:** The Triton `scratch/work` directories are automounted. If you don’t see it, enter the full name then tab complete and it will appear. It will appear after you try accessing with the full name.
• **Common problem:** These filesystems are protected with Kerberos, which means that you must be authenticated with *Kerberos tickets* to access them. This normally happens automatically, but they expire after some time. If you are using systems remotely (the shell servers) or have stuff running in the background, this may become a problem. To solve, run `kinit` and it will refresh your tickets.

**Filesystem list**

- **home:** your home directory
  - Shared with the Aalto environment, for example regular Aalto workstations, Aalto shell servers, etc.
  - Should not be used for research work, personal files only. Files are lost once you leave the university.
    - Instead, use project for research files, so they are accessible to others after you leave.
  - Quota 20 GiB.
  - Backups recoverable by `~/../.snapshot/` (on linux workstations at least).
  - SMB mounting: `smb://home.org.aalto.fi/`
- **project:** main place for shared, backed-up project files
  - `/m/$dept/project/$project/`
  - Research time storage for data that requires backup. Good for e.g. code, articles, other important data. Generally for small amount (10s-100s GiB) of data per project.
  - This is the normal place for day to day working files which need backing up.
  - Multi user, per-group.
  - Quotas: from 10s to 100s of GiB
  - Quotas are not designed to hold extremely large research data (TiBs). Ideal case would be 10s of GiB, and then bulk intermediate files on scratch.
  - Weekly backup to tape (to recover from major failure) + snapshots (recover accidentally deleted files). Snapshots go back:
    - hourly last 26 working hours (8-20)
    - daily last 14 days
    - weekly last 10 weeks
    - Can be recovered using `.snapshot/` within project directories
  - Accessible on `magi/taltta` at the same path.
  - SMB mounting: `smb://tw-cs.org.aalto.fi/project/$group/`
- **archive:**
  - `/m/$dept/archive/$project/`
  - For data that should be kept accessible for 1-5 years after the project has ended. Alternatively a good place to store a copy of a large original data (backup).
  - This is practically the same as project, but retains snapshots for longer so that data is ensured to be written to tape backups.
  - This is a disk system, so does have reasonable performance. (Actually, same system as project, but separation makes for easier management).
  - Quotas: 10s to 1000s of GiB
- Backups: same as project.
- Accessible on magi/tallta at the same path.
- SMB mounting: smb://tw-cs.org.aalto.fi/archive/$group/

- **scratch:** large file storage and work, not backed up (Triton).
  - /m/$dept/scratch/$group/
  - Research time storage for data that does not require backup. Good for temporary files and large data sets where the backup of original copy is somewhere else (e.g. archive).
  - This is for massive, high performance file storage. Large reads are extremely fast (1+ GB/s).
  - This is a lustre file system as part of triton (which is in Keilaniemi).
  - Quotas: 10s to 100s of TiB. The university has 2 PB available total.
  - In order to use this, you must have a triton account. If you don’t, you get “input/output error” which is extremely confusing.
  - On workstations, this is mounted via NFS (and accessing it transfers data from Keilaniemi on each access), so it is not fast on workstations, just large file storage. For high performance operations, work on triton and use the workstation mount for convenience when visualizing.
  - This is RAID6, so is pretty well protected against single disk failures, but not backed up at all. It is possible that all data could be lost. Don’t even think about leaving irreplaceable files here. CSC actually had a problem in 2016 that resulted in data loss. It is extremely rare (decades) thing, but it can happen. (still, it’s better than your laptop or a drive on your desk. Human error is the greatest risk here).
  - Accessible on magi/tallta at the same path.
  - SMB mounting: smb://data.triton.aalto.fi/scratch/$dept/$dir/. (Username may need to be AALTO\yourusername.)

- **Triton work:** personal large file storage and work (Triton)
  - /m/$dept/work/$username/
  - This is the equivalent of scratch, but per-person. Data is lost once you leave.
  - Accessible on magi/tallta at the same path.
  - SMB mounting: smb://data.triton.aalto.fi/work/$username. (Username may need to be AALTO\yourusername.)

- **local:** local disks for high performance
  - You can use local disks for day to day work. These are not redundant or backed up at all. Also, if your computer is reinstalled, all data is lost.
  - Performance is much higher than any of the other network filesystems, especially for small reads. Scratch+Triton is still faster for large reads.
  - If you use this, make sure you set UNIX permissions to restrict the data properly. Ask if you are not sure.
  - If you store sensitive data here, you are responsible for physical security of your machine (as in no one taking a hard drive). Unix permissions should protect most other cases.
  - When you are done with the computer, you are also responsible for secure management/wiping/cleanup of this data.
  - See the note about disk wiping under Aalto Linux (under “when you are done with your computer”). IT should do this, but if it’s important you must mention it, too.

- **tmpfs:** in-memory filesystem
- This is a filesystem that stores all data in memory. It is extremely high performance, but extremely temporary (lost on each reboot). Also shares RAM with your processes, so don’t use too much and clean up when done.
- TODO: are these available everywhere?

- **webhome**: web space for `users.aalto.fi`
  - This is the space for `users.aalto.fi` space can be accessed from the `public_html` link in your home directory.
  - This is not a real research filesystem, but convenient to note here.
  - Quota (2015) is 1 GiB. (`/m/webhome/webhome/`)
  - [https://users.aalto.fi/~USER/](https://users.aalto.fi/~USER/)

- **triton home**: triton’s home directories
  - Not part of departments, but documented here for convenience
  - The home directory on Triton.
  - Backed up daily.
  - Not available on workstations.
  - Quota: 1 GB

- **Aalto work**: Aalto’s general storage space
  - Not often used within Science-IT departments: we use project and archive above, which are managed by us and practically equivalent. You could request space from here, but expect less personalized service.
  - Aalto home directories are actually here now.
  - You may request storage space from here, email the Aalto servicedesk and request space on work. The procedures are not very well established.
  - Data is snapshotted and backed up offsite for disaster recovery.
  - Search [https://it.aalto.fi](https://it.aalto.fi) for “work.org.aalto.fi” for the latest instructions.
  - SMB mounting via `smb://work.org.aalto.fi`

- **Aalto teamwork**: Aalto’s general storage space
  - Not used directly within Science-IT departments: we have our own direct interfaces to this, and project and archive directories are actually here.
  - For information on getting teamwork space (outside of Science-IT departments), contact servicedesk.
  - Teamwork is unique in that it is arbitrarily extensible, and you may buy the space from the vendor directly. Thus, you can use external grant money to buy storage space here.
  - SMB mounting via `smb://teamwork.org.aalto.fi`

**Quota errors**

Use the `quota` command to see your quota. If you have scratch or work mounted, the quota command will hang and produce errors. For now, check your scratch/work quotas on Triton.

The scratch and work directories do quotas by unix group, and **there is a strange error about quota exceeded** that you may get sometimes when the unix group of the file or directory is wrong. See the full information at Quotas and summary below. You may have to fix this on Triton if the things below don’t work.
• Symptoms: “Quota exceeded” when you are trying to make a new file in scratch or work directory.

• Root cause: quotas are by groups, and if a directory is not setgroupid (chmod g+s), then files being created will have a different group (with no quota for that location), thus quota exceeded by default. This often happens when you copy a directory from one place to another, and then later try to make new files in that directory.

• Solution: chmod g+s $directory or find $directory -type d -exec chmod g+s {} \;
(you don’t want to make regular files g+s mode).

4.1.6 Data: outline, requesting space, requesting access

Note: Need a place to store your data? This is the place to look. First, we expect you to read and understand the top information. Then, see the instructions at bottom.

This page is about how to handle data - not the raw storage part, which you can find at data storage. Aalto has high-level information on research data management, too.

What is data management?

Data management is much more than just storage. It concerns everything from data collection, to data rights, to end-of-life (archival, opening, etc). This may seem far-removed from research practicalities, but funding agencies are beginning to require advanced planning. Luckily, there are plenty of resources at Aalto (especially in SCI), and it’s just a matter of connecting the dots.

Oh, and data management is also important because without data management, data becomes disorganized, you lose track, and as people come and go, you lose knowledge of what you have. Don’t let this happen to you or your group!

Another good starting point is the Aalto research data management pages. These pages can also help with preparing a data management plan.

Data management is an important part of modern science! We are here to help. These pages both describe the resources available at Aalto (via Science-IT), and provide pointers to issues that may be relevant to your research.

Data storage at Aalto SCI (principles and policies)

Note: This especially applies to CS, NBE, and PHYS (the core Science-IT departments). The same is true for everyone using Triton storage. These policies are a good idea for everyone at Aalto, and are slowly being developed at the university level.

Most data should be stored in a group (project) directory, so that multiple people can access it and there is a plan for after you leave. Ask your supervisor/colleagues what your group’s existing groups are and where the data is stored. Work data should always be stored in a project directory, not personal home directories. See below for how to create or join a group. Home directory data can not be accessed by IT staff, according to law and policy - data there dies when you leave.

All data in group directories is considered accessible to all members (see below).

All data stored should be Aalto or research related. Should there be questions, ask. Finnish law and Aalto policies must be followed (in that order), including by IT staff. Should there be agreements with third-parties regarding data rights, those will also be followed by IT staff, but these must be planned in advance.

All data must have an owner and lifespan. We work with large amount of data from many different people, and data without clear ownership becomes a problem. (“ownership” refers to decision-making responsibility, not IPR
ownership). Also, there must be a clear successor for when people leave or become unavailable. By default, this is the supervisor.

Personal workstations are considered stateless and, unless there is special agreement, could be reinstalled at any time and are not backed up. This should not concern day to day operations, since by default all data is stored on network filesystems.

We will, in principle, make space for whatever data is needed. However, it is required that it be managed well. If you can answer what the data contains, why it’s stored, and how the space is used, and why it’s needed, it’s probably well managed for these purposes.

Read the full Science-IT data management policy here.

Information on all physical locations how to use them is on the storage page.

Groups

Everywhere on this page, “group” refers to a certain file access group (such as a unix group), not an organizational (research) group. They will often be the same, but there can be many more access groups made for more fine-grained data access.

Data is stored in group directories. A group may represent a real research group, a specific project, or specific access-controlled data. These are easy to make, and they should be extensively used to keep data organized. If you need either finer-grained or more wide data access, request that more groups are made.

Please note, that by design all project data is accessible to every member in the group. This means that, when needed, IT can fix all permissions so that all group members can read all data. For limiting the access more fine-grained than these project groups, please have a separate group created. Data in a group is considered “owned and managed” by the group owner on file. The owner may grant access to others and change permissions as needed. Unless otherwise agreed, any group member may also request permissions to be corrected so that everyone in the group has access.

• Access control is provided by unix groups (managed in the Aalto active directory). There can be one group per group leader, project, or data that needs isolation. You should use many groups, they make overall management easier. A group can be a sub-group of another.

• Each group can get its own quota and filesystem directories (project, archive, scratch, etc). Quota is per-filesystem. Tell us requested quota when you set up a project.
  – A typical setup would be: one unix group for a research group, with more groups for specific project when that is helpful. If there are fixed multi-year projects, they can also get a group.

• Groups are managed by IT staff. To request a group, mail us with the necessary information (see next point).

• Each group has an owner, quota on filesystems, and some other metadata (see below),

• To have a group created and storage space allocated, see below.

• To get added to a group, see instructions below.

• To see your groups: use the groups command or groups $username

• To see all members of a group: getent group $groupname

Common data management considerations

Organizing data

This may seem kind of obvious, but you want to keep data organized. Data is always growing in volume and variety, so if you don’t organize it as it is being made, you have no chance of doing it later. Organize by:
• Project
• To be backed up vs can be recreated
• Original vs processed.
• Confidential or not confidential
• To be archived long-term vs to be deleted

Of course, make different directories to sort things. But also the group system described above is one of the pillars of good data organization: sort things by group and storage location based on how it needs to be handled.

Backups

Backups are extremely important, not just for hardware failure, but consider user error (delete the wrong file), device lost or stolen, etc. Not all locations are backed up. It is your responsibility to make sure that data gets stored in a place with sufficient backups. Note that personal workstations and mobile devices (laptops) are not backed up.

Confidential or sensitive data

Note: The following description is written for the CS department, but applies almost equally to NBE and PHYS. This is being expanded and generalized to other department as well. Regardless of your department, these are good steps to follow for any confidential data at Aalto.

Note: This meets the requirements for “Confidential” data, which covers most use cases. If you have extreme requirements, you will need something more (but be careful about making custom solutions).

Aalto has some guidelines for classification of confidential information, but they tend to deal with documents as opposed to practical guidelines for research data. If you have data which needs special attention, you should put it in a separate group and tell us when creating the group.

The following paragraph is a “summary for proposals”, which can be used when the CS data security needs to be documented. This is for the CS department, but similar thing can be created for other departments. A longer description is also available.

Aalto CS provides secure data storage for confidential data. This data is stored centrally in protected datacenters and is managed by dedicated staff. All access is through individual Aalto accounts, and all data is stored in group-specific directories with per-person access control. Access rights via groups is managed by IT, but data access is only provided upon request of the data owner. All data is made available only through secure, encrypted, and password-protected systems: it is impossible for any person to get data access without a currently active user account, password, and group access rights. Backups are made and also kept confidential. All data is securely deleted at the end of life. CS-IT provides training and consulting for confidential data management.

If you have confidential data at CS, follow these steps. CS-IT takes responsibility that data managed this way is secure, and it is your responsibility to follow CS-IT’s rules. Otherwise you are on your own:

• Request a new data folder in the project from CS-IT. Notify them that it will hold confidential data and any special considerations or requirements. Consider how fine-grained you would like the group: you can use an existing group, but consider how many people will have access.

• Store data only in this directory on the network drive. It can be accessed from CS computers, see data storage.
To access data from laptops (Aalto or your own), use network drive mounting, not copying. Also consider if temporary files: don’t store intermediate work or let your programs save temporary files to your own computer.

Don’t transfer the data to external media (USB drives, external hard drives, etc) or your own laptops or computers. Access over the network.

All data access should go through Aalto accounts. Don’t send data to others and or create other access methods. Aalto accounts provide central auditing and access control.

Realize that you are responsible for the day to day management of data and using best practices. You are also responsible for ensuring that people who have access to the data follow this policy.

In principle, one can store data on laptops or external devices with full disk encryption. However, in this case we does not take responsibility unless you ask us first.you must ask us about this. In general it’s best to try to adapt to the network drive workflow. (Laptop full disk encryption is a good idea anyway).

We can assist in creating more secure data systems, as can Aalto IT security. It’s probably more efficient to contact us first.

Personal data (research data about others, not about you)

“Personal data” is any data concerning an identifiable person. Personal data is very highly regulated (mainly by the Personal Data Act, soon by the General Data Protection Regulation). Aalto has a document that describes what is needed to process personal data for research, which is basically a research-oriented summary of the Personal Data Act. Depending on the type of project, approval from the Research Ethics Committee may be needed (either for publication, or for human interaction. The second one would not usually cover pure data analysis of existing data). Personal data handling procedures are currently not very well defined at Aalto, so you will need to use your judgment.

However, most research does not need data to be personally identifiable, and thus research is made much simpler. Thus, you want to try to always make sure that data is not identifiable, even to yourself using any technique (anonymization). The legal requirement is “reasonable likelihood of identification”, which can include technical and confidentiality measures, but in the end is still rather subjective. Always anonymize before data arrives at Aalto, if possible. Let us know when you have personal data, so we can make a note of it in the data project.

However, should you need to use personal data, the process is not excessively involved beyond what you might expect (informed consent, ethics, but then a notification of personal data file). Contact us for initial help in navigating the issues and RIS for full advice.

Openness

Aalto strongly encourages to share the data openly or under controlled access with a goal of 50% data shared by 2020 (see The Aalto RDM pages). In short, Aalto says that you “must” make strategic decisions about openness for the best benefits (which practically probably means you can do what you would like). Regardless, being open is usually a good idea when you can: it builds impact for your work and benefits society more.

Zenodo (https://zenodo.org/) is an excellent platform for sharing data, getting your data cited (it provides a DOI), and control what you share with different policies (http://about.zenodo.org/policies/). For larger data, there are other resources, such as IDA/AVA provided by CSC (see below).

There are lists of data repositories: r3data, and Nature Scientific Data’s list.

Datasets can and should also be listed on ACRIS, just like papers - this allows you to get credit for them in the university’s academic reporting.
Data management plans

Many funders now require data management plans when submitting grants. (Aside from this, it’s useful to do a practical consideration of how you’ll deal with data)

Please see:
- The DMP section on this site
- The Aalto data management plan page

Long-term archival

Long-term archival is important to make sure that you have ability to access your group’s own data in the long term. Aalto resources are not currently intended for long-term archival. There are other resources available for this, such as
- the EU-funded Zenodo for open published data (embargoed data and closed data is also somewhat supported).
- Finland’s IDA (for large data, closed or open). There are Aalto-specific instructions for IDA here.
- There is supposed to be an alternate Finnish digital preservation service coming in 2017, and it’s unclear what the intention of IDA is in light of that.

Archival when you leave

Unfortunately, everyone leaves Aalto sometime. Have you considered what will happen to your data? Do you want to be remembered? This section currently is written from the perspective of a researcher, not a professor-level staff member, but if you are a group leader you need to make sure your data will stay available! Science-IT (and most of these resources) are focused on research needs, not archiving a person’s personal research data (if we archive it for a person who has left, it’s not accessible anyway! Our philosophy is that it should be part of a group as described above.). In general, we can archive data as part of a professor’s group data (managed in the group directories the normal ways), but not for individuals.

- Remember that your home directories get removed when your account expires (we think in only two weeks!).
- Data in the group directories it won’t be automatically deleted. But you should clean up all your junk and leave only what is needed for future people. Remember, if you don’t take care of it, it becomes extremely hard for anyone else to. The owner of the group (professor) will be responsible for deciding what to do with the data, so make sure to discuss with them and easy for them to do the right thing!
- Make sure that the data is documented well. If it’s undocemented, then it’s unusable anyway.
- Can your data be released openly? If you can release something as open data on a reputable archive site like Zenodo, you can ensure that you will always have access to it. (The best way to back up is to let the whole internet do it for you.)
- For lightweight archival (~5 years past last use, not too big), the archive filesystem is suitable. The data must be in a group directory (probably your professor’s). Make sure that you discuss the plans with them, since they will have to manage it.
- IDA (see above) could be used for archival of any data, but you will have to maintain a CSC account (TODO: can this work, and how?). Also, these projects have to be owned by a senior-level staff person, so you have to transfer it to a group anyway.
- Finland aims to have a long-term archival service by 2017 (PAS), but this is probably not intended for own data, only well-curated data. Anyway, if you need something that long and it isn’t confidential, consider opening it.
Summary of data locations

Below is a summary of the core Science-IT data storage locations.

<table>
<thead>
<tr>
<th>Solution</th>
<th>Purpose</th>
<th>Where available?</th>
<th>Backup?</th>
<th>Group management?</th>
</tr>
</thead>
</table>
| project                | Research time storage for data that requires backup. Good for e.g. code, articles, other important data. Generally for a small amount of data per project. | Workstations, Triton login node        | Weekly backup to tape (to recover from major failure) + snapshots (recover accidentally deleted files). Snapshots go back  
  • hourly last 26 working hours (8-20)  
  • daily last 14 days - weekly last 10 weeks | yes                             |
| Archive                | Data which a longer life that project. Practically the same, but better to sort things out early. Also longer snapshot and guaranteed to get backed up to tape. | Workstations, Triton login node        | Same as above /m/$dept/project/$group. | yes                             |
| Scratch (group based)/work (per-user) | Large research data that doesn’t need backup. Temporary working storage. Very fast access on Triton. | /m/$dept/$scratch/$groupname, /m/$dept/work/$username. | scratch: yes, work: no | |

See data storage for full info.

Instructions for storage and access

Note: This applies to the Science-IT departments. If you want to apply for storage space from Aalto-IT, you can use these instructions as a model, but their processes are not yet fully developed.

You and users must accept the data policy (summary above).

Existing data groups and responsible contacts:

- CS: Existing groups and CS-IT (guru) email here
- NBE: Existing groups and NBE IT (it-nbe) email here
- PHYS:
- Aalto: Aalto IT servicedesk
Requesting to be added to a group

Send an email to the responsible contact (see above) and **CC the group owner or responsible person**, and include this information:

- Group name that you request to join
- copy and paste this statement, or something similar: “I am aware that all data stored here is managed by the group’s owner and have read the data management policies.”
- Ask the group owner to reply with confirmation.
- Do you need access to scratch or work? If so, you need a Triton account and you can request it now. If you don’t, you’ll get “input/output error” and be very confused.
- Example:

  Hi, I (account=omes1) would like to join the group myprof. I am aware that all data stored here is managed by the group's owner and have read the data management policies. $professor_name, please reply confirming my addition.

Requesting a new group

Send an email to the responsible contact (see above) with the following information. Group owners should be long-term (e.g. professor level) staff.

- Requested group name (you can check the name from the lists below)
- Owner of data (prof or long-term staff member)
- Other responsible people who can authorized adding new members to the group. (they can reply and say “yes” when someone asks to join the group.)
- Who is responsible for data should you become unavailable (default: supervisor who is probably head of department).
- Initial members
- Expiration time (default=max 2 years, extendable. max 5 years archive). We will ping you for management/renewal then.
- Which filesystems and what quota. (project, archive, scratch). See the *storage page*.
- Basic description of purpose of group.
- Is there any confidential or personal data (see above for disclaimer).
- Any other notes that CS-IT should enforce, for example check NDA before giving access.
- Example:

  I would like to request a new group coolproject. I am the owner, but my postdoc Tiina Tekkari can also approve adding members. (Should I become unavailable, my colleague Anna Algorithmi (also a professor here) can provide advice on what to do with the data)

  We would like 20GB on the project filesystem.

  This is for our day to day work in algorithms development, we don’t expect anything too confidential.
4.1.7 Science-IT data policy

Note: This was originally developed at CS, but applies to all departments managed by the Science-IT team.

In Aalto, large amounts of data with variety of requirements are being processed daily. This describes the responsibilities of IT support and users with respect to data management.

Everyone should know the summary items below. The full policy is for reference in case of doubts (items in **bold** are things which are not completely obvious).

This policy is designed to avoid the most common problems by advance planning for the majority case. Science-IT is eager to provide a higher level of service for those who need it, but users must discuss with staff. This policy is jointly implemented by department IT and Science-IT.

**Summary for users**

- Do not store research data in home directories, this is not accessible should something happen to you or when you leave. They will be automatically deleted.
- Project directories are accessible to ALL members, files not intended for access by ALL members should be stored in a separate project.
- Workstations and mobile devices are NOT backed up. Directories with backups are noted. It is your responsibility to make sure that you store in backed up places. Don’t consider only disk failure, but also user error, loss of device, etc.
- Data stored in project directories is managed by the (professor, supervisor) who owns the directory, and they can make decisions regarding access now and in the future. Any special considerations should be discussed with them.
- Data is not archived or saved for individual users. Data which must be saved should be in a shared project directory with an owner who is still at Aalto.
- There is no default named security level - of course we keep all data secure, but should you be dealing with legally confidential files, you must ask us.

**Summary for data directory owners (professors or long-term staff)**

- Data in the shared directories controlled by you and you make decisions on it.
- All data within a project is accessible by all members of that project. Make more projects if more granularity is needed.
- Data must have an expiration time, and this is extended as needed. Improperly managed data is not stored indefinitely. If data is long-term archived, it must still have an administrative owner at Aalto who can make decisions about it.
- There must be a succession plan for the data, should the data owner leave or become unavailable to answer questions. By default this is the supervisor or department head. They will make decisions about access, management, and end-of-life.
- We will try to handle whatever data you may need us to. The only prerequisite is that it is managed well. We can’t really define “managed well”, but at least it means you know what it contains and where the space is going.
Detailed policy

This is the detailed policy. The important summary for users and owners is above, but the full details are written below for avoidance of doubts.

Scope

1. This policy concerns all data stored in the main provided locations or managed by Science-IT staff (including its core departments).

Responsibilities

1. In data processing and rules we follow Finnish legislation and Aalto university policies in this order.
2. If there are agreements with a third party organization for data access those rules are honored next. Regarding this type of data we must be consulted first prior to the storing the data.
3. Users are expected to follow all Aalto and CS policies, as well as good security practices.
4. IT is expected to provided a good service, data security, and instruction on best practices.

Storage

1. All data must have owner and given lifespan. Data cannot be stored indefinitely, but of course lifespan is routinely extended when needed. There are other long-term archival services.
2. Work related data should always be stored outside users HOME directory. HOME is meant only for private and non-work related files. (IT staff is not allowed to retrieve lost research files from a user’s home directory)
3. Other centrally available folders (i.e. Project, Archive, Scratch) than HOME are meant for work related information only.
4. Desktop computers are considered as stateless. They can be re-installed at any point by IT if necessary. Data stored on local workstations is always considered as temporary data and is not backed up. IT support will still try to inform users of changes.
5. Backed-up data locations are listed. It is the user’s responsibility to ensure that data is stored in backed-up locations as needed. Mobile devices (laptops) and personal workstations are not backed up.

Ownership, and access rights, and end-of-life

1. Access rights in this policy refer only to file system rights. Other rights (e.g. IPR) to the stored information are not part of this policy.
2. There must be a clear owner and chain of responsibility (successor) for all data (who owns it and can make decisions and who to ask when they leave or become unavailable).
3. For group directories (e.g. project, archive, scratch), file system permissions (possibility to read, write, copy, modify and delete) of these files belongs to group. There is not more granular access, for example single files with more restrictive permissions. Permissions will be fixed by IT on request from group members.
4. The group owner-on-file can make any decisions related to data access, management, or end-of-life.
5. Should a data owner of a group directory become unavailable or unable to answer questions about access, management, or end-of-life, the successor they named may make decisions regarding the data access, including end-of-life. This defaults to their supervisor (e.g. head of department), but should be discussed on data opening.

6. Should researchers need a more complex access scheme, this must be discussed with IT support.

Security/Confidentiality

1. **Unless there is a notification, there is no particular guaranteed service level regarding confidential data. However, all services are expected to be as secure as possible and are designed to support confidential data.**

2. **Should a specific security level be needed, that must be agreed separately.**

3. Data stored to the provided storage location is not encrypted at rest.

4. Confidentiality is enforced by file system permissions will be set and access changes will be always confirmed from data owner.

5. All storage medium (hard drives, etc), should be securely wiped to the extend technically feasible at end of life. This is handled by IT, but if it is required it must be handled by the end users.

6. All remote data access should use strong encryption.

7. Users must notify IT support or their supervisor about any security issues or misuse of data.

8. **Security of laptops, mobile devices and personal devices is not currently guaranteed by IT support. Confidential data should use centralized IT-provided services only.**

9. Users and data owners must take primary responsibility for data security, since technical security is only one part of the process.

Communication

1. Details about centrally provided folders and best practices are available in online documentation.

2. Changes to policy will be coordinated by department management. All changes will at least be announced to data owners, but individual approvals are not needed unless a service level drops.

4.1.8 Remote Access

This page describes remote access solutions. Most of them are provided by Aalto, but there are also instruction for accessing your workstations here. See Aalto Inside for more details.

Linux shell servers

- CS department servers:
  - **magi**: Department staff server (no heavy computing, access to workstations and has file systems mounted, use kinit command first if project directories are not visible)

- Aalto servers
  - **kosh.aalto.fi, lyta.aalto.fi**: Aalto, for general login use (no heavy computing)
  - **brute.aalto.fi, force.aalto.fi**: Aalto, for “light computing” (expect them to be overloaded and not that useful). If you are trying to use these for research, you really want to be using Triton instead.
– taltta.aalto.fi: Staff server (access to workstations and has filesystems mounted, but you need to kinit to access them.) that is kind of outdated and different.

- Your home directory is shared on all Aalto shell servers, and that means .ssh/authorized_keys as well.

- You can use any of these to mount things remotely via sshfs. This is easy on Linux, harder but possible on other OSs. You are on your own here. magi is recommended since kerberos is not needed.

- The CS filesystems project and archive and Triton filesystems scratch and work are mounted on magi (and taltta.aalto.fi)(see storage).

VPN / web proxy

To access certain things, you need to be able to connect to the Aalto networks via VPN. This is easy and automatically set up on Aalto computers.

For the main Aalto instructions, see it.aalto.fi and search for “VPN”. This section has some quick reference info.

- Generic: OpenConnect/Cisco AnyConnect protocols. vpn.aalto.fi

- Aalto Linux: Status bar → Network → VPN Connections → Aalto TLS VPN.

- Aalto mac: Dock → Launchpad → Cisco AnyConnect Secure Mobility Client

- Aalto windows: Start → Search → AnyConnect


- Personal mac: use Cisco AnyConnect VPN Client

- personal windows: use Cisco AnyConnect VPN Client

For more lightweight things (though not actually easier!), you can use ssh proxy. You are on your own here. ssh -D 8080 $username@kosh.aalto.fi. Configure your web browser or other applications to use a SOCKS5 proxy on localhost:8080 for connections. Remember to revert when done or else you can’t connect to anything. The extension FoxyProxy Standard may be useful here.

Remote mounting of network filesystems

From Aalto networks (or VPN), you can mount many of the filesystems via SMB. To use this well, you want to get the VPN set up first like mentioned above. (You can also access these filesystems via ssh through the shell servers):

- In all cases, username=aalto username, domain=AALTO, password=Aalto password.

- smb://home.org.aalto.fi/ for your home directory

- smb://tw-cs.org.aalto.fi/project/$name/ for project directories ($name=project name)

- smb://tw-cs.org.aalto.fi/archive/$name/ for archive directories ($name=project name)

- For scratch directories, see Triton storage.

- smb://work.org.aalto.fi for “Aalto work” directories (different than Triton work).

Depending on your OS, you may need to use either your username directly or AALTO\username

On Ubuntu: Files → Left sidebar → Connect to server → use the URLs above. For other Linuxes, you can probably figure it out.

On mac laptops: Finder → Go menu item → Connect to server → use the URLs above.
On windows laptops: To do the mounting, Windows Explorer → Computer → Map network drive → select a free letter. smb:// becomes \\ (without the smb:), and / becomes \. For example, a full URL could be \\tw-cs.org.aalto.fi\project\mygroup. You can also just enter it into the file manager bar.

Remember that you must connect to the Aalto VPN first!

**Accessing you Linux workstation / Triton remotely**

- Remote access to desktop workstations is available via the university staff shell servers taltta.aalto.fi or department-specific servers magi.cs.aalto.fi (CS), amor.org.aalto.fi (NBE).
- You need to be the **PrimaryUser** of the desktop in order to ssh to it.
- Remote access to Triton is available from any Aalto server: taltta, kosh.aalto.fi, etc.
- SSHing directly to computers using openssh ProxyCommand:
  - Put this in your .ssh/config file under the proper Host line: ProxyCommand ssh taltta.aalto.fi -W %h:%p
  - For this to be most useful, you probably want to set up ssh keys, otherwise you will have to enter your password twice.
  - This starts getting beyond the basic level of ssh use, so you may want to read up on ssh keys, ProxyCommand, ControlMaster. It can make your experience much better.

**Remote Windows desktop**

Aalto has a windows remote desktop available. As usual, you must be on the Aalto VPN or Aalto networks.

- Server name rds01.org.aalto.fi (or rds02).
- login with Aalto credentials, username: AALTO\$username
- From Linux: the GUI program remmina lets you connect. Follow the instructions to add a server connection. Username is Aalto username and password is Aalto password. If you are on the Aalto network, you can turn the quality settings quite high (True color, Quality=best). Security=negotiate.
- From Linux: the command line program rdesktop can connect: rdesktop -u 'AALTO\$username' -g 1360x760 rds01.org.aalto.fi

As usual (on Linux), you can also access this directly through SSH forwarding instead of the VPN (remmina seems to be able to do this automatically):

- ssh -L 3389:rds01.org.aalto.fi:3389 kosh.aalto.fi
- rdesktop -u 'AALTO\$username' -g 1360x760 localhost

**4.1.9 JupyterHub**

**Note:** This page is about the JupyterHub for light use and teaching, https://jupyter.cs.aalto.fi. The **Triton JupyterHub** for research is documented at [Jupyter](https://jupyter.cs.aalto.fi).

https://jupyter.cs.aalto.fi is a JupyterHub installation for teaching and light usage. Anyone at Aalto may use this for generic light computing needs, teachers may create courses with assignments using nbgrader. Jupyter has a rich ecosystem of tools for modern computing.
Basic usage

Log in with any valid Aalto account. Our environment may be used for light computing and programming by anyone. Your persistent storage has a quota of 1GB. Your data belongs to you, may be accessed from outside, and currently is planned to last no more than one year from last login. You are limited to several CPUs and 1GB memory.

Your notebook server is stopped after 60 minutes of idle time, or 8 hours max time. Please close the Jupyter tab if you are not using it, or else it may still appear as active.

There are some general use computing environments. You will began with Jupyter in the /notebooks directory, which is your persistent storage. Your server is completely re-created each time it restarts. Everything in your home directory is re-created, only /notebooks is preserved. (Certain files like .gitconfig are preserved by linking into /notebooks/.home/...)

You begin with a computing server with the usual scipy stack installed, plus a lot of other software used in courses here.

You may access your data as a network drive by SMB mounting it on your own computer - see Accessing JupyterHub data. This allows you total control over your data.

JupyterHub has no GPUs, but you can check out the instructions for using the Paniikki GPUs with the JupyterHub data. These instructions are still under development.

Each notebook server is basically a Linux container primarily running a Juptyer notebook server. You may create Jupyter notebooks to interact with code in notebooks. To access a Linux bash shell, create a new terminal - this is a great place to learn something new.

Accessing JupyterHub data

Unlike many JupyterHub deployments, your data is yours and have many different ways to access it. Most importantly, the data is a normal Aalto network drive, and thus it can be accessed remotely, from your own computers.

On Paniikki and Aalto computers

TODO: eventually, on Paniikki, and the Aalto servers kosh, lyta, brute, and force, the JupyterHub data will be available automatically.

Remote access via network drive

Basic info

Note: These instructions are still being tested and updated, because people have many different computers and we don’t fully control the data storage ourselves. Please check back for updates.

You can do a SMB mount, which makes the data available as a network drive. You will have the same copy of the data as on the hub - actually, same data, so edits immediately take effect on both places, just like your home directory. You must be on an Aalto network, which for students practically means you must be connected to the Aalto VPN (see vpn instructions) or use an Aalto computer. The “aalto” wifi network does not work unless you have an Aalto computer.

- Linux: use “Connect to Server” from the file browser. The path is smb://jhnas.org.aalto.fi/$username. You may need to use AALTO\username as your username. If there is separate “domain” option, use AALTO for domain and just your username for the username.
• Mac: same path as Linux above, “Connect to Server”. Use AALTO\your_username as the username.

• Windows: \\jhnas.org.aalto.fi\$username, and use username AALTO\your_username. Windows sometimes caches the username/password for a long time, so if it does not work try rebooting.

You can also access course data and shared data by using jhnas.org.aalto.fi/course/ or jhnas.org.aalto.fi/shareddata/.

See also:
Mounting network drives in Windows is the same instructions, but for Aalto home directories. Anything there should apply here, too.

Using GPUs

One problem with our JupyterHub so far is that we don’t have GPUs available. But, because our data is available to other computers, you can use the Paniikki – Computer Lab For Students GPUs (quite good ones) to get all the power you need. To do this, you just need to make the data available on these classroom computers, and then start Jupyter or whatever you need.

First, log in to a Paniikki computer and open the file browser. Depending on your desktop, you can use “Places –> Connect to server”, or “Connect to Server from the file browser.

Then, enter the server address smb://jhnas.org.aalto.fi

Go to the directory with your username. At this point, you can set a bookmark that saves this for the future.

Now you have to start Jupyter there. To do that, start a terminal in the Jupyter directory. You can do this by right clicking and selecting “Open in Terminal”:

Now that you have the terminal and the data, you can do whatever you want with it. Presumably, you will start Jupyter here - but first you want to make the right software available. If you course tells you how to do that using an Anaconda
4.1. The Aalto environment
environment, go ahead and do it. (Please don’t go installing large amounts of software like anaconda in the Jupyter
data directories - they are for notebooks and small-medium data.)

Using the built-in anaconda, you can load the Python modules with `module load anaconda3` and start Jupyter
with `jupyter notebook`:

![Image of command line output]

### Terms of use

This service must be used according to the general IT usage policy of Aalto university (including no unlawful pur-
poses). It should only be used for academic purposes (but note that self-exploration and programming for own interests
is considered an academic purpose, though commercial purposes is not allowed). For more information, see the Aalto
policies. Heavy non-interactive computational use is not allowed (basically, don’t script stuff to run in the background
when you are not around. If you are using this service is person, it is OK). For research computing, see Triton user
guide.

### Courses and assignments

Some courses may use the nbgrader system to give and grade assignments. These courses have special entries in the
list. If you are a student in such a course, you will have a special environment for that course. Your instructor may
customize the environment, or it may one of our generic environments.

If your course is using nbgrader, there are some built-in features for dealing with assignments. Under the Assignment
list tab, you can see the assignments for your course (only the course you selected when starting your notebook server).
You can fetch assignments to work on them - they are then copied to your personal /notebooks directory. You can
edit the assignments there - fill out the solutions and validate them. Once you are done, you can submit them from the
same assignment list.

A course may give you access to a /coursedata folder with any course-specific data.

By default, everyone may access every course’s environment and fetch their assignments. We don’t stop you from sub-
mitting assignments to courses you are not enrolled in - but please don’t submit assignments unless you are registered,
because the instructors must then deal with it. Some courses may restrict who can launch their notebook servers: if
you can not see or launch the notebook server for a course you are registered for, please contact your instructor in this
case.

Note that the /notebooks folder is shared across all of your courses/servers, but the assignment list is specific to
the course you have started for your current session. Thus, you should pay attention to what you launch. Remember
to clean up your data sometimes.

### Instructors

**Jupyterhub instructions for course instructors**

See also:

Main article with general usage instructions: Jupyterhub for Teaching. For research purposes, see Triton JupyterHub.
nbgrader documentation is at https://nbgrader.readthedocs.io/, and is necessary reading to understand how to use it. It
is not duplicated here.
Basics

The JupyterHub installation provides a way to provide a notebook-based computational environment to students. It is best to not think of this service as a way to do assignments, but as a general light computing environment that is designed to be easy enough to be used for courses. Thus, students should feel empowered to do their own computing and this should feel like a stepping stone to using their own systems set up for scientific computing. Students’ own data is persistent as they go through courses, and need to learn how to manage it themselves. Jupyter works best for project/report type workflows, not lesson/exercise workflows but of course it can do that too. In particular, there is no real possibility for real-time grading and so on.

Optionally, you may use nbgrader (notebook grader to make assignments, submit them to students, collect them, autograde them, manually grade, and then export a csv/database of grades. From that point, it is up to you to manage everything. There is currently no integration with any other system, except that Aalto accounts are used to login.

You may find the book Teaching and Learning with Jupyter helpful.

Currently we support Python the most, but there are other language kernels available for Jupyter. For research purposes, see the Triton Jupyter page.

Basic course environment

The following are the properties of the course environment. To get started with a course, please read the below list and describe your needs from the relevant items (contact address is at the bottom of this page). Don’t worry too much about understanding or answering everything perfectly, there is a lot here - just let us know what you have as concisely as possible and we will work together to answer the rest.

If all you need is a Python environment to do assignments and projects, you don’t need to request anything special - students can just use the generic instance for their independent computational needs. If your course needs special packages which compatible with existing packages, let us know and we will install them. You would want a course environment if you want to (distribute assignments to students via the interface) and/or (collect assignments via the interface).

A course environment consists of:

1. A course slug (of the form nameYEAR, for example mlbp2018) and full name.
2. Description of general computational load expected - number of students, expected processor and memory usage, expected data size both on disk and in memory, expected schedule (will everyone be doing an assignment right before the deadline?). You should strongly discourage people from waiting until the last minute if you have hundreds of students in the course. We’ll provide feedback on how well we can handle the load.
3. Course schedule (lectures/exercise sessions/deadlines). We add this to our hub calendar, which is used to avoid maintenance during important times. You can check the calendar to avoid major deadlines at the same times as other courses. Note: we’ve heard that late night deadlines are bad for students well-being, so don’t make deadlines late at night just to reduce the peak load on our system.
4. (optional, recommended to use the default and add what you need) A list of required software, or a docker container containing the Jupyter stack and additional software. By default, we have an image based on the scipy stack and all the latest software that anyone else has requested, as long as it is mutually compatible. You can request additional software, and this is shared among all courses. If you need something special, you may be asked to take our image and extend it yourself. Large version updates to the image are done twice a year during holidays.
   a. (optional) A sample python file or notebook to test that the environment works for your course (which will be made public and open source). We also use use automated testing on our software images, so that we can be sure that our server images still work when they are updated. If you send us a file, either .py or .ipynb, we will add this to our automatic tests. The minimum amount is something like import of
the packages you need, a more advanced thing would test the libraries a little bit - do a minimal, quick calculation.

5. A course directory /course, available only to instructors. This comes by default, with a quota of a few gigabytes (combined with coursedata). Note: instructors should manage assignments and so on using git or some other version control system, because the course directory lasts only one year, and is renewed for the next year.

6. A list of instructors (Aalto emails or usernames). Instructors will be added to a Aalto unix group named jupyter-$courseslug to provide access control. To request new instructors, contact CS-IT and ask that people be added/removed from your group jupyter-$courseslug.
   a. Primary group owner (e.g. main instructor). Data is stored in a group according to Science-IT data policy, and this person is in charge of knowing what data exists, granting access and telling us what to do with the data long-term and they should be a long-term staff member. There can be deputies (e.g. head TA) which can grant access.
   b. People who should be added to the announcement mailing list - these will get updates for updates and maintenance breaks.
   c. Lead contact person, if different from instructor.

7. (optional, not recommended) A list of students (Aalto usernames). This can be null if anyone with an Aalto account should be able to access the environment (this is recommended to be as open as possible and to save manual effort). If you provide a list of students, you will have to request manual effort every time it changes, so this is not recommended.
   a. Should non-students be allowed to spawn the environment? Default yes…

8. Should the image start in “private mode”, where only instructors and students from the previous point can start the course environment? From our side, there is no major disadvantage to going public from the start.

9. (optional, not recommended) A list of computational resources per image. Default is currently 2GB and 4 processors (oversubscribed). Note that because this is a container, only the memory of the actual Python processes are needed, not the rest of the OS, and memory tends to be quite small.

10. Shared data directories. If you have nontrivial data which needs distributing, consider one of these shared directories which saves it from being copied over and over. The notebook directory itself can only support files of up to 2MB to prevent possible problems. If number of students times amount of data is more than a few hundred MB, strongly consider one of the data directories. Read more about this below.
   a. You can use the “shareddata” directory /mnt/jupyter/shareddata. shareddata is available in all notebooks on jupyter.cs.aalto.fi (even outside of your course) and also (eventually) other Aalto servers. This data should be considered public (and have a valid license), even though for now it’s only accessible to Aalto accounts.
   b. /coursedata is only available within your course’s environment (as chosen from the list). coursedata is also assumed to be public to everyone at Aalto, though you have more control over it.
   c. If you use either of these, you can embed the paths directly in your notebooks. This is easy for hub use, but makes it harder to copy the notebooks out of the hub to use on your own computers. This is something we are working on.

12. Time period and expiry date - default is six months after the course is over, by which time data will be removed. But if it will be used the next year, then we’ll keep it up until then. We intentionally replace the course directories every year both for security and to encourage you to use maintainable processes!
nbgrader

“nbgrader is a tool that facilitates creating and grading assignments in the Jupyter notebook. It allows instructors to easily create notebook-based assignments that include both coding exercises and written free-responses. nbgrader then also provides a streamlined interface for quickly grading completed assignments.” - nbgrader upstream documentation

Currently you should read the upstream nbgrader documentation, which we don’t repeat. We have some custom Aalto modifications (also submitted upstream) which are:

- Instructors can share responsibilities, multiple instructors can use the exchange to release/collect files, autograde, etc. Note that with this power comes responsibility - try hard to keep things organized.
- We can have the assignments in /notebooks while providing whole-filesystem access (so that students can also access /coursedata).
- Submissions are hidden from other students better.
- While not part of nbgrader, we have a way to isolate the grading process so that students can’t access other instructor files.

To use nbgrader:

- Request a course as above.
- Once you log in to your course’s environment, the per-course /course (instructors only) and /srv/nbgrader/exchange (instructors and students, if requested) are mounted.
- You can use the Formgrader tab at the top to manage the whole nbgrader process (this automatically appears for instructors). This is the easiest way, because it will automatically set up the course directory, create assignment directories, etc. But, you can use the nbgrader command line, too. It is especially useful for autograding.
- It’s good to know how we arrange the course directory anyway, especially if you want to manage things yourself without Formgrader. The “course directory” (nbgrader term) is /course. The original assignments go in /course/source. The other directories are /course/{nbgrader_step} and, for the most part, are automatically managed.
- New assignments should be in /course/source. Also don’t use + in the assignment filename (nbgrader #928).
- Manage your assignments with git. See below for some hints about how to do this.
- If you ever get permission denied errors, let us know. nbgrader does not support multiple instructors editing the same files that well, but we have tried to patch it in order to do this. We may still have missed some things here.
- To autograde from the command line, add the option --Autograde.create_student=True so that it will automatically add students to the grader database. This happens automatically if you click the lightning bolt to autograde from the Formgrader UI.
- nbgrader is not secure, because it runs the student’s code as the instructor. We have a custom-build solution at https://github.com/AaltoScienceIT/isolate-namespace, but it will require manual work. This requires a Linux computer.

Autograding is not secure right now. If you use autograding, contact us first well in advance so we can improve the documentation. Autograding is equivalent to accepting arbitrary code from all students and running it on your own computer automatically without checking input or outputs. Do this at your own risk, but we do not offer this as a secure service without our custom add-ons.

Using git

git is a version control system which lets you track file versions, examine history, and share. We assume you have
Aalto scientific computing guide

basic knowledge of git, and here we will give practical tips to use git to manage a course’s files. Our vision is that you should use nbgrader to manage the normal course files, not the students submissions. Thus, to set up the next year’s course, you just clone the existing git repository to the new /course directory. You backup the entire old course directory to maintain the old students work. Of course, there are other options, too.

Create a new git repository in your /course/ directory and do some basic setup:

```
cd /course/
git init
```

```git config core.sharedRepository group
```

You should make a .gitignore file excluding some common things (TODO: maybe more is needed):

```
gradebook.db
release/
submitted/
autograded/
feedback/
.nbgrader.log
.ipynb-checkpoints
```

The git repository is in /course, but the main subdirectory of interest is the source/ directory, which has the original files, along with whatever other course notes/management files you may have which are under /course. Everything else is auto-generated.

### Course data

If your course uses data, request a coursedata or sharreddata directory as mentioned above. You need to add the data there yourself, either through the Jupyter interface or SMB mounting of data.

If you use coursedata, just start the course environment and instructors should have permissions to put files in there. Please try to keep things organized!

If you use sharreddata, ask for permission to put data there - we need to make the directory for you. When asking, tell us the (computer readable short)name of the dataset. In the sharreddata directory, you find a README file with some more instructions. All datasets should have a minimum README (copy the template) which makes it minimally usable for others.

In both cases, you need to chmod -R a+rX the data directory so that the data becomes readable to students.

Note: after you are added to relevant group to access the data, it make take up to 12 hours for your account information to be updated so that it can be accessed via remote mounting.

### Public copy of assignments

Let’s say you want to make your assignments publicly available so that anyone can access them to follow along without being an Aalto student or being registered. This is also important because your course environment will go away after a few months - do you want students to be able to refer to it later? If so, do the below.

- change to the release/ directory and git init. Create a new repo here.
- Manually git add the necessary assignment files after they are generated from the source directory. Why do we need a new repo? Because you can’t have the instructor solutions/answers made public.
- Update files (git commit -a or some such) occasionally when new versions come out.
• Add a requirements.txt file listing the different packages you need installed for a student to use the notebooks. See the MyBinder instructions for different ways to do this, but a normal Python requirements.txt file is easiest for most cases. On each line, put in a name of a package from the Python Package Index. There are other formats for R, conda, etc, see the page.

• Then, push this release/ repo to a public repository (check mybinder for supported locations). Make sure you don’t ever accidentally push the course repository!

• Then, go to https://mybinder.org/ and use the UI to create a URL for the resources. You can paste this URL into your course info, but recommend people use our resources first if they can (see below for the reason).

• Note that mybinder has a limit of 100 simultaneous users for a repository, to prevent too much use for single organization’s projects. It’s possible that limits will change or decrease later. Either way, for Aalto primary academic purposes we should use our resources first to avoid over-burdening free resources, and students should be advised as such.

• If you have a /coursedata directory, you will have to provide these files some other way. You could put them in the assignment directory and the release/ git repository, but then you’ll need to have notebooks able to load them from two places: /coursedata or .. I’d recommend do this: import os, if os.path.exists('/coursedata'): DATADIR='/coursedata', else: DATADIR='. ' and then access all data files by os.path.join('DATADIR', 'filename.dat'). This has the added advantage that it’s easy to swap out DATADIR later, too.

Instructions and hints to instructors

• Request a course when you are sure you will use it. You can use the general use containers for writing notebooks before that point.

• The course directory is stored according to the Science-IT data policy. In short, all data is stored in group directories (for these purposes, the course is a group). The instructor in change is the owner of the group: this does not mean they own all files, but are responsible for granting access and answering questions about what to do with the data in the long term. There can be a deputy who can also grant access.

• Store your course data in a git repository (or some other version control system) and push it to version.aalto.fi or some such system. git and relevant tools are all installed in the images.

• You know that you are linked as an instructor to a course if, when you spawn that course’s environment, you get the /course directory.

• We have a test course which you can use as a sandbox for testing nbgrader and courses. No data here is private even after deleted, and data is not guaranteed to be persistent. Use only for testing. Use the general use notebook for writing and sharing your files (using git).

• The course environments are not captive: students can install whatever they want. Even if we try to stop them, they can use the general use images (which may get more software at any time) or download and re-upload the notebook files. Either way, autograding is done in the instructors environment, so if you want to limit the software that students can use, this must be done at the autograding stage or via other hacks.

  – 1) If you want to check that students have not used some particular Python modules, have a hidden test that they haven’t used the module, like: 'tensorflow' not in sys.modules.

  – 2) autograde in an environment which does not have these extra packages. Really, #2 is the only true solution. See the information under https://github.com/AaltoScienceIT/isolate-namespace for information on doing this.

• In all cases, it is good practice to pre-import all modules the students are expected to be able to use and tell students that other modules should not be imported.
• Students should use you, not us, as the first point of contact for problems in the system. Please announce this to students. Forward relevant problems to us.

• You can access your course data via SMB mounting at the URLs smb://jhnas.org.aalto.fi/course/$courseslug/files/ and the course data using smb://jhnas.org.aalto.fi/course/$courseslug/data/ (with Windows, use \ instead of / and don’t include smb://). This can be very nice for managing files. This may mess up group-writeability permissions. It will take up to half a day to be able to access the course files after your request your course.

• You are the data controller of any assignments which students submit. We do not access these assignments on your behalf, and a submission of an assignment is an agreement between you and the student.

• You should always do random checks of a fair fraction of notebooks, to avoid unexpected problems.

• You can tell what image you have using echo $JUPYTER_IMAGE_SPEC.

• A notebook can tell if it is in the hub environment if the AALTO_JUPYTERHUB environment variable is set.

Limits

• This is not a captive environment: students may always trivially remove their files and data, and may share notebooks across different courses. See above for the link to isolate-environment with instructions for fixing this.

• We don’t have unlimited computational resources, but we can try to procure what is necessary. Work as hard as you can to spread the load and de-peak deadlines. You should discuss estimated number of students and estimated deadlines (days of the week) before courses start so that we can spread the load some.

• There is no integration to any other learning management systems, such as the CS department A+ (yet). The only unique identifier of students is the Aalto username. nbgrader can get you a csv file with these usernames, what happens after that point is up to you.

• Currently there is nothing in place to return marked-up assignments to students. We can possibly make a root script to do this. Organize assignments by username and we can do the rest.

• There is currently no plagiarism detection support. You will have to handle this yourself somehow so far.

More info

Contact: CS-IT via the guru alias guru @ cs dot aalto.fi (students, contact your course instructors first).

For source code and reporting issues, see the main jupyterhub page.

See the separate instructors guide. This service may be either used as general light computing for your students, or using nbgrader to release and collect assignments.

Privacy policy

This system is managed by Aalto CS-IT. We do not store separate accounts or user data beyond a minimal database of usernames and technical logs of notebooks which are periodically removed (this is separate from your data). Your actual data is yours only and you are responsible for it. We do not access your data, but when necessary for the operation of the system, but we use and may look at file metadata such as permissions, timestamp, filename (stat filename). Your /notebooks directory may be deleted once your have been inactive for one year, and at the latest once your Aalto home directory is removed (after your account expires). Some courses will use the feedback/ directory to return assignments to you.
The use of your own data and submission of data to your course instructors is the responsibility of you and the instructors.

See the separate privacy policy document for longer, less useful information.

FAQ and bugs

- **I started the wrong environment and can’t get back to the course selection list.** In JupyterLab, use the menu bar, “Hub->Control Panel”. On the classic notebooks, use the “Control panel” button on the top right. Unfortunately JupyterLab isn’t made so intuitively and there’s not much we can do about it. (Emergency backup: you can always change the URL path to /hub/home).

- **Is JupyterLab available?** Yes, and it’s nice. There are two general use instances that are actually the same, the only difference is one starts JupyterLab by default and one starts classic notebooks by default. Course environments always use classic notebooks, because the nbgrader assignment list only works there. To switch back and forth in any notebook server, change /tree in the URL to /lab/tree. If you want to use JupyterLab with a course’s files, first start that course’s server, get the assignments, then change to JupyterLab (change the URL, or stop and restart your server).

- **Can I login with a shell?** Run a new terminal within the notebook interface.

- **Can I request more software be installed?** Yes, let us know and we will include it if it is easy. We aim to have featureful environments by default, but won’t go so far as to install large specialist software. It should be in standard repositories (conda or pip for Python stuff).

- **Are there other programming languages available?** Currently there is only Python, but we could install other open-source languages.

- **What can I use this for?** Intended uses include anything related to courses, own exploration of programming, own data analysis, and so on (see Terms of Service above). Long-term background processing isn’t good (but it’s OK to leave small stuff running, close the tab, and come back).

- **When using nbgrader, how do I know what assignments I have already submitted?** Currently you can’t beyond what is shown there.

- **Can I know right away what my score is after I submit an assignment with nbgrader?** nbgrader is not currently designed for this.

- **Are there backups of data?** Data storage is provided by the Aalto Teamwork system. There are snapshots available in .snapshot in every directory (you have to ls this directory in a shell using its full name for it to appear the first time). This service is not designed for long term data storage, and you should back up anything important because it will be lost after about one year or when your Aalto account expires. You should use git as your primary backup mechanism, obviously.

- **Is git installed?** Yes, and you should use it. Currently you have to configure your username and email each time you use it, because this isn’t persistent (because home directories are not persistent). Git will guide you through doing this. In the future, your Aalto directory name/email will be automatically set. As a workaround, run git config without the --global option in each repository.

- **I don’t see “Assignment list”**. You are probably not in a general use server instead of a course server. Stop your server and go spawn the notebook server of your course.

- **I’m getting an error code** Here are the ones we know about:
  - 504 Gateway error: The hub isn’t running in background. This may be hub just restarting or us doing maintenance. If it persists for more than 30 minutes, let someone know.

- **Stan/pystan/Rstan don’t work.** Stan needs to do a memory-intensive compilation when your program is run. We can’t increase our memory limits too much, but we have a workaround: you need to tell your program to use the clang compiler instead of the gcc compiler by setting the environment variables CC=clang and
CXX=clang++. For R notebooks, this should be done for you. For RStudio, we don’t know. For Python, put the following in your notebook:

```python
import os
os.environ['CC'] = "clang"
os.environ['CXX'] = "clang++"
```

We should set this the default, but want to be sure there are no problems first.

**More info**

Students, your first point of contact for course-related matters and bugs with JuptyerHub should be your instructors, not us. They will answer questions and send the relevant ones to us. But, if you can actively help with other things, feel free to comment via Github repositories below.

The preferred way to send feedback and development requests is via Github issues and pull requests. However, we’re not saying it’s best to give Github all our information, so you can also send tickets to CS-IT.

Students and others who have difficulty in usage outside of a course can contact CS-IT via the guru alias.

Notebooks are not an end-all solution: for an entertaining look at some problems, see “I don’t like notebooks” by Joel Grus. Most of these aren’t actually specific to notebooks and JupyterLab makes some better, but thinking hard about the downfalls of notebooks makes your work better no matter what you do.

Our source is open and on Github:

- single-user image (everything about a user’s environment)
- server itself (logging in, course profiles, etc).

**4.1.10 Remote Jupyter Notebook on shell servers**

**See also:**

We now have a *General use student/teaching JupyterHub* installation which may serve your uses more simply.

Here we describe how you can utilise Aalto computing resources for Jupyter Notebook remotely. The guide is targeted for UNIX users at the moment.

Aalto provides two “light computing” servers: brute.org.aalto.fi, force.org.aalto.fi. We demonstrate how to launch a Jupyter Notebook on brute and access it on your laptop.

```
ssh username@brute.org.aalto.fi

# Create your Kerberos ticket
kinit

# Create a session. I use tmux
tmux

# Load Anaconda
module use /work/modules/modulefiles/common/; module refresh; module load anaconda3

# Create your env
conda create -n env-name python=3.6 jupyter

# Activate your python environment
source activate env-name
```

(continues on next page)
Fig. 1: <System activity on Brute>
# Launch jupyter notebook in headless mode and a random port number

jupyter notebook --no-browser --port=12520

**Note:** You might get messages like **The port 12520 is already in use, trying another port** while starting the notebook server. In that case, take note of the port the server is running in, e.g.:

```
```

and replace “12520” below with the correct port number, 12470 in this case.

Now back to your laptop

```
# Forward the port
ssh -L 12520:localhost:12520 -N -f -l username brute.org.aalto.fi
```

Now launch your browser and go to http://localhost:12520 with your token.

### 4.1.11 Paniikki – Computer Lab For Students

Paniikki is a cutting edge computer lab in the computer science department. It is located in T-building C106 (right under lecture hall T1). This documentation is a Paniikki cheatsheet.

For more services directed at students, see *Welcome, students!*

**The name**

Paniikki means “panic” in English which is a fascinating name as people in panic are in panic. I don’t know which comes first, the space or the emotion. Anyway, people experience the both simultaneously.

**Hardware**

<table>
<thead>
<tr>
<th>CPU properties</th>
<th>Spec</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td>Intel(R) Xeon(R) CPU E5-1650 v4 @ 3.60GHz</td>
</tr>
<tr>
<td>Architecture</td>
<td>x86_64</td>
</tr>
<tr>
<td>CPU(s)</td>
<td>12</td>
</tr>
<tr>
<td>Thread(s) per core</td>
<td>2</td>
</tr>
<tr>
<td>max MHz</td>
<td>4000.0000</td>
</tr>
<tr>
<td>Virtualization</td>
<td>VT-X</td>
</tr>
<tr>
<td>L1d cache</td>
<td>32K</td>
</tr>
<tr>
<td>L1i cache</td>
<td>32K</td>
</tr>
<tr>
<td>L2 cache</td>
<td>256K</td>
</tr>
<tr>
<td>L3 cache</td>
<td>15360K</td>
</tr>
</tbody>
</table>
Fig. 2: < The blue box at the entrance is Paniikki >
Model | NVIDIA Quadro P5000
--- | ---
GPU properties | Spec
Core | GP104GL (Pascal-based)
Core clock | 1607 MHz
Memory clock | 1251 MHz
Memory size | 16384 MiB
Memory type | 256-bit GDDR5X
Memory bandwidth | 320
CUDA cores | 2560
CUDA compute capability | 6.1
OpenGL | 4.5
OpenCL | 1.2
Near GeForce Model | GeForce GTX 1080

| Memory properties | Spec |
--- | --- |
RAM | 32GiB |

Software

First thing first, you DO NOT have a sudo right in Aalto classroom machines and you never will. We provide mostly used SW and if you need more you could inquire via servicedesk@aalto.fi. We try to have good base software.

| What? | How? |
--- | --- |
Python via Anaconda | module load anaconda3 (anaconda2 for Python2) |
Python (system) | Default available |
Tensorflow | in the Python environments |

Modules

In short, module is a software environment management tool. With module you can manage multiple versions of software easily. Here are some sample commands:

| Command | Description |
--- | --- |
module load NAME | load module |
module avail | list all modules |
module spider NAME | search modules |
module list | list currently loaded modules |
module show NAME | details on a module |
module help NAME | details on a module |
module unload NAME | unload a module |
module save ALIAS | save module to this alias (saved in ~/.lmod.d/) |
module restore ALIAS | load saved module set (faster than loading individually) |
module purge | unload all loaded modules (faster than unloading individually) |

There are some modules set up specifically for different courses: if you just load the environment, you will have everything you need.
Read the details in Module environment page.

**Example 1**

Assume we are in Paniikki and wants to do our homework for CS-E4820 Machine Learning: Advanced probabilistic methods. In the course students use Tensorflow and Edward.

```
# Check available modules
$ module load courses/     # Tab to auto-complete

# Finally you will complete this
$ module load courses/CS-E4820-advanced-probabilistic-methods.lua

# Check the module you loaded
$ module list

Currently Loaded Modules:
   1) courses/CS-E4820-advanced-probabilistic-methods

# Check the packages
$ conda list     # You will see Tensorflow and etc.

# Launch Jupyter
$ jupyter notebook

# Do your homework

# You are done and want to un-load all the modules?
$ module purge
```

**Example 2: general Python software**

Need Python and general software? The anaconda modules have Python, a bunch of useful scientific and data packages, and machine learning libraries.

```
# Latest Python 2
$ module load anaconda2

# Latest Python 3
$ module load anaconda3
```

**Example 3: List all software**

You can check all other modules as well

```
$ module avail

You want to use Matlab?

$ module load matlab/2017b
$ matlab
```
Fig. 3: <Available modules in Paniikki as of 2018 March 8th>

Questions?

If you have any question please contact seyoung.park@aalto.fi.

4.1.12 HTCondor

Note:

- SCIP courses: look for *Introduction to distributed computing with HTCondor*

Introduction

HTCondor (formerly known as just Condor) is a computing scheduler developed at University of Wisconsin-Madison. This allows users to run their binaries on Aalto Linux workstations without explicit logging to desktop machines. Condor takes care of choosing the right workstation, setting correct job priority and taking care of cleaning the output. Condor distributes, schedules, executes and returns the result. So handmade farming is not needed.

HTCondor status at Aalto and support

Condor installations are department specific. Here is a list of departments that have HTCondor software installed on their Ubuntu workstations.
<table>
<thead>
<tr>
<th>Department / school</th>
<th>Support contact</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>PHYS &amp; NBE / SCI</td>
<td>Aalto IT servicedesk *</td>
<td>joint installation, installed on all the Ubuntu workstations</td>
</tr>
<tr>
<td>CS / SCI</td>
<td>Aalto IT servicedesk *</td>
<td>installed on all the Ubuntu workstations</td>
</tr>
<tr>
<td>MATH / SCI</td>
<td>Matti Harjula and Kenrick Bingham</td>
<td>installed on about 50 newer Ubuntu workstations</td>
</tr>
</tbody>
</table>

The instructions below are common to all the departments if not mentioned otherwise.

* Getting help: your department IT guys have responsibility over the HTCondor installation. Best way to reach them is to drop an email to the Aalto IT servicedesk including info like: your department, Linux workstation name and type of problem.

**HTCondor official manuals**

The detailed manual can be found from [http://www.cs.wisc.edu/condor/manual/](http://www.cs.wisc.edu/condor/manual/). Current version of Condor we have can be checked with `condor_q -version`.

**Before you run with Condor**

It is recommended that you compile your binary statically. If you have used shared libs (or you get from someone code that has not been compiled statically), make sure that you set your environment correctly and use `getenv = true` option in Condor submit script.

No large MPI jobs (over the net) are allowed with Condor. For any large MPI or multithread job, please either run on your local workstation only or on other resources like Triton.

Condor is well suited for short time serial runs (like overnight), or for small (2-4 CPUs) parallel runs that can be run within one machine. Long runs (over 12 hours) are possible, but remember that Condor runs on local workstations, and uses only idle CPU cycles, i.e. some currently unused workstation during the day and all of them during night. Local usage is of higher priority and thus submitted Condor job that hurts local user will be suspended.

Always use `should_transfer_files = yes` in your Condor submit script. This way you make sure that all IOs will go to local directory assigned to HTCondor on a local worker instead of shared NFS (be it /home or alike).

**Run your code with Condor**

- Discover condor pool status with `condor_status` or with `condor_status -available` to find out which machines are available for jobs. This step is to make sure that condor pool is available.
- Compile a statically linked binary.
- Create a condor submission script, like `job.cond` below
- Submit the job to condor pool with `condor_submit job.cond`
- Manage your job(s) with `condor_q`, `condor_rm`

It may take several minutes for code to start running. Check out `condor.log` for any useful log information.

**Job script examples**

CS users should use `universe = local`

---

4.1. The Aalto environment 65
# job_1.cond -- ready to run serial code example

executable = serial.bin
universe = vanilla
output = serial.out
error = serial.err
log = condor.log
should_transfer_files = YES
queue

# job_2.cond -- Condor serial job submission script example

# define job specific vars to be used later in this script
# this should be an absolute path, or path from current working dir
DIR=myrun

# setting up base directory for input, output, error and log files, executable path is not affected
initialdir = $(DIR)

# Define executable to run, it can be arch specific, or just some generic code
executable = mycode

# memory requirements, if any
#request_memory = 512 MB

# Condor universe. Default Vanilla, others haven't been configured/tested
universe = vanilla

# the file name specified with 'input' should contain any keyboard input the program requires
# note, that command-line arguments are specified by the 'arguments' command below
input = input.txt

# and output files
# note, that input, output, log and error files will/should be in 'initialdir'
output = $(cluster).out
error = $(cluster).err

# Errors, if any, will go here
log = condor.log

# email for job notifications, when it is completed or finished with errors
#notify_user = firstname.lastname@aalto.fi
#notification = Complete

# Additional environment vars
#environment = "PATH=$ENV(PATH):/home/user/bin"

# replicate your current working environment on the worker node
# useful when you have some specific vars like PATH, LD_LIBRARY_PATH or other defined with 'module'
getenv = true

# code arguments, if any

(continues on next page)
## Aalto scientific computing guide

(continued from previous page)

```bash
#arguments = -c cmd_input.conf

# Transferring your files to a system the job is going to run on
# that is the recommended method, to avoid NFS traffic
should_transfer_files = yes
transfer_input_files = cmd_input.conf,input.txt
when_to_transfer_output = ON_EXIT_OR_EVICT

# Some specific requirements, if any. By default Condor will run job on a machine which has
# the same architecture and operating system family as the machine from which it was submitted.
# Here is we want the worker node would be Ubuntu 12.04 with 4 CPU cores or more
#requirements = (OpSysLongName >= "Ubuntu 12.04") && (TotalCPus >= 4)
```

### Condor commands

- `condor_q -analyze <condor_job_id>` # your running/pending jobs diagnostics (for all your jobs at once if job_id is missing)
- `condor_q -global` # list all/everyone’s jobs at pool
- `condor_q -version` # find out installed condor version
- `condor_status -available` # list available computers for your job
- `condor_status -state -total` # Condor pool resources in total
- `condor_status HOSTNAME` # show status for a specific host (HOSTNAME.hut.fi in this case), where number of slots gives number of CPU cores available
- `condor_status -long vesku` # show all details for a specific host
- `condor_status -constraint 'OpSysLongName>="Ubuntu 12.04"'` # list Ubuntu 12.04 workstations only
- `condor_rm <condor_job_id>` # remove particular job
- `condor_rm -all` # remove all user jobs
- `condor_rm -constraint 'JobStatus != 2'` # remove all user jobs that are not currently running
- `condor_hold <job_id>` # hold your Condor job(s) in the queue
- `condor_release <job_id>` # release job(s) previously holded in the queue
- `condor_history` # list the completed jobs submitted from the workstation you run this command on

### Startup script requirements can be always tested with condor_status -constraint. Like in the above job_2.cond example:
- `condor_status -constraint '(OpSysLongName>="Ubuntu 12.04") && (TotalCPus >= 4)斩 availability`
More commands and their usage examples you can find at Condor User Manual.

Additional “requirements”/“constraints” options that have been configured on PHYS workstations only: CPUModel, CPUModelName, TotalFreeMemory. The later one in MB, reports currently available free memory according to /proc/meminfo. Can be useful for large memory jobs, see example below.

```bash
# ask for machine with more than 4GB of free memory
requirements = (TotalFreeMemory >= 4000)
```

**Checkpointing and condor_compile**

HTCondor has no checkpointing or remote system calls support on Ubuntu (according to manual pages).

**HTCondor config**

Machine in considered to be free if: no user activity within 15 min (keyboard or mouse), average load < 30%, and no condor job already running.

Running job will be suspended if: local workstation user became active (on hold) or CPU busy for more than 2 min and job has been running more than 90 sec.

Suspended job will be resumed if: machine has been free for 5 min.

Suspended job is killed if: it has been suspended for 4 hours (Vanilla universe) or hasn’t completed checkpointing within 10 min (Standard universe) or higher priority job is waiting in the queue.

Job will be preempted if: it uses more memory than available for its slot (killed and send back to queue).

**FAQ**

**Condor has support on running jobs under shared filesystem. Should I use this?**

This is a bad idea. Keep using Condor’s default local directory (somewhere on the local harddrive, department specific settings), otherwise, several jobs using NFS constantly (either home or any other remotely mounted) would make it really slow. Use

```bash
should_transfer_files = YES
transfer_input_files = file1.dat,file2.txt
```

options instead. Then condor will copy all required (specified) files to its local spool directory and run jobs locally. Only when finished, it will return files back to the original submitting directory. This original submitting directory should not be a NFS mounted directory such as your home directory, as in the Aalto environment those are mounted with Kerberos security, and if the Kerberos ticket has expired because you aren’t working on your workstations, condor will not be able to access this directory and your job results will be lost.

**My job is in ‘Idle’ state, while there are resources available**

Job may take several minutes to start, if it takes longer, check out job log (defined with log = directive in the submit script) and then run `condor_q -analyze <job_id>` to see possible reasons. More debugging options at condor_q manual.
I've copy/pasted example files from this page, but when try to run they produce some errors

Should be this wiki specific. Noticed (with `cat -A filename`) that copy/pasted text includes bunch of non-ascii characters.

Got it fixed with `perl -pi -e 's/\[[[:^ascii:]]\] //g' filename`

**Additional files/scripts**

Files that may be useful with condor:

- **cq** – A script that works as `condor_q` but also prints the executing host

```perl
#!/usr/bin/perl

use POSIX;
$user=$ENV{'LOGNAME'};
$now=`date +%s`;
$now=~s/\n//;
$str=" -cputime -submitter $user ";
for $i (0..$#ARGV) {
   $str.= $ARGV[$i-1];
}
if($ARGV[0] eq "all") {$str=" -global -cputime -currentrun";}
if($ARGV[0] eq "j") {system("condor_q -global -cputime -currentrun -submitter
   $user|egrep '(jobs|Schedd)'"),exit(0);}
if($ARGV[0] eq "rm") {$str=`condor_q -submitter $user -format "%d\n" ClusterId|xargs`;print "condor_rm $str";exit(0);}

foreach(`condor_q -long $str`) {
    s/\n//;
    s/\="/g;
    if(m/^Iwd\s*=\s*(\S+)/) { $iwd=$1; } 
    if(m/^RemoteHost\s*=\s*(\S+)/) { $rh=$1; } 
    if(m/ServerTime/) {
        $iwd=~/\s*\.(\S+)$/; 
        push(@iwds, "$rh\t $iwd")
    }
}
foreach(`condor_q $str`) {
    s/\n//;
    if(/\s*d\+/.d/) {
        $iwd=shift(@iwds);
        $_.=" 
    }
    print "$_.\n";
}
sub runtime() {
```

(continues on next page)

4.1. The Aalto environment 69
my($now, $st)=@_;  
$str=localtime($now-$st-7200);  
$str =~ s/\t/ /g;  
$str =~ s/\s*//g;  
$str =~ s/\s+/ /g;  
$split(/ /,$str);  
$d=$_[2]-1;  
$t=$_[3];  
if($d>0) { $ret="$d+$t"; } else { $ret=$t; }  
return $ret;

• turbomole.cond, run_ridft510_condor.scr—pair of scripts for running TurboMole or AMBER (thanks to Markus Kaukonen)

```bash
#!/bin/sh
source /etc/profile
source /etc/bashrc
source /etc/profile.d/fyslab-env.sh

AMBERHOME=${HOME}/bin/Amber10
TURBODIR=${HOME}/bin/Turbo5.10/

PATH=$PATH:$TURBODIR/scripts
PATH=$PATH:$TURBODIR/bin/`sysname`

export PATH
export PATH="${AMBERHOME}/exe:${AMBERHOME}/bin:${PATH}"
export PATH="${HOME}/bin:${PATH}"

ulimit -s unlimited
#ulimit -a > mylimits.out
jobex -ri -c 200 > jobex.out
```

and run_ridft510_condor.scr
4.1.13 Aalto Gitlab

https://version.aalto.fi is a Gitlab installation for the Aalto community. Gitlab is a git server and hosting facility (an open source Github, basically).

Note:

- This page is about https://version.aalto.fi, the Aalto gitlab installation.
- scicomp/git contains our pointers for Git usage in general.
- Git migration contains information on switching from subversion or other git repositories to Gitlab.

Git in general

Git seems to have become the most popular and supported version control system, even if it does have some rough corners. See the general git page on this site for pointers.

Aalto Gitlab service

Aalto has a self-hosted Gitlab installation at https://version.aalto.fi, which has replaced most department-specific Git-labs. With Aalto Gitlab, you can:

- Have unlimited private repositories
- Have whatever groups you need
- Get local support

The Aalto instructions can be found here, and general gitlab help here.

All support is provided by Aalto ITS. Since all data is stored within Aalto and is managed by Aalto, this is suitable for materials up to the “confidential” level.

Extra instructions for Aalto Gitlab

Always login with HAKA wherever you see the button. To use your Aalto account otherwise, use username@aalto.fi and your Aalto password (for example, use this with https pushing and pulling). But, you really should try to configure ssh keys for pushing and pulling.

For outside/public sharing read-only, you can make repositories public.

If you need to share with an outside collaborator, this is supported. These outside partners can access repositories shared with them, but not make new ones. They will get a special gitlab username/password, and should use that with the normal gitlab login boxes. To request an collaborator account, their Aalto sponsor should go here to the request form (employees only). (You can always set a repository as public, so anyone can clone. Another hackish method is to add ssh deploy keys (read-only or read-write) for outside collaborators, but this wouldn’t be recommended for serious cases.)

For public projects where you want to build a community, you can also consider Github. There’s nothing wrong with having both sites for your group, just make sure people know about both. Gitlab can have public projects, and Github can also have group organizations.

NOTE! If your work contract type changes (e.g. staff -> visitor, student->employee, different department), the Aalto Version blocks the access as a “security” measure. Please contact Aalto ITS Servicedesk <servicedesk@aalto.fi> to unblock you. This is annoying, but can’t be fixed yet.
The service doesn’t have quotas right now, but has limited resources and we expect everyone to use disk space responsibly. If you use too much space, you will be contacted. Just do your best to use the service well, and the admins will work with you to get your work done.

**CodeRefinery Gitlab and Gitlab CI service**

CodeRefinery is a publicly funded project (by Nordforsk / Nordic e-Infrastructure Collaboration) which provides teaching and a GitLab platform for Nordic researchers. This is functionally the same as the Aalto Gitlab and may be more useful if you have cross-university collaboration, but requires more activation to set up.

They also have a Gitlab CI (continuous integration) service which can be used for automated building and testing. This is also free for Nordic researchers, and *can be used even with Aalto Gitlab*. Check their repository site info, if info isn’t there yet, then mail their support asking about it.

**Recommendations**

version.aalto.fi is a great resource for research groups. Research groups should create a “Gitlab group” and give all their members access to it. This way, code and important data will last longer than single person’s time at Aalto. Add everyone as a member to this group so that everyone can easily find code.

Think about the long term. Will you need access to this code in 5 years, and if so what will you do?

- If you are a research group, put your code in a Gitlab group. The users can constantly switch, but the code will stay with the group.
- If you are an individual, plan on needing a different location once you leave Aalto. If your code can become group code, include it in the group repository so at least someone will keep it at Aalto.
- Zenodo is a long-term data archive. When you publish projects, consider archiving your code there. (It has integration with Github, which you might prefer to use if you are actually making your code open.) Your code is then citeable with a DOI.
- In all cases, if multiple people are working on something, think about licenses at the beginning. If you don’t, you may be blocked from using your own work.

**FAQ**

- **What password should I use?** It is best to use HAKA to log in to gitlab, in which case you don’t need a separate gitlab password. To push, it is best to use ssh keys.
- **My account is blocked!** That’s not a question, but Gitlab blocks users when your Aalto unit changes. This is unfortunately part of gitlab and hasn’t been worked around yet. Mail servicdesk@aalto.fi with your username and request “my version.aalto.fi username XXX be unblocked (because my aalto unit changed)” and they should do it.
- **What happens when I leave, can I still access my stuff?** Aalto can only support it’s community, so your projects should be owned by a group which you can continue collaborating after you leave (note that this is a major reason for group-based access control!). Email servicdesk for information on what to do to become an external collaborator.
- **When are accounts/data deleted?** The deletion policy is findable in the privacy policy. In 2017, it’s 6 months after Aalto account closed, 24 months after last login, or 12 months after last login of an external collaborator.
- **Are there continuous integration (CI) services available?** Not from Aalto, but the CodeRefinery project has free CI services to Nordics, see their site and the description above.
4.1.14 Python

The scientific python ecosystem is also available on Aalto Linux workstations, including the anaconda2 and anaconda3 modules providing the Anaconda python distribution. For a more indepth description see the generic python page under scientific computing docs.

The “neuroimaging” environment

On the Aalto Linux workstations there exists a conda environment under the anaconda3 module called “neuroimaging” which contains an extensive collection of Python packages for the analysis of neuroimaging data, such as fMRI, EEG and MEG.

To use it:

```
$ ml purge
$ ml anaconda3
$ source activate neuroimaging
```

To see the full list of packages what are installed in the environment, use:

```
$ conda list
```

Some highlights include:

- Basic scientific stack
  - numpy
  - scipy
  - matplotlib
  - pandas
  - statsmodels
- fMRI:
  - nibabel
  - nilearn
  - nitime
  - pysurfer
- EEG/MEG:
  - mne
  - pysurfer
- Machine learning:
  - scikit-learn
  - tensorflow
  - pytorch
- R:
  - rpy2 (bridge between Python and R)
  - tidyverse
Finally, if you get binaries from the wrong environment (check with `which BINARYNAME`) you may need to update the mappings with:

```
$ rehash
```

**Mayavi**

If you experience problems with the 3D visualizations that use Mayavi (for example MNE-Python’s brain plots), you can try forcing the graphics backend to Qt5:

- For the Spyder IDE, set Tools -> Preferences -> Ipython console -> Graphics -> Backend: Qt5
- For the ipython consoles, append `c.InteractiveShellApp.matplotlib = 'qt5'` to the `ipython_config.py` and `ipython_kernel_config.py` configuration files. By default, these can be found in `~/.ipython/profile/default/`.
- In Jupyter notebooks, execute the magic command `%matplotlib qt5` at the beginning of your notebook.

**Installation of additional packages**

The “neuroimaging” environment aims to provide everything you need for the analysis of neuroimaging data. If you feel a package is missing that may be useful for others as well, contact Marijn van Vliet. To quickly install a package in your home folder, use `pip install <package-name> --user`.

**Notes for admins**

The trick to getting Mayavi to play nicely with a modern Python environment is to install it from Git:

```
$ ml purge
$ ml anaconda3
$ source activate neuroimaging
$ pip install git+https://github.com/enthought/traits.
   →git@a7a83182048c08923953e302658b51b68c802132
$ pip install git+https://github.com/enthought/pyface.
   →git@13a064de48adda3c880350545717d8cf8929afad
   →git@ee8ef0a34dfc1db18a8e2c0301cc18d96b7a3e2f
$ pip install git+https://github.com/enthought/mayavi.git
```

### 4.1.15 Open Source at Aalto

**Note:** This policy was developed at the Department of Computer Science, in conjunction with experts from Research and Innovation services (both the legal and commercialization sides) with the intention of serving the wider community.

After more research, we have learned that this policy is, in fact, de-facto applicable to all of Aalto, it is just extremely unclear that open source is actually allowed. Thus, this policy can be seen as best practices for all of Aalto. However, everyone (including CS) has more rights: one does not have to use this policy. You don’t have to use an open source license. IP ownership may be in more limited hands, so that you need fewer agreements to release.
However, we strongly encourage you to use this policy anyway. If you use this, you know that you are safe and have all permissions to make open source, regardless of your particular funding situation. It also ensures that you make proper open source software, for maximum benefit and open science impact.

References at bottom.

Researchers make at least three primary outputs: publications, software, and data. This policy aims to make openly releasing all types of work as straightforward the traditional academic publishing process.

This document describes the procedure for Aalto employees releasing the output of their work openly (open source software, data, and publications). Aalto University encourages openness. This policy covers only cases where work can clearly be released openly with no bureaucracy needed. It does not cover complex cases, such as commercial software, work related to inventions, complex partnership agreements, etc. The policy is voluntary, and provides a right to release openly, but does not require it or preclude any other university process. (Thus it’s more of a guideline than a policy.) It only is relevant when the creator has an employment relationship with Aalto. If they don’t (e.g. students), they own their own work unless there is some other agreement in place (e.g. their own funding contract, grant, etc). Still, they can use this same process with no extra bureaucracy needed.

We realize that this policy does not cover all cases. We aim to cover the 99% case, and existing processes are used for complicated cases. Aalto Innovation Services provides advice on both commercialization and open source release.

This policy is for public licensing only (one to many). You must go through Research and Innovation Services for anything involving a multi-party agreement.

Why release?

The more people who see and build on our work, the more impact we can have. If this isn’t enough, you get more citations and attention. While we can’t require anything, we strongly encourage that all work is either made open source or taken through the commercialization process. If you don’t know what to do, don’t worry: they are not mutually exclusive. Proper open-source licensing can protect your ability to commercialize later. Talk to Innovation Services. They like open source, too, and will help you find the right balance. Anyway, if work matches the criteria in this policy, it probably has limited commercial potential anyway: what is more important is your own knowledge and skills that went into it.

You want to add a proper open source license to your work, rather than just putting code on some webpage. Without a license, others can not build on your code, making your impact limited. No one will build on you, and eventually your work rots and gets lost.

You always want to go through this process as soon as possible at the beginning of a project: if you don’t, it becomes much harder to track everyone down.

You shouldn’t release as open source (yet) if your work is intentionally commercial or contains patentable inventions. In these cases, contact Innovation Services. In the second case (patentable inventions), according to Finnish legislation you are actually required to report the invention to Innovation Services.

Traps and acting early

Intellectual property rights don’t give you the right to do anything - they give you the right to block others from doing something. Thus, it is very important that you don’t end up in a situation where others can block you, and that means thinking early.

Decide on a license as soon as possible. Once it goes into the repository, future contributors implicitly agree to it. Otherwise, you are stuck trying to find all past contributors and get their agreement.

Another common trap is non-open source friendly grants. Not many outright ban it, but some require permission from all partners, and if there are a lot then this becomes close to impossible. Ask in advance, but in the worst case it might be you just can’t write software at the times you are paid by these projects!
Step-by-step guide for release under this policy

1. Do these steps at the beginning of your project, not at the end!

2. Check if the work is covered under the “conditions for limited commercial potential” in the policy.

3. Choose a proper license to match your needs. See below for information. It must be open source, and you can not transfer any type of exclusive license away - Aalto keeps full right to future use.

4. Get the consent of all authors and their supervisors and/or funders. There are no particular requirements for this, the only need is proving it later in case a question ever arises. You should also make sure that your particular funding source/collaboration agreements don’t have any further requirements on you. (For example, some grant agreements may say no GPL-type licenses without consent of all partners.) Your advisor (and Research and Innovation Services) can help you with this.

If you are funded by Aalto basic funding, you by default have permission. Same goes for other big public funding agencies (Academy, EU… but the grant can always override this).

If you are in services, follow your source of funding. At the very worst, whoever is responsible for your funding can decide, but it may be someone lower too.

5. You are responsible for making sure that you have the right to release your code. For example, that any other included software has compatible licenses.

6. Put a copyright license in the source repository. In the best case, each individual source file should list copyright and authors, but in practice if you don’t do this it’s not too much of a problem. Make sure that the license disclaims any warranty (almost all licenses will do this). After this, contributors implicitly consent to the license. If you have an important case, ask explicitly too. The important thing is that you have more evidence than the amount of scrutiny you might get (low in typical projects, will be higher if your project becomes more important).

7. This policy is seen as Aalto transferring the rights to you to release, not Aalto releasing itself (just the same as with publications). Release in your own name, but you can(+should) list your affiliations.

8. Make your code public if/when you want. No particular requirements here, but see below for best practices.

Any borderline or questionable cases should be handled by the existing innovation disclosure process.

In addition to the above requirements, the following are best practices:

1. You can’t require that people cite you, but you can ask nicely. Make it easy to do this! Include the proper citations directly in the README. Make your code itself also citeable by publishing it somewhere (Github, Zenodo, . . . ).

2. Put on a good hosting location and encourage contributions. For example, Github is the most popular these days, but there are plenty of others. Welcome contributions and bug reports, and build on them. Make yourself the hub of expertise of your knowledge and methods.

Choosing a license

Under this policy, any Creative Commons, Open Source Initiative, and Free Software Foundation approved open source licenses are usable. However, you should not try to be creative, and use the most common license that serves your needs.

Top-level recommendations:

1. Use this nice site: https://choosealicense.com/. It contains everything you need to know, including what is here.

2. MIT for software which should be basically public domain, Apache 2.0 for larger almost-public domain things (the Apache license protects more against patent trolling). Anyone can use this for any purpose, including putting it in their own proprietary, non-open products.
3. **GNU General Public License (GPL)** (“v2 or any later version”) for software which you may try to commercialize in the future. This license says that others can not make it closed-source without your consent. Others can use it for commercial purposes, but all derivative work must also be made open source - so you keep an advantage.

For special cases:

1. **Lesser GNU General Public License** (LGPL, GPL with classpath exception) type licenses. Suitable where the GPL would be appropriate but the software is a library. It can be embedded within other proprietary products, but the code itself must stay open.

2. The **Affero GPL/LGPL**. These get around the “webservice loophole”: if your code is available via a webservice, the code running it must stay open.

3. **CC-BY** for other non-software output.

Discussion:

- Most public domain → MIT / Apache 2 > CC-BY > LGPL > GPL > AGPL → Most protection against proprietary use

- If you think you might want to commercialize in the future: ask innovation services and they’ll help you release as open source now and preserve commercialization possibilities for the future.

### The policy

#### Open Source Policy

#### Covered work

1. Software

2. Publications and other writing (Note that especially in this case, it is common to sign away full rights. This is a case where you do more than this policy says.)

3. Data

#### Conditions for limited commercial potential

This policy supports the release of work with limited commercial potential. Work with commercial potential should be assessed via Aalto’s innovation process.

1. If work’s entire novelty is equally contained in academic publications, there is usually little commercial value. Examples: code implementing algorithms, data handling scripts.

2. Similarly, work which only is a byproduct of academic publications or other work probably has limited commercial value, unless some other factor overrides. For example: analysis codes, blog posts, datasets, other communications.

3. Small products with limited independent value. If the time required to reproduce the work is small (one week or less), there is likely not commercial value. For example: sysadmin scripts, analysis codes, etc. Think about the time for someone else to reproduce the work given what you are publishing, not the time it took for you to create it.

4. Should a work be contributing to an existing open project, there is probably little commercial value. For example: contribution to existing open-source software, Wikipedia edits, etc.

---

4.1. The Aalto environment
5. NOT INCLUDED: Should work contain patentable elements or have commercial potential, this policy does not apply and it should be evaluated according to the Aalto innovation process. Patentable discoveries are anything which is a truly new, non-obvious, useful inventions. In case of doubt, always contact Innovation Services! Indicators for this category: actually novel, non-obvious, useful, and actually an invention. Algorithms and math usually do not count, but expressions of these can.

6. NOT INCLUDED: Software designed for mass-market consumption or business-to-business use should be evaluated according to the Aalto innovation process. Indicators for this category: large amount of effort, software being a primary output.

Ownership of intellectual property rights at Aalto

1. This policy covers work of employees whose contracts assign copyright and other intellectual property rights of their work to Aalto. However, the Aalto rules for ownership of IP are extremely difficult, so see the last point.

2. Your rights are assigned to Aalto if you are funded by external funding, or if there are other Aalto agreements regarding your work.

3. If neither of the points in (2) apply to you AND your work is independent (self-decided and self-directed), then according to Finnish law you own all rights to your own work. You may release it how you please, and the rest of this policy does NOT apply (but we recommend reading it anyway for valuable advice). Aalto Innovation Services can serve you anyway.

4. Rather than figure out the the ownership of work, this policy is written to apply to all work, so that you do not need to worry about this.

Release criteria and process

1. This policy applies to copyright only, not other forms of intellectual property. Should a work contain other intellectual property (which would not be published academically), this policy does not apply. In particular, this policy does not cover any work which contains patentable inventions.

2. The employee and supervisor must consider commercial potential. The guidelines in the “conditions for limited commercial potential” may guide you. Should there be commercial potential, go through the existing innovation disclosure processes. In particular, any work which may cover patentable inventions must be reported first.

3. If all conditions are satisfied, you, in consultation with your PI, supervisor, or project leader (whichever is applicable) and any funder/client requirements, may choose to release the work. Should the supervisor or PI have a conflict of interest or possible conflict of interest, their supervisor should also be consulted.

4. Depending on funding sources, you may have more restrictions on licensing and releasing as open source. Project proposals and grant agreements may contain provisions relevant to releasing work openly. When making project proposals, consider these topics already. When in doubt, contact the relevant staff.

5. To be covered under this policy, work must be licensed under a open/open source/free software license. In case of doubt, Creative Commons, Open Source Initiative, and Free Software Foundation approved open source licenses are considered acceptable. See below for some license recommendations.

6. All warranty must be disclaimed. The easiest way of doing this is by choosing an appropriate license. Practically all of them disclaim warranty.

7. All authors must consent to the release terms.

8. The employee should not transfer an exclusive license or ownership to a third party. Aalto maintains the right to relicense and use internally, commercially, or re-license should circumstances change.

9. Employees should acknowledge their Aalto affiliation, if this possible and within the community norms.
10. This right should not be considered Aalto officially releasing any work, but allowing the creators to release it in their own name. Thus, Aalto does not assume liability or responsibility for work released in this way. Copyright owner/releaser should be listed as the actual authors.

11. Employees are responsible for ensuring that they have the right to license their work as open source, for example ensuring that all included software and data is compatible with this license and that they have permission of all authors. Also the release must be allowed by any relevant project agreements. Should you have any doubts or concern, contact Innovation Services.

To apply this to your work, first receive any necessary permissions. In writing, by email, is sufficient. Apply the license in your name, but list Aalto University as an affiliation somewhere that makes sense. Do not claim any special Aalto approval for your work.

For clarity, raw official text is separate from the guidance on this page. Current approvals: Department of Computer Science (2017-03-17).

How to run a good open-source software project

One of the largest benefits to open source is having a community of people contributing back to you. To do this, you need to have a good environment. Open development, good style and a basic contribution guide, and encouragement is the base of this. Eventually, this section may contain some more pointers to how to create this type of community. (TODO)

References

- Aalto IP guide: FI EN: contains evidence that this policy is applicable to all Aalto.
- Aalto Innovation Services: http://innovation.aalto.fi/
- Choosing an open source license: http://choosealicense.com/
- Aalto copyright advice: http://copyright.aalto.fi/
- Practical guidelines for Open Source Projects: forthcoming, 2017

4.1.16 Standalone Matlab

General matlab hints: http://math.aalto.fi/opetus/Mattie/MattieO/matlab.html

Installation and license activation on staff-owned computers

Matlab academic license permits installation on home computers for university personnel. Triton MDCS workers are available to anyone with a Triton account, which means the workers can be utilized from personal laptops as well.

Download image

Log into http://download.aalto.fi/ with your Aalto account. Look for the link Software for employees’ home computers which will take you to the Matlab download links. Download the UNIX version for Linux and OSX or the separate separate image for Windows.

The ISO image can be burned on a DVD or mounted on a virtual DVD drive.
**Aalto scientific computing guide**

- **Windows:** Use MagicDisk or Virtual CloneDrive OR burn the image on DVD. Double click on setup.exe icon.
- **Linux:**
  ```bash
  # sudo mkdir /mnt/loop
  # sudo mount -o loop Download/Matlab_R2010b_UNIX.iso /mnt/loop
  # sudo /mnt/loop/install.sh
  ```
- **Mac OS X:** Double click on InstallForMacOSX.app icon.

**Installation steps**

Select the installer options as shown in the screenshots.

Mathworks account is required to continue with the installation.

- Enter your account information in the installer to log in. If the password has been lost, Click on the *Forgot your password?* option to receive your password in email. *OR*
- Register to Mathworks with the installer.
  1. Click on *I need to create an account.*
  2. Enter your name and email address. To be recognized as Aalto academic user the email address must end in one of aalto.fi, tkk.fi, hut.fi, hse.fi, hkkk.fi or uiah.fi domains.
  3. The installer will ask for an activation key, which is shown here in the last screenshot.

You may leave out unnecessary toolboxes and change the installation location. Remember however, that the Parallel Computing Toolbox is necessary to run any Matlab batch jobs on Triton.

**Install Triton-MDCS integration scripts**

Continue MDCS setup from Matlab Distributed Computing Server.

**FAQ**

**Matlab freezes with Out of Memory errors**

Q: Matlabs freezes and I get errors like this. What to do?:

```java
Exception in thread "Explorer NavigationContext request queue" java.lang.OutOfMemoryError: GC overhead limit exceeded
  at com.mathworks.matlab.api.explorer.FileLocation.<init>(FileLocation.java:89)
  at com.mathworks.matlab.api.explorer.FileLocation.getParent(FileLocation.java:126)
  ... ... ...
```

A1: Add more memory in Home -> Preferences -> General -> Java Heap memory
A2: Can you free up memory in your code sooner using the `clear` command? [https://se.mathworks.com/help/matlab/ref/clear.html](https://se.mathworks.com/help/matlab/ref/clear.html)
GPU acceleration?

Q: is there functional GPU acceleration? Does the acceleration even work?

A: run code:

```matlab
>> g = gpuDevice;
>> ng
```

A2: Just query some feature:

```matlab
>> fprintf(' %s

', g.ComputeCapability)
```

a3: Show multiple devices if found:

```matlab
>> for ii = 1:gpuDeviceCount
    g = gpuDevice(ii);
    fprintf(ii,'Device %i has ComputeCapability %s \n',
    g.Index,g.ComputeCapability)
end
```

4.1.17 Overleaf

Aalto provides an professional» site license to all the community. For more information, see https://www.overleaf.com/edu/aalto.

In order to link yourself to Aalto, you must register for and have an OrcID [wikipedia]. An OrdID (“Open Researcher and Contributor ID”) is some permanent ID which is used for linking researchers to their work, for example, some journals require linking to an OrcID. OrdID can be accessed directly with your Aalto account.

TODO: determine exact procedure and update here

Aalto rates overleaf as for “public” data. This doesn’t mean that Overleaf makes your data public, but just that Aalto can’t promise security. In reality, you decide if Overleaf is secure enough. If there is some legal requirement for security, you probably shouldn’t use Overleaf. If there is a collaborator requirement for security, then you must make your own choice if Overleaf is suitable.

Cheatsheets: CS, Data.
In this section, you can find some information and instructions on data management. Concrete information: main Aalto services and global services. Main Theoretical information: Aalto-specific summary and Aalto’s Research Data Management pages.

5.1 Data

Data binds all of today’s research together. Even if you don’t consider yourself to do data-based research, the results of your work becomes data before it is published. The highest levels of funding agencies are beginning to demand good data management and openness. Knowing how to manage data is probably one of the most important untaught modern skills.

It’s not just “get it done”: there are good and bad methods of managing data. For example,

- A **bad strategy** is to store everything in one folder on your own laptop: there’s a very high chance that you will someday lose it all. A **better strategy** is to use a secure centralized service - preferably an Aalto service, since you get guaranteed support for free.

- A **bad strategy** is for everyone to do their own thing and put no effort into recording what they have done: in five years, when the group is almost completely changed, that data will be unusable. A **better strategy** is to make sure things are documented and archived as soon you get them (and keep this up to date), so that the data can continue to serve you in the future.

- A **bad strategy** is to assume all data is proprietary Aalto information: eventually funders will demand more and you won’t be prepared, and Aalto will remain an island, instead of a hub that others want to work with and build on. A **better strategy** is to always consider openness, licensing, and privacy from the start (even if you don’t do it right away), and always separate data based on level of confidentiality so that you can share or open later.

You can find more formal information at the Aalto Research Data Management pages, and here we focus on the practical side of things.
5.1.1 Applying for funding

When applying for funding, you may need to submit a data management plan (DMP) along with the grant application. For hints on making one, see our data management plan page or the Aalto Research Data Management pages. However, be aware that a grant application data management plan (“Funder DMP”) usually focuses on sounding like a grant, not being a usable work plan (“Practical DMP”). Before you start accumulating data, browse the other links on this site and make sure that you organize things well! Aalto info will only help you make a funder DMP, not organize your data during the project.

Grantwriters and the Open Science and ACRIS teams can help you with producing data management plans for funding. Science-IT can help you with funding or practical data management plans.

Data management plans

Data management plans are a catchphrase these days, mainly because funders are requiring them now. This is for a good reason - researchers often focus on their papers, and making good use of the data gets forgotten. Funders pay a lot for research, and they want all the possible value for society.

However, it is worth doing a bit of planning about data, even aside from the required bureaucratic exercise. It is true that researchers focus on the next paper. Data has long-term value even inside Aalto, and if you don’t try hard it will get lost.

Practical DMP

In this section, we outline recommend ways to use Aalto resources for different use cases.

No matter what your project, you want to start by thinking how you will handle your data (this can be “real data”, notes, code, papers, etc). This will make sure that your team works together well and doesn’t end up with a big mess in a few months - or that you can’t work together because you can’t share information. For this, see the A4 DMP template. This site is focused on practical DMPs.

- Suggested DMP for large experimental data (TODO)
- Suggested DMP for simulations or computer-generated data (TODO)
- Suggested DMP for data from humans (surveys, interviews, etc) (TODO)

Funder DMP

There are plenty of other good resources about making funder DMPs.

- At Aalto, the RIS grantwriters have taken responsibility for helping to make good funder DMPs.
- The Aalto RDM pages have a subsection dedicated to data management plans.
- The DMPTuuli is a combination template, instructions, and web form which makes it easy to do the mechanical assembly of DMPs. They also have public docx/pdf templates which can be used even without the web form. Aalto recommends this service, though be aware it helps you fill out a form, not plan your work.

As some concrete suggestions:

- Funders are especially concerned about sharing, preservation, reproducibility, and dissemination but probably can’t evaluate too much about the practical side of things.
- You can mention that you will follow the Aalto RDM policy, which covers mainly opening and licensing. The policy still allows you to make your own choices, but it sounds quite good if you refer to it and say you will follow it.
• For data storage considerations, you can say that your department/Science-IT provides data storage services (for Science-IT departments) and has a data storage policy which you will follow: citation and/or full text.

Help! I need a DMP right now!

If you are reading this, you probably have a grant deadline and you need to do something right now. Use the resources above, but here is some more advice:

• Read the data management outline on this site. You should be able to pull many of the practical pieces (storage, confidentiality, archiving, etc) from here. Read this first!

• Read the Aalto-level guidelines. These are quite abstract and high level, and might tell you what people think is important but not tell you how to do stuff.

• To internally organize things, you could start with the A4 DMP template. This can’t be used for something you submit, but lets you know the big picture. If you fill this out first and give it to someone, they can guide you in making the next version.

• Use the DMP Tuuli tool to prepare the DMP. It just makes a final document you can download (you could do the same using a word processor), but breaks everything down into a nice form.
  – If you don’t like the idea of a web form, the templates seem to be available publically, too. These seem to have roughly the info as the DMPTuuli web forms.

Why do they want DMPs? What should it include? Answering these will help you to know what to write, since there is not near enough room to make a plan that contains everything you need to know personally:

• The main purpose is to make sure that other researchers can use your data as easily as they can use your published papers. Can other researchers access your data? Can your results be reproduced?

• Most likely, whoever is reading doesn’t care that much about the actual day to day data storage and so on, but more of the big picture: licensing, opening, archiving, sharing, preserving, expanding, securing.

• If you produce your own data, how can others use it? Funders want open, but by giving good justification you can do whatever you need. If the data comes from others, then can you re-distribute (even for validation) or would others need to request it from the source?

• How software you make related to data processing (and really all software) will be handled. Even if data can’t be released, software can be open sourced which allows reproduction of results and some sort of validation.

• How you preserve data for future use: both for you, and for others. This is especially important. Also, how will data be understandable in 50 years? Is the program that will read it gone? Do you have a README? Is your data in a field-specific standard structured format? Is it opened and does it go into an archive which will be around in 50-100 years (anything managed by you or Aalto specific isn’t a credible option for this)?

• You should mention how you will follow the “Aalto Research Data Management Policy and related guidance”. The policy just says “you will make strategic decisions”, so sounds good to the funder while not binding you to anything.

• For storage, organization, confidentiality, etc, you can say you will follow the Science-IT data management policy. This isn’t requirements for you, but the default services we offer for data storage (designed to keep data safe and secure, and uuushareable). It also sounds good to say. (see the outline)

Model Academy of Finland DMP

You can see the Academy’s detailed info in their supplement. This guide isn’t to replace their guidelines (there is a lot there that isn’t duplicated here), but make it clear what the Aalto correspondences are. You can also see the Aalto guidelines, but this is also a bit abstract to be immediately usable.
With all the time spent on writing your plan, don’t forget to do something useful, too.

1. General description of the data
   • No specific extra advice here - see academy guidelines.

2. Ethical and legal compliance
   • For identifiable human data, say that you will follow the Aalto personal data policy. In particular, data will only be stored only on systems meeting the Aalto guidelines for personal data storage. Preferable, store this on the department network drives only - not on personal computers. You can request ethical evaluation from the Aalto Research Ethics Committee. In Finland, this is required in quite few cases, but publishers are requiring this more and more often. Thus, you may want to check your journal requirements and request ethical evaluation anyway.
   • Data always will be made available under the Aalto data management policy. (You can commit to this, because the policy only says you should make decisions “strategically” so there are actually no obligations.)
   • Software will be made open source if it matches the criteria under the Aalto open source policy. If software exceeds that criteria, there will be discussions with Aalto innovation services for commercialization or licensing.
   • There are plenty of other intellectual property concerns which I can’t go into here, and you need to study yourself. Aalto Research and Innovation Services has lawyers which can help with this - you can consult in advance or say you will use them.

3. Documentation and metadata
   • It is harder to comment on this because it is so field-specific. Make sure you have READMEs and documents.
   • Everyone talks about “metadata” but this is such a broad term that it is essentially meaningless. I personally put this into three types:
     – Cataloging: You can say that the metadata required by your repository will be used.
     – Necessary to understand: you will use README files, use formats that are self-describing such as CSV files with useful headers and comments, include code, and whatever is needed to make someone understand the data later (including yourself).
     – Necessary to automatically process: data should be automatically usable with the least amount of manual effort. This is highly domain-specific, and depends on if your domain already has standards to make this possible. Use the best possible practices here, taking into account cost vs benefit.

4. Storage and backup during the research project.
   • Aalto really excels here. Basically, just use the Aalto network drives. This storage is large, free, shareable, snapshotted, backed up to an offsite datacenter. Access is controlled via Aalto accounts plus unix groups. If people need to make other copies (and it’s allowed for security reasons), they can. Big data is stored on Triton user guide from which it has direct access to any computational power you may need.

5. Opening, publishing, and archiving the data after the research project.
   • This gets more abstract, and really depends on what you want. There are many options, and maybe it is best to consult the Aalto page on this, though it’s again rather abstract.
   • You can check the services page to see what common services are available. If you don’t have any more specialized repository to use, Zenodo is a good choice. Always prefer a specialized, domain-specific repository if you can. Don’t say it is archived on Aalto resources, since you or Aalto can’t commit to hosting things or the long term.
   • You can say that organization of data is a part of research, though the extra requirements needed to open are small. Give some estimate of the total/extra amount of work needed.
5.1.2 During the project

Make sure that you manage data well - just think, your data is possibly worth more than all your other devices combined. Check out the core lessons to learn of the most common problems, and see if any of them apply to you.

You may want to read our welcome to researchers and outline of data management at Aalto pages. For specific Aalto storage services, see Data storage, and for other options see the general services page.

We recommend that each project or group gets a network drive, which is used as the centralized place for data storage, safekeeping, and possibly daily work. See the outline of data management at Aalto page.

Data storage services available

This page provides a list of common data storage services, and can help you select the right service for the type of data you have (see Data organization). But before we talk about services, you have to consider what your needs are.

Types of data

There are different broad categories of data:

- **Code/papers drafts**: These are absolutely critical, but quite small. You want a full history that is easy to use, too. Put in version control and in Aalto gitlab.

- **Original data**: Your original, irreplaceable data. You want this in two places: a fast, large, available place for day-to-day work, and also somewhere backed up for a fairly long time.

- **Intermediate working files**: This is what you get when you run code on original data. It’s OK if this is lost, because you have the code and original data to re-create it, right? It can go in the large, fast location.

- **Final published results/data**: You want this backed up and available for a very long time (forever?). Put in an open-access repository such as Zenodo. Once it’s in the archival, backups should be done there.

O = good, x = bad

<table>
<thead>
<tr>
<th></th>
<th>Large</th>
<th>Fast</th>
<th>Confidential</th>
<th>Frequent backups</th>
<th>Long-term archival</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code</td>
<td>OO</td>
<td>OO</td>
<td></td>
<td>OO</td>
<td>OO</td>
</tr>
<tr>
<td>Original data</td>
<td>O</td>
<td>O</td>
<td>O?</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>Intermediate files</td>
<td>OO</td>
<td>OO</td>
<td>O?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Final results/open data</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>O</td>
</tr>
</tbody>
</table>

Service index

There are different qualities we want in filesystems: large, fast, confidential, highly available, backed up, mounted everywhere, lasts forever. It is expensive to have all of these together, so there are different places with different benefits. It is up to you to balance their use so that you can accomplish what you need. Compare this table to the types of data above. Use the right place for the right data.

You often need to use different types of services, for example version.aalto.fi for day to day code management, but archive to Zenodo at the end of a project.

O = good, x = bad
## Service details

**Note:** This list is still under development (2018-03-07)

In general, if you need to

- **archive and open**, consider hosting data on Zenodo (and put a record of it in ACRIS, so you can get internal Aalto credit. If you have a discipline-specific repository, use that instead (with metadata still in ACRIS)

- For **day to day work** within Aalto, Aalto network drives are a good service and (different options below).

- For making a **data management plan**, DMPTuuli along with *our info* is good.

### Science-IT services

- The filesystems by *Triton*. Primarily scratch and work, which are very large, very fast on Triton, but only for scratch data because they are not backed up.
Departments

- **CS,NBE,PHYS** provide storage logically divided into project and archive. These are the counterparts of Triton and are backed up. They are actually Aalto “teamwork”, but the departments do the day-to-day interfacing. See *Data storage*.

- See the work and teamwork notes below in the next section. In some cases, these are managed by departments.

Aalto

- See *Data storage*.

- Also information is available from Aalto ITS, some here.

- **Aalto home directories** are small and intended mainly for personal stuff. Once you leave, this data dies, so don’t put important stuff here.

- Aalto has work and teamwork storage systems. These are actually provided at the Aalto level, but how you request space, how you use them, and what the are called varies and is not always very well defined. A little bit of info at *Data storage*.

- **Aalto laptops** are not a good place to store data because they are usually not backed up, and data is not shareable. (Even if data is backed up, once you leave, no one will even be able to get access). Most people who use laptops have the most valuable data stored on network drives.

- **Aalto webspace** can share data. See *Data storage*. This isn’t suitable for archival or long-term anything, since it is tied to user accounts. If you want to share here, maybe you could do a bit more work and handle it forever at Zenodo?

- [https://version.aalto.fi](https://version.aalto.fi) is the Aalto Gitlab. It is used for small version controlled files. It is a great place for day to day work of private files, but not for permanent archival. See *Aalto Gitlab*.

- **ACRIS** is the Aalto “research information system”, meaning it’s a record of things that everyone is doing research-wise. You should make records for datasets there as a research output. *(ACRIS + research data instructions)*

  Summary: try to host the actual data elsewhere, but always make a report of the data in ACRIS so you get credit.

  ACRIS has support for storing data itself, but that isn’t recommended most of the time since ACRIS in it’s current form isn’t guaranteed to stay around forever. However, if data needs to be kept internal, it might be OK since you can set confidentiality and share with certain people. However, you *should* always make a report of your datasets in ACRIS even if they are hosted elsewhere, so that you can get academic credit for it.

  What data sets should be included in ACRIS? We think: a) anything that is independently published with DOI.
  b) any paper which serves as a formal dataset description in a data journal, even if there is also an entry as an ACRIS article. c) any paper which serves as an informal dataset description.

  As for different roles: creator=who is involved in creating it, distributor=who can be contacted about access (if not public), owner=who has ultimate responsibility (often the PI but project dependent).

- **Eduuni** is a Finnish service for educational collaboration. It’s reported to be more secure than either Google Drive or OneDrive, but we know of few people who use it.

- The **Aalto Wiki** is sometimes mentioned as a place to store data. It’s really better for collaboration, but you can put little bits of data there if you want.
Finnish services

- **The FUNET filesender** ([https://filesender.funet.fi](https://filesender.funet.fi)) can share files with others. You log in with your Aalto account, and then you can upload files and send a link by email. Or, you can send an email that allows others to upload. Run by CSC and recommended for sharing (instead of email).

- **IDA, Etsin, and AVAA** are CSC-provided services (funded by the ministry as part of the Open Science project, ATT), which provide some data services to researchers.
  - **Etsin** is the Finnish metadata catalog. The intention is that all research data eventually gets cataloged here (open or not), but we are quite far from that goal. Ideally, there would be bidirectional imports to and from ACRIS (the Aalto system) and other repositories, but it’s not there yet. We should recommend that you make a note of your data here, but realistically do ACRIS and wait for a link.
  - **IDA** is a storage service. ([instructions](https://example.com)) It is based on iRODS, a data management layer on top of filesystems. Thus, you have to access it using a special API, command line interfaces, or other tools. Because of this, the learning curve is very steep. Currently, we think IDA would be good if your university doesn’t provide large enough free, properly backed up storage that is shareable within Finland. For long-term public storage, Zenodo is probably overall easier to use. We have some practical notes on using it here, because it takes quite a few steps to get started.
    
    It is said to be a safe place to store your data, but if you read closely a different “long-term preservation” service is coming, so IDA isn’t that. IDA might have a use case for confidential data which can’t leave Finland, but it says it claims it is not suitable for such. They also say that metadata “shall” be added, which makes you think it is only for data which is prepared enough for putting in Etsin.

    If you are dealing with a large amount of data and want to use an API to handle it, this could be good. IDA is being [renewed](https://example.com) in 2018, and will need reevaluation then.

  - **AVAA** is basically a merging of IDA and Etsin. You can set some metadata in IDA so that your data is available via the web. There are some instruction in the IDA user guide (browser, command line). Overall, having to use three different services for publishing a file takes a fair amount of work, so if you want to open data, Zenodo is faster.

- **The FSD Finnish Social Data Archive / Tietoarkisto** is run from the University of Tampere. It is a full-service archive for social data, so they can help in data preparation and curation. It is one of the few places in Finland allowed to archive personally identifiable data.

- **DMPTuuli** ([dmtuuli.fi](http://dmtuuli.fi)) is a service for making data management plans. It is primarily targeted at funder DMPs, so it won’t help you plan your actual research (and even for funder DMPs, you need to know what to say). You can check our [data management plans](https://example.com) page, including the “emergency DMP” section. Aalto also has a little bit of guidance.

EU services

- **Zenodo** ([https://zenodo.org](https://zenodo.org)) is a long-term data repository. It is the largest (thus the most stable long-term) and also has a great user interface. You get a DOI if you archive here. We recommend this service unless you have another domain-specific repository that fits your data better. If you publish data here, also make a metadata entry in ACRIS (see above).

    Zenodo is a good service, but there is little curation, so you need to make sure that your data is described well (both in the structured catalog information and within the data, so that it is usable).

    When you put data in Zenodo, also make an ACRIS dataset entry linked with the DOI.

- **EUDAT** ([http://eudat.eu](http://eudat.eu)) provides a lot of different services: B2share is a lot like Zenodo, but smaller and last we checked the user interface wasn’t as good (and it didn’t provide DOIs). B2Drop is a Dropbox-like file sharing
service (powered by nextcloud), which can be quite nice. B2Find is a metadata catalog that lets you search for data. The other services are mostly target to other large infrastructures. (EUDAT will be re-evaluated in 2018)

Global services (with special Aalto support)

- **Google Drive** is a cloud storage solution (but you probably already knew that...). You can register your Aalto account as a Google account, which gives you unlimited storage (note that this does not mean your personal account gets unlimited... a Gsuite account does. This account ends when you leave Aalto, so this should not be used for permanent storage). You have to enable your account using ITS instructions here. Access the Aalto Google Drive from [https://gdrive.aalto.fi](https://gdrive.aalto.fi). This service can be great for sharing, but because it is tied to your Aalto account, you should not store valuable research data here.

Google Drive has a “team drives” concept, which will allow you to put data into groups which can easily be inherited as time goes on, even if the original people move on.

- **Microsoft OneDrive** is like Google Drive, and Aalto has a special agreement. You can find instructions from ITS here. Theoretically, OneDrive has a higher security rating than Google Drive, but it is still not suitable for legally confidential data.

Global services

- **Github** is a code-sharing and collaboration service (using git, obviously). If you have an open source project, this is a well-known place to put it. The only downside is if you have objections to proprietary services. Github should not be used as a permanent archive, but there is Zenodo integration so that your code can be archived permanently (and even has integration with the Github “release” feature).

This is by no means a complete list...

Data organization

How should data be stored? On the simplest level, this asks “on what physical disks”, but this page is concerned about something more high-level: how you organize data on those disks.

Data organization is very important, because if you don’t do it soon, you end up with a epic mess which you will never have time to clean up. If you organize data well, then everything after becomes much easier: you can archive what you need. Others can find what they need. You can open what you need easily.

First steps: filesystems

The first step is to pick the right filesystems to store on, which you can find on other pages here.

If you’ve picked a good storage locations, then you can share the data with multiple people automatically. In this case, you really should pay attention to the advice on this page. Even if you are the only person who can access the data, these tips will be quite useful to you.

Traditional organization

This is the traditional organization within a single person’s project. The key concept is separation of code, original data, scratch data, and final outputs. Each is handled properly.

- **PROJECT/code/** - backed up and tracked in a version control system.
- **PROJECT/original/** - original and irreplaceable data. Backed up at the same time it is placed here.
• PROJECT/scratch/ - bulk data, can be regenerated from code+original
• PROJECT/doc/ - final outputs, which should be kept for a very long term.
• PROJECT/doc/paper1/
• PROJECT/doc/paper2/
• PROJECT/doc/opendata/

When the project is over, code/ and doc/ can be backed up permanently (original/ is already backed up) and the scratch directory can be kept for a reasonable time before it is removed (or put into cold storage).

The most important thing is that code is kept separate from the data. This means no copying files over and over to minor variations. Could should be adjustable for different purposes (and you can always get the old versions from version control). Code is run from the code directory, no need to copy to each folder individually.

Multi-user

The system above can be trivially adapted to suit a project with multiple users:

• PROJECT/USER1/.... - each user directory has their own code/, scratch/, and doc/ directories.
  Code is synced via the version control system. People use the original data straight from the shared folder in the project.
• PROJECT/USER2/....
• PROJECT/original/ - this is the original data.
• PROJECT/scratch/ - shared intermediate files, if they are stable enough to be shared.

For convenience, each user can create a symbolic link to the original/ data directory from their own directory.

Master project

In this, you have one long-term master project that has many different users and research themes with in. As time goes on, once users leave, their directories can be cleaned up and removed. The same can happen for the themes.

• PROJECT/USER1/SUBPROJECT1/...
• PROJECT/USER1/SUBPROJECT2/...
• PROJECT/USER2/SUBPROJECT1/...
• PROJECT/original/
• PROJECT/THEME/USER1/...
• PROJECT/THEME/USER2/...
• PROJECT/archive/

Common variants

• Simulations with different parameters: all parameters are stored in the code directory, within version control. The code knows what parameters to use when making a new run. This makes it easy to see the entire history of your simulations.
• Downloading data: this can be put into either original or scratch, depending on how much you trust the original source to stay available.
Projects

In Aalto, data is organized into project groups. Each project has members who can access the data, and different shared storage spaces (project, archive, scratch (see below)). You can apply for these whenever you need.

What should a project contain? How much should go into the same project?

• **One project that lasts forever per research group**: This is traditional. A professor will get a project allocated, and then people put data in here. There may be subdirectories for each researcher or topic, and some shared folders for common data. The problem here is that the size will grow without bound. Who will ever clean up all the old stuff? These have a way of growing forever so that the data becomes no longer manageable, but they are convenient because it keeps the organization flat.
  - If data size is small and growing slower than storage, this works for long-term.
  - It can also work if particular temporary files are managed well and eventually removed.

• **One project for each distinct theme**: A research group may become interested in some topic (for example, a distinct funded project), and they get storage space just for this. The project goes on and is eventually closed.
  - You can be more fine-grained in access, if data is confidential
  - You can ensure that the data stays together
  - You can ensure that data end-of-life happens properly. This is especially useful for showing you are managing data properly as part of grant applications.
  - You can have a master group as a member of the specific project. This allows a flat organization, where all of your members can access all data in different projects.

5.1.3 Internal reporting

Data is a top-level research output, even if not everyone considers it valuable now. Open or not, the university wants to know what data exists. Currently, this is done via ACRIS (primary instructions). In particular, you should create a “dataset” object for data you create (it doesn’t have to be open). For some hints, for now see the ACRIS point on the services page or the ACRIS instructions on data.

5.1.4 Sharing and collaboration

Obviously, you will often need to share data within projects. Emailing things back and forth is rarely a good way to do things. Check other data sharing services from our services page or Aalto’s IT services for research page.

We recommend, instead of seeing this as a sharing problem, see this as a storage problem: find a place to store data which everyone can access, and share via that. This promotes long-term organization.

5.1.5 Archival after the project

After a project is done, you may need to store data long-term for follow-up use. You shouldn’t do this just by assuming everyone keeps their copy: people leave, and eventually that a data will get lost. The easiest and recommended way of doing this is by opening data and publishing it on a reputable worldwide archive once it is time. For the most part, the
university wants to avoid creating its own internal permanent archives, because they will end up requiring large effort to maintain. It’s better to use the publically-funded and managed worldwide services.

5.1.6 Publication

See our list of storage services for recommendations on archival. If you don’t know what to pick (there isn’t something specialized for your field), use Zenodo and report it in ACRIS (see “internal reporting” above).

5.1.7 Licensing and intellectual property

Just because data is “out there” doesn’t mean it’s usable by others: big companies have ensured that data is by default closed. Luckily, it is easy to make data reusable: just add a license. There are plenty of options that can balance between “public domain, do anything” and “if you help us too”.

See our Open Source page for more info.

5.1.8 Other info

IDA data storage service

Note: This page is under development

Note: IDA has changed in 2018/2019, and these instructions may no longer be accurate.

IDA is a storage service provided by the Ministry of Culture of Education / the Finnish Open Science and Research initiative / CSC. It can be used for storing very large files securely and for a reasonably long time. Quota can be in many TB, and quota is allocated by application.

The upstream instructions can be found at http://openscience.fi/ida-user-instructions. The upstream description can be found at http://openscience.fi/ida

What it is for

Main article: http://openscience.fi/ida

IDA is for stable research data which needs safe, somewhat long term storage. (However, it isn’t for very long term archival, another system is coming for that). It isn’t for active, day-to-day use. It can link data to permanent identifiers, store metadata, and also publish data via AVAA and make it searchable via Etsin.

If you just need large storage, Triton’s scratch is good for that. However, if you have many TBs of data, then finding a backup place is difficult (scratch is not backed up). IDA can serve that need.

IDA can also serve to make small or large data open (searchable and downloadable), via Etsin and AVAA. These three go together: IDA is storage, Etsin is search, AVAA is download server.

You automatically get a quota from Academy of Finland projects (and it says they encourage its use).
Registering and applying for space

Main article: http://openscience.fi/becoming-an-ida-user

Everyone can apply for IDA space via your CSC account (but all IDA space is allocated to projects, not individual users). Anyone at a Finnish university can get a CSC account automatically. IDA space is

First, you need a CSC account. You can get this online via the Aalto authentication: https://sui.csc.fi/.

Once you have the CSC account, you need a CSC project. Only senior level staff (postdoc or above) can do this - you probably want it to be someone who will be here long term, since that is the point!. Apply for the project through the scientists user interface (SUI) (https://sui.csc.fi). The SUI can be rather confusing. First, go to eService → Resources and Applications → select “Academic CSC project”. The bottom of the page then changes to an application form. Fill this out: say you need a project for IDA (or whatever). You need to wait for an email for the CSC project to be approved.

After this project is approved, you can apply for IDA storage space to be connected to this project. Go to the SUI → eService → Resources and Applications, then go to Resources → Storage → IDA Storage Service. The application form below changes to the IDA application. Select the project which will receive the resources, then fill out the application.

You will get another email with your IDA password and path once it has been approved by Aalto’s IDA contact person. This is different from the project approval email from CSC (the right email has explicit IDA usernames and passwords in it). If you do not get the IDA info email within a day or two, ping Juha Juvonen at Aalto and ask if the project has been approved.

Confidential data

In many places, CSC states that IDA is not suitable for confidential data. This is because the command line interface does not encrypt files (though I had heard that it just doesn’t by default, but maybe it could be made to). Still, since they do not intend to support confidential data, we should not count on this for the future. However, confidential data is OK if it is strongly encrypted.

See our page on encryption for scientists.

Access

Main instructions: http://openscience.fi/ida-user-instructions

iRODS (and thus IDA) is an API-based file storage service. Thus, you use separate commands to get and put files. This comes out of the fact that this is designed for very big files and flexible, long-term storage. This is not too hard - it is like using FTP or sftp. There are also mountable filesystems, however this should not be used for daily work since they are not very efficient.

Not all tools are suitable for very large files - there are some reported problems that need to be worked around. See the CSC instructions for details and hints for large files.

Note that the IDA password is different than your CSC or Aalto passwords. Don’t use your CSC or Aalto password with IDA accidentally, some of the programs (command line tool in particular) don’t seem to handle it very securely (it is stored weakly obfuscated in a file in ~/.irods)

Browser

Through the CSC SUI, you can brows and upload files. See http://openscience.fi/ida-browser. This is probably not good for extremely large files.
Aalto scientific computing guide

Command line


irops commands: Aalto workstations and Triton have the irods command line tools (the “icommands”). Use the module system: module load irods.

Configuration file: You need to set up the config file (see the openscience.fi instructions). You need an extra path in it here:

On Aalto Linux, this is needed in the config file .irods/irods_environment.json (be careful with commas to make sure it stays valid JSON):

"irods_plugins_home": "/work/modules/Ubuntu/14.04/amd64/common/irods/4.1.9/var/lib/irods/plugins/

On Triton, the corresponding directory is "/share/apps/irods/4.1.9/var/lib/irods/plugins/"

Network mounting

The IDA directories can be mounted via DAV: http://openscience.fi/ida-network-directory

There is also an irods 4.x fuse module (irodsFs) included by default. Use it like normal, though it is quite slow.

Practical usage

To be added once we have more specific use cases which are not covered above.

More resources

Documentation

- http://openscience.fi/ida-user-instructions - Instructions from avointiede
- https://research.csc.fi/csc-guide-archiving-data-to-the-archive-servers - CSC documentation, note that this is a bit out of date and less complete than Avointiede’s one.
- The CSC archive also uses irods, but it uses version 3 which is not compatible with these command line tools.

5.1.9 External links

- Finland Open Science Initiative (ATT)
- Aalto Research Data Management pages.

Cheatsheets: Data, A4 Data management plan.
Triton is the Aalto high performance computing cluster. It is your go-to resources for anything that exceeds your desktop computer’s capacity.

6.1 Triton user guide

Triton is the Aalto high-performance computing cluster. It serves all researchers of Aalto, but is currently coordinated from within the School of Science. Access is free for researchers (students not doing research should check out our intro for students). It is similar to the CSC clusters, though CSC clusters are larger and Triton is easier to use because it is more integrated into the Aalto environment.
6.1.1 Quick contents and links

<table>
<thead>
<tr>
<th>Triton contents</th>
<th>News</th>
</tr>
</thead>
<tbody>
<tr>
<td>• About Triton</td>
<td>• Quick Reference</td>
</tr>
<tr>
<td>– Cluster overview</td>
<td>• Triton Cheatsheet</td>
</tr>
<tr>
<td>– Usage policies and legal</td>
<td>• Triton FAQ</td>
</tr>
<tr>
<td>– Acknowledging Triton</td>
<td>• Triton FAQ</td>
</tr>
<tr>
<td>– Aalto Science-IT (external link)</td>
<td>• Scicomp Garage</td>
</tr>
<tr>
<td>• Getting Help/Contact</td>
<td>Scientific computing resources</td>
</tr>
<tr>
<td>– Triton issue tracker (most requests here, login with HAKA)</td>
<td>• SCIP – Scientific Computing in Practice courses: organized by Science IT. Including Triton kickstarts and many others</td>
</tr>
<tr>
<td>– Suggestions for good support requests</td>
<td>• Parallel computing</td>
</tr>
<tr>
<td>• Quick Reference</td>
<td>• Aalto IT Services for Research</td>
</tr>
<tr>
<td>• Tutorials (start here)</td>
<td>• Software Carpentry (scientific computation basics) and Code Refinery (more focused on programming techniques)</td>
</tr>
<tr>
<td>– Triton accounts</td>
<td>General links</td>
</tr>
<tr>
<td>– About Science-IT and Triton</td>
<td>• CSC - Finland’s academic computing center.</td>
</tr>
<tr>
<td>– Connecting to Triton</td>
<td>• FGCI user’s guide at CSC: That is a general Guide to FGCI resources. Triton is one of them.</td>
</tr>
<tr>
<td>– Software Modules</td>
<td>• Taito user guide at CSC: a Triton like cluster at CSC. Similar setup, thus examples and instructions can be useful.</td>
</tr>
<tr>
<td>– Applications</td>
<td>• Aalto research data management information</td>
</tr>
<tr>
<td>– Data storage</td>
<td></td>
</tr>
<tr>
<td>– Interactive jobs</td>
<td></td>
</tr>
<tr>
<td>– Serial Jobs</td>
<td></td>
</tr>
<tr>
<td>– Array jobs</td>
<td></td>
</tr>
<tr>
<td>– Job dependencies</td>
<td></td>
</tr>
<tr>
<td>– GPU computing</td>
<td></td>
</tr>
<tr>
<td>– Parallel computing</td>
<td></td>
</tr>
<tr>
<td>• Cluster usage details</td>
<td></td>
</tr>
<tr>
<td>– Parallel jobs (coming, for now see Running programs on Triton)</td>
<td></td>
</tr>
<tr>
<td>– GPU Computing</td>
<td></td>
</tr>
<tr>
<td>• Applications</td>
<td></td>
</tr>
</tbody>
</table>

For full contents, see below.

6.1.2 Overview

Cluster overview

Shared resource

Triton is a joint installation by a number of Aalto School of Science faculties within Science-IT project, which was founded in 2009 to facilitate the HPC Infrastructure in all of School of Science. It is now available to all Aalto researchers.

As of 2016, Triton is part of FGCI - Finnish Grid and Cloud Infrastructure (predecessor of Finnish Grid Infrastructure). Through the national grid and cloud infrastructure, Triton also becomes part of the European Grid Infrastructure.

Hardware

Different types of nodes:

• 144 compute nodes HP SL390s G7, each equipped with 2x Intel Xeon X5650 2.67GHz (Westmere six-core each). 118 compute nodes wsm[1-112,137-144], have 48 GB of DDR3-1066 memory, others wsm[113-136]
have 96GB, each node has 4xQDR Infiniband port, wsm[1-112,137-144] have about 830 GB of local disk space (2 striped 7.2k SATA drives), while wsm[113-136] about 380GB on single drive. 16 nodes have by two additional SATA drives.

- 3 compute nodes gpu[20-22] are Dell PowerEdge C4130 for gpu computing. CPUs are 2x6 core Xeon E5 2620 v3 2.50GHz and memory configuration is 128GB DDR4-2133. There are 4x2 GPU K80 cards per node.
- 2 fat nodes HP DL580 G7 4U, 4x Xeon, 6x SATA drives, 1TB of DDR3-1066 memory each and 4xQDR Infiniband port.
- 48 compute nodes ivb[1-48] are HP SL230s G8 with 2x Xeon E5 2680 v2 10-core CPUs. First 24 nodes have 256 GB of DDR3-1667 memory and the other 24 are equipped with 64 GB.
- 67 compute nodes Dell PowerEdge C4130 servers with 2x Xeon E5 2680 v3 2.5GHz 12-core CPUs. 51 nodes pe[1-48, 65-67] have 128 GB of DDR4-2133 memory while 15 nodes pe[49-64] have 256GB.
- 9 compute nodes pe[83–91] Dell PowerEdge C4130 with 2x Xeon X5 E5-2680 v4 2.4 GHz 14-core CPUs.
- 5 compute nodes gpu[23–27] for gpu computing. CPUs are 2x12core Xeon E5-2680 v3 @ 2.50GHz and memory configuration 256GB DDR4-2400. There are 4x Tesla P100 16GB cards per node.
- 2 Nvidia DGX-1 compute nodes for gpu computing. CPUs are 2x20 core Xeon E5-2698 v4 @ 2.2GHz and memory configuration 512GB DDR4-2133. There are 8x Tesla V100 16GB cards per node. These nodes have a special operating system, unlike the rest of the cluster, and thus these nodes require special considerations. See the DGX page.
<table>
<thead>
<tr>
<th>Node name</th>
<th>Number of nodes</th>
<th>Node type</th>
<th>Year</th>
<th>Arch (constraint)</th>
<th>CPU type</th>
<th>Memory Configuration</th>
<th>GPUs</th>
</tr>
</thead>
<tbody>
<tr>
<td>wsm[1-112,137-144]</td>
<td>120</td>
<td>ProLiant SL390s G7</td>
<td>wsm</td>
<td>2x6 core Intel Xeon X5650 2.67GHz</td>
<td>48GB DD3-1333</td>
<td></td>
<td></td>
</tr>
<tr>
<td>wsm[113-136]</td>
<td>24</td>
<td>ProLiant SL390s G7</td>
<td>wsm</td>
<td>2x6 core Intel Xeon X5650 2.67GHz</td>
<td>96GB DD3-1333</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ivb[1-24]</td>
<td>24</td>
<td>ProLiant SL230s G8</td>
<td>ivb,avx</td>
<td>2x10 core Xeon E5 2680 v2 2.80GHz</td>
<td>256GB DDR3-1667</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ivb[25-48]</td>
<td>24</td>
<td>ProLiant SL230s G8</td>
<td>ivb,avx</td>
<td>2x10 core Xeon E5 2680 v2 2.80GHz</td>
<td>64GB DDR3-1667</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pe[1-48,65-81]</td>
<td>65</td>
<td>Dell PowerEdge C4130</td>
<td>2016</td>
<td>hsw,avx,avx2</td>
<td>2x12 core Xeon E5 2680 v3 2.50GHz</td>
<td>128GB DDR4-2133</td>
<td></td>
</tr>
<tr>
<td>pe[49-64,82]</td>
<td>17</td>
<td>Dell PowerEdge C4130</td>
<td>2016</td>
<td>hsw,avx,avx2</td>
<td>2x12 core Xeon E5 2680 v3 2.50GHz</td>
<td>256GB DDR4-2133</td>
<td></td>
</tr>
<tr>
<td>pe[83-91]</td>
<td>8</td>
<td>Dell PowerEdge C6420</td>
<td>2019</td>
<td>skl,avx,avx2,avx512</td>
<td>2x14 core Xeon Gold 6148 2.40GHz</td>
<td>128GB DDR4-2667</td>
<td></td>
</tr>
<tr>
<td>skl[1-48]</td>
<td>48</td>
<td>Dell PowerEdge C6420</td>
<td>2019</td>
<td>skl,avx,avx2,avx512</td>
<td>2x14 core Xeon Gold 6148 2.40GHz</td>
<td>192GB DDR4-2667</td>
<td></td>
</tr>
<tr>
<td>gpu[20-22]</td>
<td>3</td>
<td>Dell PowerEdge C4130</td>
<td>2016</td>
<td>hsw,avx,avx2,avx512</td>
<td>2x14 core Xeon E5 2680 v3 2.50GHz</td>
<td>128GB DDR4-2133</td>
<td></td>
</tr>
<tr>
<td>gpu[23-27]</td>
<td>5</td>
<td>Dell PowerEdge C4130</td>
<td>2017</td>
<td>hsw,avx,avx2,avx512</td>
<td>2x14 core Xeon E5 2680 v3 2.50GHz</td>
<td>256GB DDR4-2400</td>
<td></td>
</tr>
<tr>
<td>dgx[01-02]</td>
<td>2</td>
<td>Nvidia DGX-1</td>
<td>2018</td>
<td>bdw,avx,avx2,avx512</td>
<td>2x14 core Xeon E5-2680 v4 @ 2.5GHz</td>
<td>512GB DDR4-2133</td>
<td></td>
</tr>
<tr>
<td>gpu[28-37]</td>
<td>10</td>
<td>Dell PowerEdge C4140</td>
<td>2019</td>
<td>skl,avx,avx2,avx512</td>
<td>Intel Xeon Gold 6134 @ 3.2GHz</td>
<td>384GB DDR4-2667</td>
<td></td>
</tr>
</tbody>
</table>

All Triton computing nodes are identical in respect to software and access to common file system. Each node has its own unique host name and ip-address.

**Networking**

The cluster has two internal networks: Infiniband for MPI and Lustre filesystem and Gigabit Ethernet for everything else like NFS /home directories and ssh.

The internal networks are unaccessible from outside. Only the login node triton.aalto.fi has an extra Ethernet connection to outside.

High performance InfiniBand has fat-tree configuration in general. Triton has several InfiniBand segments (often called islands) distinguished based on the CPU arch. The nodes within those islands connected with ratio 2:1, thus for each 2 downlinks there is 1 uplink to spine switches. The islands are wsm[1-144] 1728 cores, ivb[1-45] 540 cores, pe[1-91] 2192 cores (keep in mind that pe[83-91] have 28 cores per node). Uplinks from those islands are mainly used for Lustre communication. Running MPI jobs possible on the entire island or its segment, but not across the cluster.
See the IB topology map at cluster technical details page.

**Disk arrays**

All compute nodes and front-end are connected to *DDN SFA12k storage system*: large disk arrays with the Lustre filesystem on top of it cross-mounted under `/scratch` directory. The system provides about 1.8PB of disk space available to end-user.

**Software**

The cluster is running open source software infrastructure: CentOS 7, with SLURM as the scheduler and batch system.

**Getting Triton help**

There are many ways to get help, and you should try them all. If you are just looking for the most important link, it is our issue tracker.

Whatever you do, these guidelines for making good support requests are very useful.

See also:

Are you just looking for a Triton account? See *Triton accounts*.

**The Triton docs**

In case you got to this page directly, you are now on the Triton and Science-IT (CS, NBE, PHYS at least) documentation site. See the main page for the index.

**Your colleagues**

Science is a collaborative process, even if it doesn’t seem so. Academic courses don’t teach you everything you need to know, so it’s worth trying to work together and learn from each other - your group is the expert in it’s work, after all.

**Issue tracker**

We keep track of cluster issues at [https://version.aalto.fi/gitlab/AaltoScienceIT/triton/issues](https://version.aalto.fi/gitlab/AaltoScienceIT/triton/issues). Feel free to post your issue there. Either admins or other users can reply — and you should feel free to reply and help others, too. The system is accessible from anywhere in the world, but you need to login with HAKA (using the button). All newly created issues are reported to admins by email.

This is primary support channel and meant for general issues like general help, troubleshooting, problems with code, new software requests, problems that may affect several users.

**Note:** If you get a message that you are blocked from version.aalto.fi, send the email to servicedesk. It’s not your fault: it automatically blocks people when their organizational unit changes. Yes, this is bad but it’s not in our control...

If you have an Aalto visitor account, login with HAKA won’t work - use your email address and Aalto password.
Email ticketing system

For private issues you can also contact us via our email alias (on our wiki pages, login required). This is primarily intended for specific issues such as requesting new accounts, quotas, etc. Please avoid sending personal mails directly to admins, because it is best for all admins to be aware of issues, people may be absent, and personal emails are likely to be lost.

Most general issues should be reported to the issue tracker instead, not by email. Email is primarily for accounts related queries.

Users’ mailing list

All cluster users are on the triton-users mailing list (automagically kept in sync with those who have Triton access). It is for announcements and open discussions mainly, for problem solving please try the tracker.

If you do not receive list emails, you’d better check out with your local Triton admin that you are on the list. Otherwise you miss all the announcements including critical ones about maintenance breaks.

In person

Come by one of the Scientific computing garages any week. It’s the best place to get problems solved fast.

You can also come and talk to us face-to-face, but of course we have to be in-office. This is especially useful when there is an open-ended question where we have to discuss what is the best solution. We may then ask you to open a ticket once there is an answer, so that we can track the progress and not forget.

Triton support team

Most of us are members of your department’s support teams, so can answer questions about balancing use of Triton and your department’s computers. We also like it when people drop by and talk with us, so that we can better plan our services. In general, don’t mail us directly - use either the issue tracker above or the support email address. You can address your request to a specific person.

<table>
<thead>
<tr>
<th>Dept</th>
<th>Name</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>PHYS/NBE</td>
<td>Janne Blomqvist</td>
<td>Otakaari 1, Y415a / Otakaari 3, F354</td>
</tr>
<tr>
<td>CS/NBE</td>
<td>Mikko Hakala</td>
<td>T-building A243 / Otakaari 3, F354</td>
</tr>
<tr>
<td>CS</td>
<td>Simo Tuomisto</td>
<td>T-building A243</td>
</tr>
<tr>
<td>ELEC</td>
<td>Tarmo Simonen</td>
<td>Otakaari 3, F417</td>
</tr>
<tr>
<td>PHYS</td>
<td>Ivan Degtyarenko</td>
<td>Otakaari 1, Y415a</td>
</tr>
<tr>
<td>CS/SCI</td>
<td>Richard Darst</td>
<td>T-building A243</td>
</tr>
</tbody>
</table>

Science-IT trainings

We have regular training in topics relevant to HPC and scientific computing. In particular, each January and June we have a “kickstart” course which teaches you everything you need to know to do HPC work. Each Triton user should come to one of these. For the schedule, see our training page.
Getting a detailed bug report with triton-record-environment

We have a script named `triton-record-environment` which will record key environment variables, input, and output. This greatly helps in debugging.

To use it to run a single command that gives an error:

```
triton-record-environment YOUR_COMMAND
Saving output to record-environment.out.txt
...
```

Then, just check the output of `record-environment.out.txt` (it shouldn’t have any confidential information, but make sure) and send it to us/attach it to the bug report.

If you use Python, add the `-p` option, matlab should use `-m`, and graphical programs should use `-x` (these options have to go before the command you execute).

Triton accounts

You need to request Triton access separately, however, the account information (username, password, shell, etc) is shared with the Aalto account so there is not actually a separate account. Triton access is available to any researcher at Aalto for free. Resources are funded by departments, and distributed by a fairshare algorithm: members of departments and schools which provide direct funding have a greater share.

Please contact us via the support email address (not personal email) and we will set up your account. Please tell us your department, supervisor, Aalto account, and see the conditions below. (For future help, you should probably use our issue tracker: see the Getting Triton help page.)

A few prerequisites:

- you must have valid Aalto account
- you must have working email address (Aalto)
- you must accept Triton usage policies: requesting access to Triton cluster at Aalto University, you accept all the conditions listed in the Triton usage policies, including the data and privacy policies.
- You should tell us your department/school in your account creation request.
- Accounts are for
  - Researchers (as in, affiliated with a research PI in any way). Please tell us who your supervisor is in your account request.
  - Students coming to one of our Scientific Computing in Practice courses which uses Triton. You will be specifically told if this is the case
  - Other students not doing research needing computational facilities should check out our introduction for students. This includes most student projects as part of courses, unless you are effectively joining a research group to do a project.

You know that you have Triton access if you are in the `triton-users` group at Aalto: `groups` shows this on Aalto linux machines.

Your department/unit

When you get an account, you get added to a unit’s group, which is “billed” for your usage. If you change Aalto units, this may need updated. Check `sshare -U` or `sshare` and if it’s wrong, let us know (the units are first on the line). (These are currently by department, so changes are not that frequent)
Password change and other issues

Since your Triton account is a regular Aalto account, for any password change, shell change etc use Aalto services. You can always do these on the server kosh.aalto.fi (at least).

If you are in doubts, in case of any account related issue your primary point of contact is your local support team member via the support email address. Do not post such issues on the tracker.

Account deactivation / remove from mailing list

Your account currently lasts as long as your Aalto account does. If you want to end your account early, contact your local support via the esupport-triton email address. This way, you will also be unsubscribed from the triton-users maillist (the mailing list is required for everyone who has an account).

Before you leave, please clean up your home/work/scratch directories data. Consider who should have your data after you are done: does your group still need access to it? You won’t have access to the files after your account is deactivated. Note that scratch/work directory data are unrecoverable after deleting, which will happen eventually.

Terms of use/privacy policy

See the Usage policies and legal page.

Usage policies and legal

Acceptable Use Policy and Terms of Service

By using the Triton cluster resources, you shall be deemed to accept these conditions of use:

1. You shall only use Triton cluster to perform work, or transmit or store data consistent with the stated goals and policies of Aalto University and in compliance with these conditions of use.

2. You shall not use Triton cluster for any unlawful purpose and not (attempt to) breach or circumvent any administrative or security controls. You shall respect copyright and confidentiality agreements and protect your credentials (e.g. user login name, password, ssh private key), sensitive data and files.

3. You shall immediately report any known or suspected security breach or misuse of Triton cluster or credentials to the cluster support team. 

4. Use of the cluster is at your own risk. There is no guarantee that the cluster will be available at any time or that it will suit any purpose.

5. Logged information, including information provided by you for registration purposes, shall be used for administrative, operational, accounting, monitoring and security purposes only in accordance with the policy below. This information may be disclosed to other organizations anywhere in the world for these purposes in the extent allowed by local laws. Although efforts are made to maintain confidentiality, no guarantees are given.

6. The cluster support team is entitled to regulate and terminate access for administrative, operational and security purposes and you shall immediately comply with their instructions.

7. You are liable for the consequences of any violation by you of these conditions of use.

8. You agree to explicitly mention and acknowledge the use of Science-IT resources in your work in any reports, workshops, papers or similar that result from such usage. Appropriate reference can be found at Acknowledgment of Triton usage.
Triton data (privacy) policy

Triton is a part of Aalto University IT systems, thus is fundamentally governed by the Aalto Privacy Policy for Employees or Privacy Policy for Students, the latest versions of which can always be found on aalto.fi.

For clarity, in this section, we describe the special cases of Triton data:

In summary:

- **The Triton account** is not a separate account, it is part of the Aalto account. We do not control that.

- **Triton usage statistics and logs.** Triton is used for university academic research only, so this information may be used for reporting and management in any way. Identifying information won’t be public, but note that

- **Data stored on Triton.** We are not the controller of this data. Data in your personal directories is controlled by you, and data in shared directories is controlled by the manager of that group. See the section below for more information on this data.

- **HAKA login data** (JupyterHub only). This is used to secure access to JupyterHub. Only your Aalto account name is requested, it is compared and immediately discarded (Triton is already linked to your Aalto account).

- **The triton-users mailing list** is automatically formed from all Aalto accounts in the triton-users group (everyone with an account). This is used to send service announcement and information related to scientific computing. This subscription is intrinsically tied to the Triton account and a requirement of the cluster usage. (Email information held by Aalto IT services).

We do not consider the Triton management data to consist of a personal data file (this is covered under Aalto policies), but for full disclosure we describe our use of data.

*Note about research data:* This section does not cover any data which users store on the cluster: for that, the user is the controller and Science-IT is only a processor. You are responsible for any administrative privacy matters. The following subsections relate only to administrative metadata.

**Controller and contact**

*Controller:* Aalto Science-IT, Aalto University, Espoo, Finland. Contact information. Please use the support email alias for account and personal data queries.

Account information comes from Aalto ITS registers.

**The purpose for processing the personal data**

Data is processed and stored in accordance with our agreement to provide a HPC cluster service including accounting and reporting, in accordance with the usage agreement. The cluster may only be used to support Aalto (not personal) activities, and all thus usage metadata represents Aalto activities and is owned by Aalto University.

**Types of data**

Triton stores the information necessary for provision of its services, including accounting, funding, and security. This includes logs of all operations and metadata of stored data. Data is only generated when a users uses the cluster. For example (including but not limited to):

- Connection logs
- Job submission and statistics logs
- Filesystem and storage metadata and logs
Aalto scientific computing guide

Uses of data

Data is used in the provision of the HPC cluster service. Primarily, this is through accounting, reporting, and scheduling of tasks. Historical data will automatically adjust future cluster priority.

Sources of information

Data is produced during the use of Triton for research purposes. This data is generated directly by users while using the cluster. Account information is provide by Aalto University, and in general not stored or processed here.

Data sharing

Data may be used for internal Aalto reporting and accounting (usually but not always aggregated at least at the group level), and used in non-identifiable forms in public reports and statistics. It may also be used as needed to investigate usage matters.

All users of the cluster may inspect the usage information and job statistics of the entire cluster (including all other users).

Timeframe

Data related to usage remains as long as the user has an active Triton account. Technical logging data allows accounting and reporting, and may be kept as long as needed for security and reporting purposes (indefinately). Where possible, this may be in anonymous form.

Legal notices

Data is stored in Finland in Aalto or CSC approved facilities. Access is only via Aalto account.

You may request rectification of your data. However, most data is technical logging information which can not be removed or changed.

You may cease using the cluster, remove your research data, and request your account be closed (this does not close your Aalto account because we do not control that), but historical usage data will remain for accounting purposes. Should technical errors in data be identified, a bug should be reported.

You may access and extract your own data using the standard interfaces described in the user guide.

Identifiable administrative metadata and accounting data is not transferred outside of the EU/EEA except under proper agreement. (We have to say that, but in reality identifiable data is never transferred out of Aalto or maybe the FGCI consortium in Finland).

You may lodge a complaint with the Aalto data protection officer (see Aalto privacy notices for up to date contact information) or the Finnish supervision authority Tietosuoja.

Research and home data stored on cluster

We provide a storage service for for data stored on the cluster (scratch and home directories):

Our responsibility is limited to keeping this data secure and providing access to the corresponding Aalto accounts. The shared directory manager should be able to make choices about data. We do not access this data except with an
explicit request, but for management purpose we do use the file metadata (`stat $filename`). For full information, see the Science-IT data policy.

- We do not look into private files without your explicit request (if you want help with something, explicitly tell us if we can look at them).
- If your files are made cluster-readable (the `chmod` “other” permissions), you give permission for others to look at contents. Note that this is not the default setting.
- Should you report a problem, we may run `stat` as superuser on relevant files to determine basic metadata without further checks.
- Should you have a problem that requires us to look at the contents of files or directories, we must first have your explicit permission (either in writing or in person).

Our data storage service is suitable for confidential data. You must ensure that permissions are such that technical access is limited.

**Acknowledging Triton**

**Acknowledgement line**

Triton and Science-IT gets funding from departments and Aalto, so it is critical that we show them the results of our work. Thus, please note that if you use the cluster for research that is published or presented in a talk or poster form you must acknowledge the Science-IT project by School of Science, that funds the Triton and affiliated resources. By published work we mean everything like articles, doctoral theses, diplomas, reports, other relevant publications. Use of triton can be anything: CPUs, GPUs, or the storage system (note that the storage system is the “scratch” system, which is cross-mounted to several different departments - you can use Triton without logging into it.)

An appropriate acknowledgement line might be one of:

We acknowledge the computational resources provided by the Aalto Science-IT project.

or

The calculations presented above were performed using computer resources within the Aalto University School of Science “Science-IT” project.

You can decide which one fits better to your text/slides. Rephrasing is also fine, the main issue is referencing to Science-IT and Aalto. (Note that this does not exist in various funding databases, this is an Aalto internal project.)

**Reporting**

We can’t automatically track all the Triton publications. We need all users to link the publications to Science-IT in ACRIS. It takes about 30 seconds if you aren’t looking at ACRIS now, or 5 when you are already there. All publications are required to be in ACRIS anyway, so this is a fast process.

You can see the already-reported publications here: https://research.aalto.fi/en/equipment/scienceit(27991559-92d9-4b3b-95ee-77147899d043)/publications.html

Instructions:

1. Log in to ACRIS: https://acris.aalto.fi
2. Find your publication: Research Output (left sidebar) -> Click on your publication
   - If your publication is not already there, then see your department’s ACRIS instructions, or the general help below.
3. Link it to Science-IT: scroll down to “Relations” -> “Facilities/Equipment” -> Search “Science-IT” and select it. (This is on the main page, not the separate “Relations” page.)

4. Click Save at the bottom of the window.

5. Repeat for all publications (and datasets, etc.)

You are done! You can see if your publications appears on the list above.

More help:

- General ACRIS help: ACRIShelp
- Manually adding journal article (most are automatically transferred): Submitting a journal article on ACRIShelp.

Should you have problems, first contact your department’s ACRIS help. If a publication or academic output somehow can’t be linked, let us know and we will make sure that we include it in our own lists.

### 6.1.3 Tutorials

These are designed to be read in-order by every Triton user when they get their accounts (except maybe the last ones).

**About Science-IT and Triton**

This is the first tutorial. The next is Connecting to Triton.

Science-IT is an Aalto infrastructure for scientific computing. Its roots was a collaboration between the Information and Computer Science department (now part of CS), Biomedical Engineering and Computational Science department (now NBE), and Applied Physics department. Now, it still serves all Aalto and is organized from the School of Science.

Our basic administrative web page is http://science-it.aalto.fi, but it has mostly administrative info. This site, http://scicomp.aalto.fi, is our practical info for users.

You are now at the first step of the Triton tutorial.

**About Triton**

Triton is a mid-sized heterogeneous computational Linux cluster. This means that we are not at a massive scale (though we are, after CSC, the largest publically known known cluster in Finland). We are heterogeneous, so we continually add new hardware and incrementally upgrade. We are designed for scientific computing and data analysis. We use Linux as an operating system (like most supercomputers). We are a cluster: many connected nodes with a scheduling system to divide work between them. The network and some storage is shared, CPUs, memory, and other storage is not shared.

On an international scale of universities, the power of Triton is relatively high and it has a very diverse range of uses, though CSC has much more. Using using this power requires more effort than using your own computer.

**Getting skills**

See also:

Main article: Training

As time goes on, computers are getting easier and easier to use. However, research is not a consumer product, and the fact is that you need more knowledge to use Triton than most people learn in academic courses.
6.1. Triton user guide
Science-IT has created a (still under development) modular training plan, divided into A (basics, use basic software), B (Linux usage), C (high performance computing), and D (advanced HPC). In order to use Triton well, you need be somewhat proficient at Linux usage (B level). In order to do parallel work, you need to be good at the B-level and also somewhat proficient at the HPC level (C-level). This tutorial and user guide covers the C-level, but it is up to you to reach the B-level first.

See our training program and plan for suggested material for self-study and lessons. We offer routine training, see our Scientific Computing in Practice lecture series page for info.

Getting help

See also:

Main article: Getting Triton help

There are many ways to get help. Most daily questions should go to our issue tracker (direct link), which is hosted on Aalto Gitlab (login with the HAKA button). This is especially important because many people end up asking the same questions, and in order to scale everyone needs to work together.

Please, don’t send us personal email, because not everyone is here all the time and you may end up asking someone other than the best person. Personal email is also very likely to get lost. By the same token, we have a service email address, but this should only be used for account matters. If it affects others (software, usage problems, etc), use the issue tracker, otherwise we will point you there and spend lots of time answering the same questions over and over.

Also, always search this scicomp docs site and old issues in the issue tracker.

We have weekly “SciComp garage” sessions where we provide help in person.

However, the most important thing is to be able to continually develop your skills to help yourself and your colleagues. See the previous section for our solution for this.

Software

Triton, being a shared system, has more complicated software requirements. In an upcoming tutorial, you will learn how to use existing software. Be aware that installing your own is possible (and people do it all the time), but does require some attention to details.

What’s next?

The next tutorial is Connecting to Triton.

Connecting to Triton

Tutorials: previous (intro), next.

All access to Triton is via Secure Shell (ssh).

You can connect to triton.aalto.fi from Aalto and CSC networks. Aalto networks include: Wired workstation networks, eduroam, and the aalto wireless network only if you are using an Aalto managed laptop (otherwise aalto is like aalto open). If you connect to the Aalto VPN, you will be on the Aalto networks.

For SSHing to Triton from outside of your department or CSC, please login first to a university server (like kosh.aalto.fi or taltta.org.aalto.fi) and then open a session to triton.aalto.fi.
Note: Are you here for a SciComp KickStart course? You just need to make sure you have an account and then be able to connect via ssh (first section here), and you don’t need to worry about the graphical application parts. Everything else, we do tomorrow.

Note: Triton uses Aalto accounts, but your account must be activated first.

See also:
The shell crash course is a prerequisite to this material.

There are different ways of connecting:

<table>
<thead>
<tr>
<th>Method</th>
<th>About</th>
<th>From where?</th>
</tr>
</thead>
<tbody>
<tr>
<td>ssh</td>
<td>Works everywhere, from everywhere. Firewalls may make things hard sometimes.</td>
<td>Aalto networks only, otherwise ssh to kosh and then Triton</td>
</tr>
<tr>
<td><a href="https://jupyter.triton.aalto.fi">https://jupyter.triton.aalto.fi</a></td>
<td>Jupyter interface, but provides shell access via web browser.</td>
<td>Whole internet</td>
</tr>
<tr>
<td><a href="https://vdi.aalto.fi">https://vdi.aalto.fi</a></td>
<td>Virtual desktop, from there you have to ssh to Triton anyway but gets you past firewalls and can run graphical programs via SSH.</td>
<td>Whole internet</td>
</tr>
</tbody>
</table>

Connecting via ssh

Linux

All Linux distributions come with an ssh client, so you don’t need to do anything. To use graphical applications, use the standard --x option, nothing extra needed.:

```bash
ssh triton.aalto.fi
# OR, if your username is different:
ssh username@triton.aalto.fi
```

If you are from outside the Aalto networks:

```bash
ssh kosh.aalto.fi       # or ssh username@kosh.aalto.fi
ssh triton.aalto.fi
```

Mac

ssh is installed by default, same as Linux. Run it from a terminal, same command as Linux. To run graphical applications, you need an to install an X server (XQuartz).

Windows

You need to install a ssh client yourself: PuTTY is the standard one. If you want to run graphical programs, you need an X server on Windows: see this link for some hints. (Side note: putty dot org is an advertisement site trying to get you to install something else.)
You should configure this with the hostname, username, and save the settings so that you can connect quickly.

If you are outside the Aalto networks, you need to first connect to kosh.aalto.fi or some other server, and then use the Linux instructions to connect to Triton.

**Advanced options**

Check the *advanced ssh information* to learn how to log in without a password, automatically save your username and other options, and more. It really will save you time.

**Exercise**

1. Connect to Triton. List your home directory and work directory `$WRKDIR`.
2. Check the uptime and load of the login node: `uptime` and `htop` (q to quit). What else can you learn about the node?
3. Check what your default shell is: `echo $SHELL`. Go ahead and change your shell to bash if it’s not yet (see below).

**Connecting via https://jupyter.triton.aalto.fi**

Jupyter is a web-based way of doing computing. But what some people forget is that it has a full-featured terminal and console included.

Go to https://jupyter.triton.aalto.fi (not .cs.aalto) and log in. Select “Slurm 5 day, 2G” and start.

To start a terminal, click File → New → Terminal - you do anything you need to do from here, same as `ssh`. If you need to edit text files, you can also do that through JupyterLab (note: change to the right directory *before* creating a new file!).

To learn more about Jupyterlab, you need to read up elsewhere, there are plenty of tutorials.

**Connecting via https://vdi.aalto.fi**

If you go to https://vdi.aalto.fi, you can access a cloud-based Aalto workstation. HTML access works from everywhere, or download the “VMware Horizon Client” for a better connection. Start a Ubuntu desktop (you get Aalto Ubuntu).

From there, you **have to use the normal ssh instructions** (via the Terminal application) using the instructions you see above: `ssh triton.aalto.fi`.

For more information, see it.aalto.fi.

**Change your shell to bash (Aalto)**

*Only needed if you shell isn’t already bash.*

The thing you are interacting with when you type is the shell - the layer around the operating system. bash is the most common shell, but the Aalto default shell used to be zsh (which is more powerful in some ways, but harder to teach with). For new users (if you joined Aalto after autumn 2018) you should have bash as the default. We recommend that you check and change your shell to bash.

You can determine if you shell is bash by running `echo $SHELL`. Does it say `/bin/bash`?
If not, ssh to kosh.aalto.fi and run chsh -s /bin/bash. It may take 15 minutes to update, and you will need to log in again.

**Transferring files**

You’ll actually learn this in the next section, the *data storage tutorial*. It is easiest to mount them using SMB, and on Aalto workstations and taltta.aalto.fi they are mounted at /m/triton/{scratch,work}/. You can also use an sftp (which works over ssh, so will work from anywhere you can access Triton) client such as Filezilla to transfer files. See the next tutorial (or FAQ).

**What’s next?**

ssh is one of the most fundamental Linux programs: by using it well, you can really do almost anything from anywhere. The .ssh/config file is valuable to set up. If ssh is annoying to use, ask for some help in getting it working well. See the advanced ssh information.

The next tutorial is about software and modules.

**Software Modules**

There are hundreds of people using Triton. They all have different software needs, including conflicting versions required! How do we handle this without making a mess?

The answer is the standard “module” system Lmod. It allows us to have unlimited number of different software packages installed, and the user can select what they want. Modules include everything from compilers (+their required development files), libraries, and programs. If you need a program installed, we will put it in the module system.

In a system the size of Triton, it just isn’t possible to install all software by default for every user.

**The summary**

A module lets you adjust what software is available. For example, let’s see what our Python is:

```
$ which python3
/usr/bin/python3
$ python3 -V
Python 3.4.9
```

Now let’s load the anaconda3 module, a more up to date Python with a lot of libraries already included:

```
$ module load anaconda3
$ which python3
/share/apps2/anaconda/anaconda3/latest/bin/python3
$ python3 -V
Python 3.6.8 :: Anaconda custom (64-bit)
```

As you see, we have a newer Python. Let’s go back to blank:

```
$ module purge
```
What is a module?

Let’s look at py-gpaw module (abbreviated output shown):

```
$ module show py-gpaw
---
/share/apps/spack/modules/linux-centos7-x86_64/Core/py-gpaw/1.3.0-openmpi-
    --scalapack-python3.lua:
---
whatis("Name : py-gpaw")
whatis("Version : 1.3.0")
whatis("Short description : GPAW is a density-functional theory (DFT) Python code based on the projector-augmented wave (PAW) method and the atomic simulation environment (ASE).")
help(["GPAW is a density-functional theory (DFT) Python code based on the projector-augmented wave (PAW) method and the atomic simulation environment (ASE)."])
load("py-scipy/1.1.0-python3")
load("libxc/3.0.0")
load("python/3.6.3")
load("fftw/3.3.8-openmpi")
load("py-ase/3.15.0-python3")
load("py-numpy/1.14.3-python3")
load("netlib-scalapack/2.0.2-openmpi")
load("openmpi/2.1.5")
load("openblas/0.3.2")
prepend_path("PATH","/share/apps/spack/software/py-gpaw/1.3.0/vmhtg6i/bin")
prepend_path("LIBRARY_PATH","/share/apps/spack/software/py-gpaw/1.3.0/vmhtg6i/lib")
prepend_path("LD_LIBRARY_PATH","/share/apps/spack/software/py-gpaw/1.3.0/vmhtg6i/lib")
prepend_path("CMAKE_PREFIX_PATH","/share/apps/spack/software/py-gpaw/1.3.0/vmhtg6i/
    --python3.6/site-packages")
prepend_path("CPATH","/share/apps/spack/software/libxml2/2.9.8/f4u6mya/include/libxml2")
setenv("PY_GPAW_ROOT","/share/apps/spack/software/py-gpaw/1.3.0/vmhtg6i")
```

You can see that it has some meta-info, then adjusts various environment paths, so that when you run gpaw it runs the program from /share/apps/spack/software/py-gpaw/1.3.0/vmhtg6i/bin. This is almost magic: we can have many versions of any software installed, and everyone can pick what they want, with no conflicts.

You can search for modules using the command `module spider`.

You can list currently loaded modules using `module list`.

What’s going on under the hood here?

In Linux systems, different environment variables like `$PATH` and `$LD_LIBRARY_PATH` to figure out how to run programs. Modules just cleverly manipulate these so that you can find the software you need, even if there are multiple versions available.

Exercise: where is Matlab?

Let’s say you want to use Matlab. You log in and try:
$ matlab
-bash: matlab: command not found

So first search for it:

module spider matlab

matlab:

Versions:
...
matlab/r2016a
matlab/r2016b
matlab/r2017b
matlab/r2018a
matlab/r2018b
matlab/r2019a

We see there are a lot of versions available.

Load the latest version of Matlab as:

module load matlab

run it to check the version you got, then close it and swap the version with the older one, try

**Loading modules**

Normally, you run `module load MODULE_NAME`. Do it in your open shell, your scripts, or whatever. You could put it in your `~/.bash_profile`, but then it will always automatically load it - perhaps even if you don’t expect it. Watch out for this if you get un-explainable bugs - it may be best to explicitly load what you need. You can load any number of modules, and there is a basic dependency/conflict system.

Each time you load a module, it resolves all the dependencies. This can mean that loading module takes a long time, but there is a solution: `module save $collection_name` and `module restore $collection_name`.

**Exercise: make a module collection**

Try loading the `graph-tool` module. How long does it take? Use `module list` to see how many things were actually loaded:

```
module load graph-tool       # 600 seconds!
module list                 # 72 modules!
```

Then, do `module save my-collection`. Then `module purge` to unload everything. Now, do `module restore my-collection`. Was it much faster?:

```
module save my-gt
module purge
module restore my-gt        # only 3 seconds
module list                 # same 72 modules
```

You may occasionally need to rebuild your collections if we re-organize things (it will tell you, just re-save).
Full reference

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>module load <code>NAME</code></td>
<td>load module</td>
</tr>
<tr>
<td>module avail</td>
<td>list all modules</td>
</tr>
<tr>
<td>module spider <code>NAME</code></td>
<td>search modules</td>
</tr>
<tr>
<td>module list</td>
<td>list currently loaded modules</td>
</tr>
<tr>
<td>module show <code>NAME</code></td>
<td>details on a module</td>
</tr>
<tr>
<td>module help <code>NAME</code></td>
<td>details on a module</td>
</tr>
<tr>
<td>module unload <code>NAME</code></td>
<td>unload a module</td>
</tr>
<tr>
<td>module save <code>ALIAS</code></td>
<td>save module to this alias (saved in ~/.lmod.d/)</td>
</tr>
<tr>
<td>module restore <code>ALIAS</code></td>
<td>load saved module set (faster than loading individually)</td>
</tr>
<tr>
<td>module purge</td>
<td>unload all loaded modules (faster than unloading individually)</td>
</tr>
</tbody>
</table>

But what about the software?

You know how to load modules software now, but what about specific software packages? See the upcoming applications tutorial for more info.

Final notes

If you have loaded modules when you build/install software, remember to load the same modules when you run the software (also in Slurm jobs). You’ll learn about running jobs later, but the `module load` should usually be put into the job script.

The modules used to compile and run a program become part of its environment and must always be loaded.

If you are compiling things and want it to work in the future, load a particular version of the module (`module load $name/$version`). Then, things will keep working even if we upgrade in the meantime (in fact, this is a primary advantage of modules).

We use the Lmod system.

Lmod uses environment variables. Thus, they must be sourced by a shell and are are only transferred to child processes. Anything that clears the environment clears loaded modules. Module loading is done by special functions (not scripts) that are shell-specific and set environment variables.

Some modules are provided by Aalto Science-IT, some by CSC. You could even make your own user modules if needed.

Exercises

Before each exercise, `module purge` to clear all modules.

1. `module avail` and check what you see. Find some examples of software that have many different versions available. Load the oldest version of that software.

2. `PATH` is an environment variable that shows from where programs are run. See its current value using `echo $PATH`. Then, load some toolchain module such as `goolfc/triton-2017a`. List what it loaded. Check the value of `PATH` again. Why is there so much stuff?

3. (Advanced of #2). Same as number 2, but use `env | sort > filename` to store environment variables, then swap to `goolfc/triton-2016a`. Do the same, and compare the two outputs using `diff`.
3. Load a module with many dependencies, such as R/3.3.2-iomkl-triton-2017a-libX11-1.6.3 and save it as a collection. Compare the time needed to load the module and the collection. (Does time not work? Change your shell to bash, see the previous tutorial.)

4. (Advanced) Load GROMACS. Use ‘which’ to find where command ‘gmx’ is and then use ‘ldd’ to find out what libraries it uses. Load incompatible toolchain e.g. goolf. Check ldd output again.

Next steps

Next, move on to the applications tutorial

Applications

The previous tutorial taught you about modules. Here, we talk a bit more about the overall process of finding, building, and compiling software. These days, installing and managing scientific software is taking more and more time, thus we need to specifically talk about it some.

See also:

This assumes that you have read the previous tutorial about modules.

Main article: Applications: General info

Available software

You can find what software we have available in different ways:

- First, you should check our applications page and see if the software you need is already available and if it has instructions.
- You should also search this site to see what you can find (though not that not everything is in the Triton section here - some applies to Aalto workstations or own computers).
- Then, you should search the issue tracker to see if there are previous issues about it - not everything is always updated.
- Check the available modules with module spider and module avail (next section) to see what’s available but undocumented.

If you find software available, you will usually load it via a module:

Common applications

For reference, here are the most common applications:

- **Python**: module load anaconda3 for the Anaconda distribution of Python 3, including a lot of useful packages. More info.
- **R**: module load r for a basic R package. More info.
- **Matlab**: module load anaconda3 for the latest Matlab version. More info.
Aalto scientific computing guide

Modules

As you learned in the previous tutorial, module is a command that allows you to get and remove access to other software - because not everything can be available at once. Refer to the section on modules for info - basically module load $NAME.

Not all of the software we have available is documented. You can module spider $NAME to try to see if you can find a module that way. Note that this is partially case sensitive so it can be hard to find things - you might need to look through module avail some, too. To see just what a module does, remember module show.

Singularity containers

See also:
Main article: Singularity Containers

Some software has gotten so hard that it just can’t be installed, and for that we use containers. A software container is basically a complete self-contained operating system environment. Another advantage of containers is that it makes it easy to move installed software from system to system, so that you can have the same environment everywhere.

You can read about singularity from its page. If you load a module that uses singularity, nothing will happen at first. You execute your software using singularity_wrapper exec, or use singularity_wrapper shell to get a shell in there.

Compilers and toolchains

See also:
Main article: Compilers and toolchains

Some people need to compile your own code. You can try to use the operating system gcc, but it is likely too old and doesn’t have the necessary libraries. Instead, load a toolchain which contains a fixed compiler and support libraries.

For GCC-based tool chains, check module spider goolf and, for example, module load goolf/triton-2017a. For Inter-based compilers, try module spider iomkl and, for example, module load iomkl/triton-2017a. (These stand for “gcc openmpi openblas lapack fftw” and “intel openmpi intel-mkl”)

Toolchains change often - check back for latest info if you need to use one.

Requesting new software

We aim to install a good base of software for our users - but it’s not possible to keep up with all requests. If you need something, submit a request to our issue tracker, but be aware that despite best efforts, we can’t do everything. See the main applications info page.

Exercises

1. Figure out how to use tensorflow (this is not a software problem, but a searching the documentation problem). Make it work enough to do python and import tensorflow – though you will get an error which you will learn to solve in a later lesson.

2. Figure out how to run openfoam by searching the docs (use the new image). Using singularity_wrapper, run foamExec so that it fails with the error message no application specified. Try singularity_wrapper shell, too.
Next steps

The next tutorial is data storage

Data storage

See also:

- Storage: Lustre (scratch)
- Storage: local drives
- Quotas
- Small files
- The storage advanced page: Storage

In this tutorial, we go over places to store data on Triton and how to access it remotely.

Basics

Triton has various ways to store data. Each has a purpose, and when you are dealing with the large data sets or intensive IO, efficiency becomes important.

Roughly, we have small home directories (only for configuration files), large Lustre (scratch and work, large, primary calculation data), and special places for scratch during computations (local disks). At Aalto, there is aalto home, project, and archive directories which, unlike Triton, are backed up but don’t scale to the size of Triton.

A file consists of its contents and metadata. The metadata is things like user, group, timestamps, permissions. To view metadata, use ls -l or stat.

Filesystem performance can be measures by both IOPS (input-output operations per second) and stream I/O speed. /usr/bin/time -v can give you some hints here. You can see the profiling page for more info.

Think about I/O before you start! - General notes

When people think of computer speed, they usually think of CPU speed. But this is missing an important factor: how fast can data get to the CPU? In very many cases, input/output (IO) is the true bottleneck and must be considered just as much as processor speed. In fact, modern computers and especially GPUs are so fast, it is very easy for a few GPUs with bad data access patterns to bring the cluster down for everyone.

The solution is similar to how you have to consider memory: there are different types of filesystems with different tradeoffs between speed, size, and performance, and you have to use the right one for the right job. Often times, you have to use several in tandem: for example, store original data on archive, put your working copy on scratch, and maybe even make a per-calculation copy on local disks.

Consider:

- How much IO in the first place? Do you continually re-read the same data?
- What’s the pattern of it, and which filesystem is best for it? If you read all at once, scratch is fine, but if there are many small files or random access, local disks may help.
- Do you write log files / checkpoints more often than is needed?
- Some programs use local disk as swap-space. Only turn on if you know it is reasonable.
There’s a checklist in the storage details page.

Avoid many small files! Use a few big ones instead. (we have a dedicated page on the matter)

Summary table

<table>
<thead>
<tr>
<th>Name</th>
<th>Path</th>
<th>Quota</th>
<th>Backup locality</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Home</td>
<td>$HOME or /home/ $username/</td>
<td>hard quota 10GB</td>
<td>Night</td>
<td>Small user specific files, no calculation data.</td>
</tr>
<tr>
<td>Work</td>
<td>$WRKDIR or /scratch/ work/ $username/</td>
<td>200GB and 1 million files</td>
<td>x</td>
<td>Personal working space for every user. Calculation data etc. Quota can be increased on request.</td>
</tr>
<tr>
<td>Scratch</td>
<td>/scratch/ $dept/ $project/</td>
<td>on request</td>
<td>x</td>
<td>Department/group specific project directories.</td>
</tr>
<tr>
<td>Local temp</td>
<td>/tmp/</td>
<td>limited by disk size</td>
<td>x</td>
<td>Primary (and usually fastest) place for single-node calculation data. Removed once user’s jobs are finished on the node.</td>
</tr>
<tr>
<td>Local persistent</td>
<td>/1/</td>
<td>varies</td>
<td>x</td>
<td>Local disk persistent storage. On servers purchased for a specific group. Not backed up.</td>
</tr>
<tr>
<td>ramfs (login nodes only)</td>
<td>$XDG_RUNTIME_DIR</td>
<td>limited by memory</td>
<td>x</td>
<td>Ramfs on the login node only, in-memory filesystem</td>
</tr>
</tbody>
</table>

Home directories

The place you start when you log in. For user init files, some small config files, etc. No calculation data. Daily backup. Usually you want to use scratch instead.

scratch and work: Lustre

This is the big, high-performance, 2PB Triton storage. The primary place for calculations, data analyzes etc. Not backed up but is reliable against hardware failures (RAID6, redundant servers), but not safe against human error.. It is shared on all nodes, and has very fast access. It is divided into two parts, scratch (by groups) and work (per-user).

In general, always change to $WRKDIR or a group scratch directory when you first log in and start doing work.

Lustre separates metadata and contents onto separate object and metadata servers. This allows fast access to large files, but a larger overhead than normal filesystems. See our info on small files.

See Storage: Lustre (scratch)

Local disks

Local disks are on each node separately. For the fastest IOs with single-node jobs. It is cleaned up after job is finished. Since 2019, things have gotten a bit more complicated since our newest (skl) nodes don’t have local disks. If you want
to ensure you have local storage, submit your job with `--gres=spindle`.
See the *Compute node local drives* page for further details and script examples.

**ramfs - fast and highly temporary storage**

On *login nodes only*, `$XDG_RUNTIME_DIR` is a ramfs, which means that it looks like files but is stored only in memory. Because of this, it is extremely fast, but has no persistence whatsoever. Use it if you have to make small temporary files that don’t need to last long. Note that this is no different than just holding the data in memory, if you can hold in memory that’s better.

**Quotas**

All directories under `/scratch` (as well as `/home`) have quotas. Two quotas are set per-filesystem: disk space and files number.

Disk quota and current usage are printed with the command `quota`. ‘space’ is for the disk space and ‘files’ for the total files number limit. There is a separate quota for groups on which the user is a member.

```
$ quota
User quotas for darstr1
   Filesystem space  quota  limit  grace files  quota  limit  grace
/home       484M   977M  1075M    10264     0    0
/scratch   3237G  200G  210G   158M    1M    1M
Group quotas
   Filesystem group  space  quota  limit  grace files  quota  limit  grace
   /scratch  domain users   132G   10M   10M    310M   5000
   /scratch  some-group    534G   524G   524G    7534  1000M
   /scratch  other-group   16T   20T   20T   1088M     5M
```

If you get a quota error, see the *quotas page* for the solution.

**Accessing and transferring files remotely**

Transferring files to/from triton is exactly the same as any other remote Linux server.

**Remote mounting using SMB**

By far, remote mounting of files is the easiest method to transfer files. If you are not on the Aalto networks (wired, eduroam, or *aalto* with Aalto-managed laptop), connect to the *Aalto VPN* first. Note that this is automatically done on some department workstations (see below) - if not, request it!

The scratch filesystem can be remote mounted using SMB inside secure Aalto networks at the URLs

- `scratch: smb://data.triton.aalto.fi/scratch/`
- `work: smb://data.triton.aalto.fi/work/$username/`

On different operating systems:
• Linux (Ubuntu for example): File manager (Nautilus) → File → Connect to server. Use the smb:// URLs above.

• Windows: In the file manager, go to Computer (in menu bar on top, at least in Windows 10) → Map Network Drive and “Map Network Drive”. In Windows 10 → “This PC” → right click → “Add Network Location”. (Note that this is different from right-click “Add network location” which just makes a folder link and has had some problems in the past.) Use the URLs above but replace smb:// with \ and / with \. For example, \data.triton.aalto.fi\scratch\.

• Mac: Finder → Go → Connect to Server. Use the smb:// URLs above.

Depending on your OS, you may need to use either your username directly or AALTO\username.

Using sftp

The sftp protocol uses ssh to transfer files. On Linux and Mac, the sftp command line program are the must fundamental way to do this, and are available everywhere.

A more user-friendly way of doing this (with a nice GUI) is the Filezilla program.

Below is an example of the “raw” sftp usage:

```
# Copying from HOME to local PC
user@pc123 $ sftp user12@triton.aalto.fi:filename
Connected to triton.aalto.fi.
Fetching /home/user12/filename to filename
# copying to HOME
user@pc123 $ sftp -b - user12@triton <<< 'put testCluster.m'
sftp> put foo
# copying to WRKDIR
user@pc123 $ sftp -b - user12@triton:/scratch/work/USERNAME/ <<< 'put testCluster.m'
...
```

Rsync

Rsync is similar to sftp, but is smarter at restarting files. Use rsync for large file transfers. rsync actually uses ssh, so you can rsync from anywhere you can ssh from.

Remote mounting using sshfs

sshfs is a neat program that lets you mount remote filesystems via ssh only. It is well-supported in Linux, and somewhat on other operating systems. It’s true advantage is that you can mount any remote ssh server - it doesn’t have to be set up specially for SMB or any other type of mounting. On Ubuntu, you can mount by “File → Connect to server” and using sftp://triton.aalto.fi/scratch/work/USERNAME.

The below uses command line programs to do the same, and makes the triton_work on your local computer access all files in /scratch/work/USERNAME. Can be done with other folders:

```
mkdir triton_work
sshfs USERNAME@triton.aalto.fi:/scratch/work/USERNAME triton_work
```

Note that ssh binds together many ways of accessing Triton, with a similar syntax and options. ssh is a very important program and binds together all types of remote access, and learning to use it well will help you for a long time.
Exercises

1. Mount your work directory by SMB and transfer a file to Triton. Note that you must be on eduroam, the aalto with Aalto laptop, or connected to the Aalto VPN.

2. Or, use rsync, sftp, or sshfs to transfer a file.

3. (Advanced) If you have a Linux on Mac computer, study the rsync manual page and try to transfer a file.

Accessing files from Department workstations

This varies per department, with some strategies that work from everywhere.

These mounts that are already on workstations require a valid Kerberos ticket (usually generated when you log in). On long sessions these might expire, and you have to renew them with kinit to keep going.

Generic

The staff shell server taltta.aalto.fi has scratch and work mounted at /m/triton, and department directories are also in the standard paths /m/{cs,nbe}/(scratch,work)/.

NBE

Work directories are available at /m/nbe/work and group scratch directories at /m/nbe/scratch/$project/.

PHYS

Directories available on demand through SSHFS. See the Data transferring page at PHYS Intranet (accessible by PHYS users only).

CS

Work directories are available at /m/cs/work/, and group scratch directories at /m/cs/scratch/$project/.

Exercises

strace is a command which tracks system calls, basically the number of times the operating system has to do something. It can be used as a rudimentary way to see how much I/O load there is.

1. Use strace -c to compare the number of system calls in ls, ls -l, ls --no-color, and ls --color. You can use the directory /scratch/scip/lustre_2017/many-files/ as a place with many files in it. How many system calls per file were there for each option?

2. Using strace -c, compare the times of find and lfs find on the directory mentioned above. Why is it different?

3. (Advanced, requires slurm knowledge from future tutorials) You will find some sample files in /scratch/scip/examples/io. Create a temporary directory and...

a) Run create_iodata.sh to make some data files in data/

b) Compare the IO operations of find and lfs find on this directory.
c) use the `iotest.sh` script to do some basic analysis. How long does it take? Submit it as a slurm batch job.

d) Modify the `iotest.sh` script to copy the `data/` directory to local storage, do the operations, then remove the data. Compare to previous strategy.

e) Use `tar` to compress the data while it is on lustre. Unpack this tar archive to local storage, do the operations, then remove. Compare to previous strategies.

**Next steps**

The next tutorial is about *interactive jobs*.

If you are doing anything IO heavy, you might want to read the *advanced storage page*.

Optimizing data storage isn’t very glamorous, but it’s an important part of high-performance computing. You can find some hints on the *profiling* page.

We have these related pages:

- Storage: Lustre (scratch)
- Storage: local drives
- Quotas
- Small files

**Interactive jobs**

**Introduction**

Triton is a large system that combines many different individual computers. At the same time, hundreds of people are using it. Thus, we don’t just have machines sitting around to run directly on. You need to share resources among everyone by applying for them using the queuing system, slurm. As you will see, this is very fast and lets you get basically whatever you need.

This page discusses what is necessary to use Triton interactively. This means no scripts, no overhead. You “**just add `srun`**”! For the small jobs that you would use this for, you will almost always get your time right away. Still, you have to request the resources you need (time/cores/memory). It also means that if you don’t do things properly, it is inefficient because you request more than you need. You should start here, but once you need more go to more advanced usage.

**Note:** Advantages of interactive running: It’s good for getting started quickly and scaling up: “just add `srun`!”. It’s good when task is so small that scripting isn’t worth it.

Downsides include: You have to be there and wait for things to run. If your shell connection gets interrupted, you lose the process. If you don’t stop interactive shells, they will continue it will count against your fairshare quota, making your jobs run slower in the future.

**Single process**

The simplest way is to use `srun`. Let’s say you run some program like this:
You switch to use srun. All input/output still goes to your terminal (but note X forwarding for graphical applications doesn’t work - see below for that).

```
srun --mem=50G --time=5:00:00 python3 -c 'import os; print("hi from", os.uname().nodename)'
```

This has some possible problems: it is connected to your shell. If your shell quits, the process gets lost. Also, this runs only one single process. If you need to do multiple things in a row, then you have to wait before each one starts. Note: srun is used directly with a command to run, not batch scripts like sbatch is (though of course you could run a shell script). srun does not look at the #SBATCH options inside of scripts.

How do you find the right time/CPU/memory requirements? Slurm (the queuing system) has extensive reporting. For example, slurm history will show you the actual run time and actual memory used of your job. You generally make a guess and adjust based on what you see. There is a little bit about this below and more in the next tutorial.

**Interactive shell**

So, let’s say you need to do something a bit fancier: what if you want an actual shell to do things interactively? You just need the extra `--pty` and run `bash`. The `--pty` says “give me a partition dedicated to interactive usage (more on this later). Full example:

```
srun -p interactive --time=HH:MM:SS --mem=nnG --pty bash
```

Now you have a shell... do whatever you need to do. **Close the shell when you are done! If you don’t, the process will keep running until your time limit. All of this time will be counted against your usage. It doesn’t cost money, but does mean that your priority will go down in the future.** (Note that we specify the interactive partition with “--pty interactive”. More on this below.)

**Interactive shell with graphics**

sinteractive is very similar to srun, however it is more clever and thus allows you to do X forwarding. In the background, it starts the job, starts a screen session on the node, then sshes to there and connects to the screen. You can also ssh to this node again and connect to the process again.

```
sinteractive --time=HH:MM:SS --mem=nnG
```

Just like with `srun --pty`, remember to close the process when done. **However, it’s even harder than before.** Since there is a separate screen session running, just closing the terminal isn’t enough. Exit all shells in the screen session on the node (C-d or “exit“), or cancel the process (see below).

**More options**

**Time/CPU/memory requirements:** The commands srun/sinteractive have many more options that let you specific resources. The most important for interactive running are probably `--mem`, `--cpus-per-task (-c)`, and `--time (-t)`.

**How much time/memory/CPU resources should you request?** The less you request, the faster you are likely to run. As for all you need, but not ridiculously large amounts. If you request something slightly less than a node size (note that we have different size nodes) or partition limit (see below), you are more likely to fit into a spare spot. We have many nodes with 12 cores, and some with 20 or 24. If you request 24, you have very limited options. If you request...
10, or 18, you will have a lot more options. Same with memory: most common cutoffs are 48, 64, 128, 256GB. Use smaller values when interactive testing, then more for batch running overnight.

**Configure your program well:** Also, note that requesting more CPUs doesn’t magically mean that your program becomes parallel. Make sure you turn that on in your code to enable that. Also specify how many CPUs to use (matching how many you request with slurm). If you don’t get an entire node, your program might try to use all CPUs, and the OS will limit the number you can use (with cgroups, if you are interested). This leads to inefficiency.

**Partitions:** Now almost always automatically set, but used to be important. Partitions are groups of nodes reserved for different purposes. For example, `-p interactive` tells us to use the interactive partition - which should always be available for quick tests. `-p debug` is a short partition for debugging. See a bit more in the *serial tutorial*.

### Monitoring your usage

When you start running in the queue, you need to be able to get information on how much time, memory, etc is being used. Without this, you won’t know how much time and memory to request. You always want to request the least possible, so that your stuff can run faster. The next tutorial (about batch jobs) goes into this in more detail. You probably want to be checking things like slurm history even if you aren’t running batch, to see how many resources you are actually using.

The command `slurm q` will tell you the currently running processes (a good way to make sure you have stopped everything). `slurm history` will tell you about recent jobs, including how much total memory they used and their total CPU time.

The command `scancel` will cancel a job by job-id (useful if something keeps running after you don’t need it anymore).

### Exercises

1. The program `/scratch/scip/examples/slurm/memory-hog.py` uses up a lot of memory to do nothing. Let’s play with it. It’s run like this: `python /scratch/scip/examples/slurm/memory-hog.py 50M`, where the last argument is however much memory you want to eat. (also available from triton-examples/slurm)
   a) Try running the program with `50M`
   b) Run the program with `50M` and `srun --mem=500M`.
   c) Increase the amount of memory allocated until the job fails. What happens?
   d) Play around with different parameters: how much memory can you use?
   e) Look at the job history using `slurm history` - can you see how much memory it actually used?

2. The program `/scratch/scip/examples/slurm/pi.py` (also available from triton-examples/slurm) calculates pi using a simple stochastic algorithm. You give it one argument: the number of trials.

   The `time` program allows you to time any program. E.g. you can `time python x.py` to print the amount of time it takes.

   a) Run the program, timing it with `time`, a few times, increasing the number of trials, until it takes about 10 seconds: `time python /scratch/scip/examples/slurm/pi.py 500` and so on.
   b) Add `srun` in front (`srun python ...`). What changes?
   c) Tell srun to use five CPUs (`-c 5`). Does it go any faster?
   d) Use the `--threads=5` option to the Python program to tell it to also use five threads. ... `python .../pi.py --threads=5`
3. Check out some of these commands: `sinfo, squeue`. Run `slurm job $jobid` on some running job - does anything look interesting?

4. Run `scontrol show node wsml` What is this?

What's next

Read the next tutorial on *serial batch jobs*. You can put these same commands into a script to run many things in the background, without you having to wait.

**Serial Jobs**

See also:

This assumes you have read *the interactive jobs tutorial* first.

**Introduction**

Triton is a large system that combines many different individual computers. At the same time, hundreds of people are using it. Thus, we must use a batch queuing system (slurm) in order to allocate resources.

The queue system takes computation requests from everyone, figures out the optimal use of resources, and allocates code to nodes. You have to start your code in a structured way in order for this to work. Our previous tutorial showed how to run things directly from the command line, without any scripting needed.

Now let’s see how to put these into scripts. A *shell script* takes any commands that you might type directly into a shell and automates them. The *slurm scripts* that we make in this lesson do do this. Scripts allow jobs to run asynchronously, in batch, and without human supervision.

**A basic script**

Let’s say we want to run `echo 'hello world'`. We have to tell the system how to run it. Here is a simple submission script, put it in a file called `hello.slrm` (you can use the editor `nano`: `nano hello.slrm`):

```
#!/bin/bash
#SBATCH --time=0-00:05:00  # 5 mins
#SBATCH --mem-per-cpu=500   # 500MB of memory
srun echo 'hello, world'
```

*Whatever your application or programming language requires, you put it in the script.*

Each `srun` is a job step, and appears as a separate row in your history - which is useful for monitoring. Then submit it with `sbatch`:

```
$ sbatch hello.slrm
```
Warning: You must use `sbatch`, not `bash` to submit the job to process the #SBATCH headers and run in the background.

This sends it to the queue to wait. Since the time requested is short, it will probably run on the debug partition, which is reserved for small test jobs (see below). Let’s see if it is in the queue:

Checking job status with `slurm q`:

```bash
$ slurm q
JOBID PARTITION NAME TIME START_TIME STATE NODELIST(REASON)
   13031249 debug hello.slrm 0:00 N/A PENDING (None)
```

Keep rerunning `slurm q` until you see it finish.

You can use `scancel` with that jobid to cancel the job before it finishes.

The output is then saved to `slurm-13031249.out` in your current directory (the number being the job ID).

Loading modules in scripts

Need to load modules for your software? Do it in the batch scripts. In general, anything you can do from the shell, you can do here:

```bash
#!/bin/bash
#SBATCH --time=0-00:05:00 # 5 mins
#SBATCH --mem-per-cpu=500 # 500MB of memory
module load anaconda3
python -V
```

Exercise: Try the Python version-printing script above. Try changing to different modules, `anaconda2`, `Python`, and others if you can find them.

Job parameters

As you can see, the above script is limited to 5 minutes and 500MB of memory. All scripts have to have limits, otherwise they can’t be efficiently scheduled. If you exceed the limits, the jobs will be killed. At least you need to set `--time`, `--mem-per-cpu` or `--mem`.

See the previous tutorial, the reference page or the details page for more information and advanced usage.

The same parameters can be used in:

- The `sbatch` script, prefixed by `#SBATCH`
- The `sbatch` command line program directly (like `--p debug` above)
- `sinteractive/srun` from the command line, which lets you run programs without a batch script.

It is important to note that slurm is a declarative system. You declare what you need, and slurm handles finding the resources without you having to worry about details. The more resources you request, the harder it will be to schedule and the longer you may have to wait. So, you should ask for enough to make sure your job can complete, but once you get experience with your code reduce resources to just what is needed.
In general, you don’t want to go submitting too short jobs (under 5 minutes) because there is a lot of startup, accounting, and scheduling overhead. If you are testing, short things are fine, but once you get to bulk production try to have each job take at least 30 minutes if possible. If you have lots of things to run, combine them into fewer jobs.

Full slurm reference

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sbatch</td>
<td>submit a job to queue (see standard options below)</td>
</tr>
<tr>
<td>srun</td>
<td>Within a running job script/environment: Run code using the allocated resources (see options below)</td>
</tr>
<tr>
<td>srun</td>
<td>On frontend: submit to queue, wait until done, show output. (see options below)</td>
</tr>
<tr>
<td>sinteractive</td>
<td>Submit job, wait, provide shell on node for interactive playing (X forwarding works, default partition interactive). Exit shell when done. (see options below)</td>
</tr>
<tr>
<td>srun--pty</td>
<td>(advanced) Another way to run interactive jobs, no X forwarding but simpler. Exit shell when done.</td>
</tr>
<tr>
<td>bash</td>
<td></td>
</tr>
<tr>
<td>scancel</td>
<td>Cancel a job in queue</td>
</tr>
<tr>
<td>salloc</td>
<td>(advanced) Allocate resources from frontend node. Use srun to run using those resources, exit to close shell when done. Read the description! (see options below)</td>
</tr>
<tr>
<td>scontrol</td>
<td>View/modify job and slurm configuration</td>
</tr>
</tbody>
</table>
### Command Option Description

<table>
<thead>
<tr>
<th>Command</th>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sbatch</td>
<td>/etc --time=hh:mm:ss</td>
<td>time limit</td>
</tr>
<tr>
<td></td>
<td>-t, --time=dd-hh</td>
<td>time limit, days-hours</td>
</tr>
<tr>
<td></td>
<td>-p, --partition=partition</td>
<td>job partition. Usually leave off and things are auto-detected.</td>
</tr>
<tr>
<td></td>
<td>--mem-per-cpu=n</td>
<td>request n MB of memory per core</td>
</tr>
<tr>
<td></td>
<td>--mem=n</td>
<td>request n MB memory per node</td>
</tr>
<tr>
<td></td>
<td>-c, --cpus-per-task=n</td>
<td>Allocate <em>n</em> CPU’s for each task. For multithreaded jobs. (compare –ntasks: -c means the number of cores for each process started.)</td>
</tr>
<tr>
<td></td>
<td>-N, --nodes=n-m</td>
<td>allocate minimum of n, maximum of m nodes.</td>
</tr>
<tr>
<td></td>
<td>-n, --ntasks=n</td>
<td>allocate resources for and start n tasks (one task=one process started, it is up to you to make them communicate. However the main script runs only on first node, the subprocesses run with “srun” are run this many times.)</td>
</tr>
<tr>
<td></td>
<td>-J, --job-name=name</td>
<td>short job name</td>
</tr>
<tr>
<td></td>
<td>-o output</td>
<td>print output into file output</td>
</tr>
<tr>
<td></td>
<td>-e error</td>
<td>print errors into file error</td>
</tr>
<tr>
<td></td>
<td>--exclusive</td>
<td>allocate exclusive access to nodes. For large parallel jobs.</td>
</tr>
<tr>
<td></td>
<td>--constraint=feature</td>
<td>request feature (see slurm features for the current list of configured features, or Arch under the hardware list). Multiple with --constraint=&quot;hsw</td>
</tr>
<tr>
<td></td>
<td>--array=0-5,7,10-15</td>
<td>Run job multiple times, use variable $SLURM_ARRAY_TASK_ID to adjust parameters.</td>
</tr>
<tr>
<td></td>
<td>--gres=gpu</td>
<td>request a GPU, or --gres=gpu:n for multiple</td>
</tr>
<tr>
<td></td>
<td>--gres=spindle</td>
<td>request nodes that have disks, spindle: n, for a certain number of RAID0 disks</td>
</tr>
<tr>
<td></td>
<td>--mail-type=type</td>
<td>notify of events: BEGIN, END, FAIL, ALL, REQUEUE (not on triton) or ALL. MUST BE used with --mail-user=your@email</td>
</tr>
<tr>
<td>srun</td>
<td>-N &lt;N_nodes&gt;</td>
<td>Print allocated nodes (from within script)</td>
</tr>
<tr>
<td></td>
<td>hostname</td>
<td></td>
</tr>
</tbody>
</table>

### Status of the jobs

Once you submit jobs, it goes into a queue. You need to be able to see the status of jobs. There are commands to do this.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>slurm j &lt;jobid&gt;</td>
<td>Status on single job (still running)</td>
</tr>
<tr>
<td>slurm history [2hours</td>
<td>5days...]</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>slurm q:slurm qq</td>
<td>Status of your queued jobs (long/short)</td>
</tr>
<tr>
<td>slurm partitions</td>
<td>Overview of partitions (A/I/O/T=active,idle,other,total)</td>
</tr>
<tr>
<td>slurm cpus &lt;partition&gt;</td>
<td>list free CPUs in a partition</td>
</tr>
<tr>
<td>slurm history [1day,2hour,...]</td>
<td>Show status of recent jobs</td>
</tr>
<tr>
<td>seff &lt;jobid&gt;</td>
<td>Show percent of mem/CPU used in job</td>
</tr>
<tr>
<td>slurm s:slurm ss &lt;partition&gt;</td>
<td>Show status of all jobs</td>
</tr>
<tr>
<td>sacct</td>
<td>Full history information (advanced, needs args)</td>
</tr>
</tbody>
</table>

### Full slurm command help:

```bash
$ slurm
```

**Show or watch job queue:**

- `slurm [watch] queue` show own jobs
- `slurm [watch] q` show user's jobs
- `slurm [watch] quick` show quick overview of own jobs
- `slurm [watch] shorter` sort and compact entire queue by job size
- `slurm [watch] short` sort and compact entire queue by priority
- `slurm [watch] full` show everything
- `slurm [w] [q|qq|ss|sf]` shorthands for above!
- `slurm qos` show job service classes
- `slurm top [queue|all]` show summary of active users

**Show detailed information about jobs:**

- `slurm prio [all|short]` show priority components
- `slurm j|job` show everything else
- `slurm steps` show memory usage of running srun job steps

**Show usage and fair-share values from accounting database:**

- `slurm h|history` show jobs finished since, e.g. "1day" (default)
- `slurm shares`

**Show nodes and resources in the cluster:**

- `slurm p|partitions` all partitions
- `slurm n|nodes` all cluster nodes
- `slurm c|cpus` total cpu cores in use
- `slurm cpus` cores available to partition, allocated and free
- `slurm cpus jobs` cores/memory reserved by running jobs
- `slurm cpus queue` cores/memory required by pending jobs
- `slurm features` List features and GRES

**Examples:**

- `slurm q`
- `slurm watch shorter`
- `slurm cpus batch`
- `slurm history 3hours`

### Other advanced commands (many require lots of parameters to be useful):

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>squeue</td>
<td>Full info on queues</td>
</tr>
<tr>
<td>sinfo</td>
<td>Advanced info on partitions</td>
</tr>
<tr>
<td>slurm nodes</td>
<td>List all nodes</td>
</tr>
</tbody>
</table>

See the full list of status commands on the reference page.
Partitions

There are different partitions, which have different limits. The “debug” partition is for short debugging, so is designed to always be available. The “batch” partition is designed for all the normal long jobs. There are also partitions for GPUs, huge memory nodes, interactive shells, and so on. Most of the time, you should leave the partition off, and slurm will use all possible partitions. You can specify your partitions with \(-p\) PARTITION_NAME to whatever command you are running, which is mainly needed if you want to force interactive or a test partition. The available partitions are listed on the reference page.

You can see the partitions in the quick reference.

Exercises

1. Basics
   a. Submit a batch job that just runs hostname.
   b. Set time to 1 hour and 15 minutes, memory to 500MB.
   c. Change the job’s name and output file.
   d. Monitor the job with slurm watch queue.
   e. Check the output. Does it match slurm history?

2. Create a simple batch script using \(\text{pi.py}\) based on the \(\text{pi}\) calculation of the interactive job tutorial exercises. Create multiple job steps (separate \(\text{srun}\) lines), each of which runs \(\text{pi.py}\) with a greater and greater number of tries. How does this appear in slurm history. When would you use extra \(\text{srun}\) commands, and when not?

3. Create a batch script which does nothing (or some pointless operation for a while), for example \(\text{sleep 300}\) (waits for 300 seconds) in the debug partition. Check the queue to see when it starts running. Then, cancel the job. What output is produced?

4. What happens if you submit a batch script with \(\text{bash}\) instead of \(\text{sbatch}\)? Does it appear to run? Does it use all the Slurm options?

4. (Advanced) Create a batch script that runs in another language. Does it run? What are some of the advantages and problems here?

Next steps

There is a full description of running jobs on Triton, and the reference page lists many useful commands.

Running multiple instances of a \(\text{sbatch}\) script is easier with array jobs.

Array jobs

Introduction

By now, you should be able to do all the basic things to run programs on Triton. Now, you want to do it... a lot. The easiest way to parallelize things is to use array jobs. With array jobs, you take a single code or script, and Slurm (the batch system) runs it many times for you, with all the parameters. This is the simplest way of parallelizing things, but only works for embarrassingly parallel problems: where your code runs independently multiple times, and you use or combine the results later. Still, for most people, this is as far as you need to go.
Basic examples

When you run an array job (with --array=1-5), Slurm runs your job script many (5) times, with one difference: the environment variable $SLURM_ARRAY_TASK_ID (1,2,3,4,5). You have your job script or code read this variable and take the right action, depending on what you need to do.

Below are four examples. The first and third are probably most recommended for starting out.

Different inputs

In the example below, the $SLURM_ARRAY_TASK_ID is used to change to the right directory, make the application read the correct input file, and to generate output in a unique directory. This script is submitted with sbatch script.sh:

```bash
#!/bin/bash
#SBATCH -n 1
#SBATCH -t 04:00:00
#SBATCH --mem-per-cpu=2500
#SBATCH --array=0-29

cd $SLURM_ARRAY_TASK_ID
srun echo I am number $SLURM_ARRAY_TASK_ID
# e.g. srun ./my_application -input input_data_$SLURM_ARRAY_TASK_ID

cd ..
```

Different parameters in script

In the example below, we have the same program, but different command line parameters. In this case, everything is hard coded in the bash script itself. You could also do this directly inside your program, and generate the parameters according to some algorithm. This can be really powerful: not only can you both hard code and generate with algorithm, but if you never edit already-run parameters, you have a single place where you can see everything that has been run before. (Personally, I always try to set up a system where parameters are defined in one place, code in another, and I can always know what has been run for each output by looking in just one place. If you are going to be configuring stuff by hand anyway, better to have it all together.):

```bash
#!/bin/bash
#SBATCH -n 1
#SBATCH -t 04:00:00
#SBATCH --mem-per-cpu=2500
#SBATCH --array=0-4

case $SLURM_ARRAY_TASK_ID in
  0) ARGS="-res 5 -arg 3" ;;
  1) ARGS="-res 6 -arg 3" ;;
  2) ARGS="-res 5 -arg 4" ;;
  3) ARGS="-res 6 -arg 4" ;;
  4) ARGS="-res 5 -arg 5" ;;
esac

 cd $SLURM_ARRAY_TASK_ID
srun I am doing $ARGS !
# e.g. python input.py $ARGS

cd ..
```
Read parameters from file

Now we do basically the same thing as above, but we have all of the parameters stored in another file `arrayparams.txt`:

```
input_561 --opt1
input_418 --opt2
input_569 --opt1
```

In our script, we use the `sed` program to read just one line from the file. This is stored in the variable `line`, and then we can use this however; in this case by using it as the parameters to the program. Don’t worry about how the `sed` command works - no one really knows, we just find it via a web search. Note that the line numbers start at one, not zero!

```
#!/bin/bash
#SBATCH -n 1
#SBATCH -t 04:00:00
#SBATCH --mem-per-cpu=2500
#SBATCH --array=1-3

n=${SLURM_ARRAY_TASK_ID}
line=`sed "${n}q;d" arrayparams.txt`

# Do whatever with arrayparams.txt
srun echo I am doing $line
```

(advanced) Grouping runs together in bigger chunks

Let’s say your tasks are very short - only a few minutes. This is still a bit short to use array jobs, because you will still have too much overhead in scheduling. We want to try for 30 minutes and if possible more. So, below is an example script that uses shell

In the below script, we take a chunk size of 100. Array #0 will run 0-99, array #1 will run 100-199, etc. The for loop handles the running. Before each one runs, it uses `test -s output_$i` to see if the output filename `output_$i` exists: only run the task if it does not exist already (see `man test` to see other types of tests you can do). This example starts using more advanced shell scripting, which might be worth learning.

```
[... all the initial stuff from above]

CHUNKSIZE=100
arrayID=${SLURM_ARRAY_TASK_ID}
indexes=`seq $((arrayID * CHUNKSIZE)) $((arrayID+1)*CHUNKSIZE - 1))`

for i in $indexes ; do
  if ! test -s output_$i ; then
    run $i
  fi
done
```

2D sampling

Here is an example that lets you sample from a 2D array, with experiments and 10 replicas (but this might be approaching hackish, ask first if it makes sense to have them together):

```
```

134 Chapter 6. Triton
experiment=$(( $SLURM_ARRAY_TASK_ID / 10 ))
replica=$(( $SLURM_ARRAY_TASK_ID % 10 ))

More control

You can specify the --array option either in the script itself using the #SBATCH syntax, or on the command line to sbatch. So, you can control what runs different ways. Let’s say you have a fixed number of parameters: put that directly in the script. Or if you are just running replicas, run them from the command line as you need more. In any case, use the command line when things fail and you need to repeat only certain runs.

You don’t have to have the job script use the variable. You could directly pass it as a command line argument to your program, use it to pattern input files, or even have your own code access the process environment and get the variable.

You can use %N, like --array=1-100%10, to limit number of jobs running at once.

Note that arrays are only a feature of sbatch. You can’t use them directly from the command line with srun: you have to make a batch script and submit with sbatch.

Hints

The array indices need not be sequential. E.g. if you discover that after the above array job is finished, the job task id’s 7 and 19 failed, you can relaunch just those jobs with --array=7,19. While the array job above is a set of serial jobs, parallel array jobs are possible. For more information, see the Slurm job array documentation.

How do you map from $SLURM_ARRAY_TASK_ID to the parameters of the job? There are different strategies

- Have a lookup table in your code or another config file (bash example in slurm script above)
- Pre-create different input files
- Programatically generate the different configs in your code.
- Don’t have different config, just use them to run multiple replicas of the same parameters. You increase the array ID until you have enough statistics to get your result.

You probably want to look at the slurm -o option to direct the script output to somewhere useful. See the sbatch manual page, -o, -e, and --open-mode options. In the filenames, use %a for array index and %A (array jobs) for array jobid. For normal jobs, use %j for the jobid. (If you use %j for array jobs, you get a different number even when things were started as part of the same array. Maybe it’s what you want).

Array jobs have less overhead for accounting and scheduling, but you still want them to not be too short. 30 minutes is a good target time, so try to combine smaller tasks to fit that.

Exercises

1. Look at man sbatch and investigate the --array parameter.
2. Using the pi.py example from the interactive tutorial, make an array job that calculates pi 10 times.
   a) The pi.py program takes an extra option: --seed=SEED. Use the array task ID as the seed.
   b) Verify that the runs worked. Average all values together to get your more accurate pi.
3. Using one of the techniques above, use memory-hog.py from the interactive tutorial. Make an array job that runs this with five different values of the memory (5M, 50M, 100M, 200M, 500M). You have to use one of the techniques above.
4. Make job array which runs every other index (like 1, 3, 5, etc). You’ll have to look at the sbatch manual page.
What's next?

The next tutorial is about GPU computing.

For more information, you can see the CSC guide on array jobs: https://research.csc.fi/taito-array-jobs.

For more detailed information about running on Triton, see the main page Running programs on Triton.

Remember to check the quick reference when needed.

GPU computing

See also:

This tutorial assumes you have read Interactive jobs.

Main article: GPU Computing

GPUs and accelerators are basically very special parallel processors: they can apply the same instructions to a big chunk of data at the same time. The speedup can be 100x or more... but only in the specific cases where your code fits the model. It happens that machine learning/deep learning methods are able to use this type of parallelism, so now these are the standard for this type of research.

On Triton, we have a large number of NVIDIA GPU cards from different generations, and are constantly getting more. Our GPUs are not your typical desktop GPUs, but specialized research-grade server GPUs with large memory, high bandwidth and specialized instructions. For scientific purposes they generally exceed the best desktop GPUs.

Some nomenclature: a GPU is a graphical processing unit, CUDA is the software interface for Nvidia GPUs. Currently we only support CUDA.

Getting started

GPUs are, just like anything, resources which are scheduled by slurm. So in addition to time, memory, and CPUs, you have to specify how many GPUs you want. This is done with the --gres (generic resources) option:

```
srun --gres=gpu:1 $my_code
```

This means you request the gpu resources, and one of them (1). Combining this with the other required slurm options:

```
srun --gres=gpu:1 -t 2:00:00 --mem=10G -c 3
```

... and you’ve got yourself the basics. Of course, once you are ready for serious runs, you should put your code into slurm scripts.

If you want to restrict yourself to a certain type of card, you should use the --constraint option. For example, to restrict to Kepler generation (K80s), use --constraint=kepler or all new cards, --constraint='pascal|volta' (note the quotes - this is very important, because | is a shell pipe symbol!).

Our available GPUs and architectures:
## Ready software

We support these machine learning packages out of the box:

- **Tensorflow**: `anaconda2/anaconda3` modules. Use `--constraint='kepler|pascal|volta'`
- Keras: same module as tensorflow
- PyTorch: same module as tensorflow
- **Detectron**: via `singularity images`
- **CNTK**: via `singularity images`

Do note that most of the pre-installed software has CUDA already present. Thus you **do not need to load CUDA** as a module when loading these. See the [application list](#) or [GPU computing reference](#) for more details.

## Compiling code yourself

To compile things for GPUs, you need to load the relevant CUDA modules:

```bash
module avail cuda
module load gcc
cuda
nvcc cuda_code.cu -o cuda_code # compile your CUDA code
```

More information is in the [reference](#), but most people will use pre-built software through channels such as Anaconda for Python.

## Making efficient use of GPUs

When running a job, you want to check that the GPU is being fully utilized. To do this, ssh to your node (while the job is running), and run `nvidia-smi`, find your process (which might take some work) and check the *GPU-Util* column. It should be close to 100%, otherwise see below.

After job has finished, you can use `slurm history` to obtain the `JobID` and run:

```
sacct -j INSERT_JOBID_HERE -o comment -p
```

This will show the GPU utilization.
Input/output

Deep learning work is intrinsically very data-hungry. Remember what we said about storage and input/output being important before (in the storage tutorial)? Now it’s really important. In fact, faster memory bandwidth is the main improvement of our server-grade GPUs compared to desktop models.

If you are loading lots of data, package the data into a container format first: lots of small files are your worst enemy, and we have a dedicated page on small files.

If your dataset consists of individual files and it is not too big, it is a good idea to have the data stored in one file, which is then copied to nodes ramdisk /dev/shm or temporary disk /tmp.

If your data is too big to fit to the disk, we recommend that you contact us for efficient data handling models.

Enough CPUs

When using a GPU, you need to also request enough CPUs to supply the data to the process. So, increase the number of CPUs you request so that you can provide the GPU with enough data. However, don’t request too many: then, there aren’t enough CPUs for everyone to use the GPUs, and they go to waste! (For the K80 nodes, we have only 1.5 CPUs per GPU, but on all others we have 4-6 CPUs/GPU)

Other

Most of the time, using more than one GPU isn’t worth it, unless you specially optimize, because communication takes too much time. It’s better to parallelize by splitting tasks into different jobs.

FAQ

If you ever get libcuda.so.1: cannot open shared object file: No such file or directory, this means you are attempting to use a CUDA program on a node without a GPU. This especially happens if you try to test GPU code on the login node, and happens (for example) even if you try to import the GPU tensorflow module in Python on the login node.

Examples

Simple Tensorflow/Keras model

Let’s run the MNIST example from Tensorflow’s tutorials:

```python
model = tf.keras.models.Sequential({
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(512, activation=tf.nn.relu),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(10, activation=tf.nn.softmax)
})
```

The full code for the example is in tensorflow_mnist.py. One can run this example with srun:

```bash
wget https://raw.githubusercontent.com/AaltoScienceIT/scicomp-docs/master/triton/examples/tensorflow/tensorflow_mnist.py
module load anaconda3/latest
srun -t 00:15:00 --gres=gpu:1 python tensorflow_mnist.py
```
or with `sbatch` by submitting `tensorflow_mnist.sh`:

```bash
#!/bin/bash
#SBATCH --gres=cpu:1
#SBATCH --time=00:15:00
module load anaconda3/latest
python tensorflow_mnist.py
```

Do note that by default Keras downloads datasets to `$HOME/.keras/datasets`.

### Simple PyTorch model

Let’s run the MNIST example from PyTorch’s tutorials:

```python
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 20, 5, 1)
        self.conv2 = nn.Conv2d(20, 50, 5, 1)
        self.fc1 = nn.Linear(4*4*50, 500)
        self.fc2 = nn.Linear(500, 10)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = F.max_pool2d(x, 2, 2)
        x = F.relu(self.conv2(x))
        x = F.max_pool2d(x, 2, 2)
        x = x.view(-1, 4*4*50)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return F.log_softmax(x, dim=1)
```

The full code for the example is in `tensorflow_mnist.py`. One can run this example with `srun`:

```bash
wget https://raw.githubusercontent.com/AaltoScienceIT/scicomp-docs/master/triton/examples/pytorch/pytorch_mnist.py
module load anaconda3/latest
srun -t 00:15:00 --gres=gpu:1 python pytorch_mnist.py
```

or with `sbatch` by submitting `pytorch_mnist.sh`:

```bash
#!/bin/bash
#SBATCH --gres=gpu:1
#SBATCH --time=00:15:00
module load anaconda3/latest
python pytorch_mnist.py
```

The Python-script will download the MNIST dataset to `data` folder.

### Simple CNTK model

Let’s run the MNIST example from CNTK’s tutorials:
The full code for the example is in `cntk_mnist.py`. One can run this example with `srun`:

```bash
wget https://raw.githubusercontent.com/AaltoScienceIT/scicomp-docs/master/triton/examples/cntk/cntk_mnist.py
module load anaconda3/latest
srun -t 00:15:00 --gres=gpu:1 python cntk_mnist.py
```

or with `sbatch` by submitting `cntk_mnist.sh`:

```bash
#!/bin/bash
#SBATCH --gres=gpu:1
#SBATCH --time=00:15:00
module load nvidia-cntk
singularity_wrapper exec python cntk_mnist.py
```

Do note that datasets in the code come from `/scratch/scip/data/cntk/MNIST`. Thus model won’t run outside of Triton. Check the CNTK GitHub repo for the whole example.

### Exercises

1. Run `nvidia-smi` on a GPU node with `srun`. Use `slurm history` to check which GPU node you ended up on. Try setting a constraint to force a different GPU architecture.

2. Copy `/scratch/scip/examples/gpu/pi.cu` to your work directory. Compile it using `cuda` module and `nvcc`. Run it. Does it say zero? Try running it with a GPU and see what happens.

3. Run one of the samples given above. Try using `sbatch` as well.

4. Modify CTNK sample slurm script in a way that it copies datasets to an unique folder in `/dev/shm` or `$TMPDIR` before running the Python code. Modify CNTK sample so that it loads data from the new location.

   HINT: Check out `mktemp --help, command output substitutions`-section from our Linux shell tutorial and the API page for Python’s `os.environ`.

   Solution to ex. 4: `cntk_mnist_ex4.py` `cntk_mnist_ex4.sh`.

### Next steps

Check out or reference information about GPU computing, including examples of different machine learning languages.

If you came straight to this page, you should also read Interactive jobs and Serial Jobs (actually you should have read them first, but don’t worry).
This guide assumes you are using pre-existing GPU programs. If you need to write your own, that’s a whole other story, and you can find some hints on the reference page.

**Parallel computing**

Parallel computing is what HPC is really all about: processing things on more than one CPU at once. By now, you should have read all of the previous tutorials.

**Parallel programming models**

Parallel programming is a completely different way of programming. Most Triton users don’t need to write their own applications, at most they will be running existing programs, but in order to understand things, we start with some introduction.

The two main models are:

- **Shared memory program** (or multithreading) runs on only one node because, like the name says, all the memory has to be accessible to all the processes. Thus, scalability is limited to a number of CPU cores available within one computational node. The code is easier to implement and the same code can still be run in a serial mode. Examples of application that utilize this model: Matlab, R, OpenMP applications, typical parallel desktop programs.

- **Message passing programming** (e.g. MPI, message passing interface) can run on multiple nodes interconnected with the network via passing data through MPI software libraries. The large-scale scientific programs are MPI. MPI can scale to thousands of CPU cores, but it’s harder to implement from the programmer point of view.

Both models, MPI and shared memory, can be combined in one application, in this case we are talking about hybrid parallel programming model.

Most historical scientific code is MPI, but these days more and more people are using shared memory models.

The important note is that a normal, serial code can’t just be run as parallel without modifications. As a user it is your responsibility to understand what parallel model implementation your code has, if any. Knowing this, you can proceed with the instructions below.

Another important note regarding parallelism is that all the applications scale good up to some upper limit which depends on application implementation, size and type of problem you solve and some other factors. The best practice is to benchmark your code on different number of CPU cores before actual production run.

**If you want to run some program in parallel, you have to know something about it - is it shared memory or MPI? A program doesn’t magically get faster when you ask more processors if it’s not designed to.**

**Shared memory: OpenMP programs**

OpenMP is a standard de facto for the multithreading implementations. There are many others, but this one is the most common, supported by all known compiler suits. For other implementations of shared memory parallelism, please consult your code docs.

Simple code compiling:

```bash
gcc -fopenmp -O2 -g omp_program.c -o omp_program
```

Running an OpenMP code:

```bash
export OMP_PROC_BIND=TRUE
srun --cpus-per-task=12 --mem-per-cpu=2000 --time=45:00 omp_program
```
The basic slurm options you need are `--cpus-per-task=N` (or `-c N`) to specify the number of cores to use within one node. If your memory needs scale with the number of cores, use `--mem-per-core=`, if you require a fixed amount of memory (per node regardless of number of processors), use `--mem`.

The SLURM batch file will look similar:

```bash
#!/bin/bash
#SBATCH --cpus-per-task=12
#SBATCH --mem-per-cpu=2000
#SBATCH --time=45:00
export OMP_PROC_BIND=TRUE
srun omp_program
```

Good to know that OpenMP is both an environment and set of libraries, but those libraries always come as part of the compiler, thus no need to load extra modules if you compile with the default gcc.

**Other programs and multithreading**

Some programs use multiple threads for their parallel computations. A good example of this kind of program is MATLAB, that uses parallel pool of workers; or R, which uses the parallel-package for its parallel applys. Threaded applications behave similarly to OpenMP applications in that one needs to specify the number of cores per task and amount of memory per core.

**Message passing programs: MPI**

For compiling/running an MPI job one has to pick up one of the MPI library suites. Big vendors provide their own (Cray, Intel) while there are other popular MPI flavors available. To compile and run code you need to pick one. Since most of the MPI codes will also use math libs, makes sense to pick a toolchain that provides all at once.

**Loading module:**

```bash
module load openmpi  # GCC + OpenMPI
```

**Compiling a code:**

```bash
mpif90 -O2 -g mpi_prog.f -o mpi_prog
```

**Running an MPI code in the batch mode:**

```bash
#!/bin/bash
#SBATCH -n 16          # 16 processes
#SBATCH --constraint=avx # run on nodes with AVX instructions
#SBATCH --time=4:00:00 # takes 4 hours all together
#SBATCH --mem-per-cpu=4000 # 4GB per process
module load openmpi  # NOTE: should same as you used to compile the code
srun ./mpi_prog
```

Triton has multiple architectures around (12, 20, 24, 40 CPU cores per node), even though SLURM optimizes resources usage and allocate CPUs within one node, which gives better performance for the app, it still makes sense to put constraints explicitly.
Monitoring performance

You can use the `seff` program (with a jobid) to list what percent of available processors and memory you used. If your processor usage is far below, your code may not be working correctly in a parallel environment.

Exercises

In `triton-examples` (at `/scratch/scip/examples`), you find some examples.

1. Find the files `openmp/hello_omp.c` and `openmp/hello_omp.slrm` that have a short example of OpenMP. Compile and run it - a slurm script is included.

2. Find the files `mpi/hello_mpi.c` and `mpi/hello_mpi.slrm` that have a short example of MPI. Compile and run it - a slurm script is included.

Next steps

See the next pages:

- You can check the *Running programs on Triton* page for the reference information on running jobs. This contains the general reference information.

- *MPI on Triton*

Job dependencies

Introduction

Job dependencies are a way to specify dependencies between jobs. The most common use is to launch a job only after a previous job has completed successfully. But other kinds of dependencies are also possible.

Basic example

Dependencies are specified with the `--dependency=<dependency list>` option. E.g. `--dependency=afterok:123:124` means that the job can only start after job ID’s 123 and 124 have both completed successfully.

Automating job dependencies

A common problem with job dependencies is that you want job B to start only after job A finishes successfully. However, you cannot know the job ID of job A before it has been submitted. One solution is to catch the job id of job A when submitting it and store it as a shell variable, and using the stored value when submitting job B. Like:

```
# idA=$(sbatch jobA.sh | awk '{print $4}"
sbatch --dependency=afterok:${idA} jobB.sh
```
Exercises

1. Look at `man sbatch` and investigate the `--dependency` parameter.

2. Create a chain of jobs A -> B -> C each depending on the successful completion of the previous job. In each job run e.g. `sleep 60` to give you time to investigate the status of the queue.

3. Continuing from the previous exercise, what happens if at the end of the job A script you put `exit 1`. What does it mean?

6.1.4 Detailed instructions

Available compilers

Please see full up to date listing of different toolchains and compilers from Applications-page. This page contains information on their usage.

GCC

Triton has the GCC set of compilers installed by default, but we recommend that you use the provided module versions. The GNU Compiler Collection (aka GCC) includes front ends for C, C++, Objective-C, Fortran, Java, Ada, and Go, as well as libraries for these languages (libstdc++, libgcj, . . .). In case of missing features, contact admins.

```
$ gcc -v
Using built-in specs.
COLLECT_GCC=gcc
COLLECT_LTO_WRAPPER=/share/apps/easybuild/software/GCCcore/5.4.0/libexec/gcc/x86_64-unknown-linux-gnu/5.4.0/lto-wrapper
Target: x86_64-unknown-linux-gnu
Configured with: ../configure --enable-languages=c,c++,fortran --enable-lto --enable-checking=release --disable-multilib --enable-shared=yes --enable-static=yes --enable-threads=posix --enable-gold=default --enable-plugins --enable-ld --with-plugin-ld=ld.gold --enable-bootstrap --prefix=/share/apps/easybuild/software/GCCcore/5.4.0 --with-local-prefix=/share/apps/easybuild/software/GCCcore/5.4.0
Thread model: posix
gcc-versio 5.4.0 (GCC)
```

Example usage:

```
$ gcc -lm -o my_code.c my_code # compiling your C code and linking with Math lib
$ gfortran -O2 -o my_code my_code.f90 # compiling your Fortran code with -O2 optimization level (see also g77 for Fortran 77)
$ g++ -O3 -funroll-loops -ffast-math -ftree-vectorize -mtune=native -o my_code my_code.cpp # compiling your C++ code with aggressive optimizaton and architecture tuning
```

Thus GCC is the default compiler and is used to build most of the software in the cluster.

See `man gcc, man gfortran` and other mans for options. Online GCC Documentation.
GCC compiling examples

```bash
$ cat hello.c
#include
int main(void) {
    printf("Hello World!");
    return 0;
}
$ gcc hello.c -o hello
```

Compiling your own code

- Use gcc, g++, and gfortran compilers for compilation
- Use mpicc, mpic++, and mpi90 for MPI (= MVAPICH, MPICH2 or OpenMPI + GCC)
- Setup your environment with `module load <toolchain>` e.g. `module load goolf/triton-2016b` for BLAS/LAPACK, FFTW3, ScALAPACK+BLACS, etc. Modules will set CPATH and LD_LIBRARY_PATH variables for -I and -L, but you can use 'module show <module>' to see the exact library paths.

MPI-code “mpihello.c”:

```c
#include
#include
int main (int argc, char **argv) {
    int rank, size;
    MPI_Init (&argc, &argv);
    printf("Hello world\n");
    MPI_Finalize();
    return 0;
}
```

Compile MPI code:

```
module load goolf
mpicc mpihello.c -o mpihello
```

Optimization options for GCC

By default GCC/G++/GFortran do NOT perform any optimization; you must add appropriate optimization flags yourself. Experiment and see what works for your program!

- Basic optimization level: `-O2`
- More aggressive optimization, arch specific: `-O3 -march=native`
- Might, or might not help: `-O3 -march=native -funroll-loops -ftree-loop-linear -fprefetch-loop-arrays`
- Might help a lot, but potentially dangerous: `-ffast-math -mrecip

OpenMP with GCC: `-fopenmp`

Note that using `-march=native` will produce an arch specific code. Thus when compiled on Haswell, that code must be run on Haswell, otherwise expect segmentation fault errors on other architectures.
Intel Compilers

Intel Composer available through module provides full set of compilers C/C++/Fortran. In addition, it comes with MKL libraries, that works well in case you need LAPACK/BLAS functionality, as well as paralle version of them with ScalAPACK and BLACS.

Example of linking for Intel Composer + MKL + OpenMPI

```bash
$ module load ioolf
$ mpif90 -o my.exe one.o two.o three.o libmpi_f90.a -lmkl_scalapack_lp64 -Wl,--start-group -lmkl_intel_lp64 -lmkl_sequential -lmkl_core -lmkl_blacs_openmpi_lp64 -Wl,--end-group -lpthread -lm
```

Debugging

Note: Also see Profiling.

Debugging is one of the most fundamental things you can do while using software: debuggers allow you to see inside of running programs, and this is a requirement of developing with any software. Any reasonable programming language will have a debugger made as one of the first tasks when it is being created.

Serial code debugging

GDB is the usual GNU debugger.

Note: the latest version of gcc/gfortran available through module require -gdwarf-2 option along with the -g to get it to work with the default gdb command. Otherwise the default version 4.4 should work normally with just -g.

Valgrind is another tool that helps you to debug and profile your serial code on Triton.

MPI debugging & profiling

GDB with the MPI code

Compile your MPI app with -g, run GDB for every single MPI rank with:

```
salloc -p play --nodes 1 --ntasks 4 srun xterm -e gdb mpi_app
```

You should get 4 xterm windows to follow, from now on you have full control of you MPI app with the serial debugger.

PADB

A Parallel Debugging Tool. Works on top of SLURM, support OpenMPI or MPICH only (as of June 2015), that is MVAPICH2 is not supported. Do not require code re-compilation, just run your MPI code normally, and then launch padb separately to analyze the code behavior.

Usage summary (for full list and explanations please consult http://padb.pittman.org.uk/):
# assume you have your openmpi module loaded already
module load padb
padb --create-secret-file  # for the very first time only

# Show all your current active jobs in the SLURM queue
padb -show-jobs

# Target a specific jobid, and reports its process state
padb --proc-summary
# or, for all running jobs
padb --all --proc-summary

# Target a specific jobid, and report its MPI message queue, stack traceback, etc.
padb --full-report=

# Target a specific jobid, and report its stack trace for a given MPI process (rank)
padb --stack-trace --tree --rank

# Target a specific jobid, and report its stack trace including information about parameters and local variables for a given MPI process (rank)
padb --stack-trace --tree --rank -Ostack-shows-locals=1 -Ostack-shows-params=1

# Target a specific jobid, and reports its MPI message queues
padb --mpi-queue

# Target a specific jobid, and report its MPI process progress (queries in loop over and over again)
padb --mpi-watch --watch -Owatch-clears-screen=no

**Nvidia DGX machines**

Triton currently has two Nvidia DGX-1 machines machines which contain 8 V100 GPUs and are optimized for deep learning.

**Warning:** The DGX usage in Slurm, and this page in general, are under development and testing. For latest changes, you can check git history using the link in the top right corner.

**Access and prerequisites**

The DGX machines have been specifically bought by several groups and these groups have priority access.

**General access:** you should us the `dgx-common` partition. This has preemption enabled, which means that if a higher priority job comes, you job can be cancelled at any time, even if it is running. The job will then be added back to the queue and possibly run again. Design your code to take this into account. Furthermore, your job will only start running when the priority partition is empty... so in effect jobs happen very slowly. If you are using general access, all of the `-p dgx` in the examples below need to be changed to `-p dgx-common`.

**Dedicated group access:** You can check the groups which may access it by running `grep PartitionName=dgx /etc/slurm/slurm.conf` and checking `AllowedGroups=`, check your groups with `groups`, and check all group members with `getent group $groupname`. If you should have access but don’t, email our support alias with a CC to your group leader, and we will fix this.

You also need a **Triton account**.

---

6.1. Triton user guide 147
Basics

The DGX machines have a special operating system from Nvidia based on Ubuntu 16.04, and thus form a very special
of a Triton node because the rest of Triton is CentOS. We have done work to make them work together, but it will
require special effort to make code run on both halves. You may find some problems, so please be aggressive about
filing issues (but also aggressive about checking the background yourself and giving us good information).

Basic reading: Connecting to Triton.

Unlike before, direct access is not available: you should connect to the login node and submit jobs via Slurm, not
running directly interactively.

Software and modules

Basic reading: Software Modules.

You should load software using the module command, just like the rest of Triton. However, since the base operating
system is different, modules are not automatically compatible. So, you can’t automatically reuse the modules you use
on the rest of Triton.

The current available modules are:

```
----------------------- /share/apps/anaconda-ci/modules ------------------------
anconda2/5.1.0-cpu    anaconda3/5.1.0-cpu
anaconda2/5.1.0-gpu (D) anaconda3/5.1.0-gpu (D)
---------------- /share/apps/singularity-ci/dgx/modules/common -----------------
nvidia-cafe/18.02-py2   nvidia-pytorch/18.11-py3   (D)
nvidia-caffe/18.02-py2   nvidia-tensorflow/18.02-py2
nvidia-cntk/18.02-py3   nvidia-tensorflow/18.02-py3
nvidia-mxnet/18.02-py2  nvidia-tensorflow/18.05-py3
nvidia-mxnet/18.08-py3  nvidia-tensorflow/19.01-py3 (D)
nvidia-mxnet/18.11-py3 (D) nvidia-theano/18.02
nvidia-pytorch/18.02-py3 nvidia-torch/18.02-py2
nvidia-pytorch/18.08-py3 singularity-tensorflow/latest
------------------------- /share/apps/modulefiles/dgx --------------------------
matlab/r2012a        matlab/r2015b        matlab/r2016b        matlab/r2018a
matlab/r2014a        matlab/r2016a        matlab/r2017b        matlab/r2018b (D)
------------------------- /usr/share/lmod/lmod/modulefiles/Core ----------------
   lmod/6.6   settarg/6.6
```

Unlike the rest of Triton, you can’t see which modules are available on the login node: currently see above (which
might go out of date) or get an interactive shell on the DGX node (see below) and run module avail yourself.

Running jobs

Basic reading: tutorials on interactive jobs, serial jobs

All runs on the DGX machines go via Slurm. For an introduction to slurm, see the tutorials linked above, and in
general all the rest of the Triton user guide. Slurm is a cluster scheduling system, which takes job requests (code,
CPU/memory/time/hardware requirements) and distributes it to nodes. You basically need to declare what your jobs
require, and tell it to run on DGX nodes.
Basic required slurm options

The necessary Slurm parameters are:

- `-p dgx` (dedicated group access) or `-p dgx-common` (general access, jobs may be killed at any time, see above) to indicate that we want to run in the DGX partitions.

- `--gres=gpu:v100:1` to request GPUs (Slurm also manages GPUs and limits you to the proper devices).
  - To request more than one graphics card, `--gres=gpu:v100:2`

- `--export=HOME,USER,TERM,WRKDIR` to limit the environment exported. Because these are a different operating system, you need to clear most environment variables. If there are extra environment variables you need, add them here.

- `/bin/bash -l`: you need to give the full path to bash and request a login shell, or else the environment won’t be properly set by Slurm.

- To set the run time, `--time=HH:MM:SS`. If you want more CPUs, add `-c N`. If you want more (system) memory, use `--mem=5GB` and so on. (These are completely generic slurm options.)

To check running and jobs: `squeue -p dgx,dgx-common` (whole cluster) or `slurm q` (for your own jobs).

Getting an interactive shell for own work

For example, to get an interactive shell, run:

```
srun -p dgx --gres=gpu:v100:1 --export=HOME,USER,TERM,WRKDIR --pty /bin/bash -l
```

From here, you can do whatever you want interactively with your dedicated resources almost as if you logged in directly. Remember to log out when done, otherwise your resources stay dedicated to you and no one else can use them!

Batch scripts

Similarly to the rest of Triton, you can make batch scripts:

```
#!/bin/bash -l
#SBATCH -p dgx
#SBATCH --gres=gpu:1
#SBATCH --mem=5G --time=5:00
#SBATCH --export=HOME,USER,TERM,WRKDIR

your shell commands here
```

Nvidia containers

Some of the Nvidia containers designed for the DGX machines are available as modules - see above. They are integrated with our Triton singularity setup, so you can use those same procedures:

```
module load nvidia-tensorflow

# Get a shell within the image:
singularity_wrapper shell
```

(continues on next page)
## Execute Python within the image

```bash
singularity_wrapper exec python3 code.py
```

`singularity_wrapper` sets the image file (from the module you loaded), important options (to bind-mount things), and starts it.

This is a minimum slurm script (submit with `sbatch`, see the slurm info above and tutorials for more info):

```bash
#!/bin/bash -l
#SBATCH -p dgx
#SBATCH --gres=gpu:1
#SBATCH --mem=5G --time=5:00
#SBATCH --export=HOME,USER,TERM,WRKDIR

module load nvidia-tensorflow
singularity_wrapper exec python -V
```

### Other notes

Note: if you are using tensorboard, just have it write data to the scratch filesystem, mount that on your workstation, and follow it that way. See the [data storage tutorial](#).

Within jobs, use `/tmp` for temporary local files. This is bind-mounted per user (not per job, make sure that you prefix by job ID or something to not get conflicts) to the `/raid` SSD area. (note: see below, this doesn’t work yet)

### Known bugs

- You have to give the full path to `/bin/bash` and give the `-l` option to make a login shell to read necessary shell initialization.
- You have to limit the environment variables you export, because they are different. But you have to export at least `HOME` and possibly more (see above).
- You can’t figure out modules are available without getting an interactive shell there.
- The `/tmp` directory is not automatically to a per-user tmpdir (or `/raid`). For large amounts of intermediates, use a per-user subdirectory of `/raid` for your work.
- `/scratch` isn’t automatically mounted for some reason. For now, we manually mount it on each reboot but this needs fixing.

### Frequently asked questions

Frequently Asked Questions. The latest are at the beginning.

**libcuda.so.1: cannot open shared object file: No such file or directory**

You are trying to run a GPU program (using CUDA) on a node without a GPU (and thus, no `libcuda.so.1`. Remember to specify that you need GPUs
Why are my jobs waiting in the queue with reason AssocGrpMemRunMinutes/AssocGrpCPURunMinutes or such

Accounts are limited in how much the can run at a time, in order to prevent a single or a few users from hogging the entire cluster with long-running jobs if it happens to be idle (e.g. after a service break). The limit is such that it limits the maximum remaining runtime of all the jobs of a user. So the way to run more jobs concurrently is to run shorter and/or smaller (less CPU’s, less memory) jobs. For an in-depth explanation see http://tech.ryancox.net/2014/04/scheduler-limit-remaining-cputime-per.html and for a graphical simulator you can play around with: https://marylou.byu.edu/simulation/grpcpurunmins.php. You can see the exact limits of your account with

```
sacctmgr -s show user $USER format=user,account,grptresrunmins%70
```

Why are my jobs in state “launch failed requeued held”

Slurm is configured such that if a job fails due to some outside reason (e.g. the node where it’s running fails rather than the job itself crashing due to a bug in the job) the job is requeued in a held state. If you’re sure that everything is ok again you can release the job for scheduling with “scontrol release JOBID”. If you don’t want this behavior (i.e. you’d prefer that such failed jobs would just disappear) then you can prevent the requeuing with

```
#SBATCH --no-requeue
```

Invalid account … error message

While submitting a job you receive an error message like

```
sbatch: error: Batch job submission failed: Invalid account or account/partition_—combination specified
```

Most probably your account is missing from SLURM database, to check it out run

```
$ sacctmgr show user $USER
-----------------------
User   Def Acct   Admin
---------   --------  -------
YOUR_LOGIN YOUR_DEPART None
```

That should return your login and associated department/school. If empty, please contact your local support team member and ask to add your account to SLURM db.

How can I find out the remaining runtime of my job/allocation

You can find out the remaining time of any job that is running with

```
squeue -h -j -o %L
```

Inside a job script or sinteractive session you can use the environment variable SLURM_JOB_ID to refer to the current job ID.

Disk quota exceeded error but I have plenty of space

Main article: Triton Quotas

6.1. Triton user guide 151
Everyone should have a group quota, but no user quota. All files need to be in a proper group (either a shared group with quota, or your “user private group”). First of all, use the ‘quota’ command to make sure that neither disk space nor number of files are exceeded. Also, make sure that you use $WRKDIR for data and not $HOME. If you actually need more quota, ask us.

Solution: add to your main directory and all your subdirectories to the right group, and make sure all directories have the group s-bit set, (SETGID bit, see man chmod). This means “any files created within this directory get the directory’s group”. Since your default group is “domain users” which has no quota, if the s-bit is not set, you get an immediate quota exceeded by default.

Why this happens: $WRKDIR directory is owned by the user and user’s group that has the same name and GID as UID. Quota is set per group, not per user. That is how it was implemented since 2011 when we got Lustre in use. Since spring 2015 Triton is using Aalto AD for the authentication which sets everyone a default group ID to ‘domain users’. If you copy anything to $WRKDIR/subdirectory that has no +s bit you copy as a ‘domain users’ member and file system refuses to do so due to no quota available. If g+s bit is set, all your directories/files copied/created will get the directory’s group ownership instead of that default group ‘domain users’. There can be very confusing interactions between this and user/shared directories.

My $WRKDIR is not visible on my department computer

Most likely your Kerberos ticket has expired. If you log in with a password or use ‘kinit’, you can get an another ticket. See page on data storage for more information.

While copying to $WRKDIR with rsync or cp I’m getting ‘Disk quota exceeded’ error, though my quota is fine.

It is related to the above mentioned issue, something like rsync -a ... or cp -p ... are trying to save original group ownership attribute, which will not work. Try this instead:

Can I change zsh to bash?

Yes. Change shell to your Aalto account and re-login to Triton to get your newly changed shell to work. For Aalto account changes one can login to kosh.aalto.fi, run kinit first and then run chsh, then type /bin/bash. To find out what is your current shell, run echo $SHELL

For the record: your default shell is not set by Triton environment but by your Aalto account.
Job fails due to missed module environment variables.

You have included ‘module load module/name’ but job still fails due to missing shared libraries or that it can not find some binary etc. That is a known ZSH related issue. In your sbatch script please use –l option (aka --login) which forces bash to read all the initialization files at /etc/profile.

```bash
#!/bin/bash -l
...
```

Alternatively, one can change shell from ZSH to BASH to avoid this hacks, see the post above.

There seems to be running a lot of jobs in the short queue that has gone for longer than 4 hours. Should that be possible?

SLURM kills jobs based on the partition’s TimeLimit + OverTimeLimit parameter. The later in our case is 60 minutes. If for instance queue time limit is 4 hours, SLURM will allow to run on it 4 hours, plus 1 hour, thus no longer than 5 hours. Though OverTimeLimit may vary, don’t rely on it. Partition’s (aka queue’s) TimeLimit is the one that end user should take into account when submit his/her job. Time limits per partition one can check with `slurm p` command.

For setting up exact time frame after which you want your job to be killed anyway, set --time parameter when submitting the job. When the time limit is reached, each task in each job step is sent SIGTERM followed by SIGKILL. If you run a parallel job, set --time with `srun` as well. See ‘man srun’ and ‘man sbatch’ for details.

```bash
#SBATCH --time=1:00:00
...
srun --time=1:00:00 ...
```

How can I print my text file to a local department printer?

We don’t have local department printers configured anywhere on Triton. But one can use SSH magic to send a file or command output to a remote printer. Run from your local workstation, insert the target printer name:

```bash
... printing text file
$ ssh user@triton.aalto.fi "cat file.txt" | enscript -P printer_name
... printing a PostScript file
$ ssh user@triton.aalto.fi "cat file.ps" | lp -d printer_name -
... printing a man page
$ ssh user@triton.aalto.fi "man -t sbatch" | lp -d printer_name -
```

How can I access my Triton files from outside?

Remote mounting

The scratch filesystem can be mounted from inside the Aalto networks by using `smb://data.triton.aalto.fi/scratch/`. For example, from Nautilus (the file manager) on Ubuntu, use “File” -> “Connect to server”. Outside Aalto networks, use the Aalto VPN. If it is not an Aalto computer, you may need to use AALTO\username as the username, and your Aalto password.

Or you can use `sshfs` – filesystem client based on SSH. Most Linux workstations have it installed by default, if not, install it or ask your local IT support to do it for you. For setting up your SSHFS mount from your local workstation: create a local directory and mount remote directory with `sshfs`
Replace `user1` with your real username and `/LOCALDIR` with a real directory on your local drive. After successful mount, use your `/LOCALDIR` `triton` directory as it would be local. To unmount it, run `fusermount -u /LOCALDIR/triton`.

**PHYS users example, assuming that Triton and PHYS accounts are the same:**

```bash
$ mkdir /localwrk/$USER/triton
$ sshfs triton.aalto.fi:/triton/PATH/TO/DIR /localwrk/$USER/triton
$ cd /localwrk/$USER/triton
... (do what you need, and then unmount when there is no need any more)
$ fusermount -u /localwrk/$USER/triton
```

### Easy access with Nautilus

The SSHFS method described above works from any console. Though in case of Linux desktops, when one has a GUI like Gnome or Unity (read all Ubuntu users) one may use Nautilus – default file manager – to mount remote SSH directory. Click `File -> Connect to Server` choose `SSH`, input `triton.aalto.fi` as a server and directory `/triton/PATH/TO/DIR` you’d like to mount, type your name. Leave password field empty if you use SSH key. As soon as Nautilus will establish connection it will appear on the left-hand side below Network header. Now you may access it as it would be your local directory. To keep it as a bookmark click on the mount point and press Ctrl+D, it will appear below Bookmark header on the same menu.

### Copying files

If your workstations has no NFS mounts from Triton (CS and NBE have, consult with your local admins for exact paths), you may always use SSH. Either copy your files from triton to a local directory on your workstation, like:

```bash
$ sftp user1@triton.aalto.fi:/triton/path/to/dir/* .
```

### How can I copy Triton files from outside of Aalto?

It is an extension of the previous question. In case you are outside of Aalto and has neither direct access to Triton nor access to NFS mounted directories on your directory servers. Say you want to copy your Triton files to your home workstation. It could be done by setting up an SSH tunnel to your department SSH server. A few steps to be done: set tunnel to your local department server, then from your department server to Triton, and then run any rsync/sftp/ssh command you want from your client using that tunnel. The tunnel should be up during whole session.

```bash
client: ssh -L9509:localhost:9509 department.ssh.server
department server: ssh -L9509:localhost:22 triton.aalto.fi
client: sftp -P 9509 localhost:/triton/own/dir/* /local/dir
```

Note that port 9509 is taken for example only. One can use any other available port. Alaternatively, if you have a Linux or Mac OS X machine, you can setup a “proxy command”, so you don’t have to do the steps above manually everytime. On your home machine/laptop, in the file `~/.ssh/config put the lines`

```bash
Host triton
  ProxyCommand /usr/bin/ssh DEPARTMENTUSERNAME@department.ssh.server "/usr/bin/nc -w 10 triton.aalto.fi 22"
  User TRITONUSERNAME
```
This creates a host alias “triton” that is proxied via the department server. So you can copy a file from your home machine/laptop to triton with a command like:

```
rsync filename triton:remote_filename
```

I need to connect to some server on a node

Let’s say you have some server (e.g. debugging server, notebook server, ...) running on a node.

**Why are my jobs waiting in the queue with reason AssocGrpMemRunMinutes/AssocGrpCPURunMinutes or such**

Accounts are limited in how much the can run at a time, in order to prevent a single or a few users from hogging the entire cluster with long-running jobs if it happens to be idle (e.g. after a service break). The limit is such that it limits the maximum remaining runtime of all the jobs of a user. So the way to run more jobs concurrently is to run shorter and/or smaller (less CPU’s, less memory) jobs. For an in-depth explanation see [http://tech.ryancox.net/2014/04/scheduler-limit-remaining-cputime-per.html](http://tech.ryancox.net/2014/04/scheduler-limit-remaining-cputime-per.html) and for a graphical simulator you can play around with: [https://marylou.byu.edu/simulation/grpcpurunmins.php](https://marylou.byu.edu/simulation/grpcpurunmins.php). You can see the exact limits of your account with

```
sacctmgr -s show user $USER format=user,account,grptresrunmins
```

**Why are my jobs in state “launch failed requeued held”**

Slurm is configured such that if a job fails due to some outside reason (e.g. the node where it’s running fails rather than the job itself crashing due to a bug in the job) the job is requeued in a held state. If you’re sure that everything is ok again you can release the job for scheduling with “scontrol release JOBID”. If you don’t want this behavior (i.e. you’d prefer that such failed jobs would just disappear) then you can prevent the requeuing with

```
#SBATCH --no-requeue
```

**Invalid account … error message**

While submitting a job you receive an error message like

```
sbatch: error: Batch job submission failed: Invalid account or account/partition combination specified
```

Most probably your account is missing from SLURM database, to check it out run

```
$ sacctmgr show user $USER
```

```
User    Def Acct   Admin
----------  ----------
YOUR_LOGIN YOUR_DEPART None
```

That should return your login and associated department/school. If empty, please contact your local support team member and ask to add your account to SLURM db.
How can I find out the remaining runtime of my job/allocation

You can find out the remaining time of any job that is running with

```
squeue -h -j -o %L
```

Inside a job script or `sinteractive` session you can use the environment variable `SLURM_JOB_ID` to refer to the current job ID.

**Disk quota exceeded error**

*Main article: ‘Triton Quotas <quotas>’*

Everyone should have a group quota, but no user quota. All files need to be in a proper group (either a shared group with quota, or your “user private group”). First of all, use the ‘quota’ command to make sure that neither disk space nor number of files are exceeded. Also, make sure that you use `SWRKDIR` for data and not `HOME`. If you actually need more quota, ask us.

*Solution:* add to your main directory and all your subdirectories to the right group, and make sure all directories have the group s-bit set, (SETGID bit, see `man chmod`). This means “any files created within this directory get the directory’s group”. Since your default group is “domain users” which has no quota, if the s-bit is not set, you get an immediate quota exceeded by default.

```
# Fix everything
# (only for $WRKDIR or group directories, still in testing):
/share/apps/bin/quotafix -sg --fix /path/to/dir/

# Manual fixing:
# Fix sticky bit:
lfs find $WRKDIR -type d --print0 | xargs -0 chmod g+s
# Fix group:
lfs find /path/to/dir ! --group $GROUP -print0 | xargs -0 chgrp $GROUP
```

**Why this happens:** `$WRKDIR` directory is owned by the user and user’s group that has the same name and GID as UID. Quota is set per group, not per user. That is how it was implemented since 2011 when we got Lustre in use. Since spring 2015 Triton is using Aalto AD for the authentication which sets everyone a default group ID to ‘domain users’. If you copy anything to `$WRKDIR`/subdirectory that has no +s bit you copy as a ‘domain users’ member and file system refuses to do so due to no quota available. If g+s bit is set, all your directories/files copied/created will get the directory’s group ownership instead of that default group ‘domain users’. There can be very confusing interactions between this and user/shared directories.

**While copying to $WRKDIR with rsync or cp I’m getting ‘Disk quota exceeded’ error, though my quota is fine.**

It is related to the above mentioned issue, something like `rsync -a ...` or `cp -p ...` are trying to save original group ownership attribute, which will not work. Try this instead:

```
## mainly one should avoid -g (as well as -a) that preserves group attributes
$ rsync -urpDxv --chmod=Dg+s somefile triton.aalto.fi:/path/to/work/directory
```

(continues on next page)
Can I change zsh to bash?

Yes. Change shell to your Aalto account and re-login to Triton to get your newly changed shell to work. For Aalto account changes one can login to kosh.aalto.fi, run kinit first and then run chsh, then type /bin/bash. To find out what is your current shell, run echo $SHELL

For the record: your default shell is not set by Triton environment but by your Aalto account.

Job fails due to missed module environment variables.

You have included ‘module load module/name’ but job still fails due to missing shared libraries or that it can not find some binary etc. That is a known ZSH related issue. In your sbatch script please use –l option (aka --login) which forces bash to read all the initialization files at /etc/profile.

Alternatively, one can change shell from ZSH to BASH to avoid this hacks, see the post above.

There seems to be running a lot of jobs in the short queue that has gone for longer than 4 hours. Should that be possible?

SLURM kills jobs based on the partition’s TimeLimit + OverTimeLimit parameter. The later in our case is 60 minutes. If for instance queue time limit is 4 hours, SLURM will allow to run on it 4 hours, plus 1 hour, thus no longer than 5 hours. Though OverTimeLimit may vary, don’t rely on it. Partition’s (aka queue’s) TimeLimit is the one that end user should take into account when submit his/her job. Time limits per partition one can check with slurm p command.

For setting up exact time frame after which you want your job to be killed anyway, set --time parameter when submitting the job. When the time limit is reached, each task in each job step is sent SIGTERM followed by SIGKILL.

If you run a parallel job, set --time with srun as well. See ‘man srun’ and ‘man sbatch’ for details.

How can I print my text file to a local department printer?

We don’t have local department printers configured anywhere on Triton. But one can use SSH magic to send a file or command output to a remote printer. Run from your local workstation, insert the target printer name:

... printing text file
$ ssh user@triton.aalto.fi "cat file.txt" | enscript -P printer_name

... printing a PostScript file
$ ssh user@triton.aalto.fi "cat file.ps" | lp -d printer_name -
How can I access my Triton files from outside?

If your workstations has no NFS mounts from Triton (CS and NBE have, consult with your local admins for exact paths), you may always use SSH. Either copy your files from triton to a local directory on your workstation, like

```bash
$ rsync -pr user1@triton.aalto.fi:/triton/path/to/dir /
```

or use SSHFS – filesystem client based on SSH. Most Linux workstations have it installed by default, if not, install it or ask your local IT support to do it for you. For setting up your SSHFS mount from your local workstation: create a local directory and mount remote directory with sshfs

```bash
$ mkdir /LOCALDIR/triton
$ sshfs user1@triton.aalto.fi:/triton/PATH/TO/DIR /LOCALDIR/triton
```

Replace `user1` with your real username and `/LOCALDIR` with a real directory on your local drive. After successful mount, use you `/LOCALDIR/triton` directory as it would be local. To unmount it, run `fusermount -u /LOCALDIR/triton`.

**PHYS users example, assuming that Triton and PHYS accounts are the same:**

```bash
$ mkdir /localwrk/$USER/triton
$ sshfs triton.aalto.fi:/triton/TFY/work/$USER /localwrk/$USER/triton
$ cd /localwrk/$USER/triton
... (do what you need, and then unmount when there is no need any more)
$ fusermount -u /localwrk/$USER/triton
```

**Easy access with Nautilus**

The SSHFS method described above works from any console. Though in case of Linux desktops, when one has a GUI like Gnome or Unity (read all Ubuntu users) one may use Nautilus – default file manager – to mount remote SSH directory. Click *File -> Connect to Server* choose *SSH*, input triton.aalto.fi as a server and directory `/triton/PATH/TO/DIR` you’d like to mount, type your name. Leave password field empty if you use SSH key. As soon as Nautilus will establish connection it will appear on the left-hand side below Network header. Now you may access it as it would be your local directory. To keep it as a bookmark click on the mount point and press Ctrl+D, it will appear below Bookmark header on the same menu.

**How can I copy Triton files from outside of Aalto?**

It is an extension of the previous question. In case you are outside of Aalto and has neither direct access to Triton nor access to NFS mounted directories on your directory servers. Say you want to copy your Triton files to your home workstation. It could be done by setting up an SSH tunnel to your department SSH server. A few steps to be done: set tunnel to your local department server, then from your department server to Triton, and then run any rsync/sftp/ssh command you want from your client using that tunnel. The tunnel should be up during whole session.

```
client: ssh -L9509:localhost:9509 department.ssh.server
department server: ssh -L9509:localhost:22 triton.aalto.fi
client: sftp -P 9509 localhost:/triton/own/dir/* /local/dir
```
Note that port 9509 is taken for example only. One can use any other available port. Alternatively, if you have a Linux or Mac OS X machine, you can setup a “proxy command”, so you don’t have to do the steps above manually everytime. On your home machine/laptop, in the file ~/.ssh/config put the lines

```
Host triton
  ProxyCommand /usr/bin/ssh DEPARTMENTUSERNAME@department.ssh.server "'/usr/bin/nc -w 10 triton.aalto.fi 22"
  User TRITONUSERNAME
```

This creates a host alias “triton” that is proxied via the department server. So you can copy a file from your home machine/laptop to triton with a command like:

```
rsync filename triton:remote_filename
```

### I need to connect to some server on a node

Let’s say you have some server (e.g. debugging server, notebook server, …) running on a node. As usual, you can do this with ssh using port forwarding. It is the same principle as in several of the above questions.

For example, you want to connect from your own computer to port AAAA on node nnnNNN. You run this command:

```
ssh -L BBBB:nnnNNN:AAAA username@triton.aalto.fi
```

Then, when you connect to port BBBB on your own computer (localhost, it gets forwarded straight to port AAAA on node nnnNNN. Thus only one ssh connection gets us to any node. It is possible for BBBB to be the same as AAAA. By the way, this works with any type of connection. The node has to be listening on any interface, not just the local interface. To connect to `localhost:AAAA` on a node, you need to repeat the above steps twice to forward from `workstation->login` and `login->node`, with the second nnnNNN being `localhost`.

### Why all of the files on triton cluster are in one color? How can I make them colorful? Like green for execution files, blue for folds

That is made intentionally due to high load on Lustre filesystem. Being a high performance filesystem Lustre still has its own bottlenecks, and one of the common Lustre troublemakers are `ls` `-lr` or `ls` `--color` which generate lots of requests to Lustre meta servers which regular usage by all users may get whole system in stuck. Please follow the recommendations given at the last section at *Data storage on the Lustre file system*

### How do I subscribe to triton-users maillist?

Having a user account on Triton also means being on the triton-users@lists.aalto.fi mailist. That is where support team sends all the Triton related announcements. All the Triton users MUST be subscibed to the list. Just in case you are not yet there, please send an email to your local team member and ask to add your email. Same also if you want to replace your email with a new one.

How to unsubscribe? You will be removed from the maillist as soon as your Triton account is deleted from the system. Otherwise no way.

### I can’t save anything to my $HOME directory, get some fsync error.

Most probably your quota has exceeded, check it out with `quota` command.
quota is a wrapper at /usr/local/bin/quota on front end which merges output from classic quota utility that supports NFS and Lustre’s lfs quota. NFS $HOME directory is limited to 10GB for everyone and intended for initialization files mainly. Grace period is set to 7 days and “hard” quota is set to 11GB, which means you may exceed your 10GB quota by 1GB and have 7 days to go below 10GB again. However none can exceed 11GB limit.

Note: Lustre mounted under /triton is the right place for your simulation files. It is fast and has large quotas.

What node names like cn[01-224] mean?

All the hardware delivered by the vendor has been labeled with some short name. In particular every single compute node has a label like Cn01 or GPU001 etc. we used this notation to name compute nodes, that is cn01 is just a hostname for Cn01, gpu001 is a hostname for GPU001 etc. Shorthands like cn[01-224] mean all the hostnames in the range cn01, cn02, cn03 .. cn224. Same for gpu[001-008], tb[003-008], fn[01-02]. Similar notations can be used with SLURM commands like:

```sh
$ scontrol show node cn[01-12]
```

What is a good scaling factor for parallel applications? What is the recommended number of processors for parallel jobs?

The good scaling factor is 1.5 or higher. It means that your program is running 1.5 times faster when you double the number of nodes.

There is no way to know in advance the exact “universal” optimal number of CPUs. It depends on many factors, like the application itself, type of MPI libraries, the initial input, I/O volume and the current network state. Certainly, you must not expect that, as many CPUs your application has got, that faster it will run. In general the scaling on Triton is good since we have Infiniband for nodes interconnect and DDN / Lustre for I/O.

Few recommendations about CPU number:

- benchmark your applications on different number of CPU cores 1, 2, 12, 24, 36, and larger. Check out with the developers, your application may have ready scalability benchmarks and recommendations for compiler, MPI libraries choice.
- benchmark on shared memory i.e. up to 12 CPU cores within one node and then on different nodes (distributed memory): involving interconnect make have huge difference
- if you are not sure about program scalability and you have no time for testing, don’t run on more than 12 CPU cores within one node
- be considerate! it is not you against others! do not try to fill up the cluster just for being cool

Can you recovery some files from my $HOME or $WRKDIR directory?

Short answer: yes for $HOME directory and no for $WRKDIR.

$HOME is slow NFS with small quota mounted through Ethernet. Intended mainly for user initialization files and for some plain configs. We make regular backups from $HOME.

$WRKDIR (aka /triton) is fast Lustre, has large quota, mounted through InfiniBand. Though no backups made from /triton, the DDN storage system as such is secure and safe place for your data, though you can always loose your data deleting them by mistake. Every user must take care about his work files himself. We provide as much diskspace to every user, as one needs and the amount of data is growing rapidly. That is the reason why the user
should manage his important data himself. Consider backups of your valuable data on DVDs/ USB drives or other resources outside of Triton.

**The cluster has a few compiler sets. Which one I suppose to use? What are the limits for commercial compilers?**

Currently there are two different sets of compilers: (i) GNU compilers, native for Linux, installed by default, (ii) Intel compilers plus MKL, a commercial suite, often the fastest compiler on Xeons.

FGI provides all FGI sites with 7 Intel licenses, thus only 7 users can compile/link with Intel at once.

**Code is compiled with shared libraries and it stops with an error message: error while loading shared libraries: libsome.so: cannot open shared object file: No such file or directory**

That means your program can’t find libraries which has been used at linking/compiling time. You may always check shared library dependencies:

```bash
$ ldd YOUR_PROGRAM # print the list of libraries required by program
```

If some of libraries is marked as not found, then you should first (i) find the exact path to that lib (suppose it is installed), then second (ii) explicitly add it to your environment variable $LD_LIBRARY_PATH.

For instance, if your code has been previously compiled with the *libmpi.so.0* but on SL6.2 it reports an error like

```bash
error while loading shared libraries: libmpi.so.0
```

try to locate the library:

```bash
$ locate libmpi.so.0
/usr/lib64/compat-openmpi/lib/libmpi.so.0
/usr/lib64/compat-openmpi/lib/libmpi.so.0.0.2
```

and the add it to your $LD_LIBRARY_PATH

```bash
export LD_LIBRARY_PATH=/usr/lib64/compat-openmpi/lib:$LD_LIBRARY_PATH # export the lib
```

or, as in case of *libmpi.so.0* we have ready module config, just run

```bash
module load compat-openmpi-x86_64
```

In case your code is missing some specific libs, not installed on Triton (say you got a binary compiled from somewhere else), you have a few choices: (i) get statically linked program or (ii) find/download missing libs (for instance from developers’ site). For the second, copy libs to your $WRKDIR and add paths to $LD_LIBRARY_PATH, in the same manner as described above.

See also:

```bash
ldconfig -p # print the list of system-wide available shared libraries
```
While compiling should I use static or shared version of some library?

One can use both, though for shared libs all your linked libs must be either in your $WRKDIR in /shared/apps or must be installed by default on all the compute nodes like vast majority of GCC and other default Linux libs.

I’ve got a binary file, may I find out somehow whether it is 32-bit or 64-bit compiled?

Use `file` utility:

```
# file /usr/bin/gcc
/usr/bin/gcc: ELF 64-bit LSB executable, AMD x86-64, version 1 (SYSV),
for GNU/Linux 2.4.0, dynamically linked (uses shared libs), not stripped
```

it displays the type of an executable or object file.

**Graphical programs don’t work (X11, -X)**

In order for graphical programs on Linux to work, a file `~/.Xauthority` has to be written. If your home directory quota (check with `quota`) is exceeded, then this can’t be written and graphical programs can’t open. If your quota is exceeded, clean up some files, close connections, and log in again. You can find where most of your space goes with `du -h $HOME | sort -hr | less`.

This is often the case if you get X11 connection rejected because of wrong authentication.

**Running programs on Triton**

Triton differs somewhat from your regular desktop computer. The large numbers and complex examples may give the impression of something far more special than it actually is: a bunch of computers. Fundamentally these are just slightly more powerful computers, with much more memory and a faster network in between. Where the major differences begin, though, is that they are shared by around 100 people from different departments with an unusual scale and variation of applications and needs. In order to even begin to accommodate everyone on the cluster, we have to use an intermediate resource manager and scheduler through which certain policies can be put into effect. The cluster is a combination of the compute nodes, our site policies, and the scheduler software which works it all out in practice.

This guide tries to give an idea of how to run programs in the cluster through the Slurm scheduler. While this certainly does not cover all the use cases, you’re welcome to ask any questions in the Issue Tracker.

**Scheduling policy and queues**

The cluster nodes (computers) are grouped into partitions (the scheduler’s concept). While the default batch partition may always be in full use, other partitions act as boundaries that keep specialized nodes, such as the GPU machines, ready and immediately available for jobs with special requirements.
Use `slurm partitions` to see more details.

The debug partition and its 60 minute time limit exists for developing code and testing job scripts and simply getting used to the cluster commands. Don’t run anything here unless it is your current work focus.

**Most of the time, you should not need to specify any partition (debug, interactive, or your group’s dedicated partitions).** When you submit a job, there is a script (`job_submit.lua`) that runs in the background and automatically selects partitions. If you notice the script doing something wrong, submit an issue and we can look at it. It roughly uses this logic:

- Do you use `--gres=gpu`? If so, do GPU partitions.
- Otherwise, default to batch
- If your time limit is less than the short time limit, also add in the short partitions, too
- If you use large amounts of memory, add hugemem.

It can be worth looking at your job’s partition list to make sure it is optimal: “slurm j $jobid”

### Interactive logins

Triton mainly runs non-interactive batch jobs in its current configuration. There is a (small) partition which is meant for interactive jobs. There are two main options for running interactive shells:

- Cluster frontend (login) machine `triton.aalto.fi` for editing, compiling and testing code.
- Interactive jobs started with the “sinteractive” command.

We ask you to refrain from running multi-GB, many-core applications of the frontend. The login machine `triton.aalto.fi` is mainly intended for editing code and submission scripts, sorting your files, checking jobs and of course submitting the jobs scripts through Slurm to the actual execution nodes (called `cn-something`, `ivy*`, `gpu*` or `tb*`).

Interactive and computationally intensive applications should be run on the interactive partition. Still, to maximise the resource usage it’s best to structure your workflow such that you can use normal batch jobs.

You can access an interactive shell with the "sinteractive" command from the frontend machine `triton.aalto.fi`:

<table>
<thead>
<tr>
<th>Partition</th>
<th>Max job size</th>
<th>Mem/core (GB)</th>
<th>Tot mem (GB)</th>
<th>Cores/node limits</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;default&gt;</td>
<td>2 nodes</td>
<td>2.66 - 12</td>
<td>32-256</td>
<td>12,20,24</td>
<td>If you leave off all possible partitions will be used (based on time/mem)</td>
</tr>
<tr>
<td>debug</td>
<td>16 nodes</td>
<td>2.66 - 12</td>
<td>32-256</td>
<td>12,20,24</td>
<td>testing and debugging short interactive work. 1 node of each arch.</td>
</tr>
<tr>
<td>batch</td>
<td>8 nodes</td>
<td>4 - 12</td>
<td>48-256</td>
<td>12,20,24</td>
<td>primary partition, all serial &amp; parallel jobs</td>
</tr>
<tr>
<td>huge-mem</td>
<td>1 node</td>
<td>43</td>
<td>1024</td>
<td>24</td>
<td>huge memory jobs, 1 node only</td>
</tr>
<tr>
<td>gpu</td>
<td>1 node, 2-8GPUs</td>
<td>2 - 10</td>
<td>24-128</td>
<td>12</td>
<td>GPU computing</td>
</tr>
<tr>
<td>gpushort</td>
<td>4 nodes, 2-8GPUs</td>
<td>2 - 10</td>
<td>24-128</td>
<td>12</td>
<td>GPU computing</td>
</tr>
<tr>
<td>interactive</td>
<td>2 nodes</td>
<td>5</td>
<td>128</td>
<td>24</td>
<td>for sinteractive command, longer interactive work</td>
</tr>
</tbody>
</table>
Launch 2 hour interactive session

```bash
$ sinteractive -t 2:0
```

See also the interactive usage section below for advanced examples.

**Job examples**

**Submit a short batch job**

Batch job is by default a single-CPU job that will run specified commands in series and optionally save output into a file.

A number of nodes have been dedicated for jobs that run under four hours, which makes it more likely that resources are immediately available.

Short batch example

```bash
#!/bin/bash
#SBATCH -p short
#SBATCH --time=04:00:00  # 4 hours
#SBATCH --mem-per-cpu=1000  # 1G of memory

cd $WRKDIR/mydata/
srun myprog params

cd ..
```

A batch job can have as many “srun” steps as needed or just one. At the end of the day SBATCH script is just a BASH script with a set of specific directives for SBATCH. Being so, the script can be as simple as a few #SBATCH lines plus “srun” or may consists of hundreds of BASH lines. The best practice is to join the tasks into the same job and avoid short runs (that take seconds or minutes).

**Submit an array job (batch job for repeated tasks)**

Slurm supports so-called array jobs, where one can easily handle a large number of similar jobs. An array job is essentially a set of independent jobs. In this example we run an array job consisting of 30 different array tasks. In the job script, the environment variable SLURM_ARRAY_TASK_ID is the ID of the current task. In the example below, this is used to make the application read the correct input file, and to generate output in a unique directory.

```bash
#!/bin/bash
#SBATCH -n 1
#SBATCH -t 04:00:00
#SBATCH --mem-per-cpu=2500
#SBATCH --array=0-29


cd $SLURM_ARRAY_TASK_ID

cd ..
```

The array indices need not be sequential. E.g. if you discover that after the above array job is finished, the job task id’s 7 and 19 failed, you can relaunch just those jobs with “--array=7,19”. While the array job above is a set of serial jobs, parallel array jobs are possible. For more information, see the Slurm job array documentation.
Submit a multithreaded job

Programs using multiple threads must have their behaviour described to Slurm in terms of the number of threads needed. To launch a multithreaded job we tell slurm that we want a single task, but that that one task requires several CPU’s. This is done with the `--cpus-per-task=N` (or the short form `-c N`) option and should match the number of computational threads used by your application.

When moving a program from a Linux workstation to the cluster, please note than simply increasing the Slurm reservation size usually does not affect the running behavior of the program. Take a moment to see how many threads were using CPU on a workstation, and use that as a starting point (try the `top` command and press the H key to see separate threads). Not all tasks scale well to 12 (or 20, 24) threads, so run a few benchmarks in the play partition (`-p debug`) to test things before committing a lot of cluster resources to an application that may not utilize all of it. Amount of threads should be no more than number of CPU cores on the node.

For OpenMP programs the information about Slurm reservation size is passed with environment variable `OMP_NUM_THREADS`, which controls how many OpenMP threads will be used for the job (equal to `--n #`). However by default all allocated threads are used, so you need to specify `OMP_NUM_THREADS` only if you want to launch a job step using fewer than the allocated CPU’s. Other multi-threaded programs may have similar ways to control the number of threads launched. When using OpenMP, additionally one should bind threads to CPU’s with the `OMP_PROC_BIND` environment variable.

OpenMP example

```bash
#!/bin/bash
#SBATCH --cpus-per-task=12
#SBATCH --time=40:00
#SBATCH --mem-per-cpu=2000
export OMP_PROC_BIND=true
srun /path/to/openMP_executable
```

Submit a MPI job

Slurm’s “srun” works as a wrapper to traditional “mpirun” command, it takes care of setting up a correct environment for MPI. For more information, see the slurm MPI guide.

Triton has several generations of different architectures, as of Oct 2018 we have Westmere, IvyBridge, Haswell, and Broadwell Xeons. They have different number on CPU cores per node: 12 for Westmere, 20 on IvyBridge, 24 on Haswell, and 28 on Broadwell.

Submit a small MPI job

A job that fit to one node: single-node job. Here we use the “-N 1” option which tells slurm to allocate all tasks on a single node. The “-n X” tells to SLURM how many MPI tasks you want to run.

Small MPI example using mvapich2

```bash
#!/bin/bash
#SBATCH -N 1 # on one node
#SBATCH -n 4 # 4 processes
#SBATCH --time=4:00:00 # 4 hours
#SBATCH --mem-per-cpu=2000 # 2GB per process

module load gmvolf/triton-2016a # MVAPICH + GCC + math libs modules
srun /path/to/mpi_program params
```
For “-n” less or equal to 12 this job will fit on any of the available nodes, if you put something more that 12 but below 20, it will go to either Haswell or IvyBridge nodes, and in case of up to 24 to Haswell only. Independently on the requested “-n X” one can always define the “--constraint=” and explicitly request specific CPU arch. See large MPI jobs examples below.

**Submit a large MPI job**

Large MPI-job, the one that does not fit to a single node. You should ask for a number of tasks that is a multiple of number of CPU cores on the node. Use the “exclusive” option to ensure that entire nodes are allocated, removing interference from other jobs and minimizing the number of nodes required to fulfill the allocation. One must specify type of requested CPU, number of tasks and corresponding number of nodes in the SBATCH script.

**MPI example using Open MPI**

```
#!/bin/bash
#SBATCH --time=2:00:00   # two hours job
#SBATCH --mem-per-cpu=1500  # 1.5GB of memory per process
#SBATCH --exclusive  # allocate whole node
#SBATCH --constraint=hsw   # require Haswell CPUs with 24 cores per node
#SBATCH -N 2           # on two nodes
#SBATCH -n 48         # 48 processes to run (2 x 24 = 48)

module load goolf/triton-2016a   # OpenMPI + GCC + math libs
srun /path/to/mpi/program params
```

----------------- 12 CPU cores case ---------------------------------------

```
#SBATCH --constraint=wsm  # require Westemers, 12 cores per node
#SBATCH -N 4            # on four nodes
#SBATCH -n 48          # 48 processes to run (4 x 12 = 48)
```

----------------- 20 CPU cores case ---------------------------------------

```
#SBATCH --constraint=ivb   # require IvyBridges, 20 cores per node
#SBATCH -N 2             # on two nodes
#SBATCH -n 40           # 40 processes to run (2 x 20 = 40)
```

**Submit a hybrid MPI/OpenMP job**

Batch file for running an application using a hybrid parallelization scheme with both MPI and OpenMP. Each MPI rank (process) runs a number of OpenMP threads. From the slurm perspective, this is essentially a combination of the above examples of parallel and multithreaded jobs. In this example we launch 8 MPI processes, and each MPI process runs 6 threads. The job thus uses a total of 8*6=48 cores. We explicitly require Haswell CPUs to run 4 MPI processes per node. You need to experiment with your application to see what is the best combination. Example below uses goolf-2016a with OpenMPI and GCC.

**Hybrid MPI/OpenMP example using Open MPI**

```
#!/bin/bash
#SBATCH --time=30:00
#SBATCH --mem-per-cpu=2500
```

(continues on next page)
For mvapich2 you need to disable affinity, as mvapich2 has no way of specifying that each MPI rank needs N processors. One also cannot use the OpenMP affinity features, as the lack of any MPI affinity otherwise causes all MPI ranks on a node to be bound to the same cores. Example script below:

Hybrid MPI/OpenMP example using mvapich2

```bash
#!/bin/bash
#SBATCH --time=30:00
#SBATCH --mem-per-cpu=2500
#SBATCH --exclusive
#SBATCH --constraint=[opt|wsm]
#SBATCH -n 8
#SBATCH -c 6
#SBATCH -N 4

module load gmvolf/triton-2016a
export MV2_ENABLE_AFFINITY=0
srun /path/to/MPI_OpenMP_program params
```

If using other MPI flavors, please check the manual and do some tests to verify that CPU binding works correctly.

Submit a parallel (non-MPI) job

It is possible to launch parallel jobs that do not use MPI. However in this case you are responsible for setting up any necessary communication between the different tasks (processes) in the job. Depending on the job script, resources may be allocated on several nodes, so your application must be prepared to communicate over the network. The slurm "srun" command can be used to launch a number of identical executables, one for each task. Example:

Parallel job

```bash
#!/bin/bash
#SBATCH --time=01:00 --mem-per-cpu=500
#SBATCH --exclusive
#SBATCH --constraint=[hsw|ivb|wsm]
#SBATCH -N 4

srun -N 4 hostname
```

This will print out the 4 allocated hostnames. The “-N 4” ensures that we run a task on all 4 allocated nodes. If we instead want to launch one process per allocated CPU, we can instead do “srun -n 48 executable” (4*12=48).

In a case where the program in question uses Master-Worker-paradigm, where there exists a single worker that coordinates the rest, see Compute node local drives

Most of the compute nodes are equipped with one SATA disk drive though there are some with 2 and 4. See slurm features for the full list. A node with a specific amount of drives can be requested as:

```
```
#SBATCH --gres=spindle:4

GPU cards with –gres=

See details at Slurm commands

Links and other additional materials

- SLURM: Simple Linux Utility for Resource Management

**GPU Computing**

See also:

Introductory tutorial: GPU computing (read this first)

**Overview**

Triton has GPU cards from four different NVIDIA generations, as described below.

**Hardware breakdown**

<table>
<thead>
<tr>
<th>Card</th>
<th>total amount</th>
<th>nodes</th>
<th>architecture</th>
<th>compute threads per GPU</th>
<th>memory per card</th>
<th>CUDA compute capability</th>
<th>Slurm feature name</th>
<th>Slurm gres name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tesla K80*</td>
<td>12</td>
<td>gpu[20-22]</td>
<td>Kepler</td>
<td>2x2496</td>
<td>2x12GB</td>
<td>3.7</td>
<td>kepler</td>
<td>teslak80</td>
</tr>
<tr>
<td>Tesla P100</td>
<td>20</td>
<td>gpu[23-27]</td>
<td>Pascal</td>
<td>3854</td>
<td>16GB</td>
<td>6.0</td>
<td>pascal</td>
<td>teslap100</td>
</tr>
<tr>
<td>Tesla V100</td>
<td>40</td>
<td>gpu[28-37]</td>
<td>Volta</td>
<td>5120</td>
<td>32GB</td>
<td>7.0</td>
<td>volta</td>
<td>v100</td>
</tr>
<tr>
<td>Tesla V100</td>
<td>16</td>
<td>dgx[01-02]</td>
<td>Volta</td>
<td>5120</td>
<td>16GB</td>
<td>7.0</td>
<td>volta</td>
<td>v100</td>
</tr>
</tbody>
</table>

- Note: Tesla K80 cards are in essence two GK210 GPUs on a single chip
- Note: V100 cards are part of DGX machines, which were purchased by several groups and are currently special access only. They are also a different operating system, please see Nvidia DGX machines.


**Using GPU nodes**
GPU partitions

There are two queues governing these nodes: `gpu` and `gpushort`, where the latter is for jobs up to 4 hours. Partitions are automatically selected.

The latest details can always be found with the following command:

```
$ slurm p | grep gpu
```

GPU node allocation

For `gpu` resource allocation one has to request a GPU resource, with `--gres=gpu:N`, where `N` stands for number of requested GPU cards. To request a GPU with a certain architecture, use `--constraint=GENERATION`. To request a specific card, one must use syntax `--gres=gpu:CARD_TYPE:N`. See the table below for “slurm feature name” or “Slurm gres name”. For the full current list of configured SLURM gpu cards names run `slurm features`.

Example usages:

```
--gres=gpu:2
--gres=gpu:1 --constraint=pascal
--gres=gpu:telsap100:1
```

When using multiple GPU’s please verify that the code actually uses them with instructions given in *Monitoring GPU usage*.

GPU nodes environment and CUDA

User environment on `gpu*` nodes is the same as on other nodes, the only difference is that they have nvidia kernel modules for Tesla cards. CUDA comes through module.

```
$ module avail cuda       # list installed CUDA modules
$ module load cuda/10.0.130 # load CUDA environment that you need
$ nvcc --version          # see actual CUDA version that you got
```

Running a GPU job in serial

Quick interactive run:

```
$ module load cuda
$ srun -t 00:30:00 --gres=gpu:1 $WRKDIR/my_gpu_binary
```

Allocating a gpu node for longer interactive session, this will give you a shell sessions:

```
$ module load cuda
$ sinteractive -t 4:00:00 --gres=gpu:1
gpuXX$ .... run something
gpuXX$ exit
```

Run a batch job

```
$ sbatch gpu_job.sh
```

Where `gpu_job.sh` is
#!/bin/bash
#SBATCH --time=01:15:00 ## wallclock time hh:mm:ss
#SBATCH --gres=gpu:teslak80:1 ## one K80 requested
module load cuda
## run my GPU accelerated executable, note the --gres
srun --gres=gpu:1 $WRKDIR/my_gpu_binary

Monitoring GPU usage

Currently there isn’t a good way of monitoring the gpu usage non-interactively. Interactively one can (when the job is running) ssh to the gpu node in question and run

```
login2$ ssh gpuxx
gpuxx$ watch -n 1 nvidia-smi
```

CTRL + C quits the command.

This shows the gpu usage with 1 second interval. The GPU utilized by process with PID X is shown in the first column of the second table. The first table lists the GPUs by their ID Checking the Volatile GPU-Util column gives the utilization of GPU. If your code uses less than 50% of the GPU you should try to improve the data loading / CPU part of your code as the GPU is underutilized.

If you run multi-GPU job you should verify that the all GPUs are properly utilized. For many applications one needs to use multiple CPUs to fill the GPUs with data. With badly implemented data handling multi-GPU setups can be slower than single-GPU setups.

Development

Compiling

In case you either want to compile a CUDA code or a code with GPU support, you must do it on one of the gpu nodes (because of nvidia libs installed on those nodes only).

```
$ sinteractive -t 1:00:00 --gres=gpu:1 # open a session on a gpu node
$ module load cuda # set CUDA environment
$ nvcc cuda_code.cu -o cuda_code # compile your CUDA code
.. or compile normally any other code with 'make'
```

Debugging

CUDA SDK provides an extension to the well-known gnu debugger gdb. Using cuda-gdb it is possible to debug the device code natively on the GPU. In order to use the cuda-gdb, one has to compile the program with option pair -g -G, like follows:

```
$ nvcc -g -G cuda_code.cu -o cuda_code
```

See CUDA-GDB User Guide for a more information on cuda-gdb.
Applications and known issues

Check the Applications for most software.

nvidia-smi utility

Could be useful for debugging, in case one want to see the actual gpu cards available on the node. If this command returns an error, you should report that something is wrong on the node.

```
gpuxx$ nvidia-smi -L  # gives a list of GPU cards on the node
```

cuDNN

cudnn is available as a module. The latest version can be found with module spider cudnn. Note that (at least the later versions of) cudnn require newer cards and cannot be used on the old fermi cards. E.g. tensorflow does not run on the older fermi cards for this reason.

Nvidia MPS

Nvidia Multi-Process Service (MPS) provides a way to share a single GPU among multiple processes. It can be used to increase the GPU utilization by timesharing the GPU access, e.g. one process can upload data to the GPU while another is running a kernel. To use it one must first start the MPS server, and then CUDA calls are automatically routed via the MPS server. At the end of the job one must remember to shut it down. Example job script:

```
#!/bin/bash -l
#SBATCH --time=01:15:00  ## wallclock time hh:mm:ss
#SBATCH --gres=gpu:teslak80:1  ## one K80 requested
module load cuda
## Start the MPS server
CUDA_MPS_LOG_DIRECTORY=nvidia-mps srun --gres=gpu:1 nvidia-cuda-mps-control -d&
## run my GPU accelerated executable
srun --gres=gpu:1 $WRKDIR/my_gpu_binary
## Shut down the MPS server
echo "quit" | nvidia-cuda-mps-control
```

CUDA samples

There are CUDA code samples provided by Nvidia that can be useful for a sake of testing or getting familiar with CUDA. Placed at $CUDA_ROOT/samples. To play with:

```
$ sinteractive -t 1:00:00 --gres=gpu:1
$ module load cuda
$ cp -r $CUDA_ROOT/samples $WRKDIR
$ cd $WRKDIR/samples
$ make TARGET_ARCH=x86_64
```

(continues on next page)
$ ./bin/x86_64/linux/release/deviceQuery  
...  
$ ./bin/x86_64/linux/release/bandwidthTest  
...  

Attachments and useful links

- CUDA C Programming Guide
- CUDA Zone on NVIDIA
- CUDA FAQ

Grid computing

Note: not directly related to local usage covered by other pages. The whole thing is about an upper level cluster resources usage through Grid interface.

Grid computing on FGCI

The FGCI (Finnish Grid and Cloud Infrastructure) is a joint pool of resources that consists of a number of Linux clusters placed all around Finland. Triton is one of them. Being a part of FGCI Triton provides its CPU and disk space resources to grid users. Having your local account on Triton means having access to Triton local resources only, in order to use whole grid one has to get grid certificate.

To get started with grid computing on FGCI, please consult the https://research.csc.fi/fgci-user-guide pages.

Getting help

CSC provides grid users with support at grid-support(at)csc.fi.

Triton local usage vs. Grid computing

In other words should I start thinking of running on Grid? Why yes.

FGCI is homogeneous with respect to hardware/software. You may expect same Xeon CPUs connected through Infiniband and same Linux environment on all the clusters. Thus tested binaries on Triton will work on any other FGCI resource.

Grid is the best suited for from 30 minutes up to 24 hours long jobs. It is for production runs mainly, you do the development/testing locally on Triton and then send your jobs to the grid.

One can run MPI or large number of serial jobs, no CPU units/limits, no overloaded queues, there always be a free slot for your job on one of FGCI clusters.

Grid job management can be done from anywhere you prefer, including Triton, CSC servers or your department workstation. Grid certificate is independent on your Aalto / Triton / CSC account and valid for one year.
Grid computing 2

FGI pages at CSC

Sneak peek: http://www.nordugrid.org/monitor/

Obtaining personal certificate

https://confluence.csc.fi/display/fgi/Getting+started+with+grid

Installing client tools

http://www.nordugrid.org/documents/arc-client-install.html

Using the grid

https://confluence.csc.fi/display/fgi/FGI+Hands+on+tutorial+material

Monitoring jobs

Basics

Before you start, if you have SLURM experience but new to Triton, you may want to check out the `slurm` command on Triton’ paragraph below. There we introduce Triton specific tool that is widely used for jobs monitoring and many other issues.

There are two quick ways to see your own jobs. One is to list every job and their current priority in the queues.

```
$ slurm q
PRIORITY JOBID PARTITION NAME ST TIME START_TIME
˓NODELIST(REASON)
12971826 batch-wsm DA_vk_400k_5M 0:00 N/A PENDING
12971825 batch-wsm DA_vk_400k_50M 0:00 N/A PENDING
12971818 batch-hsw DA_vk_400k_5M 0:06 2016-10-17T15:12 RUNNING
˓→ivb28
12971817 short-ivb update.12.sh 0:06 2016-10-17T15:12 RUNNING
˓→ivb13
12971816 batch-ivb FI_vk_2000k_50 0:07 2016-10-17T15:12 RUNNING
˓→ivb28
... 
```

But mostly a simple overview will do. Here is the same information as above, only this time formatted for brevity. The leftmost value is the combined job count.

```
$ slurm qq
PARTITION CPUS NODES MIN_MEM FEATURES QOS STATE REASON
1 batch-wsm 1 1 4G ivb|wsm| normal PENDING AssocGrpMemRunMinutes
1 batch-wsm 1 1 8G ivb|wsm| normal PENDING AssocGrpMemRunMinutes
242 batch-wsm 1 1 25G (null) normal PENDING AssocGrpMemRunMinutes
2 coin 1 1 4G (null) normal RUNNING None
```

(continues on next page)
For “watching” your jobs progress, use ‘watch’ option

```
slurm watch q
```

Since the cluster is in heavy use most of the time, there are other users whose jobs will of course affect what happens with yours. The state of the entire queue, including running either `slurm short` or `slurm ss`.

```
slurm s
```

With the `s` or `short` option, the output is sorted by priority and reflects the scheduler’s execution order as nodes become available. The queue position can change at any time, either from new or submissions or based on historical usage accounting.

You can choose to display the one partition you are interested in:

```
slurm ss gpu
```

Show detailed information about running job(s):

```
slurm j
```

### Job status while pending

There are a number of reasons that your job may be sitting in the queue. Listing your pending jobs with `squeue -u $USER -t PD` will help determine why your job is not running. Look at the `NODELIST(REASON)`. A pending job may have these reasons:

- **(Priority):** Other jobs have priority over your job. Just wait.
- **(Resources):** Your job has enough priority to run, but there aren’t enough free resources to run it. Just wait.
- **(RegNodeNotAvail):** You request something that is not available. Check memory requirements per CPU, CPUs per node. Possibly time limit is the issue. Could be that due to scheduled maintenance break, all nodes are reserved and thus your -t parameter can’t be larger than time left till the break.
- **(QOSResourceLimit):** Your job exceeds the QOS limits. The QOS limits include wall time, number of jobs a user can have running at once, number of nodes a user can use at once, etc. This may or may no be a permanent status. If your job requests a wall time greater than what is allowed or exceeds the limit on the number of nodes a single job can use, this status will be permanent. However, your job may be in this status if you currently have jobs running and the total number of jobs running or aggregate node usage is at your limits. In this case, jobs in this state will become eligible when your existing jobs finish.
- **(AssociationResourceLimit):** The job exceeds some limit set on the association. On triton, this in practice means the per-user GrpCPURunMins limit, which currently is 1.5M minutes per user. Wait a while for running jobs to proceed, so that new jobs may start. Also, shorter job time limits help. See `GrpCPURunMins visualizer`.

In case of the first two one can check currently estimated time the job will be started. Run `slurm j <jobid>`, look at `StartTime=`
Job states

Possible states for jobs are:

- PENDING (PD)
- RUNNING (R)
- SUSPENDED (S)
- COMPLETING (CG)
- COMPLETED (CD)
- CONFIGURING (CF)
- CANCELLED (CA)
- FAILED (F)
- TIMEOUT (TO)
- PREEMPTED (PR)
- NODE_FAIL (NF)

Modifying the job after submission

The question asked time to time: “Can one modify job parameters after it has been submitted?”. The answer is, yes it is possible, but only some parameters. For instance change memory/CPU requirements for pending job or set another time limit off running/pending job. Think carefully before you submit a job, but if you ended up in the situation that modification is needed, please contact your Triton support team member.

Needless to say that there is no way to impact on your job priority and make sure that it goes to run asap?

Viewing finished jobs

Information about finished and cancelled jobs are available via the `slurm history` command. Most notable pieces are memory use and also exit code, in case the jobs did not finish cleanly.

```
$ slurm history 2hours
JobID JobName Start MaxVMem MaxRes TotalCPU Elapsed
Tasks CPUs Nodes ExitCode State
1052748 helloworld 2012-04-10T19:05 - - 00:00.015 00:00:00
none 1 1 1:0 FAILED
    batch * 2012-04-10T19:05 - - 00:00.015 00:00:00
    1 1 1 1:0 FAILED
1052751 testarr 2012-04-10T19:07 - - 00:00.074 00:02:30
none 5 1 0:0 COMPLETED
    batch * 2012-04-10T19:07 393M 6M 00:00.055 00:02:30
    1 1 1 0:0 COMPLETED
    1052751.0 runtask 2012-04-10T19:07 99M 1M 00:00.002 00:00:30
    1 1 1 0:0 COMPLETED
    1052751.1 runtask 2012-04-10T19:08 99M 1M 00:00.003 00:00:30
    1 1 1 0:0 COMPLETED
    1052751.2 runtask 2012-04-10T19:08 99M 1M 00:00.003 00:00:30
    1 1 1 0:0 COMPLETED
```
Recognized time forms are \( n \) min, \( n \) hours, \( n \) days, \( n \) weeks (without space).

*Elapsed* is the wall clock time from job start to finish.

*MaxVMem* is the highest amount of virtual memory your program allocated during its lifetime. If the slurm job’s memory limit is set below it, your job would be killed.

*MaxRes* is the resident (physical) memory the program actually used of its virtual memory allocation, which may be of interest when monitoring program behavior.

### Cancelling jobs

```
$ scancel # cancel a job
$ scancel `echo {5205484..5205533}` # cancel jobs in the range
$ scancel --state=PENDING --user=$USER --partition=debug # kill all of your pending jobs on debug queue
```

### Job priority

Triton queues are not first-in first-out, but “fairshare”. This means that every person has a priority. The more you run the lower your user priority. As time passes, your user priority increases again. The longer a job waits in the queue, the higher its job priority goes. So, in the long run (if everyone is submitting an never-ending stream of jobs), everyone will get exactly their share.

Once there are priorities, then: jobs are scheduled in order of priority, then any gaps are backfilled with any smaller jobs that can fit in. So small jobs usually get scheduled fast regardless.

*Warning: from this point on, we get more and more technical, if you really want to know the details. Summary at the end.*

What’s a share? Currently shares are based on department and their respective funding of Triton (*sshare*). Shares are shared among everyone in the department, but each person has their own priority. Thus, for medium users, the 2-week usage of the rest of your department can affect how fast your jobs run. However, again, things are balanced per-user within departments. (However, one heavy user in a department can affect all others in that department a bit too much, we are working on this)

Your priority goes down via the “job billing”: roughly time\times power. CPUs are billed at 1/s (but older, less powerful CPUs cost less!). Memory costs .2/GB/s. But: you only get billed for the max of memory or CPU. So if you use one CPU and all the memory (so that no one else can run on it), you get billed for all memory but no CPU. Same for all CPUs and little memory. This encourages balanced use. (this also applies to GPUs).

GPUs also have a billing weight, currently 2/GPU/s for newer models and 1/GPU/s for the older ones.

If you submit a long job but it ends early, you are only billed for the actual time you use (but the longer job might take longer to start at the beginning). Memory is always billed for the full reservation even if you use less, since it isn’t shared.

The “user priority” is actually just a record how much you have consumed lately (the billing numbers above). This number goes down with a half-life decay of 2 weeks. Your personal priority your share compared to that, so we get the effect described above: the more you (or your department) runs lately, the lower your priority.
If you want your stuff to run faster, the best way is to more accurately specify your time (may make that job can find a place sooner) and memory (avoids needlessly wasting your priority).

While your job is pending in the queue SLURM checks those metrics regularly and recalculates job priority constantly. If you are interested in details, take a look at multifactor priority plugin page (general info) and depth-oblivious fair-share factor for what we use specifically (warning: very in depth page). On Triton, you can always see the latest billing weights in /etc/slurm/slurm.conf

Numerically, job priorities range from 0 to 2^32-1. Higher is sooner to run, but really the number doesn’t mean much itself.

These commands can show you information about your user and job priorities:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>slurm s</td>
<td>list of jobs per user with their current priorities</td>
</tr>
<tr>
<td>slurm full</td>
<td>as above but almost all of the job parameters are listed</td>
</tr>
<tr>
<td>slurm shares</td>
<td>displays usage (RawUsage) and current FairShare weights (FairShare, higher is better) values for all users</td>
</tr>
<tr>
<td>slurm j &lt;jobid&gt;</td>
<td>shows &lt;jobid&gt; detailed info including priority, requested nodes etc.</td>
</tr>
</tbody>
</table>

TL;DR: Just select the resources you think you need, and slurm tries to balance things out so everyone gets their share. The best way to maintain high priority is to use resources efficiently so you don’t need to over-request.

**slurm command on Triton**

A nice tool originally developed by Tapio Leipälä specifically for Triton user needs and developed by Triton support team nowadays.

The slurm command can show most often needed information about jobs, resources and the cluster state. It’s a wrapper script to the native SLURM commands. New features are added from time to time. Running it without parameters prints a list of available commands. Most have some soft of shortcuts for convenience.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>slurm q:slurm qq</td>
<td>Status of your queued jobs (long/short)</td>
</tr>
<tr>
<td>slurm partitions</td>
<td>Overview of partitions (A/I/O/T=active,idle,other,total)</td>
</tr>
<tr>
<td>slurm cpus &lt;partition&gt;</td>
<td>list free CPUs in a partition</td>
</tr>
<tr>
<td>slurm history [1day,2hour,...]</td>
<td>Show status of recent jobs</td>
</tr>
<tr>
<td>seff &lt;jobid&gt;</td>
<td>Show percent of mem/CPU used in job</td>
</tr>
<tr>
<td>slurm j &lt;jobid&gt;</td>
<td>Job details (only while running)</td>
</tr>
<tr>
<td>slurm s;slurm ss &lt;partition&gt;</td>
<td>Show status of all jobs</td>
</tr>
<tr>
<td>sacct</td>
<td>Full history information (advanced, needs args)</td>
</tr>
</tbody>
</table>

**Full slurm command help:**

```
$ slurm
```

Show or watch job queue:

- `slurm [watch] queue` show own jobs
- `slurm [watch] q` show user's jobs
- `slurm [watch] quick` show quick overview of own jobs
- `slurm [watch] shorter` sort and compact entire queue by job size
- `slurm [watch] short` sort and compact entire queue by priority
- `slurm [watch] full` show everything
- `slurm [w] [q|qq|ss|sf]` shorthands for above!
slurm qos  show job service classes
slurm top [queue|all]  show summary of active users
Show detailed information about jobs:
slurm prio [all|short]  show priority components
slurm j job  show everything else
slurm steps  show memory usage of running srun job steps
Show usage and fair-share values from accounting database:
slurm h | history  show jobs finished since, e.g. "1day" (default)
slurm shares
Show nodes and resources in the cluster:
slurm p | partitions  all partitions
slurm n | nodes  all cluster nodes
slurm c | cpus  total cpu cores in use
slurm cpus  cores available to partition, allocated and free
slurm cpus jobs  cores/memory reserved by running jobs
slurm cpus queue  cores/memory required by pending jobs
slurm features  List features and GRES

Examples:
  slurm q
  slurm watch shorter
  slurm cpus batch
  slurm history 3hours

Other advanced commands (many require lots of parameters to be useful):

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>squeue</td>
<td>Full info on queues</td>
</tr>
<tr>
<td>sinfo</td>
<td>Advanced info on partitions</td>
</tr>
<tr>
<td>slurm nodes</td>
<td>List all nodes</td>
</tr>
</tbody>
</table>

Native slurm commands

While Triton has a slurm utility that hides most of original SLURM commands, you still may want to learn more. If need something else that slurm can not do, the native commands with their full functionality are at your service. For the details, please consult corresponding man pages (man squeue, etc).

- **squeue** – view information about jobs located in the Slurm scheduling queue
  
  $ squeue -n gpu[1-22]  # reports only jobs allocated to specific nodes
  $ squeue -t PD -i 5 -u $USER  # reports your pending jobs, with the 5s interval

- **sinfo** – view node & partition information

- **sshare** – show statistics from the accounting database

- **scontrol** – various function, end user mostly interested in scontrol show...

  scontrol show node ivb1  # show specific node config

- **sprio** - Show calculated priority factors for jobs waiting in the queue

- **sacct** - Historical info about jobs
Customizable output for \texttt{slurm}

The \texttt{slurm} command output can be customized. Look in the for variable names in /usr/local/bin/slurm and place them into your own $HOME/.config/slurmvars file.

For example, more detailed node info for those interested to know which kind of machines are free. This changes the look of \texttt{slurm} partitions.

\begin{verbatim}
fmt_s_parts="%10P %.10l %.15F %s %N"
\end{verbatim}

\section*{Libraries}

\subsection*{BLAS}

\textbf{Basic Linear Algebra Subroutine (BLAS)} is a \textit{de facto} application programming interface standard for publishing libraries to perform basic linear algebra operations such as vector and matrix multiplication (from wikipedia).

On triton a number of different BLAS implementations are available. The recommended ones are \texttt{MKL} and \texttt{OpenBLAS}. Other available BLAS libraries are ATLAS, GotoBLAS2, ACML, and the Netlib reference BLAS. For benchmark results see \texttt{DGEMM benchmark on triton}

\subsection*{Using MKL}

In order to use the MKL library you need to load the module. MKL is provided both in the “mkl” modules and in the “intel” modules; the “intel” module additonally give you the Intel compilers and debuggers. Linking with MKL is a bit tricky and the exact link options varies from version to version. Intel provides a webpage to build the correct linking options at http://software.intel.com/en-us/articles/intel-mkl-link-line-advisor.

\subsection*{Using OpenBLAS}

OpenBLAS is installed on all the triton nodes on the default library directory (/usr/lib64). 3 different variants of the library are provided:

- Serial version: Link using “-lopenblas”
- OpenMP version: Link using “-lopenblaso”
- pthreads version: Link using “-lopenblasp”

\subsection*{Other BLAS libraries}

In general MKL and OpenBLAS are recommended since they both provide good performance on all the node types we have. Other BLAS libraries have various issues such as crashing when running on the incorrect node (e.g. running an Haswell optimized library on an Westmer node or vice versa), or poor performance. In particular, the Netlib reference BLAS has \textbf{VERY} poor performance and should be avoided at all cost. Use it only for testing or if you need to debug numeric output. As can be seen on the \texttt{DGEMM benchmark results} performance for large matrices is an order of magnitude worse than the optimized versions. For a real example see for instance tracker.triton.aalto.fi/issue194 where 50% performance loss was seen for a complete application.
LAPACK

Linear Algebra Package (LAPACK) is a library of numerical linear algebra algorithms, built on top of BLAS. The recommended LAPACK implementation on triton is MKL. Alternatively, our OpenBLAS modules and libraries also contain the LAPACK library compiled against the OpenBLAS BLAS library.

Scalapack

MKL contains an implementation of ScaLAPACK as well, please try to use that one first. Again, see the BLAS section for how to link with MKL.

Benchmark is done with full Scalapack LIN/EIG testsuite with 24 processors. Scalapack is compiled with -O3 using architecture optimized gotoblas2. Given numbers are WallClockTimes in seconds.

<table>
<thead>
<tr>
<th>Library</th>
<th>Xeon processors</th>
<th>Opteron processors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Xeon-gcc optimized scalapack</td>
<td>342s</td>
<td>-na-</td>
</tr>
<tr>
<td>Opteron-gcc optimized scalapack</td>
<td>338s</td>
<td>291s</td>
</tr>
<tr>
<td>Intel-compiler-mkl</td>
<td>-na-</td>
<td>-na-</td>
</tr>
</tbody>
</table>

Scalapack libs are available under /share/apps/scalapack/2.0.1/

FFT

The FFTW library is available on triton, in several different variants. The recommended one is the one provided by MKL; see the BLAS section above for how to link to it.

<table>
<thead>
<tr>
<th>Library</th>
<th>module</th>
<th>Link line</th>
</tr>
</thead>
<tbody>
<tr>
<td>FFTW 3.2.1</td>
<td>•</td>
<td>-lfftw3 / -lfftw3_threads / -lfftw3f / -lfftw3f_threads / -lfftw3l / -lfftw3l_threads</td>
</tr>
<tr>
<td>FFTW 3.3.2</td>
<td>fftw/3.3.2</td>
<td>-lfftw3 / -lfftw3_mpi</td>
</tr>
</tbody>
</table>

Links

- BLAS at www.netlib.org/blas
- LAPACK at www.netlib.org/lapack
- GotoBLAS FAQ at [www.tacc.utexas.edu/tacc-projects/gotoblas2/faq](http://www.tacc.utexas.edu/tacc-projects/gotoblas2/faq)
- ACML User’s Guide (pdf file) at developer.amd.com/assets

Storage: local drives

See also:

*the storage tutorial.*
Local disks on computing nodes are the preferred place for doing your IO. The general idea is use network storage as a backend and local disk for actual data processing.

- In the beginning of the job cd to /tmp and make a unique directory for your run
- copy needed input from WRKDIR to there
- run your calculation normally forwarding all the output to /tmp
- in the end copy relevant output to WRKDIR for analysis and further usage

Pros
- You get better and steadier IO performance. WRKDIR is shared over all users making per-user performance actually rather poor.
- You save performance for WRKDIR to those who cannot use local disks.
- You get much better performance when using many small files (Lustre works poorly here).
- Saves your quota if your code generate lots of data but finally you need only part of it
- In general, it is an excellent choice for single-node runs (that is all job’s task run on the same node).

Cons
- Not feasible for huge files (>100GB). Use WRKDIR instead.
- Small learning curve (must copy files before and after the job).
- Not feasible for cross-node IO (MPI jobs). Use WRKDIR instead.

How to use local drives on compute nodes

NOT for the long-term data. Cleaned every time your job is finished.

You have to use --gres=spindle to ensure that you get a hard disk (note 2019-january: except GPU nodes).

/tmp is a bind-mounted user specific directory. Directory is per-user (not per-job that is), if you get two jobs running on the same node, you get the same /tmp.

Interactively

How to use /tmp when you login interactively

```bash
$ sinteractive -t 1:00:00           # request a node for one hour
(node)$ mkdir /tmp/$SLURM_JOB_ID    # create a unique directory, here we use
(node)$ cd /tmp/$SLURM_JOB_ID        # create a unique directory, here we use
... do what you wanted ...
(node)$ cp your_files $WRKDIR/my/valuable/data # copy what you need
(node)$ cd; rm -rf /tmp/$SLURM_JOB_ID # clean up after yourself
(node)$ exit
```

In batch script

Batch job example that prevents data lost in case program gets terminated (either because of scancel or due to time limit).
Batch script for thousands input/output files

If your job requires a large amount of files as input/output using tar utility can greatly reduce the load on the $WRKDIR-filesystem.

Using methods like this is recommended if you’re working with thousands of files.

Working with tar balls is done in a following fashion:

1. Determine if your input data can be collected into analysis-sized chunks that can be (if possible) re-used
2. Make a tar ball out of the input data (tar cf <tar filename>.tar <input files>)
3. At the beginning of job copy the tar ball into /tmp and untar it there (tar xf <tar filename>.tar)
4. Do the analysis here, in the local disk
5. If output is a large amount of files, tar them and copy them out. Otherwise write output to $WRKDIR

A sample code is below:

```bash
#!/bin/bash

#SBATCH --time=0-12:00:00 --mem-per-cpu=2000 # time and memory requirements
mkdir /tmp/$SLURM_JOB_ID # get a directory where you will send all output from your program
cd /tmp/$SLURM_JOB_ID

## set the trap: when killed or exits abnormally you get the output copied to $WRKDIR/$SLURM_JOB_ID anyway
trap "rm -rf /tmp/$SLURM_JOB_ID; exit" TERM EXIT

## run the program and redirect all IO to a local drive assuming that you have your program and input at $WRKDIR
srun $WRKDIR/my_program $WRKDIR/input > output

mv /tmp/$SLURM_JOB_ID/output $WRKDIR/SOMEDIR # move your output

Batch script for thousands input/output files

If your job requires a large amount of files as input/output using tar utility can greatly reduce the load on the $WRKDIR-filesystem.

Using methods like this is recommended if you’re working with thousands of files.

Working with tar balls is done in a following fashion:

1. Determine if your input data can be collected into analysis-sized chunks that can be (if possible) re-used
2. Make a tar ball out of the input data (tar cf <tar filename>.tar <input files>)
3. At the beginning of job copy the tar ball into /tmp and untar it there (tar xf <tar filename>.tar)
4. Do the analysis here, in the local disk
5. If output is a large amount of files, tar them and copy them out. Otherwise write output to $WRKDIR

A sample code is below:

```bash
#!/bin/bash

#SBATCH --time=0-12:00:00 --mem-per-cpu=2000 # time and memory requirements
mkdir /tmp/$SLURM_JOB_ID # get a directory where you will send all output from your program
cd /tmp/$SLURM_JOB_ID

## set the trap: when killed or exits abnormally you get the output copied to $WRKDIR/$SLURM_JOB_ID anyway
trap "rm -rf /tmp/$SLURM_JOB_ID; mv -f /tmp/$SLURM_JOB_ID $WRKDIR/$SLURM_JOB_ID; exit" TERM EXIT

## run the program and redirect all IO to a local drive assuming that you have your program and input at $WRKDIR
srun $WRKDIR/my_program $WRKDIR/input > output

mv /tmp/$SLURM_JOB_ID/output $WRKDIR/SOMEDIR # move your output
```
Storage: Lustre (scratch)

See also:
the storage tutorial.

Lustre is scalable high performance file system created for HPC. It allows MPI-IO but mainly it provides large storage capacity and high sequential throughput for cluster applications. Currently the total capacity is 2PB. The basic idea in Lustre is to spread data in each file over multiple storage servers. With large (larger than 1GB) files Lustre will significantly boost the performance.

Working with small files

As Lustre is meant for large files, the performance with small (smaller than 10MB) files will not be optimal. If possible, try to avoid working with multiple small files.

Note: Triton has a default stripe of 1 already, so it is by default optimized for small files (but it’s still not that great). If you use large files, see below.

If small files are needed (i.e. source codes) you can tell Lustre not to spread data over all the nodes. This will help in performance.

To see the striping for any given file or directory you can use following command to check status

```
lfs getstripe -d /scratch/path/to/dir
```

You can not change the striping of an existing file, but you can change the striping of new files created in a directory, then copy the file to a new name in that directory.

```
lfs setstripe -c 1 /scratch/path/to/dir
cp somefile /scratch/path/to/dir/newfile
```

Working with lots of small files

Large datasets which consist mostly of small (<1MB) files can be slow to process because of network overhead associated with individual files. If it is your case, please consult Compute node local drives page, see the tar example over there or find some other way to compact your files together into one.

Working with large files

By default Lustre on Triton is configured to stripe a single file over a single OST. This provides the best performance for small files, serial programs, parallel programs where only one process is doing I/O, and parallel programs using a file-per-process file I/O pattern. However, when working with large files (>> 10 GB), particularly if they are accessed in parallel from multiple processes in a parallel application, it can be advantageous to stripe over several OST’s. In this case the easiest way is to create a directory for the large file(s), and set the striping parameters for any files subsequently created in that directory:

```
cd $WRKDIR
mkdir large_file
lfs setstripe -c 4 large_file
```

The above creates a directory large_file and specifies that files created inside that directory will be striped over 4 OST’s. For really large files (hundreds of GB’s) accessed in parallel from very large MPI runs, set the stripe count to “-1” which tells the system to stripe over all the available OST’s.
To reset back to the default settings, run

```
lfs setstripe -d path/to/directory
```

**Lustre: common recommendations**

- Minimize use of `ls -l` and `ls --color` when possible

Several excellent recommendations are at

- [http://www.nics.tennessee.edu/computing-resources/file-systems/io-lustre-tips](http://www.nics.tennessee.edu/computing-resources/file-systems/io-lustre-tips)

They are fully applicable to our case.

Be aware, that being a high performance filesystem Lustre still has its own bottlenecks, and even non-proper a usage by a single user can get whole system in stuck. See the recommendations at the link above how to avoid those potential situations. Common Lustre troublemakers are `ls -lR`, creating many small files, `rm -rf`, small random i/o, heavy bulk i/o.

For advanced user, these slides can be interesting: [http://www.eofs.eu/fileadmin/1ad2012/06_Daniel_Kobras_S_C_Lustre_FS_Bottleneck.pdf](http://www.eofs.eu/fileadmin/1ad2012/06_Daniel_Kobras_S_C_Lustre_FS_Bottleneck.pdf)

**MPI on Triton**

The basic parallel programming model for PC-clusters is message passing. MPI (Message Passing Interface) is a library specification for message-passing, proposed, nowadays, as a standard. OpenMPI and MVAPICH are two different (but doing the same) MPI implementations, both are installed on triton. MVAPICH is the recommended one. MPI is suitable for both distributed memory computers (thus through interconnect between nodes) and shared memory architectures (computing within one node).

Check out the latest version of MVAPICH / OpenMPI with

```
$ module avail
```

By setting up your environment with `module`, you will set all the variables like `$MPIRUN`, `$MPIHOME` etc.

Load MVAPICH2:

```
$ module load mvapich2/1.8a2-gcc-4.4.6
```

Load OpenMPI (for software compiled on recently updated SL6.2):

```
$ module load openmpi/1.4.5-gcc-4.4.6
```

Load OpenMPI (for software compiled on SL6.1):

```
$ module load compat-openmpi-x86_64
```

Compiling after that is quite straight forward:

```
$ mpif90 your_code.f90 -o your_code
$ mpicc your_code.c -o your code
```

Parallel programs are to be run as batch jobs, even test runs. See Executing jobs / Batch system for examples of running in parallel.
You have code, you want it to run fast. This is what Triton is for. But how do you know if your code is running as fast as it can? We are scientists, and if things aren’t quantified we can’t do science on them. Programming can often seem like a black box: modern computers are extremely complicated, and people can’t predict what is actually making code fast or slow anymore. Thus, you need to profile your code: get detailed performance measurements. These measurements let you know how to make it run faster.

There are many tools for profiling, and it really is one of the fundamental principles for any programming language. You really should learn how to do quick profile just to make sure things are OK, even if you aren’t trying to optimize things: you might find a quick win even if you didn’t write the code yourself (for example, 90% of your time is spent on input/output).

This page is under development, but so far serves as an introduction. We hope to expand it with specific Triton examples.

Summary: profiling on Linux

First off, look at your language-specific profiling tools.

- Generic Linux profiling tools (big and comprehensive list, also some presentations): http://www.brendangregg.com/linuxperf.html
- Profiling in C and Python (introduction + examples): http://rkd.zgib.net/scicomp/profiling/profiling.html

CPU profiling

This can give you a list of where all your processor time is going, either per-function or per-line. Generally, most of your time is in a very small region of your code, and you need to know what this is in order to improve just that part.

See the C and Python profiling example above.

GNU gprof

gprof is a profiler based on instrumenting your code (build with -pg). It has relatively high overhead, but gives exact information e.g. for the number of times a function is called.

Perf

perf is a sampling profiler, which periodically samples events originating e.g. from the CPU performance monitoring unit (PMU). This generates a statistical profile, but the advantage is that the overhead is very low (single digit %),
and one can get timings at the level of individual asm instructions. For a simple example, consider a (naive) matrix multiplication program:

Compile the program (-g provides debug symbols which will be useful later on, at no performance cost):

```
$ gfortran -Wall -g -O3 mymatmul.f90
```

Run the program via the profiler to generate profile data:

```
$ perf record ./a.out
```

Now we can look at the profile:

```
$ perf report
# Samples: 1251
# # Overhead Command Shared Object Symbol
# ........ .............. ............................. ......
# 85.45% a.out ./a.out [.] MAIN\_\_
4.24% a.out /usr/lib/libgfortran.so.3.0.0 [.] \_gfortran\_arandom\_r4
3.12% a.out /usr/lib/libgfortran.so.3.0.0 [.] kiss\_random\_kernel
```

So 85% of the runtime is spent in the main program (symbol MAIN\_), and most of the rest is in the random number generator, which the program calls in order to generate the input matrices.

Now, let’s take a closer look at the main program:

```
$ perf annotate MAIN__
---
Percent \| Source code & Disassembly of a.out
---

```

... 

Unsurprisingly, the inner loop kernel takes up practically all the time.

For more information on using perf, see the perf tutorial at

https://perf.wiki.kernel.org/index.php/Tutorial
Input/output profiling

This will tell you how much time is spent reading and writing data, where, and what type of patterns it has (big reads, random access, etc). Note that you can see the time information when CPU profiling: if input/output functions take a lot of time, you need to improve IO.

/usr/bin/time -v prints some useful info about IO operations and statistics.

Lowest level: use strace to print the time taken in every system call that accesses files. This is not that great:

```bash
# Use strace to print the total bytes
strace -e trace=desc $command |& egrep 'write' | awk --field-separator='=' '{ x+=$NF } END { print x }'

strace -e trace=desc $command |& egrep 'read' | awk --field-separator='=' '{ x+=$NF } END { print x }'

# Number of calls only
strace -e trace=file -c $command
```

Memory profiling

Less common, but it can tell you something about what memory is being used.

If you are making your own algorithms, memory profiling becomes more important because you need to be sure that you are using the memory hierarchy efficiently. There are tools for this.

MPI and parallel profiling

mpiP

mpiP: Lightweight, Scalable MPI Profiling http://mpip.sourceforge.net/. Collects statistical information about MPI functions. mpiP is a link-time library, that means that it can be linked to the object file, though it is recommended that you have recompiled the code with -g. Debugging information is used to decode the program counters to a source code filename and line number automatically. mpiP will work without -g, but mileage may vary.

Usage example:

```bash
# assume you have you MPI flavor module loaded
module load mpip/3.4.1

# link or compile your code from scratch with -g
mpif90 -g -o my_app my_app.f90 -lmpip -lm -lblfd -liberty -lunwind
# or
mpif90 -o my_app my_app.o -lmpip -lm -lblfd -liberty -lunwind

# run the code normally (either interactively with salloc or as usual with sbatch)
salloc -p play -n 4 srun mpi_app
```

If everything works, you will see the mpiP header preceding your program stdout, and there will be generated a text report file in your work directory. File is small, no worries about quota. Please, consult the link above for the file content explanation. During runtime, one can set MPIP environment variables to change the profiler behavior.

Example:

```bash
export MPIP="-t 10.0 -k 2"
```
Scalasca

Available through module load scalasca

Quotas

Triton has quotas which limit both the space usage and number of files. The quota for your home directory is 10GB, for $WRKDIR by default is 200GB, and project directories depending on request. These quotas exist to avoid usage exploding without anyone noticing. If you ever need more space, just ask. We’ll either give you more or find a solution for you.

The file quota is because scratch is not that great for too many small files. If you have too many small files, see the page on small files.

Normally, things just work, but there are certain intrinsic problems in scratch, so if you ever get a “disk quota exceeded” error, then read on.

Note:

To try a quick fix, you can: quotafix -gs --fix /path/to/the/directory
If that fixes something, and problem recurs, then: module load teflon

How quotas work

There are both quotas for users and projects (/m/$dept/$project). However, Lustre (scratch) can currently only do quotas by user or group, not by file path. If you $ ls -l a file or directory, you see both user and group. Unfortunately, with lustre it only really works for one of them at a time. So, on Triton, we use user private groups: everyone has a group with the same name as your user, and in $WRKDIR all files should have your group, and in project directories the group of that project. We have things set up so that things will Just Work if you do normal things.

$ ls -l test
  drwxrwsr-x 3 darstr1 darstr1 4096 Jan 25 15:13 test/
  ^ ^^^^^^^ ^^^^^^^
  | ^-- user ^-- group
  ^-- SETGID

Important! If a file has a group of domain users or triton-users, which occurs by default, then there is no quota for the files! To get around this, we have all directories “SETGID” (chmod g+s) and then files automatically are made in the correct group. That leads to the next point...

Disk quota exceeded error but I have plenty of space!

If the quota command says you have plenty of space AND sufficient number of files, then you’ve hit a common problem. Probably, the directory does not have the SETGID bit set, so when you try to make a new file, it appears as group ‘domain users’, and there’s no quota assigned, so it fails!

Quick fixes

Run either quotafix which will try to do things automatically, or this find command. You can only fix $WRKDIR on Triton, since the user-private-group does not exist on Aalto Linux workstations.
# AUTOMATIC ON TRITON: Fix everything.
# (only for $WRKDIR or group directories, still in testing):
/share/apps/bin/quotafix -sg --fix /path/to/dir/

# MANUAL ON TRITON: use find yourself.
# $GROUP is your username for work, or project-group name for scratch.
lfs find /path/to/dir -type d -print0 | xargs -0 chmod g+s
lfs find /path/to/dir ! -group $GROUP -print0 | xargs -0 chgrp $GROUP

# AALTO WORKSTATIONS: use "find" instead of "lfs find" above.

Details

Check the SETGID bit and group ownership for directories: drwxrwsr-x. Directories must have “s” there and the right group, otherwise when you try to make new files in that directory, they are group= ‘domain users’ and it fails.

I can’t rsync/sftp/etc

It is related to the above mentioned issue, something like rsync -a ... or cp -p ... are trying to save original group ownership attribute, which will not work. Try this instead:

```bash
## mainly one should avoid -g (as well as -a) since it preserves the old group (with...
$ rsync -urlptDxv --chmod=Dg+s somefile triton.aalto.fi:/path/to/work/directory

## avoid '-p' with cp, or if you want to keep timestamps, mode etc, then use '--...
$ cp -r --preserve=mode,timestamps somefile /path/to/mounted/triton/work/directory
```

You may need similar things for other different programs.

Details: Some programs change the group or don’t preserve the SETGID bit. This especially happens when you try to copy a directory from somewhere else to Triton while preserving the SETGID bit. You get a directory in the wrong group, or directory without SETGID bit so new files are in the wrong group, so no quota.

Other solutions

teflon

This is a new hack we are working on and hasn’t been extensively tested. Teflon is “anti-SETGID” which stops any program from changing either the group or SETGID bit, using LD_PRELOAD magic. It should work with any program, currently probably only 64-bit though. This is still under development. Please report problems or success stories.

You have to run quotafix or chmod/chgrp commands above first.

```bash
# Use via a module - applies to everything in this session.
module load teflon

# OR: Run a single program under teflon
```

(continues on next page)
Singularity Containers

For more information see: https://www.sylabs.io/docs/

Basic Idea

The basic idea behind Singularity containers is that software is packaged into a container (basically an entire self-contained operating system!) that is based on a Docker image that can then be run by the user. This allows hard to install software to be easily packaged and used - because you are packaging the entire OS!

During runtime, the root file system / is changed to the one inside the image and file systems are brought into the container through bind mounts. This sounds complicated, but in practice this is easy due to singularity_wrapper written for Triton.

Basic Usage

While the image itself is read-only, remember that /home, /m, /scratch and /l etc. are not. If you edit/remove files in these locations within the image, that will happen outside the image as well.

On Triton, you just need to load the proper module. This will set some environment variables and enable the use of singularity_wrapper.

singularity_wrapper

singularity_wrapper is written so that when you load a module written for a singularity image, all the important options are already handled for you. It has three basic commands:

1. singularity_wrapper shell <shell> - Gives user a shell within the image (specify <shell> to say which shell you want).
2. singularity_wrapper exec <cmd> - Executes a program within the image.
3. `singularity_wrapper run <parameters>` - Runs the singularity image. What this means depends on the image in question - each image will define a “run command” which does something. If you don’t know what this is, use the first two instead.

Under the hood, `singularity_wrapper` does this:

1. Choosing appropriate image based on module version
2. Binding of basic paths (`-B /l:/l, /m:/m, /scratch:/scratch`)
3. Loading of system libraries within images (if needed) (e.g. `-B /lib64/nvidia:/opt/nvidia`)
4. Setting working directory within image (if needed)

**Power usage**

These are the “raw” singularity commands. If you use these, you have to configure the images and bind mounts yourself (which is done automatically by `singularity_wrapper`). If you run `module show` the module you can get hints about what happens.

Singularity enables three base commands to user:

1. `singularity shell <image>` - Gives user a shell within the image (see `singularity shell --help` for more information on flags etc.)
2. `singularity exec <image> <cmd>` - Executes a program within the image (see `singularity exec --help` for more information on flags etc.)
3. `singularity run <image> <parameters>` - Runs the singularity image. What this means depends on the image in question. (see `singularity run --help` for more information on flags etc.)

**Creating your own Singularity images to run in Triton**

All images used in Triton are built from Docker images stored in our private Docker registry in `registry.cs.aalto.fi`. They build automatically from Docker pushes using our continuous integration builder. If you want to build your own Singularity image to Triton, we can create a user for you to the registry and add your image to the automatic build.

Even though the system is in production it is still being tested.
Thus there might be changes in the future.

Steps to get your images building are outlined below. You’ll need to do steps 1 to 3 only once.

**Step 1: Log in to registry.cs.aalto.fi**

Go to `registry.cs.aalto.fi` and click Gitlab under Social logins. This will redirect you to a Gitlab page that you can use for authentication. In this page use your Aalto username and password to login.

In the future we’ll improve the authentication page.

**Step 2: Create an application token**

For added security you cannot use your main password for `docker login`. By clicking on your username, you’ll get to your user settings. From there, do the following:

1. Click Create new token in the Application tokens-section.
Aalto scientific computing guide

2. Choose name for the token and click create.
3. Copy the application token that is visible on the right side of your screen.

Step 3: Docker login

On your own workstation run:

```
docker login registry.cs.aalto.fi
```

Your username is same as your Aalto username. As a password give the application token you created in step 2.

Step 4: Push your images to registry

If you have an existing image in Dockerhub, you can run:

```
docker pull <dockerhub user>/<image>:<tag>
docker tag <dockerhub user>/<image>:<tag> registry.cs.aalto.fi/<your username>/<image>:

docker push registry.cs.aalto.fi/<your username>/<image>:<tag>
```

For example:

```
docker pull library/ubuntu:latest

docker tag library/ubuntu:latest registry.cs.aalto.fi/$USER/ubuntu:latest

docker push registry.cs.aalto.fi/$USER/ubuntu:latest
```

If you are building your image from Dockerfile, you can run:

```
docker build -it registry.cs.aalto.fi/$USER/my_image:latest /path/to/my/dockerfile

docker push registry.cs.aalto.fi/$USER/my_image:latest
```

Step 5: Let us know what image you want to have in Triton

**Warning:** Do note that images built to Triton are visible to all users. Do not include sensitive code/data in the docker images. You should retrieve such data from your project/work folder during job runtime.

We need the following information for the automatic build:

- What is the Docker url of the image (e.g. `registry.cs.aalto.fi/$USER/my_image`)?
- What tags do you want built (we recommend you use `latest` and `dev`)?
- Does the image utilize GPUs?

After that we’ll set up the automated build. Every time you push a newer version of said `image:tag` the image will update in Triton if the build was successful.

After the build has been done you can load up your new image in Triton with:

```
module use /share/apps/singularity-ci/centos/modules/$USER
module load my_image:latest
```

and launch the programs within using the `singularity_wrapper exec`. 

192 Chapter 6. Triton
Small files

Millions of small files are a huge problem on any filesystem. You may think /scratch, being a fast filesystem, doesn’t have this problem, but it’s actually worse here. Lustre (scratch) as like an object store, and stores files separately from metadata. This means that each file access requires multiple different network requests, and making a lot of files brings your research (and managing the cluster) to a halt. What counts as a lot? Your default quota is 1e6 files. 1e4 for a project is not a lot. 1e6 for a single project is.

You may have been directed here because you have a lot of files. In that case, welcome to the world of big data, even if your total size isn’t that much! (it’s not just size, but difficulty of handling using normal tools) Please read this and see what you can learn, and ask us if you need help.

This page is mostly done, but specific examples could be expanded.

See also:

• Data storage on the Lustre file system, especially the bottom.

• Compute node local drives

Contents

The problem with small files

You know Lustre is high performance and fast. But, there is a relatively high overhead for accessing each file. Below, you can see some sample transfer rates, and you can see that total performance drops drastically when files get small. (These numbers were for the pre-2016 Lustre system, it’s better now but the same principle applies.) This isn’t just a problem when you are trying to read files, it’s also a problem when managing, moving, migrating, etc.

<table>
<thead>
<tr>
<th>File size</th>
<th>Net transfer rate, many files of this size</th>
</tr>
</thead>
<tbody>
<tr>
<td>10GB</td>
<td>1100 MB/s</td>
</tr>
<tr>
<td>100MB</td>
<td>990 MB/s</td>
</tr>
<tr>
<td>1MB</td>
<td>90 MB/s</td>
</tr>
<tr>
<td>10KB</td>
<td>.9 MB/s</td>
</tr>
<tr>
<td>512B</td>
<td>.04 MB/s</td>
</tr>
</tbody>
</table>

Why do people make millions of small files?

We understand there reasons people make lots of files: it’s convenient. Here are some of the common problems (and alternative solutions) people may be trying to solve with lots of files.

• Flat files are universal format. If you have everything in its own file, then any other program can look at any data individually. It’s convenient. This is a fast way to get started and use things.

• Compatibility with other programs. Same as above.

• Ability to use standard unix shell tools. Maybe your whole preprocessing pipeline is putting each piece of data in its own file and running different standard programs on it. It’s the Unix way, after all. Using filesystem as your index. Let’s say you have a program that reads/writes data which is selected by different keys. It needs to locate the data for each key separately. It’s convenient to put all of these in their own files: this takes the role of a database index, and you simply open the file with the name of the key you need. But the filesystem is not a good index.

  – Once you get too many files, a database is the right tool for the job. There are databases which operate as single files, so it’s actually very easy.
• Concurrency: you use filesystem as the concurrency layer. You submit a bunch of jobs, each job writes data to its own file. Thus, you don’t have to worry about problems with appending to the same file/database synchronization/locking/etc. This is actually a very common reason.

  – This is a big one. The filesystem is the most reliable way to join the output of different jobs (for example an array job), and it’s hard to find a better strategy. It’s reasonable to keep doing this, and combine job outputs in a second stage to reduce the number of files

• Safety/security: the filesystem isolates different files from each other, so if you modify one, there’s less chance of corrupting any other ones. This goes right along with the reason above.

• You only access a few files at a time in your day to day work, so you never realize there’s a problem. However, when we try to manage data (migrate, move, etc), then a problem comes up.

• Realize that forking processes has similar overhead. Small reads are also non-ideal, but less bad(?).

Strategies

• Realize you will have to have to change you workflow. You can’t do everything with grep, sort, wc, etc. anymore. Congratulations, you have big data.

• Consider right strategy for your program: a serious program should provide options for this.

  – For example, I’ve seen some machine learning frameworks which provide an option to compress all the input data into a single file that is optimized for reading. This is precisely designed for this type of case. You could read all the files individually, but it’ll be slower. So in this case, one should first read the documentation and see there’s a solution. One would take all the original files and make the processed input files. Then, take the original training data, package it together in one compressed archive for long-term storage. If you need to look at individual input files, you can always decompress one by one.

• Split - combine - analyze

  – Continue like you have been doing: each (array?) job makes different output files. Then, after running, combine the outputs into one file/database. Clean up/archive the intermediate files. Use this combined DB/file to analyze the data in the long term. This is perhaps the easiest way to adapt your workflow.

• HDF5: especially for numerical data, this is a good format for combining your results. It is like a filesystem within a file, you can still name your data based on different keys for individual access.

• Unpack to local disk, pack to scratch when done.

  – Main article: Compute node local drives

  – This strategy can be combined with many of the other strategies below

  – This strategy is especially good when your data is write-once-read-many. You package all of your original data into one convenient archive, and unpack it to the local disk when you need it. You delete it when you are done.

• Use a proper database suitable for your domain (sqlite): Storing lots of small data where anything can be quickly findable and you can do computation efficiently is exactly what databases do. It can be difficult to have a general purpose database work for you, but there are a wide variety of special-purposes databases these days. Could one of them be suitable for storing the results of your computation for analysis?

  – Note that if you are really doing high-performance random IO, putting a database on scratch is not a good idea, and you need to think more.

  – Consider combining this with local disk: You can copy your pre-created database file to local disk and do all the random access you need. Delete when done. You can do modification/changes directly on scratch if you want.
• **key-value stores**: A string key stores arbitrary data.
  
  – This is a more general database, basically. It stores arbitrary data for a certain key.

• **Read all data to memory.**
  
  – A strategy for using many files. Combine all data into one file, read them all into memory, then do the random access in memory.

• **Compress them down when done.**
  
  – It’s pretty obvious: when you are done with files, compress all of them into one. You have the archive and can always unpack when needed. You should especially at least do this when you are done with a project: if everyone did this, the biggest problems could be solved.

• **Make sure you have proper backups for large files, mutating files introduces risks!**
  
  – If you do go using these strategies, make sure you don’t accidentally lose something you need. Have backups (even if it’s on scratch: backup your database files)

• If you do have to keep many small files, check the link above for lustre performance tuning.
  
  – *Data storage on the Lustre file system*

• If you have other programs that can only operate on separate files
  
  – This is a tough situation, investigate what you can do combining the strategies above. At least you can pack up when done, and possibly copying to local disk while you are accessing is a good idea.

• **MPI-I/O**: if you are writing your own MPI programs, this can parallelize output

**Specific example: HDF5 for numerical data, or some database**

HDF5 is essentially a database for numerical data. You open a HDF5 file and access different data by path - the path is like a filename. There are libraries for accessing this data from all relevant programming languages.

If you have some other data that is structured, there are other databases that will work. For example, sqlite is a single-file, serverless database for relational data, and there are other similar things for time serieses or graphs.

**Specific example: Unpacking to local disk**

You can see examples at *compute node local drives*

**Specific example: Key-value stores**

Let’s say you have written all your own code and want an alternative to files. Instead, use a key-value database. You open one file, and store your file contents under different keys. When you need the data out, you request it by that key again. The keys take the place of filenames. Anytime you would open files, you just access from these key-value stores. You also have ways of dumping and restoring the data if you need to analyze it from different programs.

**Performance tuning for small files**

See here: *Data storage on the Lustre file system*
Storage

See also:

The storage tutorial is a prerequisite.

These pages are also related and include solutions to common storage problems:

- Storage: Lustre (scratch)
- Storage: local drives
- Quotas
- Small files

This pages gives an overview of more advanced storage topics. You should read the storage tutorial first.

Checklist

Do any of these apply to you? If so, consider your situation and ask us for help!

If you have been sent this checklist because your jobs may be having a lot of IO, don’t worry. It’s not necessarily a problem but please go through this checklist and let us know what applies to you so we can give some recommendations.

- Many small files being accessed in jobs (hundred or more).
- Files with extremely random access, in particular databases or database-like things (hdf5).
- Files being read over and over again. Alternatives: copy to local disks, or read once and store in memory.
- Number of files growing, for example all your runs have separate input files, output files, Slurm output files, and you have many runs.
- Constantly logging to certain files, writing to files from many parallel jobs at the same time.
- Reading from single files from many parallel jobs or threads at the same time.
- Is all your IO concatenated at one point, or spread out over the whole job?

( and if we’ve asked you specifically about your jobs, could you also describe what kind of job it is, the type of disk read and write happens, and in what kinds of pattern? Many small files, a few large ones, reading same files over and over, etc. How’s it spread out across jobs? )

If you think your IO may have bad patterns or even you just want to talk to make sure, let one of the Triton staff know or submit an issue in the issue tracker.

Checking your stats

You can check the total disk read and write of your past jobs using:

```
# All your recent jobs:
sacct -o jobid%10,user%8,jobname%10,NodeList,MaxDiskRead,MaxDiskWrite -u $USER
# A single jobid
sacct -o jobid%10,user%8,jobname%10,NodeList,MaxDiskRead,MaxDiskWrite -j $jobid
```

(we will add more tools to this later)
**Loading data for machine learning**

As we’ve said before, modern GPUs are super data-hungry when used for machine learning. If you try to open many files to feed it the data, “you’re going to have a bad time”. Luckily, different packages have different solutions to the problem.

In general, at least try to combine all of your input data into some sort of single file that can be read in sequence.

Try to do the least amount of work possible in the core training loops: any CPU usage, print, logging, preprocessing, postprocessing, etc. reduces the amount of time the GPU is working unless you do it properly (Amdhal’s law).

- Tensorflow: data input pipelines

(more coming later)

**Compilers and toolchains**

**Individual compilers**

Some of our compilers are available as individual modules. If you simply want a compiler, we recommend you use these modules.

<table>
<thead>
<tr>
<th>Compiler</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>gcc</td>
<td>5.5.0</td>
</tr>
<tr>
<td>gcc</td>
<td>6.3.0</td>
</tr>
<tr>
<td>llvm</td>
<td>6.0.1-python2</td>
</tr>
<tr>
<td>llvm</td>
<td>6.0.1-python3</td>
</tr>
</tbody>
</table>

If you need libraries or tools (e.g. openmpi or cmake) to compile your software, use modules with lower-case names. These modules are individual modules that do not have extra toolchain requirements.

**Toolchains**

Some modules in Triton are organized in so-called toolchains. These are collections of compilers and tools that are used for compiling specialized software.

Typically a toolchain contains a compiler and a MPI implementation, but it can also contain additional mathematical and computational libraries.

Naming convention is from EasyBuild that is used to administer the software collections. It goes like:

<compiler><mpi><blas><lapack><fftw><cuda>

eg. GCC,OpenMPI,OpenBLAS,LAPACK,FFTW,CUDA would result in toolchain **goolfc**
### Toolchains in detail

<table>
<thead>
<tr>
<th>Toolchain</th>
<th>Compiler version</th>
<th>MPI version</th>
<th>BLAS version</th>
<th>ScaLAPACK version</th>
<th>FFTW version</th>
<th>CUDA version</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>GOOLF Toolchains:</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>goolf/triton-2016a</td>
<td>GCC/4.9.3</td>
<td>OpenMPI/1.10</td>
<td>2OpenBLAS/0.2</td>
<td>ScaLAPACK/2</td>
<td>FFTW/3.3.4</td>
<td>•</td>
</tr>
<tr>
<td>goolf/triton-2016b</td>
<td>GCC/5.4.0</td>
<td>OpenMPI/1.10</td>
<td>3OpenBLAS/0.2</td>
<td>ScaLAPACK/2</td>
<td>FFTW/3.3.4</td>
<td>•</td>
</tr>
<tr>
<td>goolf/triton-2016a</td>
<td>GCC/4.9.3</td>
<td>OpenMPI/1.10</td>
<td>2OpenBLAS/0.2</td>
<td>ScaLAPACK/2</td>
<td>FFTW/3.3.4</td>
<td>7.5.18</td>
</tr>
<tr>
<td>goolf/triton-2016a</td>
<td>GCC/5.4.0</td>
<td>OpenMPI/2.0.1</td>
<td>OpenBLAS/0.2</td>
<td>ScaLAPACK/2</td>
<td>FFTW/3.3.4</td>
<td>8.0.61</td>
</tr>
<tr>
<td><strong>GMPOLF Toolchains:</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>gmpolf/triton-2016a</td>
<td>GCC/4.9.3</td>
<td>MPICH/3.0.4</td>
<td>OpenBLAS/0.2</td>
<td>ScaLAPACK/2</td>
<td>FFTW/3.3.4</td>
<td>•</td>
</tr>
<tr>
<td>gmpolf/triton-2016a</td>
<td>GCC/4.9.3</td>
<td>MPICH/3.0.4</td>
<td>OpenBLAS/0.2</td>
<td>ScaLAPACK/2</td>
<td>FFTW/3.3.4</td>
<td>7.5.18</td>
</tr>
<tr>
<td><strong>GMVOLF Toolchains:</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>gmvolf/triton-2016a</td>
<td>GCC/4.9.3</td>
<td>MVAPICH2/2.0</td>
<td>OpenBLAS/0.2</td>
<td>ScaLAPACK/2</td>
<td>FFTW/3.3.4</td>
<td>•</td>
</tr>
<tr>
<td>gmvolf/triton-2016a</td>
<td>GCC/4.9.3</td>
<td>MVAPICH2/2.0</td>
<td>OpenBLAS/0.2</td>
<td>ScaLAPACK/2</td>
<td>FFTW/3.3.4</td>
<td>7.5.18</td>
</tr>
<tr>
<td><strong>IOOLF Toolchains:</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ioolf/triton-2016a</td>
<td>icc/2015.3.187</td>
<td>OpenMPI/1.10</td>
<td>2OpenBLAS/0.2</td>
<td>ScaLAPACK/2</td>
<td>FFTW/3.3.4</td>
<td>•</td>
</tr>
<tr>
<td><strong>IOMKLL Toolchains:</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>iomkl/triton-2016a</td>
<td>icc/2015.3.187</td>
<td>OpenMPI/1.10</td>
<td>2imkl/11.3.1.150</td>
<td>imkl/11.3.1.150</td>
<td>imkl/11.3.1.150</td>
<td>•</td>
</tr>
<tr>
<td>iomkl/triton-2016b</td>
<td>icc/2015.3.187</td>
<td>OpenMPI/1.10</td>
<td>3imkl/11.3.1.150</td>
<td>imkl/11.3.1.150</td>
<td>imkl/11.3.1.150</td>
<td>•</td>
</tr>
</tbody>
</table>

Some of our software is compiled against these toolchains and we update them to newer versions if needed. If you require older versions of e.g. GCC we will install them as individual modules.

When asking for specialized software, these will be used as the starting point. E.g. Armadillo/6.700.3-goolf-triton-2016a-Python-2.7.11 uses goolf-triton-2016a as the base.

### 6.1.5 Applications

See our *general information* and the full list below:
Applications: General info

See also:

Intro tutorial: Applications (this is assumed knowledge for all software instructions)

When you need software, check the following for instructions (roughly in this order):

• This page.
• Search the SciComp site using the search function.
• Check module spider and module avail to see if something is available but undocumented.
• The issue tracker for other people who have asked - some instructions only live there.

Compilers

See Compilers and toolchains.

Modules

See Software Modules. Modules are the standard way of loading software.

Singularity

See Singularity Containers. Singularity are software containers that provide an operating system within an operating system. Software will tell you if you need to use it via Singularity.

Software installation and policy

We want to support all software, but unfortunately time is limited. In the chart below, we have these categories (which don’t really mean anything, but in the future should help us be more transparent about what we are able to support):

• A: Full support and documentation, should always work
• B: We install and provide best-effort documentation, but may be out of date.
• C: Basic info, no guarantees

If you know some application which is missing from this list but is widely in use (anyone else than you is using it) it would make sense install to /share/apps/ directory and create a module file. Send your request to the tracker. We want to support as much software as possible, but unfortunately we don’t have the resources to do everything centrally.

Software is generally easy to install if it is in Easybuild (use the find file feature to see if an easyconfig is there) or Spack (check that package list page). If it has easy-to-install Ubuntu packages, it will be easy to do via singularity.

Software list

<table>
<thead>
<tr>
<th>Name</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Python</td>
<td>A</td>
</tr>
</tbody>
</table>
FHI-aims

FHI-aims (Fritz Haber Institute ab initio molecular simulations package) is an electronic structure theory code package for computational molecular and materials science. FHI-aims density functional theory and many-body perturbation calculations at all-electron, full-potential level.

FHI-aims is licensed software with voluntary payment for an academic license. While the license grants access to the FHI-aims source code each holder of a license can use pre-built binaries available on Triton. To this end, contact Ville Havu at the PHYS department after obtaining the license.

On Triton the most recent version of FHI-aims is available via the module FHI-aims/latest-iomkl-triton-2017a. It is compiled using the Intel compiler suite and the MKL library in the toolchain iomkl/triton-2017a. The MPI environment is OpenMPI from the toolchain iompi/triton-2017a. The binaries are available in /share/apps/easybuild/software/FHI-aims/iomkl-2017a/bin as aims.YYMMDD.scalapack.mpi.x where YYMMDD indicates the version stamp.

Running FHI-aims on Triton

To run FHI-aims on Triton a following example batch script can be used:

```bash
#!/bin/bash -l
#SBATCH --time=01:00:00
#SBATCH --constraint=avx # FHI-aims build requires at least AVX instruction set
#SBATCH --mem-per-cpu=2000
#SBATCH -N 1
#SBATCH -n 24
export OMP_NUM_THREADS=1
module load FHI-aims/latest-iomkl-triton-2017a
srun aims.YYMMDD.scalapack.mpi.x
```

Armadillo

supportlevel C

Armadillo http://arma.sourceforge.net/ is C++ linear algebra library that is needed to support some other software stacks. To get best performance using MKL as backend is advised.

The challenge is that default installer does not find MKL from non-standard location.

1. module load mkl
2. Edit "./build_aux/cmake/Modules/ARMA_FindMKL.cmake" and add MKL path to "PATHS"
3. Edit "./build_aux/cmake/Modules/ARMA_FindMKL.cmake" and replace mkl_intel_thread with mkl_sequential (we do not want threaded libs on the cluster)
4. Edit "include/armadillo_bits/config.hpp" and enable ARMA_64BIT_WORD
5. cmake . & & make
6. make install DESTDIR=/share/apps/armadillo/<version>

Boost

supportlevel C
Boost is a numerical library needed by some other packages. There is a rpm-package of this in the default SL/RHEL repositories. In case the repository version is too old, a custom compilation is required.

To setup see the manual and follow the few simple steps to bootstrap and compile/install.

http://www.boost.org/doc/libs/1_56_0/more/getting_started/unix-variants.html

**COMSOL Multiphysics**

Comsol in Triton is best run in Batch-mode. To check which versions of Comsol are available, run:

```
module spider comsol
```

- To run Comsol in a single node use the “-np” instead of “-clustersimple”:

```
#!/bin/bash

# Ask for e.g. 20 compute cores
#SBATCH -N 1
#SBATCH -n 20

cd $WRKDIR/my_comsol_directory
module load comsol

# Details of your input and output files
INPUTFILE=input_model.mph
OUTPUTFILE=output_model.mph

comsol batch -np $SLURM_NTASKS -inputfile $INPUTFILE -outputfile $OUTPUTFILE -
→tmpdir $TMPDIR
```

- Comsol can run even bigger jobs over multiple computing nodes with “-clustersimple”. For this, please refer to Comsol online manual or ask for further help.

- Comsol uses a lot of temp file storage, which by default goes to $HOME. Fix a bit like the following:

```
$ rm -rf ~/.comsol/
$ mkdir /scratch/work/$USER/comsol_recoveries/
$ ln -sT /scratch/work/$USER/comsol_recoveries/ ~/.comsol/
```

**Deep learning software**

This page has information on how to run deep learning frameworks on Triton GPUs.

**Theano**

**Installation**

The recommended way of installing theano is with an anaconda environment.
Detectron

Detectron uses Singularity containers, so you should refer to that page first for general information.

Detectron-image is based on a Dockerfile from Detectron’s repository. In this image Detectron has been installed to /detectron.

Usage

This example shows how you can launch Detectron on a gpu node. To run example given in Detectron repository one can use the following Slurm script:

```bash
#!/bin/bash
#SBATCH -n 1
#SBATCH --gres=gpu:teslap100:1
#SBATCH -t 00:30:00
#SBATCH --mem=8G
#SBATCH -o detectron.out

module load singularity-detectron
mkdir -p $WRKDIR/detectron/outputs

singularity_wrapper exec python2 /detectron/tools/infer_simple.py \
   --cfg /detectron/configs/12_2017_baselines/e2e_mask_rcnn_R-101-FPN_2x.yaml \
   --output-dir $WRKDIR/detectron/outputs \
   --image-ext jpg \
   /detectron/demo
```

Now example can be run on GPU node with:

```
sbatch detectron.slurm
```

In typical usage one does not want to download models for each run. To use stored models one needs to:

1. Copy detectron sample configurations from the image to your own configuration folder:

```
module load singularity-detectron
mkdir -p $WRKDIR/detectron/
singularity_wrapper exec cp -r /detectron/configs $WRKDIR/detectron/configs
cd $WRKDIR/detectron
```

2. Create data directory and download example models there:

```
mkdir -p data/ImageNetPretrained/MSRA
```
4. Edit the weights-parameter in configuration file `12_2017_baselines/e2e_mask_rcnn_R-101-FPN_2x.yaml`:

```bash
$WRKDIR/detectron/data/ImageNetPretrained/MSRA/R-101.pkl
```

5. Edit Slurm script to point to downloaded weights and models:

```bash
#!/bin/bash
#SBATCH -n 1
#SBATCH --gres=gpu:teslap100:1
#SBATCH -t 00:30:00
#SBATCH --mem=8G
#SBATCH -o detectron.out

module load singularity-detectron
mkdir -p $WRKDIR/detectron/outputs

singularity_wrapper exec python2 /detectron/tools/infer_simple.py \
  --cfg $WRKDIR/detectron/configs/12_2017_baselines/e2e_mask_rcnn_R-101-FPN_2x.yaml \ 
  --output-dir $WRKDIR/detectron/outputs \ 
  --image-ext jpg \ 
  --wts $WRKDIR/detectron/Data/coco_2014_train:coco_2014_valminusminival/\ 
  generalized_rcnn/model_final.pkl \ 
  /detectron/demo
```

6. Submit job:

```
sbatch detectron.slrm
```

Fenics

This uses Singularity containers, so you should refer to that page first for general information.

Fenics-images are based on these images.

Usage

This example shows how you can run a fenics example. To run example one should first copy the examples from the image to a suitable folder:

```
mkdir -p $WRKDIR/fenics
cd $WRKDIR/fenics
module load singularity-fenics
singularity_wrapper exec cp -r /usr/local/share/dolfin/demo demo
```

The examples try to use interactive windows to plot the results. This is not available in the batch queue so to fix this one needs to specify an alternative matplotlib backend. This patch file fixes example `demo_poisson.py`. Download it into $WRKDIR/fenics and run
patch -d demo -p1 < fenics_matplotlib.patch

to fix the example. After this one can run the example with the following Slurm script:

```bash
#!/bin/bash
#SBATCH -p short
#SBATCH -t 00:15:00
#SBATCH -n 1
#SBATCH --mem=1G
#SBATCH -o fenics_out.out

module purge
module load singularity-fenics

cd demo/documented/poisson/python/

srun singularity_wrapper run demo_poisson.py
```

To submit the script one only needs to run:

```
sbatch fenics.slrm
```

Resulting image can be checked with e.g.:

```
eog demo/documented/poisson/python/poisson.png
```

**FMRIPrep**

```bash
module load singularity-fmriprep
```

fmriprep is installed as a singularity container. By default it will always run the current latest version. If you need a version that is not currently installed on triton, please open an issue at https://version.aalto.fi/gitlab/AaltoScienceIT/triton/issues

To run fmriprep for one subject, without free-surfer reconall, using ica-aroma

```
singularity_wrapper exec fmriprep <path-to-bids> <your-scratch-folder> participant --participant-label 01 --use-aroma --fs-no-reconall --fs-license-file /scratch/shareddata/set1/freesurfer/license.txt
```

**POST-processing**

Fmriprep does the minimal preprocessing. There is no smoothing, no temporal filtering and in general you need to regress out the estimated confounds. The most simple way is:

```bash
module load fsl
fsl_regfilt -i $inputniifile -d "$file_with_bold_confounds.tsv" -o $outputniifile -f -1,2,3,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31
```

There are also tools for post-processing such as:

- https://github.com/HBClab/NuisanceRegression
- https://xcpengine.readthedocs.io/
• https://fitlins.readthedocs.io/en/latest/
• https://github.com/arielletambini/denoiser

These are not installed on the singularity image, hence you need to experiment with these on your own.

**Freesurfer**

```
module load freesurfer
```

Follow the instruction to source the init script specific to your shell.

**FSL**

```
module load fsl
```

Follow the instruction to source the init script specific to your shell.

**GPAW**

There is GPAW version installed in GPAW/1.0.0-goolf-triton-2016a-Python-2.7.11. It has been compiled with GCC, OpenBLAS and OpenMPI and it uses Python/2.7.11-goolf-triton-2016a as its base Python. You can load it with:

```
$ module load GPAW/1.0.0-goolf-triton-2016a-Python-2.7.11
```

You can create a virtual environment against the Python environment with:

```
$ export VENV=/path/to/env
$ virtualenv --system-site-packages $VENV
$ cd $VENV
$ source bin/activate
# test installation
$ python -c 'import gpaw; print gpaw'
```

GPAW site: https://wiki.fysik.dtu.dk/gpaw/

**Julia language on triton**

The Julia programming language is a high-level, high-performance dynamic programming language for technical computing, in the same space as e.g. MATLAB, Scientific Python, or R. For more details, see the web page http://julialang.org/.

**Interactive usage**

Julia is available in the module system. By default the latest stable release is loaded:

```
module load julia
julia
```
Batch usage

Running Julia scripts as batch jobs is also possible. An example batch job is

Batch script for Julia job:

```
#!/bin/sh
#SBATCH -p play
#SBATCH -n 1
#SBATCH --time=00:01:00
#SBATCH --mem-per-cpu=1G
module load julia
srun julia juliascript.jl
```

Number of threads to use

By default Julia uses up to 16 threads for linear algebra (BLAS) computations. However, the julia module on triton sets the environment variable OMP_NUM_THREADS to 1, so only a single thread is used. If you wish to use more threads than that (e.g. you have launched a batch job with multiple threads per task with the “#SBATCH -c N” option), you can set the OMP_NUM_THREADS environment variable to some other value, or alternatively inside julia you can use the blas_set_num_threads() function.

Jupyter

Note: Quick link

Triton’s JupyterHub is available at http://jupyter.triton.aalto.fi.

Note: For new users

Are you new to Triton and want to access JupyterHub? Triton is a high-performance computing cluster, and Jupyter-Hub is just one of our services - one of the easiest ways to get started. You still need a Triton account. This site has many instructions, but you should read at least:

- About us, how to get help, and acknowledging Triton usage (this JupyterHub is part of Triton, and thus Science-IT must be acknowledged in publications).
- The accounts page, in order to request a Triton account.
- Possibly the storage page to learn about the places to store data and how to transfer data.
- The JupyterHub section of this page (below).

If you want to use Triton more, you should finish the entire tutorials section.

Fig. 1: < Triton JupyterHub Demo >

Jupyter notebooks are a way of interactive, web-based computing: instead of either scripts or interactive shells, the notebooks allow you to see a whole script + output and experiment interactively and visually. They are good for developing and testing things, but once things work and you need to scale up, it is best to put your code into proper programs (more info). You must do this if you are going to large parallel computing.
Triton’s JupyterHub is available at https://jupyter.triton.aalto.fi. You can try them online at try.jupyter.org (there is a temporary notebook with no saving).

You can always run notebooks yourself on your own (or remote) computers, but on Triton we have some facilities already set up to make it easier.

### How Jupyter notebooks work

- Start a notebook
- Enter some code into a cell.
- Run it with the buttons or Control-enter or Shift-enter to run a cell.
- Edit/create new cells, run again. Repeat indefinitely.
- You have a visual history of what you have run, with code and results nicely interspersed. With certain languages such as Python, you can plots and other things embedded, so that it becomes a complete reproducible story.

JupyterLab is the next iteration of this and has many more features, making it closer to an IDE or RStudio.

Notebooks are without a doubt a great tool. However, they are only one tool, and you need to know their limitations. See our other page on limitations of notebooks.

### JupyterHub

**Note:** JupyterHub on Triton is still under development, and features will be added as they are needed or requested. Please use the Triton issue tracker.

The easiest way of using Jupyter is through JupyterHub - it is a multi-user jupyter server which takes a web-based login and spawns your own single-user server. This is available on Triton.

### Connecting and starting

Currently jupyterHub is available only within Aalto networks, or from the rest of the internet after a first Aalto login: https://jupyter.triton.aalto.fi.

Once you log in, you must start your single-user server. There are several options available that trade off between long run time and short run time but more memory available. Your server runs in the Slurm queue, so the first start-up takes a few seconds but after that it will stay running even if you log out. The resources you request are managed by slurm: if you go over the memory limit, your server will be killed without warning or notification (but you can see it in the output log, ~/jupyterhub_slurmspawner_*.*.log). The Jupyter server nodes are oversubscribed, which means that we can allocate more memory and CPU than is actually available. We will monitor the nodes to try to ensure that there are enough resources available, so do report problems to us. **Please request the minimum amount of memory you think you need** - you can always restart with more memory. You can go over your memory request a little bit before you get problems.

When you use Jupyter via this interface, the slurm billing weights are lower, so that the rest of your Triton priority does not decrease by as much.
Usage

Once you get to your single-user server Jupyter running as your own user on Triton. You begin in a convenience directory which has links to home, scratch, etc. You can not make files in this directory (it is read-only), but you can navigate to the other folders to create your notebooks. You have access to all the Triton filesystems (not project/archive) and all normal software.

We have some basic extensions installed:

- Jupyterlab (to use it, change /tree in the URL to /lab). Jupyterlab will eventually be made the default.
- modules integration
- jupyter_contrib_nbextensions - check out the variable inspector
- diff and merge tools (currently does not work somehow)

The log files for your single-user servers can be found in, see ~/jupyterhub_slurmspawner_*.log. When a new server starts, these are automatically cleaned up when they are one week old.

For reasons of web security, you can’t install your own extensions (but you can install your own kernels). Send your requests to us instead.

Problems? Requests?

This service is currently in beta and under active development. If you notice problems or would like any more extensions or features, let us know. If this is useful to you, please let us know your user store, too. In the current development stage, the threshold for feedback should be very low.

Currently, the service level is best effort. The service may go down at any time and/or notebooks may be killed whenever there is a shortage of resources or need of maintenance. However, notebooks auto-save and do survive service restarts, and we will try to avoid killing things unnecessarily.

Software and kernels

We have various kernels automatically installed (these instructions should apply to both JupyterHub and sjupyter):

- Python (2 and 3 via anacondaN/latest modules + a few more Python modules.)
- Matlab (latest module)
- Bash kernel
- R (a default R environment you can get by module load r-triton. (“R (safe)” is similar but tries to block some local user configuration which sometimes breaks things, see FAQ for more hints.)
- Julia: currently doesn’t seem to play nicely with global installations (if anyone knows something otherwise, let us know). Just load two modules: module load julia, module load jupyterhub/live, and then install the kernel julia and Pkg.add("IJulia") and it will install locally for you.
- We do not yet have a kernel management policy. Kernels may be added or removed over time. We would like to keep them synced with the most common Triton modules, but it will take some time to get this automatic. Send requests and problem reports.

Since these are the normal Triton modules, you can submit installation requests for software in these so that it is automatically available.

Installing kernels from virtualenvs or Anaconda environments:

- Activate the environment (however you do it normally)
Aalto scientific computing guide

- (Install ipykernel if not already there: do this if next step does not work: `pip install ipykernel`)
- `python -m ipykernel install --user --name=python-YOURENV
   --display-name="YOUR-ENV"`

If you want to install your own non-Python kernels:

- First, `module load jupyterhub/live`. This loads the anaconda environment which contains all the server code and configuration. (This step may not be needed for all kernels)
- Follow the instructions you find for your kernel. You may need to specify `--user` or some such to have it install in your user directory.
- You can check your own kernels in `~/.local/share/jupyter/kernels/`.

If your kernel involves loading a module, you can either a) load the modules within the notebook server (“softwares” tab in the menu), or b) update your `kernel.json` to include the required environment variables (see kernel-spec). (We need to do some work to figure out just how this works). Check `/share/apps/jupyterhub/live/miniconda/share/jupyter/kernels/ir/kernel.json` for an example of a kernel that loads a module first.

Git integration

You can enable git integration on Triton by using the following lines from inside a git repository. (This is normal nbdime, but uses the centrally installed one so that you don’t have to load a particular conda environment first. The `sed` command fixes relative paths to absolute paths, so that you use the tools no matter what modules you have loaded):

```
/share/apps/jupyterhub/live/miniconda/bin/nbdime config-git --enable
sed --in-place -r 's@(= )[ a-z/-]*\1@git-nb@' /share/apps/jupyterhub/live/miniconda/
   \bin/\2@' .git/config
```

FAQ/common problems

- **Jupyterhub won’t spawn my server:** “Error: HTTP 500: Internal Server Error (Spawner failed to start [status=1]).” Is your home directory quota exceeded? If that’s not it, check the `~/jupyterhub_slurmspawner_*` logs then contact us.
- **My server has died mysteriously.** This may happen if resource usage becomes too much and exceed the limits - Slurm will kill your notebook. You can check the `~/jupyterhub_slurmspawner_*` log files for jupyterhub to be sure.
- **My server seems inaccessible / I can’t get to the control panel to restart my server.** Especially with JupyterLab. In JupyterLab, use File→Hub Control Panel. If you can’t get there, you can change the URL to `/hub/home`.
- **My R kernel keeps dying.** Some people seem to have global R configuration, either in `.bashrc` or `.Renviron` or some such which globally, which even affects the R kernel here. Things we have seen: pre-loading modules in `.bashrc` which conflict with the kernel R module; changing `RLIBS` in `.Renviron`. You can either (temporarily or permanently) remove these changes, or you could **install your own R kernel**. If you install your own, it is up to you to maintain it (and remember that you installed it).
- **“Spawner pending” when you try to start - this is hopefully fixed in issue #1534/#1533 in JupyterHub. Current recommendation: wait a bit and return to JupyterHub home page and see if the server has started. Don’t click the button twice!**
See also

- https://jupyter.org
  - Online demos and live tutorial: https://jupyter.org/try (use the Python one)
- Jupyter basic tutorial: https://www.youtube.com/watch?v=HW29067qVWk (this is just the first link on youtube - there are many more too)
- More advanced tutorial: Data Science is Software (this is not just a Jupyter tutorial, but about the whole data science workflow using Jupyter. It is annoying long (2 hours), but very complete and could be considered good “required watching”)
- Pitfalls of Jupyter Notebooks
- CSC has this service, too, however there is no long term storage yet so there is limited usefulness for research: https://notebooks.csc.fi/

Our configuration is available on Github. Theoretically, all the pieces are here but it is not yet documented well and not yet generalizable. The Ansible role is a good start but the jupyterhub config and setup is hackish.

- Ansible config role: https://github.com/AaltoScienceIT/ansible-role-fgci-jupyterhub
- Configuration and automated conda environment setup: https://github.com/AaltoScienceIT/triton-jupyterhub

Keras

```yaml
supportlevel
pagelastupdated 2018
maintainer
```

Keras is a neural network library which runs on tensorflow (among other things).

Basic usage

Keras is available in the anaconda2/3 modules (GPU version) and some other anaconda modules. Run `module spider anaconda3` to list available modules. The `-cpu` modules have a tensorflow that will run on CPUs. The others have GPU-only versions, so you have to run in the GPU queues. The other information in the `tensorflow` page also applies, especially the `--constraint` options to restrict to the GPUs that have new enough features.

Example

```bash
srun -p gpu --pty bash
module load anaconda3
python3
>>> import keras
Using TensorFlow backend.
>>> keras.__version__
'2.2.4'
```
Lammps

supportlevel C
pagelastupdated 2014

http://lammps.sandia.gov/

Building LAMMPS as a library

```
import os
os.chdir('src')
# default g++ compilation with system g++
module load openmpi/1.8.1-gcc
make -f Makefile.lib serial
```

Using Mathematica on Triton

Login with X support

For a sake of GUI interface, login to triton.aalto.fi with -X, i.e. X11 forwarding enabled.

```
ssh -X triton.aalto.fi
```

Load Mathematica through module

```
module load mathematica/10.2
```

See available versions with `module avail mathematica`.

Running

Run in GUI mode

```
$ mathematica &
```

Run in text based mode

```
$ wolfram
```

*For the first time runners*, choose ‘Other ways to activate’” then “Connect to a network license server”, paste “lic-mathematica.aalto.fi” (without quotes).

Matlab

This page will guide you through the serial computing with Matlab at Triton cluster. (Note (2017): We used to have the Matlab Distributed Computing Server (MDCS), but because of low use we no longer have a license. You can still run in parallel on one node, up to 20-28 cores depending on how new.)
Matlab configuration

Matlab writes session data, compiled code and additional toolboxes to ~/.matlab. This can quickly fill up your $HOME quota. To fix this we recommend that you replace the folder with a symlink that points to a directory in your working directory.

    rsync -lrt ~/.matlab/ $WRKDIR/matlab-config/ && rm -r ~/.matlab
    ln -sT $WRKDIR/matlab-config ~/.matlab
    quotafix -gs --fix $WRKDIR/matlab-config

Interactive usage

Interactive usage is currently available via the sinteractive tool. Do not use the cluster front-end for doing heavy task. Only meant for submitting jobs/compiling. Using MDCS for sending jobs is ok.

    ssh -X user@triton.aalto.fi
    sinteractive
    module load matlab
    matlab &

Simple serial script

Running a single core Matlab job is easy through the slurm queue. A sample slurm script is provided underneath:

```bash
#!/bin/bash -l
#SBATCH -p short
#SBATCH -t 00:05:00
#SBATCH -n 1
#SBATCH --mem-per-cpu=100
#SBATCH -o serial_Matlab.out
module load matlab
n=3
m=2
srun matlab -nojvm -nosplash -r "serial_Matlab($n,$m) ; exit(0)"
```

The above script can then be saved as a file (e.g. matlab_test.slrm) and the job can be submitted with `sbatch matlab_test.slrm`. The actual calculation is done in `serial_Matlab.m`-file:

```matlab
function C = serial_Matlab(n,m)
    try
        A=0:(n*m-1);
        A=reshape(A,[2,3]).'

        B=2:(n*m+1);
        B=reshape(B,[2,3]).'

        C=0.5*ones(n,n)
        C=A*(B.' + 2.0*C
    catch error
        disp(getReport(error))
        exit(1)
    end
end
```
Remember to always set exit into your slurm script so that the program quits once the function `serial_Matlab` has finished. Using a try-catch-statement will allow your job to finish in case of any error within the program. If you don’t do this, Matlab will drop into interactive mode and do nothing while your cluster time wastes.

NOTE: Starting from version r2019a the launch options `-r ...; exit(0)` can be easily replaced with the `-batch` option which automatically exits matlab at the end of the command that is passed (see here for details). So the last command from the slurm script above for Matlab r2019a will look like:

```
  srun matlab -nojvm -nosplash -batch "serial_Matlab($n,$m);"
```

### Multiple serial batchjobs

The most common way to utilize Matlab is to write a single .M-file that can be used to run tasks as a non-interactive batch job. These jobs are then submitted as independent tasks and when the heavy part is done, the results are collected for analysis. For these kinds of jobs the Slurm array jobs is the best choice; for more information on array jobs see Array jobs in the Triton user guide.

Below you will find an example how-to prepare and run such type of jobs.

**run.m file doing the actual calculation task**

The file below calculates Sin-function in the interval 0-2*PI and stores the results into a file. The interval is divided into blocks that are distributed over the nodes.

```matlab
function run(blockIndex,pointsPerBlock,totalBlocks)
    % blockindex runs from 0..totalblocks-1
    % range 0..2pi
    length=2*pi;
    % values to setup even spacing between given range
    % and splitting the spacings to even number of points per block
    totalPoints=pointsPerBlock*totalBlocks;
    step=length/(totalPoints-1);
    start=blockIndex*pointsPerBlock*step;
    % do some calculations, store the results so arrays A and B
    for index=0:pointsPerBlock-1
        i=index+1;
        x=start+index*step;
        y=sin(x);
        A(i)=x;
        B(i)=y;
    end
    % save the results based on the blockIndex to a file
    filename=strcat('output-',int2str(blockIndex));
    save( filename, 'A', 'B', 'blockIndex');
    % display message to output (log) that we have reached this far.
    disp(sprintf('SUCCESS blockIndex %d',blockIndex));
    % exit as this is a batch-job
    exit;
```

**Submission of 10 independent tasks**

Below the `run.m` is executed as an array job with 10 array tasks, which will execute independently, potentially in parallel if there are enough idle resources. Note that it is using play partition with 5min time limit.

```bash
matslurm.sh:
```
Aalto scientific computing guide

Submit the job with “sbatch matslurm.sh” (or whatever you called the batch job script above).

Collecting the results

Finally a wrapper script to read in the .mat files and plots you the Sin-function calculated in parallel with 10 tasks:

```matlab
function collectResults(numberOfBlocks)
    X=[];
    Y=[];
    for index=0:numberOfBlocks-1
        % read the output from the jobs
        filename = strcat( 'output-', int2str( index ) );
        load( filename );
        % catenate results to a single arrays
        X=cat(2,X,A);
        Y=cat(2,Y,B);
    end
    plot(X,Y,'b+:')
```

Seeding the random number generator

Note that by default MATLAB always initializes the random number generator with a constant value. Thus if you launch several matlab instances e.g. to calculate distinct ensembles, then you need to seed the random number generator such that it’s distinct for each instance. In order to do this, you can call the `rng()` function, passing the value of `$SLURM_ARRAY_TASK_ID` to it.

Parallel Matlab with Matlab’s internal parallelization

Matlab has internal parallelization that can be activated by requesting more than one cpu per task in the Slurm script and using the `matlab_multithread` to start the interpreter:

```bash
#!/bin/bash -l
#SBATCH --time=00:15:00 --nodes=1 --ntasks=4
#SBATCH --cpus-per-task=4
#SBATCH --mem-per-cpu=2G
#SBATCH --array=0-9
module load matlab
srun time -p matlab_multithread -nojvm -nosplash -r "int_parallel() ; exit(0)"
```

An example function is provided in this script
function int_parallel()
    try
        tic;
        A = rand(2000,2000);
        A = A + A.);
        B = pinv(A);
        max(max(B * A))
        toc
    catch error
        disp('Error occurred');
        exit(0)
    end
end

Parallel Matlab with parpool

Often one uses Matlab's parallel pool for parallelization. When using parpool one needs to specify the number of workers. This number should match the number of CPUs requested. parpool uses JVM so when launching the interpreter one needs to use -nodisplay instead of -nojvm. Example Slurm script:

#!/bin/bash -l
#SBATCH -p short
#SBATCH -t 00:15:00
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=4
#SBATCH --mem-per-cpu=2G
#SBATCH -o parpool_parallel.out

module load matlab

srun matlab_multithread -nodisplay -r "parpool_parallel($SLURM_CPUS_PER_TASK) ; exit(0)"

An example function is provided in this script

function parpool_parallel(n)
    % Try-catch expression that quits the Matlab session if your code crashes
    try
        % Initialize the parallel pool
        c=parcluster();
        t=tempname()
        mkdir(t)
        c.JobStorageLocation=t;
        parpool(c,n);
        % The actual program calls from matlab's example.
        % The path for r2017b
        addpath(strcat(matlabroot, '/examples/distcomp/main'));
        % The path for r2016b
        % addpath(strcat(matlabroot, '/examples/distcomp'));

        % simulate 10000 blackjack hands with 100 players
        tic;
        pctdemo_aux_parforbench(10000,100,n);
        toc
    end
try
    c=parcluster();
    t=tempname()
    mkdir(t)
    c.JobStorageLocation=t;
    parpool(c,n);
    % The actual program calls from matlab's example.
    % The path for r2017b
    addpath(strcat(matlabroot, '/examples/distcomp/main'));
    % The path for r2016b
    addpath(strcat(matlabroot, '/examples/distcomp'));
    pctdemo_aux_parforbench(10000,100,n);
    catch error
        getReport(error)
        disp('Error occurred');
        exit(0)
    end
end

Hints for Condor users

The above example also works (even nicer way) for condor.

A wrapper script to execute matlab on the department workstation.

```bash
#!/bin/bash -l
# a wrapper to run Matlab with condor
```
block=$1
pointsPerBlock=10
totalBlocks=10
matlab -nojvm -r "run($block,$pointsPerBlock,$totalBlocks)"

Condor submission script
Condor actually contains ArrayJob functionality that makes the task easier.

```bash
## Condor submit description (script) file for my_program.exe.
## 1. Specify the [path and] name for the executable file...
Executable = run.sh
## 2. Specify Condor execution environment.
Universe = vanilla
notify = Error
## 3. Specify remote execution machines running Linux (required)...
Requirements = ((OpSys == "Linux") || (OpSysName == "Ubuntu"))
## 4. Define input files and arguments
#Input = stdin.txt.$(Process)
Arguments = $(Process)
## 5. Define output/error/log files
Output = log/stdout.$(Process).txt
Error = log/stderr.$(Process).txt
Log = log/log.$(Process).txt
## 6. Tell Condor which files need to be transferred and when.
Transfer_input_files = run.m
Transfer_output_files = output-$(Process).mat
Transfer_executable = true
Should_transfer_files = YES
When_to_transfer_output = ON_EXIT
## 7. Add 10 copies of the job to the queue
Queue 10
```

FAQ / troubleshooting

If things randomly don’t work, you can try removing or moving either the ~/.matlab directory or ~/.matlab/Rxxxxxy directory to see if it’s caused by configuration.

Random error messages about things not loading and/or something (Matlab Live Editor maybe) doesn’t work: ls *.m, do you have any unexpected files like pathdef.m in there? Remove them.

Also, check your home quota. Often .matlab gets large and fills up your home directory. Check the answer at the very top of the page, under “Matlab Configuration”.

MLPack

**pagelastupdated** 2014

**supportlevel** C

http://www.mlpack.org/

1. module load cmake; module load armadillo/4.3-mkl; module load mkl
2. mkdir build && cd build

6.1. Triton user guide
3. cmake -D ARMADILLO_LIBRARY=$ARMADILLO_LIBRARY -D ARMADILLO_INCLUDE_DIR=$ARMADILLO_INCLUDE ../
4. make
5. bin/mlpack_test
6. make install CMAKE_INSTALL_PREFIX=/share/apps/mlpack/1.0.8

For newer boost library also load boost module and tell cmake where to find boost

```
module load boost
...
cmake -D BOOST_ROOT=$BOOST_ROOT -D ARMADILLO_LIBRARY=$ARMADILLO_LIBRARY -D ARMADILLO_INCLUDE_DIR=$ARMADILLO_INCLUDE ../
```

### Notes

- 1.0.10 installation failed when installing doc to /usr/local (install prefix defined ad /share/apps/mlpack/1.0.10). The solution was manually tune install prefix at cmake_install.cmake

### MNE

```
pagelastupdate 2018
maintainer
```

```
module load mne
```

Follow the instruction to source the init script specific to your shell. In the directory:

```
$MNE_ROOT/.. 
```

you can find the relase notes, the manual, and some sample data.

### Octave

**From Octave's web page:** GNU Octave is a high-level language, primarily intended for numerical computations. It provides a convenient command line interface for solving linear and nonlinear problems numerically, and for performing other numerical experiments using a language that is mostly compatible with Matlab. It may also be used as a batch-oriented language.

Octave has extensive tools for solving common numerical linear algebra problems, finding the roots of nonlinear equations, integrating ordinary functions, manipulating polynomials, and integrating ordinary differential and differential-algebraic equations. It is easily extensible and customizable via user-defined functions written in Octave’s own language, or using dynamically loaded modules written in C++, C, Fortran, or other languages.

### Getting started

Simply load the latest version of Octave.

```
module load octave
octave
```


It is best to pick a version of octave and stick with it. Do `module spider octave` and use the whole name:

```
module load octave/4.4.1-qt-python2
```

To run octave with the GUI, run it with:

```
octave --force-gui
```

### Installing packages

Before installing packages you should create a file `~/.octaverc` with the following content:

```
package_dir = ['/scratch/work/', getenv('USER'), '/octave'];
eval(['pkg prefix ',package_dir, ';']);
setenv("CXX","g++ -std=gnu++11")
setenv("DL_LD","g++ -std=gnu++11")
setenv("LD_CXX","g++ -std=gnu++11")
setenv("CC","gcc")
setenv("F77","gfortran")
```

This sets up `/scratch/work/$USER/octave` to be your Octave package directory and sets `gcc` to be your compiler. By setting Octave package directory to your work directory you won’t run into any quota issues.

After this you should load `gcc`- and `texinfo`-modules. This gives you an up-to-date compiler and tools that Octave uses for its documentation:

```
module load gcc
module load texinfo
```

Now you can install packages in octave with e.g.:

```
 pkg install -forge -local io
```

After this you can unload the `gcc`- and `texinfo`-modules:

```
module unload gcc
module unload texinfo
```

### OpenFOAM (with ParaView)

This uses Singularity containers, so you should refer to that page first for general information.

OpenFOAM and ParaView have been installed from the Ubuntu 16.04 Docker image provided by OpenFOAM people. It has minimal amount of other software installed.

 Parallelization is done against Triton’s OpenMPI, so using this container with other OpenMPI modules is discouraged.

### New image (singularity-openfoam)

Loading: simply `module load singularity-openfoam` and use `singularity_wrapper`.

OpenFOAM is installed in `/opt/OpenFOAM`. The OpenFOAM `bashrc` file is automatically sourced when you exec or shell within the image to set `PATH` and so on.
Old image (OpenFOAM)

This is quite similar to the new image.

Within the container OpenFOAM is installed under /opt/openfoam4/ and ParaView under /opt/paraviewopenfoam50/. PATH is automatically appended with their respective paths so all program calls are available automatically.

Usage

(This has not been updated to the new image yet. To change to new image, don’t do the module use and instead just load singularity-openfoam.)

This example shows how you can run damBreak example. Firstly, let’s load the OpenFOAM module and create a folder for the example:

```
module use /share/apps2/singularity/modules
module load OpenFOAM
mkdir damBreak
cd damBreak
```

Secondly, let’s use singularity shell to copy example data files to the folder and to initialize the simulation:

```
cp -r /opt/openfoam4/tutorials/multiphase/interFoam/laminar/damBreak/damBreak/0 .
cp -r /opt/openfoam4/tutorials/multiphase/interFoam/laminar/damBreak/damBreak/constant .
blockMesh
decomposePar
exit
```

After this one can submit the following slurm script with sbatch to solve the problem:

```
#!/bin/bash
#SBATCH -p short
#SBATCH -t 00:30:00
#SBATCH -n 4
#SBATCH --mem=4G
module use /share/apps2/singularity/modules
module purge
module load OpenFOAM
srun singularity_wrapper exec interFoam -parallel
```

Paraview can be started similarly with this script:

```
#!/bin/bash
#SBATCH -p short
#SBATCH -t 00:10:00
#SBATCH -n 1
#SBATCH --mem=8G
module use /share/apps2/singularity/modules
module purge
module load OpenFOAM
```

(continues on next page)
singularity_wrapper exec paraview

**OpenPose**

This uses *Singularity containers*, so you should refer to that page first for general information. OpenPose has been compiled against Atlas and Caffe, CUDA and cuDNN. Image is based on a `bvlc/caffe:gpu` base image.

Dockerfile for this image is available [here](#).

Within the container OpenPose is installed under `/opt/openpose-master`. Due to the way the examples are organized, the `singularity_wrapper` changes the working directory to `/opt/openpose-master`.

Example `sbatch` script is shown below.

```
#!/bin/bash
#SBATCH -t 00:10:00
#SBATCH -n 1
#SBATCH --gres=gpu:teslak80:1
#SBATCH --mem=8G

module load singularity-openpose/v1.4.0

# Print out usage flags
singularity_wrapper exec openpose --help

# Run example
singularity_wrapper exec openpose --video examples/media/video.avi --display 0 --
                         --write_video $WRKDIR/openpose.avi
```

**Paraview**

**As a module**

A serial version is available on login2. You will need to use the “forward connection” strategy by using ssh port forwarding. For example, run `ssh -L BBBB:nnnNNN:AAAA username@triton`, where BBBB is the server you connect to locally and nnnNNN is the node name and AAAA is the port on that node. See this FAQ question.

See issue #13: [https://version.aalto.fi/gitlab/AaltoScienceIT/triton/issues/13](https://version.aalto.fi/gitlab/AaltoScienceIT/triton/issues/13) for some user experiences. (Note: the author of this entry is not a paraview expert, suggestions welcome.)

**As a container**

You can also use paraview via *Singularity containers*, so you should refer to that page first for general information. It is part of the *OpenFOAM (with ParaView)* container.
Python

Python is widely used programming language where we have installed all basic packages on every node. Yet, python develops quite fast and the system provided packages are often not complete or getting old.

**Python distributions**

<table>
<thead>
<tr>
<th>Most of the use cases, but sometimes different versions of modules needed</th>
<th>Python to use</th>
<th>How to install own packages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Most of the use cases, but sometimes different versions of modules needed</td>
<td>Anaconda 2/3</td>
<td>conda environment + conda</td>
</tr>
<tr>
<td>Simple programs with common packages, not switching between Pythons often</td>
<td>Anaconda 2/3</td>
<td>pip install --user</td>
</tr>
<tr>
<td>Special advanced cases.</td>
<td>Python from module system</td>
<td>virtualenv + pip install</td>
</tr>
</tbody>
</table>

There are two main versions of Python: 2 and 3. There are also different distributions: The “regular” CPython, Anaconda (a package containing CPython + a lot of other scientific software all bundled together), PyPy (a just-in-time compiler, which can be much faster for some use cases). Triton supports all of these.

- For general scientific/data science use, we suggest that you use Anaconda. It comes with the most common scientific software included, and is reasonably optimized.
- There are many other “regular” CPython versions in the module system. These are compiled and optimized for Triton, and are highly recommended. The default system Python is old and won’t be updated.
- PyPy is still mainly for advanced use (it can be faster under certain cases, but does not work everywhere). It is available in a module.

**Quickstart**

Use `module load anaconda3` (or `module load anaconda2`) to get a modern Python.

If you have simple needs, use `pip install --user` to install packages. For complex needs, use `anaconda + conda environments` to isolate your projects.

**Install your own packages easily**

Installing your own packages with `pip install` won’t work, since it tries to install globally for all users. Instead, you should do this (add `--user`) to install the package in your home directory (`~/.local/lib/pythonN.N/`):

```
pip install --user $package_name
```

This is quick and effective best used for leaf packages without many dependencies and if you don’t switch Python modules often.

**Warning!** If you do this, then the module will be shared among all your projects. It is quite likely that eventually, you will get some incompatibilities between the Python you are using and the modules installed. In that case, you are on your own (simple recommendation is to remove all modules from `~/.local/lib/pythonN.N` and reinstall). If you get incompatible module errors, our first recommendation will be to remove everything installed this way and use `conda/virtual environments` instead. It’s not a bad idea to do this when you switch to environments anyway. Note: `pip` installs from the Python Package Index.
Anaconda and conda environments

Anaconda is a Python distribution by Continuum Analytics. It is nothing fancy, they just take a lot of useful scientific packages and put them all together, make sure they work, and do some sort of optimization. They also include most of the most common computing and data science packages. It is also all open source, and is packaged nicely so that it can easily be installed on any major OS.

To load anaconda, use the module system (you can also load specific versions):

```
module load anaconda3  # python3
module load anaconda2  # python2
```

Conda environments

virtualenv does not work with Anaconda, use conda instead.

A conda environment lets you install all your own packages. Your home directories are very small, so it requires some initial steps. You see module load teflon here a lot: conda does bad things with permissions, thus messing up quota accounting. This prevents that.

- Initial setup: link the conda cache to your work directory (an rsync error because ~/.conda doesn’t exist is OK):

```
# Move your package cache to your work directory. The following does it automatically.
rsync -lrt ~/.conda/ $WRKDIR/conda/ && rm -r ~/.conda
ln -sT $WRKDIR/conda ~/.conda
quotafix -gs --fix $WRKDIR/conda
```

- Load the anaconda version you want to use. You will need to always load same version each time you source the environment:

```
# Load anaconda first. This must always be done before activating the env!
module load anaconda3  # or anaconda3
```

- Create an environment:

```
# create environment with package pip in it
module load teflon
conda create --prefix PATH/TO/DIR python pip ipython ...
module unload teflon
```

- Activating and using the environment, installing more packages, etc. can be done either using conda install or pip install:

```
# This must be run in each shell to set up the environment variables properly. # make sure module is loaded first.
source activate PATH/TO/DIR

# Install more packages, either conda or pip
module load teflon
conda search PACKAGE_NAME
conda install PACKAGE_NAME
pip install PACKAGE_NAME
module unload teflon
```

- Leaving the environment when done (optional):
• If you run into “quota exceeded” problems, you need to do the first steps above which move the .conda directory to another folder. The quotafix command may be useful to try to reset things (see above), but if that doesn’t work: in the worst case, remove everything and recreate it.:

```bash
# Remove anything installed with pip install --user.
rm -r ~/.local/lib/python*.*/
```

A few notes about conda environments:

• Once you use a conda environment, everything goes into it. Don’t mix versions with, for example, local packages in your home dir and `pip install --user`. Things installed (even previously) with `pip install --user` will be visible in the conda environment and can make your life hard! Eventually you’ll get dependency problems.

• Often the same goes for other python based modules. We have setup many modules that do use anaconda as a backend. So, if you know what you are doing this might work.

• The commands below will fail:
  - `conda create --prefix $WRKDIR/foo --clone root` # will fail as our anaconda module has additional packages (e.g. via pip) installed.

### Python: virtualenv

Virtualenv is default-Python way of making environments, but does **not** work with Anaconda. We generally recommend using anaconda, since it includes a lot more stuff by default, but virtualenv works on other systems easily so it’s good to know about.

```bash
# Create environment
virtualenv DIR

# activate it (in each shell that uses it)
source DIR/bin/activate

# install more things (e.g. ipython, etc.)
pip install PACKAGE_NAME

# deactivate the virtualenv
deactivate
```
Python optimized for Triton

There are Python modules installed with the typical software setup against EasyBuild toolchains. While some of the more general packages available with anaconda installation might be missing, the Numpy and Scipy installations on these modules are highly optimized against the installed linear algebra libraries. A typical module loading using these toolchains could be

```
module load Python/2.7.11-goolf-triton-2016a
module load numpy/1.11.1-goolf-triton-2016a-Python-2.7.11
module load scipy/0.18.0-goolf-triton-2016a-Python-2.7.11
```

Use module spider Python to see available modules. More specialized modules like Tensorflow, Theano etc. will be installed against these modules so that they can be in optimal settings. Submit your issue in tracker if you wish some other Python modules to be included in these installations.

IPython Parallel

ipyparallel is a tool for running embarrassingly parallel code using Python. The basic idea is that you have a controller and engines. You have a client process which is actually running your own code.

Preliminary notes: ipyparallel is installed in the anaconda{2,3}/latest modules.

Let’s say that you are doing some basic interactive work:

- Controller: this can run on the frontend node, or you can put it on a script. To start: `ipcontroller --ip="*"`
- Engines: `srun -N4 ipengine`: This runs the four engines in slurm interactively. You don’t need to interact with this once it is running, but remember to stop the process once it is done because it is using resources. You can start/stop this as needed.
- Start your Python process and use things like normal:

```
import os
import ipyparallel
client = ipyparallel.Client()
result = client[:].apply_async(os getpid)
pid_map = result.get_dict()
print (pid_map)
```

This method lets you turn on/off the engines as needed. This isn’t the most advanced way to use ipyparallel, but works for interactive use.

See also: IPython parallel for a version which goes in a slurm script.

Background: pip vs python vs anaconda vs conda vs virtualenv

Virtual environments are self-contained python environments with all of their own modules, separate from the system packages. They are great for research where you need to be agile and install whatever versions and packages you need. We highly recommend virtual environments or conda environments (below)

- Anaconda: use conda, see below
- Normal Python: virtualenv + pip install, see below

You often need to install your own packages. Python has its own package manager system that can do this for you. There are three important related concepts:
• pip: the Python package installer. Installs Python packages globally, in a user’s directory (--user), or anywhere. Installs from the Python Package Index.

• virtualenv: Creates a directory that has all self-contained packages that is manageable by the user themself. When the virtualenv is activated, all the operating-system global packages are no longer used. Instead, you install only the packages you want. This is important if you need to install specific versions of software, and also provides isolation from the rest of the system (so that you work can be uninterrupted). It also allows different projects to have different versions of things installed. virtualenv isn’t magic, it could almost be seen as just manipulating PYTHONPATH, PATH, and the like. Docs: http://docs.python-guide.org/en/latest/dev/virtualenvs/

• conda: Sort of a combination of package manager and virtual environment. However, it only installed packages into environments, and is not limited to Python packages. It can also install other libraries (c, fortran, etc) into the environment. This is extremely useful for scientific computing, and the reason it was created. Docs for envs: http://conda.pydata.org/docs/using/envs.html.

So, to install packages, there is pip and conda. To make virtual environments, there is venv and conda. Advanced users can see this rosetta stone for reference.

On Triton we have added some packages on top of the Anaconda installation, so cloning the entire Anaconda environment to local conda environment will not work (not a good idea in the first place but some users try this every now and then).

Examples

Running Python with internal parallelization (OpenMP)

A simple parallel Python script using OpenMP. Both anaconda modules and optimized Python modules support OpenMP, but optimized versions are faster.

Python OpenMP example

parallel_Python.slrm:

```bash
#!/bin/bash
#SBATCH -p short
#SBATCH -t 00:10:00
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=4
#SBATCH --mem-per-cpu=2G
#SBATCH -o parallel_Python.out
module load anaconda3
export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK
srun -c $SLURM_CPUS_PER_TASK python parallel_Python.py
done
```

parallel\Python.py:

```python
import numpy as np
a = np.random.random([2000,2000])
a = a + a.T
b = np.linalg.pinv(a)
print(np.amax(np.dot(a,b)))
```
Running MPI parallelized Python with mpi4py

MPI parallelized Python requires a valid MPI installation that support our SLURM scheduler. Thus anaconda is not the best option. We have installed MPI-supporting Python versions to different toolchains.

Using mpi4py is quite easy. Example is provided below.

Python mpi4py

A simple script mpi4py.py that utilizes mpi4py.

```python
#!/usr/bin/env python

""
Parallel Hello World
""

from mpi4py import MPI
import sys

size = MPI.COMM_WORLD.Get_size()
rank = MPI.COMM_WORLD.Get_rank()
name = MPI.Get_processor_name()

sys.stdout.write("Hello, World! I am process $d of $d on $s.\n"
% (rank, size, name))
```

Running mpi4py.py using only srun:

```
module load Python/2.7.11-goolf-triton-2016b
srun --time=00:10:00 -n 4 -p debug python helloworld.py
```

Example sbatch script mpi4py.slrm when running mpi4py.py through sbatch:

```
#!/bin/bash
#SBATCH --time=00:10:00
#SBATCH -n 4
#SBATCH -p debug

module load Python/2.7.11-goolf-triton-2016b
mpiexec -n $SLURM_NTASKS python mpi4py.py
```

R

R is a language and environment for statistical computing and graphics with wide userbase. There exists several packages that are easily imported to R.

Getting started

Simply load the latest R.

```
module load r
```

It is best to pick a version of R and stick with it. Do module spider r and use the whole name:
module load r/3.4.3-python-2.7.14

If you want to detect the number of cores, you should use the proper Slurm environment variables (defaulting to default):

library(parallel)
as.integer(Sys.getenv('SLURM_JOB_CPUS_PER_NODE', parallel::detectCores()))

Installing packages

There are two ways to install packages.

1. You can usually install packages yourself, which allows you to keep up to date and reinstall as needed. Good instructions can be found here, for example:

   R
   > install.packages('L1pack')

   This should guide you to selecting a download mirror and offer you the option to install in your home directory. Before installing packages you should set a package location, because the default location of the home directory can quickly fill up and loading them from the home directory is very slow. Example of doing this is here:

   module load R
   export R_LIBS=$WRKDIR/R/$EBVERSIONR
   mkdir -p $R_LIBS

   Afterwards setting

   export R_LIBS=$WRKDIR/R/$EBVERSIONR

   after loading R module will point R to the correct library location (you can put this in your .bashrc file). More info on R library paths can be found here. Looking at R startup can also be informative.

   2. You can also put a request to the triton issue tracker and mention which R-version you are using.

Simple R serial job

Serial R example

serial_R.slrm:

```bash
#!/bin/bash
#SBATCH -p short
#SBATCH -t 00:05:00
#SBATCH -n 1
#SBATCH --mem=1000
#SBATCH --mem=100
#SBATCH -o serial_R.out
module load R
n=3
m=2
srun Rscript --vanilla serial_R.R $n $m
```

serial_R.R:
```r
args = commandArgs(trailingOnly=TRUE)
n< as.numeric(args[1])
m< as.numeric(args[2])
print(n)
print(m)
A<-t(matrix(0:5,ncol=n,nrow=m))
print(A)
B<-t(matrix(2:7,ncol=n,nrow=m))
print(B)
C<-matrix(0.5,ncol=n,nrow=n)
print(C)
C<-A %*% t(B) + 2*C
print(C)
```

### Simple R job using OpenMP for parallelization

**R OpenMP Example**

```bash
#!/bin/bash
#SBATCH -p batch
#SBATCH -t 00:15:00
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=4
#SBATCH --mem=2G
#SBATCH -o r_openmp.out

module load R
export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK
time srun Rscript --default-packages=methods,utils,stats R-benchmark-25.R
```

The benchmark script is available [here](#) (more information about it is available [here](#) page).

### Simple R parallel job using `parallel`-package

**Parallel R example**

```bash
#!/bin/bash
#SBATCH -p short
#SBATCH -t 00:20:00
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=4
#SBATCH --mem=2G
#SBATCH -o r_parallel.out

# Set the number of OpenMP-threads to 1,
# as we're using parallel for parallelization
export OMP_NUM_THREADS=1
```

(continues on next page)
# Load the version of R you want to use
module load r

# Run your R script
srun Rscript r_parallel.R

r_parallel.R:

```r
callibrary(pracma)
callibrary(parallel)
callinvertRandom <- function(index) {
callA <- matrix(runif(2000*2000), ncol=2000, nrow=2000); 
callA <- A + t(A); 
callB <- pinv(A); 
callreturn(max(B %*% A)); 
}

ptm <- proc.time()
mclapply(1:16, invertRandom, mc.cores=Sys.getenv('SLURM_CPUS_PER_TASK'))
proc.time() - ptm
```

When constrained to opt-architecture, run times for different core numbers were

<table>
<thead>
<tr>
<th>ncores</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>runtime</td>
<td>380.757</td>
<td>182.185</td>
<td>125.526</td>
<td>84.230</td>
</tr>
</tbody>
</table>

RStan

- **supportlevel**: B
- **pagelastupdated**: 2018-07-26
- **maintainer**

RStan is an R interface to Stan. Stan is a platform for modeling.

Basic installation

RStan is installed as an R package and there is nothing too special about it.

First, load the R module you need to use. There are different options, using different compilers. Do **not** use an `iomkl` R version, because it requires the intel compilers to work on the nodes to compile every time you run, and they aren’t available there. If you load a `goolf` R version, it will work (you could work around this by pre-compiling models, if you wanted):

```
$ module spider R
...
R/3.4.1-goolf-triton-2017a
R/3.4.1-iomkl-triton-2017a
$ module load R/3.4.1-goolf-triton-2017a
```

If you change R versions (from intel to gcc) or get errors about loading libraries, you may have installed incompatible libraries. Removing your `~/R` directory and reinstalling all of your libraries is a good first place to start.
Notes

You should detect the number of cores with:

```r
as.integer(Sys.getenv('SLURM_JOB_CPUS_PER_NODE', parallel::detectCores()))
```

Common Rstan problems

- Models must be compiled on the machine that is running them, Triton or other workstations. The compiled model files aren’t necessarily portable, since they depend on the libraries available when build. One symptom of this problem is error messages which talk about loading libraries and `GLIBC_2.23` or some such.

- In order to compile models, you must have the compiler available on the nodes. Thus, the Intel compilers (`iomkl`) won’t work. It also won’t work if the Intel compiler license servers are down. Using the GNU compiler toolchains are more reliable.

Example

RStudio

```r
supportlevel C
pagelastupdated 2014

http://www.rstudio.com/ is an IDE for R

module load R/3.1.1-openblas boost/1.56 cmake/2.8.12.2 gcc/4.9.1 PrgEnv-gnu/0.1 qt/4.
˓→8.6
mkdir build && cd build
cmake .. -DRSTUDIO_TARGET=Desktop -DCMAKE_BUILD_TYPE=Release -DCMAKE_INSTALL_PREFIX=/
˓→share/apps/rstudio/0.98/ -DBOOST_ROOT=$BOOST_ROOT
```

Siesta & Transiesta

Copy-pasted Makefiles from Rocks. Should be used as a starting point. If you have a fully working version for SL6.2, send us a copy please.

See old wiki: https://wiki.aalto.fi/display/Triton/Applications

Rename siesta-3.0.arch.make.xxx => siesta-3.0-b/Obj/arch.make

Your own notebooks via sjupyter

Note: Now that Jupyterhub exists, this method of running Jupyter is not so important. It is only needed if you need more resources than JupyterHub can provide.

We provide a command `sjupyter` which automates launching your own notebooks in the Slurm queue. This gives you more flexibility in choosing your nodes and resources than Jupyterhub, but also will after your and your department’s Triton priority more because you are blocking others from using these resources.

6.1. Triton user guide
Set up the proxy

When running Jupyter on another system, the biggest problem is always making the connection securely. To do this here, we use a browser extension and SSH Proxy.

- Install the proxy extension
  - Install the extension FoxyProxy Standard (Firefox or Chrome). Some versions do not work properly: the 5.x series for Firefox may not work, but older and newer does.

- Create a new proxy rule with the pattern *.int.triton.aalto.fi* (or jupyter.triton.aalto.fi if you want to connect to that using the proxy).
  - Proxy type: SOCKS5, Proxy URL: localhost, port 8123.
  - DNS through the proxy: on.

- SSH to triton and use the `-D 8123`. This starts a proxy on your computer on port 8123. This has to always be running whenever you connect to the notebook.
  - If you are in Aalto networks: `ssh -D 8123 username@triton.aalto.fi`.
  - If you are not in Aalto networks, you need to do an extra hop through another Aalto server: `ssh -D 8123 username@triton.aalto.fi -o ProxyCommand='ssh username@kosh.aalto.fi -W %h:%p'`.

Now, when you go to any address matching *.int.triton.aalto.fi*, you will automatically connect to the right place on Triton. You can use Jupyter like normal. But if the ssh connection goes down, then you can’t connect and will get errors, so be aware (especially with jupyter.triton.aalto.fi which you might expect to always work).

Starting sjupyter

We have the custom-built command `sjupyter` for starting Jupyter on Triton.

To run in the Triton queue (using more resources), just use `sjupyter`. This will start a notebook on the interactive Slurm queue. All the normal rules apply: timelimits, memory limits, etc. If you want to request more resources, use the normal Slurm options such as `-t`, `--mem`, etc. Notebooks can only last as long as your job lasts, and you will need to restart them. Be efficient with resource usage: if you request a lot of resources and leave the notebook idle, no one else can use them. Thus, try to use the (default) interactive partition, which handles this automatically.

To run on the login node, run `sjupyter --local`. This is good for small testing and so on, which doesn’t use too much CPU or memory.

Spyder

```
  supportlevel C
  pagelastupdated 2014
```

Spyder is the Scientific PYthon Development EnviRonment:https://pythonhosted.org/spyder/

This guide shows you how to set this up with different version of Qt4 and python compared to the default version provided by operating system. Virtual environment makes this encapsulated from the rest of the environment and thus you can install different versions of python packages and also make the environment more portable.
Load pre-set environment modules

```bash
module load triton/python/2.7.6
module load qt/4.8.6
```

Setup you virtualenv

```bash
mkdir -p /local/mhhakala/virtualenv && cd /local/mhhakala/virtualenv
virtualenv spyder_env
source spyder_env/bin/activate
```

Install SIP + PyQt to the virtualenv

```bash
# note, that we now have the virtualenv spyder_env activated
# SIP/PyQt4 do not install with pip, so download first to some location
    tar zxf sip-4.16.7.tar.gz
    cd sip-4.16.7
    python configure.py
    make && make install

    tar zxf PyQt-x11-gpl-4.10.4.tar.gz
    cd PyQt-x11-gpl-4.10.4
    python configure.py
    make
    make install
```

Install spyder to the virtualenv

```bash
# still under activated spyder_env
pip install spyder
```

Tensorflow

- `supportlevel A`
- `pagelastupdated` 2019-06-03
- `maintainer`

Tensorflow is a commonly used Python package for deep learning.

Basic usage

First, check the tutorials up to and including `GPU computing`.

With tensorflow, you have to decide at `install time` if you want a version that runs on CPUs or GPUs. This means that we can’t install it for everyone and expect it to work everywhere - you have to load something different if you want it to run on login node/regular nodes (probably for testing) or GPU nodes. You probably want to use GPUs.
The basic way to use is via the Python in the anaconda3 module (or anaconda2) - but these modules have the GPU version installed, so you can’t run or test on the login node.

If you module spider anaconda3 (or 2), you can see several versions ending in -cpu or -gpu. These have respectively the CPU and GPU versions of tensorflow installed. Don’t load any additional CUDA modules, anaconda includes everything.

If you use GPUs, you need --constraint='kepler|pascal|volta' in order to select a GPU new enough to run tensorflow. (Note that as we get never cards, this will need further updating).

**Simple Tensorflow/Keras model**

Let’s run the MNIST example from Tensorflow’s tutorials:

```python
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(512, activation=tf.nn.relu),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])
```

The full code for the example is in tensorflow_mnist.py. One can run this example with `srun`:

```
wget https://raw.githubusercontent.com/AaltoScienceIT/scicomp-docs/master/triton/examples/tensorflow/tensorflow_mnist.py
module load anaconda3/latest
srun -t 00:15:00 --gres=gpu:1 python tensorflow_mnist.py
```

or with `sbatch` by submitting `tensorflow_mnist.sh`:

```
#!/bin/bash
#SBATCH --gres=gpu:1
#SBATCH --time=00:15:00

module load anaconda3/latest
python tensorflow_mnist.py
```

Do note that by default Keras downloads datasets to `~/.keras/datasets`.

**Common problems**

- **ImportError: libcuda.so.1: cannot open shared object file: No such file or directory.** GPU tensorflow can only be imported on GPU nodes (even though you’d think that you can import it and just not use the GPUs). So you can only run this code in the GPU queue. You could try something where you use CPU tensorflow for testing on login and GPU tensorflow for running in batch.

- Random CUDA errors: don’t load any other CUDA modules, only anaconda. Anaconda includes the necessary libraries in compatible versions.

**Theano**

```bash
supportlevel
pagelastupdated
```
If you’re using the theano library, you need to tell theano to store compiled code on the local disk on the compute
node. Create a file ~/.theanorc with the contents

```
[global]
base_compiledir=/tmp/%(user)s/theano
```

Also make sure that in your batch job script you create this directory before you launch theano. E.g.

```
mkdir -p /tmp/${USER}/theano
```

The problem is that by default the base_compiledir is in your home directory (~/.theano/), and then if you
first happen to run a job on a newer processor, a later job that happens to run on an older processor will crash with an
“Illegal instruction” error.

**VASP**

**VASP** (Vienna Ab initio Simulation Package) is a computer program for atomic scale materials modelling, e.g.
electronic structure calculations and quantum-mechanical molecular dynamics, from first principles.

VASP is licensed software, requiring the licensee to keep the vasp team updated with a list of user names. Thus, in
order to use VASP arrange with the “vaspmaster” for your group to be put on the vasp licensed user list. Afterwards,
contact your local triton admin who will take care of the IT gymnastics, and CC the vaspmaster so that he is aware of
who gets added to the list.

For the PHYS department, the vaspmaster is Janne Blomqvist.

For each VASP version, there are 3 binaries compiled. All versions are MPI versions.

- **vasp_std**: The “standard” vasp, compiled with NGZhalf
- **vasp_gam**: Gamma point only. Faster if you use only a single k-point.
- **vasp_ncl**: For non-collinear spin calculations

**VASP 5.4.4**

The binaries are compiled with the Intel compiler suite and the MKL library, the used toolchain module is iomklc/
triton-2017a. Example batch script

```
#!/bin/bash -l
#SBATCH -n 8
#SBATCH -t 0-6
#SBATCH --mem-per-cpu=1500
ml vasp/5.4.4
srun vasp_std
```

**VASP 5.4.1**

Currently the binaries are compiled with GFortran instead of Intel Fortran (the Intel Fortran binaries crashed, don’t
know why yet). Example batch script
#!/bin/bash -l
#SBATCH -n 8
#SBATCH -p batch
#SBATCH -t 0-6
#SBATCH --mem-per-cpu=1500
module load vasp/5.4.1-gmvolf-triton-2016a
srun vasp_std

Potentials

Potentials are stored at /share/apps/vasp/pot.

VASP on old triton (obsolete, for reference only!)

For each VASP version, there are two binaries compiled with slightly different options:

```bash
vasp.mpi.NGZhalf
vasp.mpi
```

Both are MPI versions. The first one is what you should normally use; it is compiled with the NGZhalf option which reduces charge density in the Z direction, leading to less memory usage and faster computation. The second version is needed for non-collinear spin calculations. The binaries can be found in the directory /share/apps/vasp/$VERSION/.

For those of you who need to compile your own version of VASP, the makefiles used for these builds can be used as a starting point, and are found in the directory /share/apps/vasp/makefiles.

VASP 5.3.5

The binaries are optimized for the Xeon Ivy Bridge nodes, although they will also work fine on the older Xeon Westmere and Opteron nodes. Note that for the moment only the NGZhalf version has been built. If you need the non-NGZhalf version for non-collinear spin calculations please contact triton support. Example job script below:

```bash
#!/bin/sh
#SBATCH -p batch
#SBATCH -N 1
#SBATCH -t 0-5 # 5 hours
#SBATCH -n 12
#SBATCH --mem-per-cpu=2500
#SBATCH --constraint=[xeonib|xeon|opteron]
module load vasp/5.3.5
srun vasp.mpi.NGZhalf
```

The relative time to run the vasptest v2 testsuite on 12 cores (so a full node for Xeon Westmere and Opteron nodes, and 12/20 cores on a Xeon Ivy Bridge node) is for Xeon IB/Xeon Westmere/Opteron 1.0/2.0/2.8. So one sees that the Xeon Ivy Bridge nodes are quite a lot faster per core than the older nodes (with the caveat that the timings may vary depending on other jobs that may have been running on the Xeon IB node during the benchmark).
**VASP 5.3.3**

The binaries are optimized for the Xeon nodes, although they also work on the Opteron nodes. Some simple benchmarks suggest that the Opteron nodes are a factor of 1.5 slower than the Xeon nodes, although it is recommended to write the batch script such that Opteron nodes can also be used, as the Opteron queue is often shorter. An example script below:

```bash
#!/bin/sh
#SBATCH -p batch
#SBATCH -N 1
#SBATCH -t 0-5 # 5 hours
#SBATCH -n 12
#SBATCH --mem-per-cpu=2500
#SBATCH --constraint=[xeon|opteron]
module load vasp/5.3.3
srun vasp.mpi.NGZhalf
```

**VASP 5.3.2 and older**

The binaries are optimized for the Intel Xeon architecture nodes, and are not expected to work on the Opteron nodes. An example job script is below (Note that it is different from the script for version 5.3.3 and newer above!):

```bash
#!/bin/sh
#SBATCH -p batch
#SBATCH -N 1
#SBATCH -t 1-0 # 1 day
#SBATCH -n 12
#SBATCH --mem-per-cpu=3500
#SBATCH --constraint=xeon
module load vasp/5.3.2
srun vasp.mpi.NGZhalf
```

**Potentials**

PAW potentials for VASP can be found in the directory `/share/apps/vasp/pot`. The recommended potentials are the ones in the Apr2012.52 subdirectory. For reference, an older set of potentials dating back to 2003 can be found in the “2003” subdirectory.

**Validation**

The vasp.mpi.NGZhalf builds have been verified to pass all the tests in the `vasptest` suite.

**Other**

Old makefiles

Here is a number of Makefiles copy-pasted from old Rocks installation. Can be useful in general, though may require adaptation to new installation. Please, send us a fully working copy if you have one.
Aalto scientific computing guide

See old wiki: https://wiki.aalto.fi/display/Triton/Applications

Rename vasp.x.y.makefile => vasp.x.y/makefile

VisIT

This uses Singularity containers, so you should refer to that page first for general information.

Visit has been compiled using the build_visit-script from the VisIT page on an Ubuntu image. It has minimal amount of other software installed.

Parallelization is done against Triton’s OpenMPI, so using this container with other OpenMPI modules is discouraged.

Within the container VisIT is installed under /opt/visit/. PATH is automatically appended with their respective paths so all program calls are available automatically.

Usage

This example shows how you can launch visit on the login node for small visualizations or launch it in multiprocess state on a reserved node. Firstly, let’s load the module:

```bash
module use /share/apps2/singularity/modules
module load Visit
```

Now you can run VisIT with:

```bash
singularity_wrapper exec visit
```

If you want to run VisIT with multiple CPUs, you should reserve a node with sinteractive:

```bash
sinteractive -t 00:30:00 -n 2 -N 1-1
singularity_wrapper exec visit -np 2
```

Do note the flag -N 1-1 that ensures that all of VisITs processes end up on the same node. Currently VisIT encounters problems when going across the node lines.

6.1.6 Reference and Examples

Triton quick reference

In this page, you have all important reference information
Modules

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>module load \textit{NAME}</td>
<td>load module</td>
</tr>
<tr>
<td>module avail</td>
<td>list all modules</td>
</tr>
<tr>
<td>module spider \textit{NAME}</td>
<td>search modules</td>
</tr>
<tr>
<td>module list \textit{NAME}</td>
<td>list currently loaded modules</td>
</tr>
<tr>
<td>module show \textit{NAME}</td>
<td>details on a module</td>
</tr>
<tr>
<td>module help \textit{NAME}</td>
<td>details on a module</td>
</tr>
<tr>
<td>module unload \textit{NAME}</td>
<td>unload a module</td>
</tr>
<tr>
<td>module save \textit{ALIAS}</td>
<td>save module to this alias (saved in ~/.lmod.d/)</td>
</tr>
<tr>
<td>module restore \textit{ALIAS}</td>
<td>load saved module set (faster than loading individually)</td>
</tr>
<tr>
<td>module purge</td>
<td>unload all loaded modules (faster than unloading individually)</td>
</tr>
</tbody>
</table>

Storage

<table>
<thead>
<tr>
<th>Name</th>
<th>Path</th>
<th>Quota</th>
<th>Backup</th>
<th>Locality</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Home</td>
<td>$\text{HOME}$ or $\text{/home/}$ or $\text{/home/}$ $\text{username/}$</td>
<td>hard quota 10GB</td>
<td>Nightly</td>
<td>all nodes</td>
<td>Small user specific files, no calculation data.</td>
</tr>
<tr>
<td>Work</td>
<td>$\text{WRKDIR}$ or $\text{/scratch/}$ $\text{work/}$ or $\text{/home/}$ $\text{username/}$</td>
<td>200GB and 1 million files</td>
<td>x</td>
<td>all nodes</td>
<td>Personal working space for every user. Calculation data etc. Quota can be increased on request.</td>
</tr>
<tr>
<td>Scratch</td>
<td>$\text{/scratch/}$ $\text{dept/}$ $\text{project/}$</td>
<td>on request</td>
<td>x</td>
<td>all nodes</td>
<td>Department/group specific project directories.</td>
</tr>
<tr>
<td>Local temp</td>
<td>$\text{/tmp/}$</td>
<td>limited by disk size</td>
<td>x</td>
<td>single-node</td>
<td>Primary (and usually fastest) place for single-node calculation data. Removed once user’s jobs are finished on the node.</td>
</tr>
<tr>
<td>Local persis-</td>
<td>$\text{/l/}$</td>
<td>varies</td>
<td>x</td>
<td>dedicated group servers only</td>
<td>Local disk persistent storage. On servers purchased for a specific group. Not backed up.</td>
</tr>
<tr>
<td>tent</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ramfs</td>
<td>$\text{$\text{XDG_RUNTIME_D$}$}$</td>
<td>limited by memory</td>
<td>x</td>
<td>single-node</td>
<td>Ramfs on the login node only, in-memory filesystem</td>
</tr>
<tr>
<td>(login nodes only)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Partitions

<table>
<thead>
<tr>
<th>Partition</th>
<th>Max size</th>
<th>Mem/core (GB)</th>
<th>Tot mem (GB)</th>
<th>Cores/node limits</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;default&gt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>debug</td>
<td>2 nodes</td>
<td>2.66 - 12</td>
<td>32-256</td>
<td>12,20,24</td>
<td>15 min testing and debugging short interactive. work. 1 node of each arch.</td>
</tr>
<tr>
<td>batch</td>
<td>16 nodes</td>
<td>2.66 - 12</td>
<td>32-256</td>
<td>12, 20,24</td>
<td>5d primary partition, all serial &amp; parallel jobs</td>
</tr>
<tr>
<td>short</td>
<td>8 nodes</td>
<td>4 - 12</td>
<td>48-256</td>
<td>12, 20,24</td>
<td>4h short serial &amp; parallel jobs, +96 dedicated CPU cores</td>
</tr>
<tr>
<td>huge-mem</td>
<td>1 node</td>
<td>43</td>
<td>1024</td>
<td>24</td>
<td>3d huge memory jobs, 1 node only</td>
</tr>
<tr>
<td>gpu</td>
<td>1 node, 2-8GPUs</td>
<td>2 - 10</td>
<td>24-128</td>
<td>12</td>
<td>5d GPU computing</td>
</tr>
<tr>
<td>gpushort</td>
<td>4 nodes, 2-8 GPUs</td>
<td>2 - 10</td>
<td>24-128</td>
<td>12</td>
<td>4h GPU computing</td>
</tr>
<tr>
<td>interactive</td>
<td>2 nodes</td>
<td>5</td>
<td>128</td>
<td>24</td>
<td>1d for sinteractive command, longer interactive work</td>
</tr>
</tbody>
</table>

Use slurm partitions to see more details.

Job submission

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sbatch</td>
<td>submit a job to queue (see standard options below)</td>
</tr>
<tr>
<td>srun</td>
<td>Within a running job script/environment: Run code using the allocated resources (see options below)</td>
</tr>
<tr>
<td>srun</td>
<td>On frontend: submit to queue, wait until done, show output. (see options below)</td>
</tr>
<tr>
<td>sinteract</td>
<td>Submit job, wait, provide shell on node for interactive playing (X forwarding works, default partition interactive). Exit shell when done. (see options below)</td>
</tr>
<tr>
<td>srun --pty bash</td>
<td>(advanced) Another way to run interactive jobs, no X forwarding but simpler. Exit shell when done.</td>
</tr>
<tr>
<td>scancel &lt;jobid&gt;</td>
<td>Cancel a job in queue</td>
</tr>
<tr>
<td>salloc</td>
<td>(advanced) Allocate resources from frontend node. Use srun to run using those resources, exit to close shell when done. Read the description! (see options below)</td>
</tr>
<tr>
<td>scontrol</td>
<td>View/modify job and slurm configuration</td>
</tr>
<tr>
<td>Command</td>
<td>Option</td>
</tr>
<tr>
<td>---------</td>
<td>--------</td>
</tr>
<tr>
<td>sbatch</td>
<td><code>-t</code></td>
</tr>
<tr>
<td>sbatch</td>
<td><code>-p</code>, <code>--partition</code></td>
</tr>
<tr>
<td>sbatch</td>
<td><code>--mem-per-cpu=</code></td>
</tr>
<tr>
<td>sbatch</td>
<td><code>--mem=</code></td>
</tr>
<tr>
<td>sbatch</td>
<td><code>-c</code>, <code>--cpus-per-task=</code></td>
</tr>
<tr>
<td>sbatch</td>
<td><code>--nodes=n-m</code></td>
</tr>
<tr>
<td>sbatch</td>
<td><code>--ntasks=n</code></td>
</tr>
<tr>
<td>sbatch</td>
<td><code>-J</code>, <code>--job-name=</code></td>
</tr>
<tr>
<td>sbatch</td>
<td><code>-o output</code></td>
</tr>
<tr>
<td>sbatch</td>
<td><code>-e error</code></td>
</tr>
<tr>
<td>sbatch</td>
<td><code>--exclusive</code></td>
</tr>
<tr>
<td>sbatch</td>
<td><code>--constraint=</code></td>
</tr>
<tr>
<td>sbatch</td>
<td><code>--array=0-5,7,10-15</code></td>
</tr>
<tr>
<td>sbatch</td>
<td><code>--gres=gpu</code></td>
</tr>
<tr>
<td>sbatch</td>
<td><code>--gres=spindle</code></td>
</tr>
<tr>
<td>sbatch</td>
<td><code>--mail-type=</code></td>
</tr>
<tr>
<td>sbatch</td>
<td><code>--mail-user=</code></td>
</tr>
<tr>
<td>srun</td>
<td><code>-N</code></td>
</tr>
</tbody>
</table>

### Full slurm command help:

```
$ slurm
```

(continues on next page)
Show or watch job queue:
slurm [watch] queue show own jobs
slurm [watch] q show user's jobs
slurm [watch] quick show quick overview of own jobs
slurm [watch] shorter sort and compact entire queue by job size
slurm [watch] short sort and compact entire queue by priority
slurm [watch] full show everything
slurm [w] [q|qq|ss|s|f] shorthands for above!
slurm qos show job service classes
slurm top [queue|all] show summary of active users

Show detailed information about jobs:
slurm prio [all|short] show priority components
slurm j|job show everything else
slurm steps show memory usage of running srun job steps

Show usage and fair-share values from accounting database:
slurm h|history show jobs finished since, e.g. "1day" (default)
slurm shares

Show nodes and resources in the cluster:
slurm p|partitions all partitions
slurm n|nodes all cluster nodes
slurm cpus total cpu cores in use
slurm cpus cores available to partition, allocated and free
slurm cpus jobs cores/memory reserved by running jobs
slurm cpus queue cores/memory required by pending jobs
slurm features list features and GRES

Examples:
slurm q
slurm watch shorter
slurm cpus batch
slurm history 3hours

Other advanced commands (many require lots of parameters to be useful):

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>squeue</td>
<td>Full info on queues</td>
</tr>
<tr>
<td>sinfo</td>
<td>Advanced info on partitions</td>
</tr>
<tr>
<td>slurm nodes</td>
<td>List all nodes</td>
</tr>
</tbody>
</table>
## Toolchains

<table>
<thead>
<tr>
<th>Toolchain</th>
<th>Compiler version</th>
<th>MPI version</th>
<th>BLAS version</th>
<th>ScaLAPACK version</th>
<th>FFTW version</th>
<th>CUDA version</th>
</tr>
</thead>
<tbody>
<tr>
<td>GOOLF Toolchains:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>goofl/triton-2016a</td>
<td>GCC/4.9.3</td>
<td>OpenMPI/1.10</td>
<td>2OpenBLAS/0.2</td>
<td>ScaLAPACK/2</td>
<td>FFTW/3.3.4</td>
<td></td>
</tr>
<tr>
<td>goofl/triton-2016b</td>
<td>GCC/5.4.0</td>
<td>OpenMPI/1.10</td>
<td>3OpenBLAS/0.2</td>
<td>ScaLAPACK/2</td>
<td>FFTW/3.3.4</td>
<td></td>
</tr>
<tr>
<td>gooflfc/triton-2016a</td>
<td>GCC/4.9.3</td>
<td>OpenMPI/1.10</td>
<td>2OpenBLAS/0.2</td>
<td>ScaLAPACK/2</td>
<td>FFTW/3.3.4</td>
<td>7.5.18</td>
</tr>
<tr>
<td>gooflfc/triton-2017a</td>
<td>GCC/5.4.0</td>
<td>OpenMPI/2.0.1</td>
<td>OpenBLAS/0.2</td>
<td>ScaLAPACK/2</td>
<td>FFTW/3.3.4</td>
<td>8.0.61</td>
</tr>
<tr>
<td>GMPOLF Toolchains:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>gmpolf/triton-2016a</td>
<td>GCC/4.9.3</td>
<td>MPICH/3.0.4</td>
<td>OpenBLAS/0.2</td>
<td>ScaLAPACK/2</td>
<td>FFTW/3.3.4</td>
<td></td>
</tr>
<tr>
<td>gmpolfc/triton-2016a</td>
<td>GCC/4.9.3</td>
<td>MPICH/3.0.4</td>
<td>OpenBLAS/0.2</td>
<td>ScaLAPACK/2</td>
<td>FFTW/3.3.4</td>
<td>7.5.18</td>
</tr>
<tr>
<td>GMVOLF Toolchains:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>gmvolf/triton-2016a</td>
<td>GCC/4.9.3</td>
<td>MVAPICH2/2.0</td>
<td>OpenBLAS/0.2</td>
<td>ScaLAPACK/2</td>
<td>FFTW/3.3.4</td>
<td></td>
</tr>
<tr>
<td>gmvolfc/triton-2016a</td>
<td>GCC/4.9.3</td>
<td>MVAPICH2/2.0</td>
<td>OpenBLAS/0.2</td>
<td>ScaLAPACK/2</td>
<td>FFTW/3.3.4</td>
<td>7.5.18</td>
</tr>
<tr>
<td>IOOLF Toolchains:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ioolf/triton-2016a</td>
<td>icc/2015.3.187</td>
<td>OpenMPI/1.10</td>
<td>2OpenBLAS/0.2</td>
<td>ScaLAPACK/2</td>
<td>FFTW/3.3.4</td>
<td></td>
</tr>
<tr>
<td>IOMKLI Toolchains:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>iomkl/triton-2016a</td>
<td>icc/2015.3.187</td>
<td>OpenMPI/1.10</td>
<td>2imkl/11.3.1.150</td>
<td>imkl/11.3.1.150</td>
<td>imkl/11.3.1.150</td>
<td></td>
</tr>
<tr>
<td>iomkl/triton-2016b</td>
<td>icc/2015.3.187</td>
<td>OpenMPI/1.10</td>
<td>3imkl/11.3.1.150</td>
<td>imkl/11.3.1.150</td>
<td>imkl/11.3.1.150</td>
<td></td>
</tr>
</tbody>
</table>

### 6.1. Triton user guide
## Hardware

<table>
<thead>
<tr>
<th>Node name</th>
<th>Number of nodes</th>
<th>Node type</th>
<th>Year</th>
<th>Arch (constraint)</th>
<th>CPU type</th>
<th>Memory Configuration</th>
<th>GPUs</th>
</tr>
</thead>
<tbody>
<tr>
<td>wsm[1-112,137-144]</td>
<td>120</td>
<td>ProLiant SL390s G7</td>
<td></td>
<td>wsm</td>
<td>2x6 core Intel Xeon X5650 2.67GHz</td>
<td>48GB DD3-1333</td>
<td></td>
</tr>
<tr>
<td>wsm[113-136]</td>
<td>24</td>
<td>ProLiant SL390s G7</td>
<td></td>
<td>wsm</td>
<td>2x6 core Intel Xeon X5650 2.67GHz</td>
<td>96GB DD3-1333</td>
<td></td>
</tr>
<tr>
<td>ivb[1-24]</td>
<td>24</td>
<td>ProLiant SL230s G8</td>
<td></td>
<td>ivb,avx</td>
<td>2x10 core Xeon E5 2680 v2 2.80GHz</td>
<td>256GB DDR3-1667</td>
<td></td>
</tr>
<tr>
<td>ivb[25-48]</td>
<td>24</td>
<td>ProLiant SL230s G8</td>
<td></td>
<td>ivb,avx</td>
<td>2x10 core Xeon E5 2680 v2 2.80GHz</td>
<td>64GB DDR3-1667</td>
<td></td>
</tr>
<tr>
<td>pe[1-48,65-81]</td>
<td>65</td>
<td>Dell PowerEdge C4130</td>
<td>2016</td>
<td>hsw,avx,avx2</td>
<td>2x12 core Xeon E5 2680 v3 2.50GHz</td>
<td>128GB DDR4-2133</td>
<td></td>
</tr>
<tr>
<td>pe[49-64,82]</td>
<td>17</td>
<td>Dell PowerEdge C4130</td>
<td>2016</td>
<td>hsw,avx,avx2</td>
<td>2x12 core Xeon E5 2680 v3 2.50GHz</td>
<td>256GB DDR4-2133</td>
<td></td>
</tr>
<tr>
<td>pe[83-91]</td>
<td>8</td>
<td>Dell PowerEdge C4130</td>
<td>2016</td>
<td>bwd,avx,avx2</td>
<td>2x14 core Xeon E5 2680 v4 2.40GHz</td>
<td>128GB DDR4-2400</td>
<td></td>
</tr>
<tr>
<td>skl[1-48]</td>
<td>48</td>
<td>Dell PowerEdge C6420</td>
<td>2019</td>
<td>skl,avx,avx2,avx54</td>
<td>2x20 core Xeon Gold 6148 2.40GHz</td>
<td>192GB DDR4-2667</td>
<td></td>
</tr>
<tr>
<td>gpu[20-22]</td>
<td>3</td>
<td>Dell PowerEdge C4130</td>
<td>2016</td>
<td>bwd,avx,avx2</td>
<td>2x12 core Xeon E5 2620 v3 2.50GHz</td>
<td>128GB DDR4-2133</td>
<td>4x2 GPU K80</td>
</tr>
<tr>
<td>gpu[23-27]</td>
<td>5</td>
<td>Dell PowerEdge C4130</td>
<td>2017</td>
<td>bwd,avx,avx2</td>
<td>2x20 core Xeon E5 2698 v4 @ 2.2GHz</td>
<td>512GB DDR4-2133</td>
<td>8x V100</td>
</tr>
<tr>
<td>dgx[01-02]</td>
<td>2</td>
<td>Nvidia DGX-1</td>
<td>2018</td>
<td>bwd,avx,avx2</td>
<td>2x20 core Xeon E5 2698 v4 @ 2.2GHz</td>
<td>512GB DDR4-2133</td>
<td>8x V100</td>
</tr>
<tr>
<td>gpu[28-37]</td>
<td>10</td>
<td>Dell PowerEdge C4140</td>
<td>2019</td>
<td>skl,avx,avx2,avx54</td>
<td>2x20 core Xeon Gold 6134 @ 3.2GHz</td>
<td>384GB DDR4-2667</td>
<td>4x V100 32GB</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Node type</th>
<th>CPU count</th>
</tr>
</thead>
<tbody>
<tr>
<td>48GB Xeon Westmere (2012)</td>
<td>1404</td>
</tr>
<tr>
<td>24GB Xeon Westmere + 2x GPU (2012)</td>
<td>120</td>
</tr>
<tr>
<td>96GB Xeon Westmere (2012)</td>
<td>288</td>
</tr>
<tr>
<td>1TB Xeon Westmere (2012)</td>
<td>48</td>
</tr>
<tr>
<td>256GB Xeon Ivy Bridge (2014)</td>
<td>480</td>
</tr>
<tr>
<td>64GB Xeon Ivy Bridge (2014)</td>
<td>480</td>
</tr>
<tr>
<td>128GB Xeon Haswell (2016)</td>
<td>1224</td>
</tr>
<tr>
<td>256GB Xeon Haswell (2016)</td>
<td>360</td>
</tr>
<tr>
<td>128GB Xeon Haswell + 4x GPU (2016)</td>
<td>36</td>
</tr>
</tbody>
</table>

### Examples
Master-Worker Example

Following example shows how to manage host list using the python-hostlist package and run different tasks for master task and worker task.

This kind of structure might be needed if one wants to create a e.g. Spark cluster or use some other program that uses master-worker-paradigm, but does not use MPI.

It is important to make sure that in case of job cancellation all programs started by the scripts will be killed gracefully. In case of Spark or other programs that initialize a cluster using SSH and then forking a process, these forked processes must be killed after job allocation has ended.

hostlist-test.slrm:

```bash
#!/bin/bash
#SBATCH --t 00:10:00
#SBATCH --N 3
#SBATCH --n 5
#SBATCH --p batch
#SBATCH --o hostlist-test.out

# An example of a clean_up-routine if the master has to take e.g. ssh connection to
# start program on workers
function clean_up {
    echo "Got SIGTERM, will clean up my workers and exit."
    exit
}
trap clean_up SIGHUP SIGINT SIGTERM

# Actual script that defines what each worker will do
srun bash run.sh
```

run.sh:

```bash
#!/bin/bash

# Get a list of hosts using python-hostlist
nodes=`hostlist --expand $SLURM_NODELIST|xargs`

# Determine current worker name
me=$(hostname)

# Determine master process (first node, id 0)
master=$(echo $nodes | cut -f 1 -d ' ')

# SLURM_LOCALID contains task id for the local node
localid=${SLURM_LOCALID}

if [[ "$me" == "$master" && "$localid" -eq 0 ]]; then
    # Run these if the process is the master task
    echo "I'm the master with number "$localid" in node "$me". My subordinates are "$nodes"
    else
    # Run these if the process is a worker
    echo "I'm a worker number "$localid" in node "$me"
fi
```

Example output:
I'm a worker number 1 in node opt469
I'm a worker number 2 in node opt469
I'm the master with number 0 in node opt469. My subordinates are opt469 opt470 opt471
I'm a worker number 0 in node opt471
I'm a worker number 0 in node opt470

Python OpenMP example

parallel_Python.slrm:

#!/bin/bash
#SBATCH -p short
#SBATCH -t 00:10:00
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=4
#SBATCH --mem-per-cpu=2G
#SBATCH -o parallel_Python.out

module load anaconda3
export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK
srun -c $SLURM_CPUS_PER_TASK python parallel_Python.py

parallel\Python.py:

```
import numpy as np
a = np.random.random([2000,2000])
a = a + a.T
b = np.linalg.pinv(a)
print(np.amax(np.dot(a,b)))
```

Serial R example

serial_R.slrm:

#!/bin/bash
#SBATCH -p short
#SBATCH -t 00:05:00
#SBATCH -n 1
#SBATCH --mem=100
#SBATCH -o serial_R.out
module load R
n=3
m=2
srun Rscript --vanilla serial_R.R $n $m

serial_R.R:

```
args = commandArgs(trailingOnly=TRUE)
n<-as.numeric(args[1])
m<-as.numeric(args[2])
print(n)
print(m)
```
(continues on next page)
A<-t(matrix(0:5,ncol=n,nrow=m))
print(A)
B<-t(matrix(2:7,ncol=n,nrow=m))
print(B)
C<-matrix(0.5,ncol=n,nrow=n)
print(C)
C<-A %*% t(B) + 2*C
print(C)

Parallel R example

r_parallel.slrm:

#!/bin/bash
#SBATCH -p short
#SBATCH -t 00:20:00
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=4
#SBATCH --mem=2G
#SBATCH -o r_parallel.out

# Set the number of OpenMP-threads to 1, 
# as we're using parallel for parallelization
export OMP_NUM_THREADS=1

# Load the version of R you want to use
module load r

# Run your R script
srun Rscript r_parallel.R

r_parallel.R:

library(pracma)
library(parallel)
invertRandom <- function(index) {
  A<-matrix(runif(2000*2000),ncol=2000,nrow=2000);
  A<-A + t(A);
  B<-pinv(A);
  return(max(B %*% A));
}
ptm<-proc.time()
mclapply(1:16,invertRandom, mc.cores=Sys.getenv('SLURM_CPUS_PER_TASK'))
proc.time()-ptm

When constrained to opt-architecture, run times for different core numbers were

<table>
<thead>
<tr>
<th>ncores</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>runtime</td>
<td>380.757</td>
<td>182.185</td>
<td>125.526</td>
<td>84.230</td>
</tr>
</tbody>
</table>
R OpenMP Example

r_openmp.slrm:

```bash
#!/bin/bash
#SBATCH -p batch
#SBATCH -t 00:15:00
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=4
#SBATCH --mem=2G
#SBATCH -o r_openmp.out

module load R
export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK

time srun Rscript --default-packages=methods,utils,stats R-benchmark-25.R
```

The benchmark script is available here (more information about it is available here page).

Python

IPython parallel

A example batch script that uses IPython parallel (ipyparallel) within slurm. See also the interactive hints on the Python page.

ipyparallel uses global state in your home directory, so you can only run _one_ of these at a time! You can add the --profile= option to name different scripts (you could use $SLURM_JOB_ID). But then you will get a growing number of unneeded profile directories at ~/.ipython/profile_*, so this isn’t recommended. Basically, ipy-parallel is more designed for one-at-a-time interactive use rather than batch scripting (unless you do more work…).

ipyparallel.slrm is an example slurm script that sets up ipyparallel. It assumes that most work is done in the engines. It has inline Python, replace this with python your_script_name.py

```bash
#!/bin/bash
#SBATCH -N 4

module load anaconda3
set -x

ipcontroller --ip="*" &
sleep 5
# Run the engines in slurm job steps (makes four of them, since we use # the -N slurm option)...
srun ipengine --location=`hostname -f` &
sleep 5
# Put the actual Python isn't in a job step. This is assuming that # most work happens in engines
python3 <<EOF
import os
import ipyparallel
client = ipyparallel.Client()
result = client[:].apply_async(os.getpid)
pid_map = result.get_dict()
EOF
```

(continues on next page)
Python MPI4py

A simple script mpi4py.py that utilizes mpi4py.

```python
#!/usr/bin/env python

"""
Parallel Hello World
"""

from mpi4py import MPI

import sys

size = MPI.COMM_WORLD.Get_size()
rank = MPI.COMM_WORLD.Get_rank()
name = MPI.Get_processor_name()

sys.stdout.write(
    "Hello, World! I am process %d of %d on %s\n"
    % (rank, size, name))
```

Running mpi4py.py using only srun:

```bash
module load Python/2.7.11-goolf-triton-2016b
srun --time=00:10:00 -n 4 -p debug python helloworld.py
```

Example sbatch script mpi4py.slrm when running mpi4py.py through sbatch:

```bash
#!/bin/bash

#SBATCH --time=00:10:00
#SBATCH -n 4
#SBATCH -p debug

module load Python/2.7.11-goolf-triton-2016b
mpiexec -n $SLURM_NTASKS python mpi4py.py
```

Cheatsheets: Triton
In this section, you find general (not Aalto specific) scientific computing resources.

### 7.1 Scientific computing

#### 7.1.1 Encryption for researchers

This page describes the basics of encryption to an audience of researchers. It describes how it may be useful (and when not needed) in a professional researcher environment, in order to secure data. It doesn’t describe encryption for personal use (there are plenty of other guides for that). It doesn’t go into very deep details about cryptography. It doesn’t get into deep details. Cryptography is the type of things where there are a huge number of subtle details, and everyone has their own opinion. This guide is designed to provide an understanding for basic use, not cover every single subtle point.

Status: this is somewhat complete, but is not a complete guide. It will be extended as needed.

**Summary**

Modern cryptography is quite well developed, and available many places. However, the human side is very difficult. Encrypting something, but not keeping the key or password secure, has no benefits. To use your encryption, you need to decide what your goals are (who should access, who you want to keep safe from) and then plan accordingly. The security of cryptography is decided more by how you manage the keys and process than the deepest technical details.

**Key management**

The point of encryption is to trade a hard problem (keeping a lot of data secure) to a more limited problem (keeping a single key or password secure). These keys can be managed separately from the data. This immediately takes us to the importance of key management. Let’s say you can’t send data over email unless it is encrypted. If you encrypt it and send the password in the same email as the encrypted data, you have managed to technically satisfy the requirement while adding no real security at all. A better strategy would be to give the password to someone when you meet them in
person, send it by another channel (e.g. SMS, but then it is only secure as SMS+email), or even better use asymmetric encryption (see below).

Deciding how you will manage keys is the hardest part of using encryption. For some fun, next time you hear someone talk about using encryption, see if they mention how they keep the keys secure. Usually, the don’t, and you have no way of knowing if they actually are doing it securely.

**Symmetric vs asymmetric encryption**

There are two main types of cryptography. They can both be considered equally secure, but have different ways of managing keys.

**Symmetric encryption** uses the same password/key for encrypting and decrypting. It is good because it is simple, because there is only one key or password you need to know and it is easy to think “one data=one password”. However, everyone needs to know the same password, and it can’t be changed. Since the same password has to be everywhere, this can be a bit insecure depending on the use, and you can argue it’s a bit complicated to keep that key password secure (if there are many people, or if it needs to be automated).

**Asymmetric encryption** has different keys for encrypting and decrypting. So, you use a “public key” to do an encryption (which requires no password - everyone in the world can know this key and your data is still secure). You have a separate private key (+password) which allows only you to decrypt it. This separation of encryption and decryption was a major mathematical breakthrough. Then, anyone who needed to receive data securely would have their own public/private key, and all the public keys are, well, public. When you want to send data to someone, you just encrypt it using their public key, and there is no need to manage sharing a password. This allows you to: encrypt so that multiple people can read it, encrypt automatically without password, and encrypt to someone not involved in the initial process.

With asymmetric encryption, there are some more things to consider. How do you make sure that you have the right public key?

**Encryption programs**

This lists some common programs, but this should not be taken to mean that using these programs makes your data safe. Security depends no how you use the program, and security will only decrease over time as new analysis is done. It is usually best to choose well-supported open source programs where possible. More detailed instructions will be provided as needed.

**7zip**

7zip is a file archiver (like zip). It can symmetrically encrypt files with a passphrase.

**PGP**

PGP is a set of encryption standards (and also a program). It has a full suite of encryption tools, and is quite stable and well-supported. You often hear about PGP in the context of email encryption, but it can be used for many things.

On Linux systems, it is normally found as the program gpg (Gnu Privacy Guard). This guide uses gpg.

**Full disk encryption**

Programs can encrypt the entire hard disk of your computer. This means that any data on it is safe, should your computer be lost. There are programs to do this for every operating system, and Aalto laptops now come encrypted by
Using symmetric encryption with gpg

Encryption:

```bash
gpg --symmetric input.file
```

Decryption:

```bash
gpg input.file.gpg
```

This will ask you for a password. If you do not want it to, you can use `--passphrase-fd` to pass it automatically. Normally, keeping a password in a file is considered quite insecure! Make sure that the permissions are restrictive. Anyone that can read this file once be able to read your data forever. The file could be backed up and spread all over the place - is that what you want? IT admins will be technically able to see the passphrase (though they do not). Is this all within the scope of your requirements?

```bash
cat pass.txt | gpg --passphrase-fd 0 --symmetric input.file
```

Using asymmetric encryption with gpg

When using asymmetric (public key) encryption, you have to generate two keys: public and private (they are made at the same time). The private key must be kept private, and has a passphrase on it too. This provides an added level of security on top of the file permissions.

There are plenty of guides on this available. Some examples:

- [https://www.madboa.com/geek/gpg-quickstart/](https://www.madboa.com/geek/gpg-quickstart/)
- [https://gnupg.org/documentation/index.html](https://gnupg.org/documentation/index.html)

You can encrypt a single file to multiple keys. This means that the owner of any of the private keys can decrypt the file. This can be useful for backups and disaster recovery.

General warnings

- Strong encryption is serious business. It is designed so that no one can read the data should the keys or passwords be lost. If you mess this up and lose the key/password, your data is gone forever. You must have backups (and those backups must also be secure), …

- If you keep passwords in files, or send them insecurely anyhow, then the technical security of your data is only as great as of that key/password.

- The strength of your encryption also depends on the strength of your password (there is the reason it is often called a “passphrase” - a phrase is more secure than a standard password). Choose it carefully.

Advanced / to do

- How much security is enough?
- Set cipher to AES (pre 16.04)
7.1.2 Git

Git is a version control system. This page collects various Git tutorials and resources.

Version control systems track changes to files. This means that as you are working on your projects (code, LaTeX, notes, etc), you can track history. This means that you can see former history, and collaborate better. Using one for at least for code should probably be one of the minimum standards of computational research.

“Git is a distributed version control system designed to handle everything from small to very large projects with speed and efficiency. Git is easy to learn and has a tiny footprint with lightning fast performance. It outclasses SCM tools like Subversion, CVS, Perforce, and ClearCase with features like cheap local branching, convenient staging areas, and multiple workflows.”

Note:

- This page is git in general, not Aalto specific.
- [aalto/git](https://aalto/git) contains advice on the Aalto Gitlab, a repository for the Aalto community integrated to Aalto systems.

Basic git tutorials

- There is an interactive git tutorial from codeschool and github. Good for your first use.
- Software carpentry has a good tutorial focused on researchers.
- Gitlab cheatsheet.

More references

- You can search for many tutorials online.
- [software-carpentry.org](https://software-carpentry.org) (an organization that teaches development to scientists) has a very good tutorial online.
- The book “Pro Git” is online.
  - Read chapters 1-3 for a good introduction to using git for your own projects.
  - Read chapter 5 for a good introduction to using git to collaborate with others.
- There’s a somewhat official documentation place - including videos.
- There is an official tutorial but it is probably too theoretical.
- All git commands have very good but very detailed manual pages - type `man git COMMAND` or `git help COMMAND` to see them.
- Interactive git cheatsheet. (very good once you know the basics)
- A Visual Git Reference

Gitlab-specific information:

- A tutorial

Other hosting services

Realistically, use version.aalto.fi for most projects related to Aalto research, and Github if you want to make something open-source with a wider community (but you can also make open repos in Aalto Gitlab, just harder for random people to contribute). For non-work private repos, you have to make your own choice.
• Github is a proprietary commercial service, but extremely popular. No free private repositories or groups (but you can pay).

• Bitbucket is also somewhat popular, limit of free 5 private repositories (but you can pay for more).

• Gitlab.com is a commercial service but makes the open-source Gitlab edition. Gitlab.com offers unlimited private repositories.

• source.coderefinery.org is another Gitlab hosted by the Coderefinery project, a pan-Nordic academic group. It might be useful if you have a very distributed project, but realistically for Aalto projects, use Aalto gitlab.

### 7.1.3 Pitfalls of Jupyter Notebooks

Jupyter Notebooks are a great tool for research, data science type things, and teaching. But they are not perfect - they support exploration, but not other parts of the coding phase such as modularity and scaling. This page lists some common limitations and pitfalls and what you can do to avoid them.

*Do* use notebooks if you like, but *do* keep in mind their limitations, how to avoid them, and you can get the best of both worlds.

None of the limitations on this page are specific to notebooks - in fact we’ve seen most of them in scripts long before notebooks were popular.

#### Modularity

We all agree that code modularity is important - but Jupyter encourages you to put most code directly into cells so that you can best use interactive tools. But to make code the most modular, you want lots of functions, classes, etc. Put another way, the most modular code has nothing except function/class/variable/import definitions touching the left margin - but in Jupyter, almost everything touches the left margin.

Solution: be aware of the transition to modules - do it when you need to. See the next point. Try to plan so it’s not too painful to do this.

#### Transitioning to modules

You may start coding in notebooks, but once your project gets larger, you will need to start using your code more places. Do you copy and paste? At this point, you will want to split your core code into regular Python modules, import them into your notebooks, and use the notebooks as an interface to them - so that modules are somewhat standard working code and notebooks are the exploration and interactive layer. But when does that happen? It is difficult to make that transition unless you really try hard, because it’s easier to just keep on going.

Solution: remember that you will probably need to form a proper module eventually. Plan for it and do it quickly once you need to. You can set modules to automatically reload with `%load_ext autoreload`, `%autoreload 1`, and then `%import module_name`. Then your edits to the Python source code are immediately used without restarting and your work is not slowed down much. See more at the IPython docs on autoreload (note: this is Python kernel specific).

#### Difficulty to test

For the same reasons modularity outlined above, it’s hard to test notebooks using the traditional unit testing means (if you can’t import notebooks into other modules, you can’t do much). Testing is important to ensure the accuracy of code.

Solution: Include mini-tests / assertions liberally. Split to modules when it is necessary - maybe you only create a proper testing system once you transition to modules.
Version control

Notebooks can’t be version controlled well. Of course, they can be version controlled (and should be), and tools like nbdiff (notebook diff and merge) work well for what they try to do.

Solution: don’t let this stop you. Do version control your notebooks (and don’t forget to commit often!), and use nbdiff, Jupyter git integration, and so on well.

Hidden state is opposed to reproducibility

This is a bit of an obscure one: people always say that notebooks are good for reproducibility. But they also allow you to run cells in different orders, delete cells after it has run, change code after you run it, and so on. And the whole point of notebooks is to be able to re-edit and re-run. So it’s very easy to get into a state where you have variables defined which aren’t in your current code and you don’t remember how you got them. Since old output is saved, you might not realize this until it’s too late.

Solution: Use “Restart and run all” liberally. Unless you do, you can’t be sure that your code will make your output. So do that when needed. (But wait… part of the point of notebooks is that you can keep data in memory because it might take a while to calculate… so “Restart and run all” defeats the purpose of that, so… balance it out.)

Notebooks aren’t named by default

This is really small, but notebooks aren’t named by default. If you don’t name them well, you will end up with a big mess. Also somewhat related, notebooks tend to purpose drift: they start for one thing then end up with a lot of random stuff in them. How do you find what you need? Obviously this isn’t specific to notebooks, but the interactive nature and modularity-second makes the problem more visible.

Solution: Remember to name notebooks well. Keep mind of when they start to feature drift too much, and take some time to sort your code logically once that happens.

Difficult to integrate into other execution systems

A notebook is designed for interactive use - you can run them all with nbconvert --to notebook --execute input.ipynb --output executed.ipynb. But there’s no good command line interface to pass arguments, input and output, and so on. So you write one notebook, but can’t easily turn it into a flexible script to be used many times.

Solution: Probably a lot of the solution is the same as modularizing. Use notebooks to explore, scripts to run in bulk. Run notebooks in batch using nbconvert if you think you can manage it well enough and not have it become a mess. I’ve had an idea that environment variables could be used to pass parameters into notebooks for batch execution, but could that be trying too hard?

Summary

The notebooks can be great for starting projects and interactive exploration. However, as a project gets more advanced, you will eventually find that the linear nature of notebooks is a limitation because code can not really be reused. It is possible to define functions/classes within the notebook, but you lose the power of inspection (they are just seen as single blocks) and can’t share code across notebooks (and copy and paste is bad). This doesn’t mean to not use notebooks: but do keep this in mind, and once your methods are mature enough (you are using the same code in multiple places), try to move the core functions and classes out into a separate library, and import this into the day-to-day exploration notebooks. For more about problems with notebooks and how to avoid them, see this fun talk “I don’t like notebooks” by Joel Grus. These problems are not specific to notebooks, and will make your science better.
In a cluster environment, notebooks are inefficient for big calculations because you must reserve your resources in advance, but most of the time the notebooks are not using all their resources. Instead, use notebooks for exploration and light calculation. When you need to scale up and run on the cluster, separate the calculation from the exploration. Best is to create actual programs (start, run, end, non-interactive) and submit those to the queue. Use notebooks to explore and process the output. A general rule of thumb is “if you would be upset that your notebook restarted, it’s time to split out the calculation”.

Notebooks are hard to version control, so you should look at the Jupyter diff and merge tools. Just because notebooks is interactive doesn’t mean version control is any less important! The “split core functions into a library” is also related: that library should be in version control at least.

Don’t open the same notebook more than once at the same time - you will get conflicts.

References

- This funny talk “I don’t like notebooks” by Joel Grus provided a starting point of this list.

7.1.4 Python

Note For triton specific instructions see triton python page. For Aalto Linux workstation specific stuff, see Aalto python page.

Python is widely used high level programming language that is widely used in many branches of science.

Python distributions

<table>
<thead>
<tr>
<th>Use case</th>
<th>Python to use</th>
<th>How to install own packages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple programs with common packages, not switching between Python often</td>
<td>Anaconda 2/3</td>
<td>pip install --user</td>
</tr>
<tr>
<td>Most of the use cases, but sometimes different versions of modules needed</td>
<td>Anaconda 2/3</td>
<td>conda environment + conda</td>
</tr>
<tr>
<td>Special advanced cases.</td>
<td>Python from module system</td>
<td>virtualenv + pip install</td>
</tr>
</tbody>
</table>

There are two main versions of python: 2 and 3. There are also different distributions: The “regular” CPython that is usually provided with the operating system, Anaconda (a package containing cython + a lot of other scientific software all bundled togeter), PyPy (a just-in-time compiler, which can be much faster for some use cases).

- For general scientific/data science use, we suggest that you use Anaconda. It comes with the most common scientific software included, and is reasonably optimized.

- PyPy is still mainly for advanced use (it can be faster under certain cases, but does not work everywhere). It is available in a module.

Installing your own packages with “pip install” won’t work unless you have administrator access, since it tries to install globally for all users. Instead, you have these options:

- pip install --user: install a package in your home directory (~/.local/lib/pythonN.N/). This is quick and effective, but if you start using multiple versions of Python, you will start having problems and the only recommendation will be to delete all modules and reinstall.

- Virtual environments: these are self-contained python environment with all of its own modules, separate from any other. Thus, you can install any combination of modules you want, and this is most recommended.
Installing own packages: Virtualenv, conda, and pip

You often need to install your own packages. Python has its own package manager system that can do this for you. There are three important related concepts:

- **pip**: the Python package installer. Installs Python packages globally, in a user’s directory (`--user`), or anywhere. Installs from the Python Package Index.

- **virtualenv**: Creates a directory that has all self-contained packages that is manageable by the user themself. When the virtualenv is activated, all the operating-system global packages are no longer used. Instead, you install only the packages you want. This is important if you need to install specific versions of software, and also provides isolation from the rest of the system (so that you work can be uninterrupted). It also allows different projects to have different versions of things installed. virtualenv isn’t magic; it could almost be seen as just manipulating PYTHONPATH, PATH, and the like. Docs: [http://docs.python-guide.org/en/latest/dev/virtualenvs/](http://docs.python-guide.org/en/latest/dev/virtualenvs/)

- **conda**: Sort of a combination of package manager and virtual environment. However, it only installed packages into environments, and is not limited to Python packages. It can also install other libraries (C, Fortran, etc) into the environment. This is extremely useful for scientific computing, and the reason it was created. Docs for envs: [http://conda.pydata.org/docs/using/envs.html](http://conda.pydata.org/docs/using/envs.html).

So, to install packages, there is pip and conda. To make virtual environments, there is venv and conda.

Advanced users can see this rosetta stone for reference.

Anaconda

Anaconda is a Python distribution by Continuum Analytics. It is nothing fancy, they just take a lot of useful scientific packages and put them all together, make sure they work, and do some sort of optimization. They also include all of the libraries needed. It is also all open source, and is packaged nicely so that it can easily be installed on any major OS. Thus, for basic use, it is a good base to start with. virtualenv does not work with Anaconda, use conda instead.

Conda environments

A conda environment lets you install all your own packages. For instructions how to create, activate and deactivate conda environments see [http://conda.pydata.org/docs/using/envs.html](http://conda.pydata.org/docs/using/envs.html).

A few notes about conda environments:

- Once you use a conda environment, everything goes into it. Don’t mix versions with, for example, local packages in your home dir. Eventually you’ll get dependency problems.

- Often the same goes for other python based modules. We have setup many modules that do use anaconda as a backend. So, if you know what you are doing this might work.

- The commands below will fail:
  - `conda create -n foo pip` # tries to use the global dir, use the `--user` flag instead
  - `conda create --prefix $WRKDIR/foo --clone root` # will fail as our anaconda module has additional packages (e.g. via pip) installed.
### Basic pip usage

`pip install` by itself won’t work, because it tries to install globally. Instead, use this:

```
pip install --user
```

**Warning!** If you do this, then the module will be shared among all your projects. It is quite likely that eventually, you will get some incompatibilities between the Python you are using and the modules installed. In that case, you are on your own (simple recommendation is to remove all modules from 

If you get incompatible module errors, our first recommendation will be to remove everything installed this way and not do it anymore.

### Python: virtualenv

Virtualenv is default-Python way of making environments, but does **not** work with Anaconda.

```
# Create environment
virtualenv DIR

# activate it (in each shell that uses it)
source DIR/bin/activate

# install more things (e.g. ipython, etc.)
pip install PACKAGE_NAME

# deactivate the virtualenv
deactivate
```

### 7.1.5 Linux shell crash course

**Note:** This is a kickstart for the Linux shell, to teach the minimum amount needed for any scientific computing course. For more, see the [linux shell course](#) or the references below.

This is **basic B-level**: no prerequisites.

If you are reading this case, you probably need to do some sort of scientific computing involving the **Linux shell**, or command line interface. You may wonder why we are still using a command line today, but the answer is somewhat simple: once you are doing **scientific computing**, you eventually need to script and automate something. The shell is the only method that gives you the power to do **anything you may want**.

These days, you don’t need to know as much about the shell as you used to, but you **do** need to know a few important commands because the command line works when nothing else does - and you can’t do scripting without it.

**What’s a shell?**

It’s the old-fashioned looking thing where you type commands with a keyboard and get output to the screen. It seems boring, but the real power is that you can script (program) commands to run automatically - which is the point of scientific computing.

You type a **command**, which may include **arguments**. Output gets shown to the screen. Spaces separate commands and arguments. Example: `cp -i file1.txt file2.txt`. `cp` is the command, `-i` is an option, `file1.txt` and `file2.txt` are arguments.
Files are represented by filenames, like file.txt. Directories are separated by /, for example mydir/file.txt is file.txt inside of mydir.

**Exercise:** Start a shell. On Linux or Mac, the “terminal” application does this.

### Editing and viewing files

`nano` is an editor which allows you to edit files directly from the shell. This is a simple console editor which always gets the job done.

`less` is a pager (file viewer) which lets you view files without editing them. (q to quit, / to search, n/N to research forward and backwards, < for beginning of file, > for end of file)

### Listing and moving files

`ls` lists the current directory. `ls -l` shows more information, and `ls -a` shows hidden files. The options can be combined, `ls -la` or `ls -l -a`. This pattern of options is standard for most commands.

`mv` will move or rename files. For example, `mv file.old file.new`.

`cp` will make a copy of a file, with the exact same syntax as `mv`: `cp file.old file.copy`.

`rm` will remove a file: `rm file.txt`. To remove a directory, use `rm -r`. Note that `rm` does not have backups and does not ask for confirmation!

`mkdir` makes a directory: `mkdir dirname`.

### Current directory

Unlike with a graphical file browser, there is a concept of current working directory: each shell is in a current directory. If you `ls`, it lists files in your current directory. If a program tries to open a file, it opens it relative to that directory.

`cd dirname` will change working directories for your current shell. Normally, you will `cd` to a working directory, and use relative paths from there. `/ alone refers to the root directory, the parent of all files and directories.

`cd ..` will change to the parent directory (dir containing this dir). By the same token, `../` the parent of the parent, and so on.

**Exercise:** Change to some directory and then another. What do (cd -) and (cd with no arguments) do? Try each a few times in a row.

### Online manuals for any command

`man` is an on-line manual. Type `man ls` to get help on the `ls` command. The same works for almost any program. Some are easy to read, some are impossible. In general you look for what you need, not read everything. Use q to quit or / to search (n and N to search again forward and backwards).

`--help` or `-h` is a standard argument that prints a short help directly.

**Exercise:** briefly look at the manual pages and --help output for the commands we have learned thus far. How can you make `rm` ask before removing a file?
Tab completion

Annoyed at typing so much? Everyone is, so shells have tab completion. Type the first few letters of any command or filename and push tab once or twice... it will either complete it or show you the options. This is so important that it’s used often, many command arguments can also be completed.

Exercise: Play around with tab completion. Type python and push TAB. (erase that then start over) Then type p and push TAB twice. (erase that and start over) Then ls, space, and the first few letters of a filename, then push TAB.

See also

- The linux shell course has much more detail.
- Software Carpentry has a basic shell course. Sections one to 3 are details of what is above (the rest is about shell scripting).

Exercise: for some fun, look at the manual pages for cat, head, tail, grep.

Exercise (advanced): read the Linux shell course and understand what “pipes” and “piping” are.

7.1.6 SSH

See also:

For triton specific instructions see Connecting to triton page.

ssh is a easy, secure way of connecting to remote computers. The Internet is practically run on it. This page tells you how to make ssh work nicer.

Basic use: connect to a server

ssh username@host.fi is the basic method of use - username is the username, and host.fi is the server to which you connect, for example triton.aalto.fi. See connecting to triton.

Login without password: ssh keys

You may get tired of typing a password all the time: and you should, using a key is faster and more secure. You make a ssh key on your own computer, copy the public key to the other server, and then can login without a password.

Note: this section is only for connecting to Triton. Once you are connected the first time, a key for internal connections is automatically made.

Linux

We highly recommend you follow these steps on the first login to set up passwordless SSH. This will make your life much more pleasant, and can be used when connecting to computers other than Triton. Using keys will save you the trouble of entering passwords every time, since ssh stores the key once and uses it for logging you in in the future.

First, create the keypair on your own computer. Do not copy private keys from other computers - one computer—one private key, and copy only the public key (.pub) to any computer you want to log in to. Protect your SSH keyfiles with a passphrase. When asked to enter one, use 3+ words, mixing languages, CAsE, or inflection, but make it something you can remember without sticky notes. xkcd has some opinions on this. A key without a passphrase is like a password just sitting on disk - so be careful here. Passwordless keys are OK in certain cases, such as internal triton connections.
ssh-keygen -o

Then, copy the key to computers you want to log into: Use the ssh-copy-id script to copy the public key file to Triton. This will put the key in ~/.ssh/authorized_keys (you can check this file to see everything that’s there). (To do this manually, put the contents of .ssh/id_rsa.pub file into ~/.ssh/authorized_keys on Triton. If you do this yourself, you may set set the permissions on .authorized_keys file: chmod u=rwx .ssh/, chmod u=rw .ssh/authorized_keys.)

Finally, you should be able to login automatically. A program called ssh-agent (or gnome-keyring) decrypts the key once and holds it and uses it each time you need to connect. If it doesn’t work automatically, try running ssh-add yourself once.

Mac

You can follow same instructions from Linux.

Windows

Realistically, on windows setting up keys takes some time. You don’t need to worry about it (you will still have an ssh key on triton that is used for internal connections).

You can make keys using puttygen. Here is a tutorial. You should make a new key for each computer you have.

Config file: don’t type so many options

Openssh on Linux and Mac can be made nicer if you set up a config file (.ssh/config):

```
# Host alias triton: "ssh triton" instead of "ssh triton.aalto.fi".
# You can set more options here.
Host triton
    User YOUR_USERNAME
    Hostname triton.aalto.fi
    # Only if not on Aalto networks:
    # Next line *automatically* proxies you through kosh.aalto.fi. You
    # probably want to set up a "kosh" host if username is different, and
    # set up public key authentication on kosh too.
    ProxyCommand ssh kosh.aalto.fi -W %h:%p

# Defaults for all hosts.
Host *
    # Following two lines allow SSH to reuse connections - second connections
    # open very fast. If problems (channels exceeded), disable it.
    ControlMaster auto
    ControlPath /tmp/.ssh-USERNAME-mux-%r@%h:%p
```

References

- https://blog.0xbadc0de.be/archives/300

262 Chapter 7. Scientific computing
7.1.7 The Zen of Scientific computing

Have you ever felt like all your work was built as a house of cards, ready to crash down at any time?
Have you ever felt that you are far too inefficient to survive?
No, you’re not alone. Yes, there is a better way.

Production code vs research code

Yes, many things about software development may not apply to you:
- Production code: * you sort of know what the target is * code is the main result
- Research code: * you don’t know what the target is * code is secondary

But research code still can’t be an unmaintainable mess. . .

Research code pyramid

I know that not all research code will be perfect.
But if you don’t build on a good base, you will end up with misery.

Yes, you can’t do everything perfectly

Not everything you do will be perfect. But it has to be good enough to:
- be correct
- be changed without too much difficulty
- be run again once reviews come in
- ideally, not wasted once you do something new

Even as a scientist, you need to know the levels of maturity so that you can do the right thing for your situation.
It takes skill and practice to do this right. But it is part of being a scientist.

This talk’s outline: * Describe different factors that influence code quality * Describe what the maturity levels are and when you might need them
Version control

Version control allows you to track changes and progress.
For example, you can figure out what you just broke or when you introduced a bug. You can always go back to other versions.
Version control is essential to any type of collaboration.

- L0: no version control
- L1: local repo, just commit for yourself
- L2: shared repo, multiple collaborators push directly
- L3: shared repo, pull-request workflow

Resources:
- https://coderefinery.org/lessons/ (git-intro and git-collaborative)

Modular code

Modularity is one of the basic prerequisites to be able to understand, maintain, and reuse things.

- L0: bunch of copy-and-paste scripts
- L1: important code broken out into functions
- L2: separation between well-maintained libraries and daily working scripts.

Resources:

Organized workspaces

- L0: no particular organization system
- L1: different types of data separated (original data/code/scratch/outputs)
- L2: projects cleanly separated, named, and with a purpose

Resources:
- I don’t know of good sources for this.

Workflow/pipeline automation

When you are doing serious work, you can’t afford to just manage stuff by hand. Task automation allows you to do more faster.
Something such as make can automatically detect changed input files and code and automatically generate the outputs.

- L0: bunch of scripts you have to run and check output of by hand.
- L1: hand-written management scripts, each output can be traced to its particular input and code.
- L2: make or other workflow management tool to automate things.
- L3: Full automation from original data to final figures and data
Resources:

- CodeRefinery: https://coderefinery.github.io/reproducible-research/

Reproducibility of environment

Is someone else able to (know and) install the libraries needed to run your code? Will a change in another package break your code?

Scientific software is notoriously bad at managing its dependencies.

- L0: no documentation
- L1: state the dependencies somewhere, tested to ensure they work
- L2: pin exact versions used to generate your results
- L3: containerized workflow or equivalent

Resources:

- CodeRefinery: https://coderefinery.github.io/reproducible-research/

Documentation

If you don’t say what you do, there’s no way to understand it. You won’t be able to understand it later, either.

At minimum, there should be some README files that explain the big picture. There are fancier systems, too.

- L0: nothing except scattered code comments
- L1: script-level comments and docstrings explaining overall logic
- L2: simple README files explaining big picture and main points
- L3: dedicated documentatin including tutorials, reference, etc.

Resources:

- CodeRefinery: https://coderefinery.github.io/documentation/

Testing

You have to test your code at least once when you first run it. How do you know you don’t break something later?

Testing gives you a way to ensure things always work (and are correct) in the future by letting you run every test automatically.

There’s nothing more liberating than knowing “tests still pass, I didn’t break anything”. It’s extremely useful for debugging, too.

- L0: ad-hoc and manually
- L1: defensive programming (assertions), possibly some test data and scripts
- L2: structured, comprehensive unit/integration/system tests (e.g. pytest)
- L3: continuous integration testing on all commits (e.g. travis-ci.org)

If code is easy to test, it is usually easy to reuse, too. Furthermore, making code testable makes it reusable.

Resources:

- CodeRefinery: https://coderefinery.github.io/testing/
Licensing

You presumably want people to use your work so they will cite you. If you don’t have a license, they won’t (or they might and not tell anyone).

Equally, you want to use other people’s work. You need to check their licenses.

- L0: no license given / copy and paste from other sources
- L1: license file in repo / careful to not copy incompatible code
- L2: license tracked per-file and all contributors known.

Resources:
- CodeRefinery software-licensing: https://coderefinery.org/lessons/
- https://choosealicense.com/

Distribution

Code can be easy to reuse, but not easy to get. Luckily there are good systems for sharing code.

- L0: code not distributed
- L1: code provided only if someone asks
- L2: code on a website
- L3: version control system repo is public
- L4: packaged, tagged, and versioned releases

Resources:
- for Python: https://packaging.python.org/tutorials/packaging-projects/

Reuse

Are you aware of what what others have already figured out through their great effort?

- L0: reinvent everything yourself
- L1: use some existing tools and libraries
- L2: deep study of existing solutions and tools, reuse them when appropriate

Collaboration

Is science like monks working in their cells, or a community effort? In your classes, you just learn certain academic skills. There’s a whole other art of applying those skills which isn’t taught. If you don’t work together, you will fall behind.

- L0: you work alone and re-invent everything
- L1: you occasionally talk about results or problems
- L2: collaborative package development
- L3: code reviews, pair programming, etc.
- L4: community project welcoming other contributors
The future

Science with computers can be extremely enjoyable… or miserable.
We are here to help you. You are here to others.
Will we?

Cheatsheets: git for normal people, Gitlab (produced by Gitlab, with Aalto link)
We have various recommended training courses for researchers who deal with computation and data. These courses are selected by researchers, for researchers and grouped by level of skill needed.

### 8.1 Training

Scientific computing and data science require special, practical skills in programming and computer use. However, these aren’t often learned in courses.

This page is your portal for training. The focus is practical, hands-on courses for scientists, not theoretical academic courses. As a broad classification, we divide the skills a scientist would need into four big levels A-D:

<table>
<thead>
<tr>
<th>Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A (basics)</td>
<td>Having a basic knowledge of university resources, so that you use the right tool for the right job and don’t lose your data.</td>
</tr>
<tr>
<td>B (scientific computing)</td>
<td>When you are doing science but existing software isn’t enough: you have to connect things together (or make your own).</td>
</tr>
<tr>
<td>D (advanced high performance computing)</td>
<td>Catch-all for everything past level C.</td>
</tr>
<tr>
<td>Special tracks</td>
<td>Programming, scientific papers/posters/presentations, etc. Can be at any level.</td>
</tr>
</tbody>
</table>

If you’re starting research, ask your advisor what level of skill they expect you to have. There are both courses and self-study materials below. A few hours now can save you days of time during your career.

Subpages:

#### 8.1.1 Linux shell tutorial
Course basics

Linux Shell tutorial by Science IT at Aalto University.

Abstract: This course consists of two parts: Linux Shell Basics and Linux Shell Scripting. The first covers introductory level shell usage (which also is a backdoor introduction to Linux basics). The second covers actual BASH scripting, using learning by doing.

Linux Shell Basics: 2 sessions x 3h

Linux Shell Scripting: 4 sessions x 3h, session rough schedule 2x1h25m with 10m break in between.

Setting up instructions for the lecturer: Main terminal white&black with the enlarged font size. One small terminal at the top that shows commands to the learners.

- `export PROMPT_COMMAND='history -a'` # .bashrc or all the terminals one launches commands
- `tail -n 0 -F .bash_history`

Alternatively, `script` allows to follow the session even after sshing to a remote host plus command appear as soon as they are run. The regular expression can be adapted to the lecturer’s PS1, this one assumes `$command`.

- `script -f demos.out` # action window
- `tail -n 1 -f demos.out | while read line; do [[ "$line" =~ \]\$ ([^ ]+.)$ ]] && echo ${BASH_REMATCH[1]}; done`

Starred exercises (*) are for advanced users who would like further stimulation.

Based on

- `man bash v4.2` (Triton default version in Feb 2018)
- Advanced BASH scripting guide¹
- UNIX Power Tools, Shelley Powers etc, 3rd ed, O’Reilly
- common sense and 20+ years Linux experience
- see also other references in the text

PART #1. Linux Shell Basics

1.1 session: processes and files

First touch: getting a BASH shell

Set yourself up with a BASH shell. Connect to a server or open on your own computer. Examples and demos given during the lecture are done on Triton, though should work on all other Linux installations.

- Linux and Mac users: just open a terminal window. If you wish you can login to Triton or any other Aalto Linux server.
- Windows users: install PuTTY² then SSH to any interactive server at Aalto or your department.

---

¹ http://tldp.org/LDP/abs/html/index.html
² https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html
About the Linux Shell

- A shell is what you get when your terminal window is open. It is a command-line (CLI), an interface that interpreters and executes the commands.
- The name comes from being a “shell” (layer) around the operating system. It connects and binds all programs together.
- This is the basic, raw method of using UNIX-like systems. It may not be used everyday, but it’s really good (necessary) for any type of automation and scripting - as is often needed in science, when connecting pieces together, or when using Triton.
- There are multiple shells. This talk is about bash, which is the most common one. zsh is another common shell which is somewhat similar but has some more powerful features. tcsh is another shell from a completely different family (the csh family), which has quite different syntax.
- bash is a “Bourne shell”: the “bourne-again shell”. An open source version of original Bourne shell.
- It may not be obvious, but the concepts here also apply to Windows programs and will help you understand them. They also apply more directly to Mac programs, because Mac is unix under the hood.

Basic shell operation

- You type things on the screen (standard input or stdin). The shell uses this to make a command.
- The shell takes the command, splits it into words, does a lot more preprocessing, and then runs it.
- When the command runs, the keyboard (still standard input) goes to the process, output (standard output) goes to the screen.

What’s a UNIX process?

- To understand a shell, let’s first understand what processes are.
- All programs are a process: process is a program in action.
- Processes have:
  - Process ID (integer)
  - Name (command being run)
  - Command line arguments
  - input and output: stdin (input, like from keyboard), stdout (output, like to screen), stderr (like stdout)
  - Return code (integer) when complete
  - Working directory
  - environment variables: key-values which get inherited across processes.
- These concepts bind together all UNIX programs, even graphical ones.

Process listing commands (feel free to try, but we play more with them later):

```
top         # (q to quit)
htop        # (q to quit)
pstree
```

(continues on next page)
You can find info about your user (try them right away):

```plaintext
id
echo $SHELL
```

Is your default shell is a `/bin/bash`? Login to kosh/taltta and run `chsh -s /bin/bash`

Another way to find out what SHELL you are running:

```plaintext
ps -p $
```

Where am I: `pwd` (this shows the first piece of process information: current directory)

### Getting help in terminal

Before you Google for the command examples, try:

```plaintext
man command_name
```

Your best friend ever – `man` – collection of manuals. Type `/search_word` for searching through the man page. But… if it’s a builtin, you need to use `help`.

### Built-in and external commands

There are two types of commands:

- shell built-in: `cd`, `pwd`, `echo`, `test`, etc.
- external: `ls`, `date`, `less`, `lpr`, `cat`, etc.
- some can be both: e.g. `test`. Options not always the same!
- For the most part, these behave similarly, which is a good thing! You don’t have to tell which is which.

**Hint** `type -a` to find what is behind the name

- `echo`: prints out `echo something` to `type` # types whatever you put after

Disable built-in command `enable -n echo`, after this `/usr/bin/echo` becomes a default instead of built-in `echo`

### Working with processes

All processes are related, a command executed in shell is a child process of the shell. When child process is terminated it is reported back to parent process. When you log out all shell child processes terminated along with the shell. You can see the whole family tree with `ps af`. One can kill a process or make it “nicer”.

```plaintext
pgrep -af <name>
kill <PID>
pkill <name>
renice #priority <PID>
```
Making process “nicer”, `renice 19 <PID>`, means it will run only when nothing else in the system wants to. User can increase nice value from 0 (the base priority) up to 19. It is useful when you backup your data in background or alike.

**Foreground and background processes**

The shell has a concept of foreground and background processes: a foreground process is directly connected to your screen and keyboard. A background process doesn’t have input connected. There can only be one foreground at a time (obviously).

If you add `&` right after the command will send the process to background. Example: `firefox --no-remote &`, and same can be done with any terminal command/function, like `man pstree &`. In the big picture, the `&` serves the same role as `;` to separate commands, but backgrounds the first and goes straight to the next.

If you have already running process, you can background with Ctrl-z and then `bg`. Drawback: there is no easy way to redirect the running task output, so if it generates output it covers your screen.

List the jobs running in the background with `jobs`, get a job back online with `fg` or `fg <job_number>`. There can be multiple background jobs.

Kill a foreground job: Ctrl-c

**Hint:** For running X Window apps while you logged in from other Linux / MacOS make sure you use `ssh -X ...` to log in. For Windows users, you need to install Xming\(^3\) on your workstation.

**Hint:** For immediate job-state change notifications, use `set notify`. To automatically stop background processes if they try writing to the screen use `stty tostop`.

**Exit the shell and ‘screen’ utility**

`logout` or Ctrl-d (if you don’t want Ctrl-d to quit, set `export IGNOREEOF=1` in `.bashrc`).

Of course, quitting your shell is annoying, since you have to start over. Luckily there are programs so that you don’t have to do this. In order to keep your sessions running while you logged out, you should learn about the `screen` program.

- `screen` to start a session
- `Ctrl-a d` to detach the session while you are connected
- `screen -ls` to list currently running sessions
- `screen -rx <session_id>` to attach the session, one can use TAB for the autocompletion or skip the `<session_id>` if there is only one session running
- `tmux` is a newer program with the same style. It has some extra features and some missing features still.

Some people have their `screen` open forever, which just keeps running and never gets closed. Wherever they are, they ssh in, connect, and resume right where they left off.

Example: `irssi` on kosh / lyta

[Lecture notes: that should be a first half, then joint hands-on/break ~30 mins]

**Exercise 1.1.1**

- for Aalto users: set your SHELL to BASH if you have not yet done so: `chsh -s /bin/bash` on kosh

\(^3\) [http://www.straightrunning.com/XmingNotes/](http://www.straightrunning.com/XmingNotes/)
Aalto scientific computing guide

- find out with man how to use top / pstree / ps to list all the running processes that belong to you
  Tip: top has both command line options and hot keys.
  - (*) see man ps and find out how to list a processes tree with ps, both all processes and
    only your own (but all your processes, associated with all terminals)
- with pgrep list all bash and then zsh sessions on kosh or triton
- log in to triton/kosh and run man ps, send it to background, and logout, then log in again.
  Is it still there? Play with the screen, run a session, then detach it and log out, then log in
  back and get your original screen session back.
- run man htop, send it to backround, and then kill it with kill. Tip: one can do it by
  background job number or by PID.
- Imagine a use case: your current ssh session got stuck and does not response. Open another ssh
  session to the same remote host and kill the first one. Tip: echo $$ gives you current bash
  PID.
  - (*) get any X Window application (firefox, xterm, etc) to run on Triton / kosh

Files and directories

Files contain data. They have a name, permissions, owner (user+group), contents, and some other metadata.
Filenames may contain any character except ‘/’, which is reserved as a separator between directory and filenames. The
special characters would require quotation while dealing, with such filenames, though it makes sense to avoid them
anyway.

Path can be absolute, starts with ‘/’ or relative, that is related to the current directory.

ls is the standard way of getting information about files. By default it lists your current directory (i.e. pwd), but there
are many options:

```
# list directory content
ls /scratch/work

# list directory files including dot files (i.e. hidden ones)
ls -A ~/directory1

# list all files and directories using long format (permissions, timestamps, etc)
ls -lA ../../directory2
```

Special notations and expansions in BASH, can be used with any command:

```
./, ../, ~, *, ?, [], [[]], {abc,xyz}, {1..10}
```

For the quotation:

```
'', "", \
```

Quotation matters ""$USER"" vs echo "$USER"

BASH first expand the expansions and substitute the wildcards, and then execute the command. Could be as complex
as:

```
ls -l ~/[!abc]??/?dir(123,456)/filename*.{1..9}.txt
```

There are a variety of commands to manipulate files/directories:
cd, mkdir, cp, cp -r, rm, rm -r, mv, ln, touch

For file/directory meta information or content type:
ls, stat, file

Note that cd is a shell builtin which change’s the shell’s own working directory. This is the base from which all other commands work: ls by default tells you the current directory, . is the current directory, .. is the parent directory, ~ is your HOME. This is inherited to other commands you run. cd with no options drops your to your $HOME.

# copy a directory preserving all the metadata to two levels up
cp -a dir1/ ../../

# move all files with the names like filename1.txt, filename_abc.txt etc to dir2/
mv filename*.txt dir2/

# remove a directories/files in the current dir without asking for the confirmation
rm -rf dir2/ dir1/ filename*

# create an empty file if doesn’t exist or update its access/modification time
touch filename

# create several directories at once
mkdir dir3 dir4 dir5
# -or-
mkdir dir{3,4,5}

# make a link to a target file (hard link by default, -s for symlinks)
l n target_file ../link_name

Discover other ls features ls -lX, ls -ltr, ls -Q
You may also find useful rename utility implemented by Larry Wall.

File/directory permissions

- Permissions are one of the types of file metadata.
- They tell you if you can read a file, write a file, and execute a file/list directory
- Each of these for both user, group, and others
- Here is a typical permission bits for a file: -rw-r--r--
- In general, it is rwxrwxrwx – read, write, execute/search for user, group, others respectively
- ls -l gives you details on files.

Modifying permissions: the easy part

chmod/chown is what will work on all filesystems:

```bash
chmod u+rwx,g-rwx,o-rwx <files>  # u=user, g=group, o=others, a=all
# -or-
chmod 700 <files>  # r=4, w=2, x=1
```

(continues on next page)
# recursive, changing all the subdirectories and files at once
chmod -R <perm> <directory>

# changing group ownership (you must be a group member)
chgrp group_name <file or directory>

Extra permission bits:

- s-bit: setuid/setgid bit, preserves user and/or group IDs.
- t-bit: sticky bit, for directories it prevents from removing file by another user (example /tmp)

Setting default access permissions: add to .bashrc umask 027. The umask is what permissions are removed from any newly created file by default. So umask 027 means “by default, g-w,o-rwx any newly created files”. It’s not really changing the permissions, just the default the operating system will create with.

**Hint:** even though file has a read access the top directory must be searchable before external user or group will be able to access it. Sometimes on Triton, people do chmod -R o-rwx $WRKDIR; chmod o+x $WRKDIR. Execute (x) without read (r) means that you can access files inside if you know the exact name, but not list the directory. The permissions of the files themselves still matter.

**Modifying permissions: advanced (*)**

Access Control Lists (ACLs) are advanced access permissions. They don’t work everywhere, for example mostly do no work on NFS mounted directories. They are otherwise supported on ext4, lustre, etc (thus works on Triton $WRKDIR).

- In “normal” unix, files have only “owner” and “group”, and permissions for owner/group/others. This can be rather limiting.
- Access control lists (ACLs) are an extension that allows an arbitrary number of users and groups to have access rights to files. The basic concept is that you have:
  - ACLs don’t show up in normal ls -l output, but there is an extra plus sign: -rw-rw-xr--+. ACLs generally work well, but there are some programs that won’t preserve them when you copy/move files, etc.
  - POSIX (unix) ACLs are controlled with getfacl and setfacl
    - Allow read access for a user setfacl -m u:<user>:r <file_or_dir>
    - Allow read/write access for a group setfacl -m g:<group>:rw <file_or_dir>
    - Revoke granted access setfacl -x u:<user> <file_or_dir>
    - See current stage getfacl <file_or_dir>

**File managers** on Triton we have installed Midnight Commander – mc

**Advanced file status** to get file meta info stat <file_or_dir>

[Lecture notes: hands-on ~30 mins till the end of this session]

**Exercise 1.1.2**

- mkdir in your $HOME (or $WRKDIR if on Triton), cd there and touch a file. Rename it. Make a copy and then remove the original. What does touch do?
- list all files in /usr/bin and /usr/sbin that start with non-letter characters with one ls command

---

8 https://www.computerhope.com/unix/uumask.htm
Aalto scientific computing guide

• (*) list with `ls` dot files/directories only (by default it lists all files/directories but not those that begin with `.`). “dotfiles” are a convention where filenames that begin with `.` such as `.bashrc` are considered “hidden”.

• Explore `stat` file output. What metadata do you find? Try to stat files of different types (regular file, directory, link, special device in `/dev`, named pipe)

• create a directory, use `chmod` to allow user and any group members full access and no access for others

• (*) change that directory group ownership with `chown` or `chgrp` (any group that you belong to is fine), set s-bit for the group and apply t-bit to a directory, check that the upper directory has o+x bit set: now you should have a private working space for your group. Tip: see groups that you are a member of `id -Gn`

• `ls -ld` tells you that directory has permissions `rw+xSr--`. Do group members have access there?

• create a directory (in WRKDIR if on Triton and in `/tmp` if on any other server), use `setfacl` to set its permissions so that only you and some user/group of your choice would have access to it.

• (*) create a directory and a subdirectory in it and set their permissions to 700 with one command.

1.2 session: interactive usage

`find`

- `find` is a very unixy program: it finds files, but in the most flexible way possible.
- It is an amazingly complicated program
- It is a number one in searching files in shell

With no options, just recursively lists all files starting in current directory:

```
find
```

The first option gives a starting directory:

```
find /etc/
```

Other search options: by modification/accessing time, by ownership, by access type, joint conditions, case-insensitive, that do not match, etc\(^5\):\(^6\):

```
# -or- 'find ~ $WRKDIR -name file.txt' one can search more than one dir at once
find ~ -name file.txt

# look for jpeg files in the current dir only
find . -maxdepth 1 -name '*.jpg' -type f

# find all files of size more than 10M and less than 100M
find . -type f -size +10M -size -100M

# find everything that does not belong to you
```

(continues on next page)

\(^5\) [https://alvinalexander.com/unix/edu/examples/find.shtml](https://alvinalexander.com/unix/edu/examples/find.shtml)

\(^6\) [http://www.softpanorama.org/Tools/Find/index.shtml](http://www.softpanorama.org/Tools/Find/index.shtml)
```bash
find ~ ! -user $USER | xargs ls -ld

# open all directories to group members
# tip: chmod applies x-bit to directories automatically
find . -type d -exec chmod g+rw {} \;

# find all s-bitted executable binaries
find /usr/{bin,sbin} -type f -perm -u+x,u+s

# find and remove all files older than 7 days
find path/dir -type f -mtime +7 -exec rm -f {} \;
```

Find syntax is actually an entire boolean logic language given on the command line: it is a single expression evaluated left to right with certain precedence. There are match expressions and action expressions. Thus, you can get amazingly complex if you want to. Take a look at the ‘EXAMPLES’ section in `man find` for the comprehensive list of examples and explanations.

**find on Triton** On Triton’s WRKDIR you should use `lfs find`. This uses a raw lustre connection to make it more efficient than accessing every file. It has somewhat limited abilities as comparing to GNU find. For details see `man lfs` on Triton.

**Fast find – locate** Another utility that you may find useful `locate <pattern>`, but on workstations only. This uses a cached database of all files, and just searches that database so it is much faster.

**Too many arguments** error solved with the `find ... | xargs`

**File archiving**

tar is the de-facto standard tool for saving many files or directories into a single archive file. Archive files may have extensions `.tar`, `.tar.gz` etc depending on compression.

```bash
# create tar archive gzipped on the way
tar -caf archive_name.tar.gz directory_to_be_archived/

# extract files
tar -xf archive_name.tar.gz -C path/to/directory
```

Other command line options: `r` - append files to the end of an archive, `l` - list archive content, `f` is for the filename, and `a` selects the compression method based on the archive file suffix (in this example gzip, due to the `.gz` suffix. Without compression files/directories are simply packed as is.

```bash
# xz has better compression ratio than gzip, but is very slow
tar -caf archive_file.tar.xz dir1/ dir2/
```

Individual files can be compressed directly, e.g. with `gzip`:

```bash
# file.gz is created, file is removed in the process.
gzip file

# Uncompress

gunzip file.gz
```
Transferring files (+archiving on the fly)

For Triton users the ability to transfer files to/from Triton is essential. Same applicable to file transfer between your home workstation and kosh etc.

Several use cases:

```bash
# transferring a file from your HOME on kosh to your home workstation
sftp AALTO_LOGIN@kosh.aalto.fi:file_to_copy

# transferring files from Triton to your Aalto workstation
sftp triton.aalto.fi:/scratch/work/LOGIN_NAME/some/files/* path/to/copy/to
```

(*) Another use case, copying to Triton, or making a directory backup with rsync:

```bash
rsync -urlptDxv --chmod=Dg+s somefile triton.aalto.fi:/scratch/work/LOGIN_NAME
# copy a file to $WRKDIR
rsync -urlptDxv --chmod=Dg+s dir/ triton.aalto.fi:/scratch/work/LOGINNAME/dir/  # sync two directories
```

(*) Transferring and archiving your Triton data on the fly to some other place:

```bash
# login to Triton
cd $WRKDIR
tar czf - path/to/dir | ssh kosh.aalto.fi 'cat > path/to/archive/dir/archive_file.tar.gz'
```

[Lecture notes: this session has three theory+excersise hands-ons, roughly 40+20 minutes each]

**Exercise 1.2.1**

- Find with `find` all the files in your $HOME that are readable or writable by everyone
  - (*) apply `chmod o-rwx` to all recently found files with `find`
- Make a tar.gz archive of any of your directory at your HOME (or WRKDIR if on Triton), when done list the archive content, then append another file/directory to the existing archive.
  - (*) Extract only one particular file to some subdirectory from the archive
- Transfer just created archive using either `sftp` or `rsync`.
  - (*) Try ssh+tar combo to make transfer and archive on the fly.

How to make things faster: hotkeys

- Is it annoying to have to type everything in the shell? No, because we have hotkeys. In fact, it can become much more efficient and powerful to use the shell.
- Most important key: **TAB**: autocomplete. You should never be typing full filenames or command names. TAB can complete almost anything

Common hotkeys:

- **TAB** – autoocompletion
- **Home** or **Ctrl-a** – start of the command line
- **End** or **Ctrl-e** – end
- **Ctrl-left/right arrows** or **Alt-b/Alt-f** - moving by one word there and back
• up/down arrows – command history
• Ctrl-l – clear the screen
• Ctrl-Shift-c – copy
• Ctrl-Shift-v – paste
• Ctrl-Shift-z – undo the last changes on cli
• Alt-r – undo all changes made to this line
• Ctrl-r – command history search: backward (hit Ctrl-r, then start typing the search word, hit Ctrl-r again to go through commands that have the search word in it)
• Ctrl-s – search command history furtherword (for this to work one needs to disable default suspend keys stty -ixon)
• Ctrl-u – remove beginning of the line, from cursor
• Ctrl-k – remove end of the line, from cursor

inputrc Check /etc/inputrc for some default key bindings, more can be defined ~/.inputrc (left as a home exercise)

CDPATH helps changing directories faster. When you type cd dirname, the shell tries to go to one of the local subdirectories and if it is not found shell will try the same command from every directory listed in the $CDPATH.

```bash
export CPATH=$HOME:$WRKDIR:$WRKDIR/project
```

Initialization files and configuration

• When the shell first starts (when you login), it reads some files. These are normal shell files, and it evaluates normal shell commands to set configuration.
• You can always test things in your own shell and see if it works before putting it in the config files. Highly recommended!
• You customize your environment means setting or expanding aliases, variables, functions.
• The config files are:
  – .bashrc (when SSH) and
  – .bash_profile (interactive login to a workstation)
  – they are often a symlink from one to another
• To get an idea how complicated .bashrc can be take a look at <https://www.tldp.org/LDP/abs/html/sample-bashrc.html>

One of the things to play with: command line prompt defined in PS1

```bash
PS1="[\d \t \u@\h:\w ] $ "
```

For special characters see PROMPTING at man bash. To make it permanent, should be added to .bashrc like export PS1.

---

Creating/editing/viewing file

- A text editor edits files as ASCII. These are your best friend. In fact, text files are your best friend: rawest, most efficient, longest-lasting way of storing data.
- “pager” is a generic term for things that view files or data.

Linux command line text editors like:

- nano - simplest
- vim - minimal. To save&quit, ESC :wq
- emacs - or the simplest one nano. To save&quit: Ctrl-x Ctrl-c

To view contents of a file in a scrollable fashion: less

Quick look at the text file cat filename.txt (dumps everything to screen- beware of non-text binary files or large files!)

Other quick ways to add something to a file (no need for an editor)

echo 'Some sentence, or whatever else 1234567!+->$#' > filename.txt
cat > filename2.txt to finish typing and write written to the file, press enter, then Ctrl-d.

The best text viewer ever less -S (to open a file in your EDITOR, hit v, to search through type /search_word)

Watching files while they grow tail -n 0 -f <file>

Try: add above mentioned export PS1 to .bashrc. Remember source .bashrc to enable changes

Exercise 1.2.2

- link .bash_profile to .bashrc. Tip: see ln command from the previous session.
- open ~/.bashrc for eiding and add there CDPATH example from above, customize it for your needs and test. Tip: remember source ~/.bashrc.
- add umask 027 to .bashrc, try creating files. Tip: umask -S prints your current setting.
- customize a prompt $PS1 and add it to your .bashrc, make sure is has a current directory name and the hostname in it in the format hostname:/path/to/current/dir. Hint: save the original PS1 like oldPS1=$PS1 to be able to recover it any time.
- (*) Set some default options for the less program in your bashrc. Examples: case-insensitive searching, long prompt, wrapping lines.

Utilities: the building blocks of shell

- wide range of all kind of utilities available in Linux
- shell is a glue to bind them all together
- commandline is often a long list of those utilities joint into pipe that pass output of each other further

We catch many of them on the way.
Input and output: redirect and pipes

- Programs can display something: `echo this is some output` or `cat`
- Programs can take some input: e.g. `less` by default displays input if no filename given.
  - `cat /etc/bashrc` dumps that file to `standard output` (`stdout`)
  - `cat /etc/bashrc | less` gives it to `less` on `standard input` (`stdin`)

Pipe: output of the first command as an input for the second one `command_a | command_b`:

```
# send man page to a default printer
man -t ls | lpr

# see what files/directories use the most space, including hidden ones
du -hs *.[!.]* | sort -h

# count a number of logged in users
w -h | wc -l

# to remove all carriage returns and Ctrl-z characters from a Windows file
cat win.txt | tr -d \'\15\32\' > unix.txt

# to list all matching commands
history | grep -w 'command name'

# print all non-printable characters as well
ls -1A | cat -A

# print the name of the newest file in the directory (non-dot)
ls -lt | grep -v -E '*/|@' | head -1
```

Redirects:

- Like pipes, but send data to/from files instead of other processes.
- Replace a file: `command > file.txt`
- Append to a file: `command >> file.txt` (be careful you do not mix them up!)
- Redirect file as STDIN: `command < file` (in case program accepts STDIN only)

```
echo Hello World > hello.txt
ls -1H >> current_dir_ls.txt

# join two files into one
cat file1 file2 > file3

# extract user names and store them to a file
getent passwd | cut -d: -f1,5 > users

# join file1 and 2 lines one by one using : as a delimiter
paste -s -d : file1 file2 > file3

# go through file1 and replace spaces with a new line mark, then output to file2
tr -s ' ' '\n' < file1 > file2
# -or- in more readable format
cat file1 | tr -s ' ' '\n' > file2
```
**This is the unix philosophy** and the true power of the shell. The **unix philosophy** is a lot of small, specialized, good programs which can be easily connected together. The beauty of the cli are elegant one-liners i.e. list of commands executed in one line.

To dump output of all commands at once: group them.

```
{ command1; command2; } > filename  # commands run in the current shell as a group
(command1; command2; ) > filename   # commands run in external shell as a group
```

**Coreutils by GNU** You may find many other useful commands at [https://www.gnu.org/software/coreutils/manual/coreutils.html](https://www.gnu.org/software/coreutils/manual/coreutils.html)

**Pipelines:** ;, &&, and ||

- You can put several commands on the same line using different separators.
- The shell term for this is pipelines.

**Chaining:** `command_a ; command_b`: always runs both commands.

Remember exit codes? In shell, 0=sUCCESS and anything 1-255=failure. Note that this is opposite of normal Boolean logic!

The `&&` and `||` are short-circuit (lazy) boolean operators. They can be used for quick conditionals.

- `command_a && command_b`
  - If `command_a` is successful, also run `command_b`
  - final exit code is last evaluated one, which has the role of Boolean **and**.
- `command_a || command_b`
  - If `command_a` is not successful, also run `command_b`
  - final exit code is that of the last evaluated command, which has the role of Boolean **or**.

**Hint** `command_a && command_b || command_c`

Try: `cd /nonexistent_dir && ls /nonexistent_dir` compare with `cd /nonexistent_dir; ls /nonexistent_dir`

Try: `ping -c 1 8.8.8.8 > /dev/null && echo online || echo offline`

**grep**

Later on you’ll find out that ```grep``` is one of the most useful commands you ever discover on Linux (except for all the **other** most useful commands ever)

```
grep <pattern> <filename>  # grep lines that match <pattern>
```

```
 command | grep <pattern>  # grep lines from stdin
```

```
# search all the files in the dir/ and its subdirs, to match the word 'is', case-sensitive
grep -R -iw 'is' dir/
```

```
# grep all lines from *command* output, except those that have 'comment' in it
*command* | grep -v comment
```

(continues on next page)
# displaying 2 extra lines before and after the match (-A just after, -B just before)
grep -C 2 'search word' file

# counts the number of matches
grep -c <pattern> file(s)

# shows only the matched part of the string (by default grep shows whole line)
grep -o <pattern> file(s)

# accepts way more advanced regular expressions as a search pattern
grep -E <extended_regexpr> file(s)

For details on what <pattern> could be, look for REGULAR EXPRESSIONS at man grep. Some examples:

```bash
# grep emails to a list
grep -Eio "\b[a-z0-9._%+-]+@[a-z0-9.-]+\.[a-z]{2,6}\b" file.txt

# grep currently running firefox processes
ps auxw | grep firefox

# grep H1 and H2 header lines out of HTML file
grep "<\[Hh\][12]>" file.html
```

Exercise 1.2.3

- make a pipe that counts number of files/directories (including dot files) in your directory
- grep directories out of `ls -l`
- grep all but blank lines in triton:/etc/bashrc
  - expand the previous one to filter out commented lines also (line starts with #). Note that lines may have spaces before # mark.
- count unique logged in users on triton. Tip: `w` or `users` gives you a list of all currently login users, many of them have several sessions open.
- (*) Play with the commands grep, cut: find at least two ways to extract IP addresses only out of `/etc/hosts`. Tip: `grep` has `-o` option, thus one can build a regular expression that will grab exactly what you need.
- (*) Using pipes and commands echo/tr/uniq, find doubled words out of `My Do Do list`: Find a a Doubled Word. Any easier way to do it?

PART #2. Linux Shell Scripting

Quoting, substitutions, aliases

Last time, we focused on interactive things from the command line. Now, we build on that some and end up with making our own scripts.

Command line processing and quoting

So, shell is responsible for interpreting the commands you type. Executing commands might seem simple enough, but a lot happens between the time you press RETURN and time your computer actually does something.
• When you enter a command line, it is one string.
• When a program runs, it always takes an array of strings (the \texttt{argv} in C, \texttt{sys.argv} in Python, for example). How do you get from one string to an array of strings? Bash does a lot of processing.
• The simplest way of looking at it is everything separated by spaces, but actually there is more: variable substitution, command substitution, arithmetic evaluation, history evaluation, etc.

The partial order of operations is (don’t worry about exact order: just realize that the shell does a lot of different things in same particular order):

• history expansion
• brace expansion (\{1..9\})
• parameter and variable expansion (\$VAR, \${VAR})
• command substitution (\$( ))
• arithmetic expansion ($((1+1)))
• word splitting
• pathname expansion (*, ?, [a,b])
• redirects and pipes

One thing we will start to see is shell quoting. There are several types of quoting (we will learn details of variables later):

\begin{verbatim}
# Double quotes: disable all other characters except $, ', \ echo "\$SHELL"

# Single quotes: disable all special characters
echo 'SHELL'

# backslash disables the special meaning of the next character
ls name\ with\ space
\end{verbatim}

By special characters we mean:
\begin{verbatim}
# & * ? [ ] ( ) { } = | ^ ; < > ` $ " '
\end{verbatim}

There are different rules for embedding quoting in other quoting. Sometimes a command passes through multiple layers and you need to really be careful with multiple layers of quoting! This is advanced, but just remember it.

\begin{verbatim}
echo 'What's up? how much did you get $$?' # wrong, ' can not be in between ''
echo "What's up? how much did you get $$$?" # wrong, $$ is a variable in this case
echo "What's up? how much did you get "$\$$"?" # correct
echo "What's up? how much did you get "'$$'"?" # correct
\end{verbatim}

At the end of the line \ removes the new line character, thus the command can continue to a next line:

\begin{verbatim}
ping -c 1 8.8.8.8 > /dev/null && \ echo online || \ echo offline
\end{verbatim}

\textbf{Substitute a command output}

• Command substitutions execute a command, take its stdout, and place it on the command line in that place.
$(command) or alternatively `command`. Could be a command or a list of commands with pipes, redirections, grouping, variables inside. The $( ) is a modern way, supports nesting, works inside double quotes. To understand what is going on in these, run the inner command first.

```bash
# get the latest modified file to a variable
newest=$(ls -Flt | grep -v */ | head -1)

# save current date to a variable
today=$(date +%Y-%m-%d)

# create a new file with current timestamp in the name (almost unique filename)
touch file.$(date +%Y-%m-%d-%H-%M-%S)

# archive current directory content, where new archive name is based on current path and date
tar czf $(basename $(pwd)).$(date +%Y-%m-%d).tar.gz .

# counting directories and files on the fly
echo Number of directories $(ls -lA | grep ^d | wc -l) files $(ls -lA | grep ^- | wc -l)
```

This is what makes BASH powerful!

Note: $(command || exit 1) will not have an effect you expect, command is executed in a subshell, exiting from inside a subshell, closes the subshell only not the parent script. Subshell can not modify its parent shell environment, though can give back exit code or signal it:

```bash
# this will not work, echo still will be executed
dir=nonexistent
echo $(ls -l $dir || exit 1)

# this will not work either, since || evaluates echo's exit code, not ls
echo $(ls -l $dir) || exit 1

# this will work, since assignment a comman substitution to a var returns exit code of the executed command
var=$(ls -l $dir) || exit 1
echo $var
```

More about redirection, piping and process substitution

**STDIN, STDOUT and STDERR**: reserved file descriptors 0, 1 and 2. They always there whatever process you run.

/dev/null file (actually special operating system device) that discards all data written to it.

```bash
# discards STDOUT only
command > /dev/null

# discards both STDOUT and STDERR
command &> /dev/null
command > /dev/null 2>&1 # same as above, old style notation

# redirects outputs to different files
command 1>file.out 2>file.err

# takes STDIN as an input and outputs STDOUT/STDERR to a file
command < input_file &> output_file
```
Note, that &> and >& will do the same, redirect both STDOUT and STDERR to the same place, but the former syntax is preferable.

```
# what happens if 8.8.8.8 is down? How to make the command more robust?
ping -c 1 8.8.8.8 > /dev/null && echo online || echo down

# takes a snapshot of the directory list and send it to email, then renames the file
ls -l > listing && { mail -s "ls -l $(pwd)" jussi.meikalainen@aalto.fi < listing; mv listing listing.$(date +"%Y-%m-%d-%H-%M") }

# a few ways to empty a file
> filename
cat /dev/null > filename

# read file to a variable
var=$(< path/to/file)

# extreme case, if you can't get the program to stop writing to the file...
ln -s /dev/null filename
```

Pipes are following the same rules with respect to standard output/error. In order to pipe both STDERR and STDOUT |&.

If ! precedes the command, the exit status is the logical negation.

```
tee in case you still want output to a terminal and to a file command | tee filename
```

But what if you need to pass to another program results of two commands at once? Or if command accepts file as an argument but not STDIN?

One can always do this in two steps, run commands and save results to file(s) and then use them with the another command. Though BASH helps to make even this part easier (or harder), the feature called Process Substitution, looks like <(command) or >(command), no spaces in between parentheses and < signs. It emulates a file creation out of command output and place it on a command line. The command can be a pipe, pipeline etc.

The actual file paths substituted are /dev/fd/<n>. The file paths can be passed as an argument to the another command or just redirected as usual.

```
# BASH creates a file that has an output of *command2* and pass it to *command1*
# file descriptor is passed as an argument, assuming command1 can handle it
command1 <(command2)

# same but redirected (like: cat < filename)
command1 < <(command2)

# in the same way one can substitute results of several commands or command groups
command1 <(command2) <(command3 | command4; command5)

# example: comparing listings of two directories
diff <(ls dir1) <(ls dir2)

# and vice versa, *command1* output is redirected as a file to *command2*
command1 > >(command2)

# essentially, in some cases pipe and process substitution do the same
ls -s | cat

```

8.1. Training
Aliases

- Alias is nothing more than a shortcut to a long command sequence
- With alias one can redefine an existing command or name a new one
- Alias will be evaluated only when executed, thus it may have all the expansions and substitutions one normally has on the cli
- They are less flexible than functions which we will discuss next

```bash
# your own listing command
alias l='ls -lAF'

# shortcut for checking space usage
alias space='du -hs .[!.]* | sort -h'

# prints in the compact way login/group
alias me="echo "$(id -un)' '$(id -gn)'""

# redefine rm
alias rm='rm -i'
alias rm='rm -rf'
```

Aliases go to `.bashrc` and available later by default (really, anywhere they can be read by the shell).

[Lecturer’s notes: about 40 mins joint hands-on session + break]

Exercise 2.1

- Define above mentioned `ping ...` command as an alias (you name it) in `~/.bashrc` once you verify it works. Then `source ~/.bashrc` and try the new alias. Tip: any path that starts with `~` means the file or directory is in your HOME.
- Create a directory structure, that has five directories and five subdirs in each directory like `dir1/subdir1`, `dir1/subdir2`, ... `dir5/subdir5` with one command. Tip: use Brace expansions and see `mkdir -p ...`
- Use command substitution to create an empty file with the date the in the name, like `file. YYYY-MM-DD.out`. Tip: investigate `date +"..."` output format.
- Create a one-liner with `ls`, `echo`, redirections etc that takes a file path and says whether this file/directory exists or not. Redirect STDOUT/STDERR to `/dev/null`. See our `ping -c 8. 8.8.8 ...` as an example.
- Use any of the earlier created files to compare there modification times with `stat -c '%y' filename`, `diff` and the process substitution.
- (*) Make a one-liner that copies a small dir (small (!), to save time/traffic) from your Triton’s `$WRKDIR` (or any other remote server) and sends confirmation to your email with the directory listing attached but use process substitution instead of saving directory listing to a file. Tip: use examples in the text.
- (*) Using pipes and commands `echo`, `tr`, `uniq` find doubled words out of `My Do Do list`: Find a a Duplicated Word.
- (*) Pick up `/scratch/scip/BASH/windows.txt` file and convert it to UNIX format using `tr` and redirects only. Tip: remind first session examples.
- (*) Using `find`, duplicate current directory tree (to some other dir, only tree, no content)
Variables, functions, environment

Your ~/bin and PATH

The PATH is an environment variable. It is a colon delimited list of directories that your shell searches through when you enter a command. Binaries are at /bin, /usr/bin, /usr/local/bin etc. The best place for your own is ~/bin:

```bash
# add to .bashrc
export PATH="$PATH:$HOME/bin"
# after you have your script written, set +x bit and run it
chmod +x ~/bin/script_name.sh
script_name.sh
```

You can find where a program is using `which` or `type -a`, we recommend the later one:

```bash
type -a ls  # a binary
type -a cd  # builtin
```

Other options:

```bash
# +x bit and ./
chmod +x script.sh
./script.sh  # that works if script.sh has `#!/bin/bash` as a first line
# with no x bit
bash script.sh  # this will work even without `#!/bin/bash`
```

Extension is optional note that .sh extension is optional, script may have any name

Functions as part of your environment

Alias is a shortcut to a long command, while function is a piece of programming that has logic and can accept input parameters. Functions can be defined on-the-fly from the cli, or can go to a file. Let us set ~/bin/functions and collect everything useful there:

```bash
# cd to the directory and lists it at once
# can be run as: lcd <path/to/directory>
lcd() {
    cd $1
    ls -FlA
}

# in one line, note spaces and ; delimiters
lcd() { cd $1; ls -FlA; }
# -or- in a full format
function lcd { cd $1; ls -FlA; }
```

By now function has been defined, to run it, one has to invoke it:

```bash
source ~/bin/functions
lcd dir1
```

The function refers to passed arguments by their position (not by name), that is $1, $2, and so forth:
Functions in BASH have return but it only returns the exit code. Useful in cases where you want to ‘exit’ the function and continue to the rest of the script. By default functions’ variables are in the global space, once chaged in the function is seen everywhere else. local can be used to localize the vars. Compare:

```
var=2; f() { var=3; }; f; echo $var
var=2; f() { local var=3; }; f; echo $var
```

If you happened to build a function in an alias way, redefining a command name while using that original command inside the function, you need to type command before the name of the command, like:

```
rm() { command rm -i "$@"; }
```

here you avoid internal loops (forkbombs).

Exporting a function with export -f function_name lets you pass a function to a sub-shell, by storing that function in a environment variable. Helpful when you want to use it within a command substitution, or any other case that launches a subshell, like find ... -exec bash -c 'function_name {}' \;.

### Variables

In shell, variables define your environment. Common practice is that environmental vars are written IN CAPITAL: $HOME, $SHELL, $PATH, $PS1, $RANOM. To list all defined variables printenv. All variables can be used or even redefined. No error if you call an undefined var, it is just considered to be empty:

```
# assign a variable, note, no need for ; delimiter
var1=100 var2='some string'

# calling a variable is just putting a $ dollar sign in a front
echo "var1 is $var1"

# re-assign to another var
var3=$var1

# when appending a variable, it is considered to be a string
var+=<string>/<integer>
        var1+=50 # var1 is now 10050
        var2+=' more' # var2 is 'some string more'

# we come later to how to deal with the integers (Arithmetic Expanssions $((())) below)
```

There is no need to declare things in advance: there is flexible typing. In fact, you can access any variable, defined or not. However, you can still declare things to be of a certain type if you need to:

```
declare -r var=xyz  # read-only
declare -i var     # must be treated as an integer, 'man bash' for other declare options
```

BASH is smart enough to distinguish a variable inline without special quoting:

```
dir=$HOME/dir1 fname=file fext=xyz echo "$dir/$fname.$fext"
```

though if variable followed by a number or a letter, you have to explicitly separate it with the braces syntax:

```
echo ${dir}2/${file}abc.$fext
```

Built-in vars:
• $? exit status of the last command
• $$ current shell pid
• $# number of input parameters
• $0 running script name, full path
• $FUNCNAME function name being executed, [ note: actually an array $\{\text{FUNCNAME}[\ast]\} ]
• $1, $2 ... input parameter one by one (function/script)
• "$@" all input parameters as is in one line

```bash
example() { echo -e " number of input params: $#
input params: $@
shell process
\→ id: $$
\→ script name: $0
\→ function name: $FUNCNAME"; return 1; }; f arg1 arg2;
\→ echo "exit code: $?"
```

What if you assing a variable to a variable like:

```bash
var2='something'
var1=$var2
echo $var1       # will return '$var2' literally
```

# BASH provides built-in 'eval' command that reads the string then re-evaluate it
# if variables etc found, they are given another chance to show themselves

```bash
eval echo $var1   # returns 'something'
```

In more realistic examples it is often used to compose a command string based on input parameters or some conditionals and then evaluate it at very end.

**Magic of BASH variables**

BASH provides wide abilities to work with the vars “on-the-fly” with $\{\text{var}\ldots\}$ like constructions. This lets you do simple text processing easily. These are nice, but are easy to forget so you will need to look them up when you need them.

• Assign a $\text{var}$ with default value if not defined: $\{\text{var}:=\text{value}\}$
• Returns $\text{var}$ value or a default value if not defined: $\{\text{var}:-\text{value}\}$
• Print an error_message if var empty: $\{\text{var}?:\text{error_message}\}$
• Extract a substring: $\{\text{var}:\text{offset}:	ext{length}\}$, example var=abcde; echo $\{\text{var}:1:3\} returns 'bcd'
• Variable’s length: $\{\#\text{var}\}$
• Replace beginning part: $\{\text{var}\#\text{prefix}\}$
• Replace trailing part: $\{\text{var}\%\text{suffix}\}$
• Replace pattern with the string: $\{\text{var}/\text{pattern}/\text{string}\}$
• Modify the case of alphabetic characters: $\{\text{var},\} for lower case or $\{\text{var}^^\}$ for upper case

```bash
# will print default_value, which can be a variable
var=''; echo $\{\text{var}:-default_value\}
var1=another_value; var=''; echo $\{\text{var}:-$var1\}
```

(continues on next page)
# assign the var if it is not defined
# note that we use ':' no operation command, to avoid BASH's 'command not found'

: ${var:=default_value}

# will print 'not defined' in both cases
var=''; echo ${var:?not defined}
var='' err='not defined'; echo ${var:?$err}

# will return 'love you'
var='I love you'; echo ${var:2:8}

# will return 15, that is a number of characters
var='I love you too!'; echo $\{\#var\}

# returns file.ext
var=26_file.ext; echo ${var#[0-9][0-9]_}

# in both cases returns 26_file
var=26_file.ext; echo ${var%.ext}
var=26_file.ext; echo ${var%.[a-z][a-z][a-z]}

# returns 'I hate you'
var='I love you'; echo ${var/love/hate}

# other options for substitutions
var=' some text ';
echo ${var/# /} # returns without the first space
echo ${var/% /} # without the last space
echo ${var// /} # without spaces at all

Except for the := the variable remains unchanged. If you want to redefine a variable:

var='I love you'; var=${var/love/hate}; echo $var # returns 'I hate you'

BASH allows indirect referencing, consider:

var1='Hello' var2=var1
echo $var2 # returns text 'var1'
echo ${!var2} # returns 'Hello' instead of 'var1'

To address special characters:

# replacing all tabs with the spaces in the var
var=${var//\t/' '}

[ Lecturer’s note: ~20 minutes for the hands-on exercises. Solution examples can be given at very end.]

**Exercise 2.2**

- Expand `lcd()` function to have WRKDIR as a default directory in case function is invoked without any input parameter.

- Expand the `Exersice 2.1’s ls ... && echo .. || echo example`, make a function that check any file/directory existense given as an argument, like `checkexist path/to/file`. If no argument given, function must return an error message “File or directory not found”.

- Implement a `spaceusage()` function with `du ... | sort` (see Aliases part examples) that takes directory path as an argument, and if missing uses current directory.
• Using `find` utility, implement a ‘fast find’ function `ff` word. The function should return a long listing (ls -ldA) of any file or directory names that contain the `<word>`. Make search case insensitive. Note: your newly created functions should go to `~/bin/functions` file.

• Write two functions `get_filename()` and `get_extension()`. Both should accept a full filename like `path/to/filename.ext` of any length and return `filename` or `ext` correspondingly. Extension can be of any length. Function should handle missing argument case correctly.

• Expand `get_filename()` so that it would accept extension pattern as a second argument (if given) and return `filename` out of `path/to/filename.tar.gz` or alike. I.e. `get_filename path/to/ filename.tar.gz tar.gz`

• (*) By now one should be able to explain: `:() { :|:&; };:` & in this case sends process to background. [WARNING: it is a forkbomb]

• (*) On Triton write a function that `lfs find` all the dirs/files at $WRKDIR that do not belong to your group and fix the group ownership. Use `find ... | xargs`. Tip: on Triton at WRKDIR your username $USER and group name are the same. On any other filesystem, $(id -gn) returns your group name. One can

• (*) Expand the function above to set group’s s-bit on all the $WRKDIR directories.

### Conditionals

#### Tests: `[[ ]]`

- `[[ expression ]]` returns 0=true/success or 1=false/failure depending on the evaluation of the conditional expression.
- `[[ expression ]]` is a new upgraded variation on test (also known as `[ ... ]`), all the earlier examples with single brackets that one can find online will also work with double
- Inside the double brackets it performs tilde expansion, parameter and variable expansion, arithmetic expansion, command substitution, process substitution, and quote removal
- Conditional expressions can be used to test file attributes and perform string and arithmetic comparisons

#### Selected examples file attributes and variables testing:

- `-f file` true if is a file
- `-r file` true if file exists and readable
- `-d dir` true if is a directory
- `-e file` true if file/dir/etc exists in any form
- `-z string` true if the length of string is zero (always used to check that var is not empty)
- `-n string` true if the length of string is non-zero
- `file1 -nt file2` true if `file1` is newer (modification time)
- many more others

```bash
# checks that file exists
[[ -f $file ]] && echo $file exists || { echo error; exit 1; }

# check that directory does not exist before creating one
[[ -d $dir ]] || mkdir $dir
```

8.1. Training
Note that integers have their own construction `(( expression ))` (we come back to this), though `[[ ]]` will work for them too. The following are more tests:

- `==` strings or integers are equal (`=` also works)
- `!=` strings or integers are not equal
- `string1 < string2` true if `string1` sorts before `string2` lexicographically
- `>` vice versa, for integers greater/less than
- `string =~ pattern` matches the pattern against the string
- `&` logical AND, conditions can be combined
- `||` logical OR
- `!` negate the result of the evaluation
- `()` group conditional expressions

In addition, double brackets inherit several operands to work with integers mainly:

- `-eq, -ne, -lt, -le, -gt, -ge` equal to, not equal to, less than, less than or equal to, greater than, or greater than or equal

```bash
# the way to check input arguments, if no input, exit (in functions
# 'return 1'). Remember, $# is special variable for number of arguments.
[[ $# -eq 0 ]] && { echo Usage: $0 arguments; exit 1; }

aalto=Aalto hy=HY utu=UTU

# the result will be true (0), since Aalto sorts before HY
[[ $aalto < $hy ]]; echo $?

# though with a small modification, the way around is going to be true also
[[ ! $aalto > $hy ]]; echo $?

# this will return also true, here we compare lengths, Aaaaalto has a longer... name
[[ ${#aalto} -gt ${#hy} ]]; echo $?

# true, since Aalto in both cases sorted before HY and UTU
[[ $aalto < $hy && $aalto < $utu ]]; echo $?

# false, since both fail
[[ ( $aalto < $hy && $aalto > $utu ) || $hy > $utu ]]; echo $?

# note that [[ ]] always require spaces before and after brackets
```

The matching operator `=~` brings more opportunities, because regular expressions come in play. Even more: matched strings in parentheses assigned to `$BASH_REMATCH[]` array elements!

- Regular expressions (regexs) are basically a mini-language for searching within, matching, and replacing text in strings.
- They are extremely powerful and basically required knowledge in any type of text processing.
- Yet there is a famous quote by Jamie Zawinski: “Some people, when confronted with a problem, think ‘I know, I’ll use regular expressions.’ Now they have two problems.” This doesn’t mean regular expressions shouldn’t be used, but used carefully. When writing regexs, start with a small pattern and slowly build it up, testing the matching at each phase, or else you will end up with a giant thing that doesn’t work and you don’t know why and can’t debug it. There are also online regex testers which help build them.
• While the basics (below) are the same, there are different forms of regexes! For example, the `grep` program has regular regexes, but `grep -E` has extended. The difference is mainly in the special characters and quoting. Basically, check the docs for each language (Perl, Python, etc) you want to use regexes in.

Selected operators:

• . matches any single character
• ? the preceding item is optional and will be matched, at most, once
• * the preceding item will be matched zero or more times
• + the preceding item will be matched one or more times
• {N} the preceding item is matched exactly N times
• {N,} the preceding item is matched N or more times
• {N,M} the preceding item is matched at least N times, but not more than M times
• [abd], [a-z] a character or a range of characters/numbers
• ^ beginning of a line
• $ the end of a line
• () grouping items, this what comes to `${BASH_REMATCH[@]}

```bash
# match an email
email='jussi.meikalainen@aalto.fi'; regex='(.*)@(.*)'; [[ "$email" =~ $regex ]]; echo ${BASH_REMATCH[@]}

# a number out of the text
txt='Some text with #1278 in it'; regex='#[0-9]+'; [[ "$txt" =~ $regex ]] && echo ${BASH_REMATCH[1]} || echo do not match

# case insensitive matching
var1=ABCD, var2=abcd; [[ ${var1,,} =~ ${var2,,} ]] && ... 
```

For case insensitive matching, alternatively, in general, set `shopt -s nocasematch` (to disable it back `shopt -u nocasematch`)

**Conditionals: if/elif/else**

Yes, we have `[[ ]]]` but scripting style is more logical with `if/else` construction:

```
if condition; then
  command1
elif condition; then
  command2
else
  command3
fi
```

At the `condition` place can be anything what returns an exit code, i.e. `[[ ]]]`, command/function, an arithmetic expression `$(( ))`, or a command substitution.

```
# to compare two input strings/integers
if [[ "$1" == "$2" ]]
then
```

(continues on next page)
echo The strings are the same
else
echo The strings are different
fi

# checking command output
if ping -c 1 8.8.8.8 &> /dev/null; then
  echo Online
elif ping -c 1 127.0.0.1 &> /dev/null; then
  echo Local interface is down
else
  echo No external connection
fi

# check input parameters
if [[ $# == 0 ]]; then
  echo Usage: $0 input_arg
  exit 1
fi

... the rest of the code

case

Another option to handle flow, instead of nested if's, is case.

read -p "Do you want to create a directory (y/n)? " yesno # expects user input
case $yesno in
  y|yes)
dir='dirname'
  echo Creating a new directory $dir
  mkdir $dir
  cd $dir
  ;;
  n|no)
  echo Proceeding in the current dir $(pwd)
  ;;
  *)
  echo Invalid response
  exit 1
  ;;
esac

# $yesno can be replaced with ${yesno,,} to convert to a lower case on the fly

In the example above, we introduce read, a built-in command that reads one line from the standard input or file descriptor.

case tries to match the variable against each pattern in turn. Understands patterns rules like *, ?, [], |

Here is the case that could be used as an idea for your ~/.bashrc

host=$(hostname)
case $host in
  myworkstation*)
    export PRINTER=mynearbyprinter
    # making your prompt smiling when exit code is 0 :)
  ;;
The following example is useful for Triton users: array jobs, where one handles array subtasks based on its index.

**Exercise 2.3**

- Using BASH built-in functionality implement `my_grep` pattern string script that picks up a pattern ($1) and a string ($2) as an input and reports whether pattern matches any part of the string or not. Tip: have your scripts in `~/bin`
  - The script must check that number of input parameters is correct (i.e. equal 2).
  - (*) Expand `my_grep` script to make search case insensitive
- Write a function `pathvalid` (add to `~/bin/functions` file) that validates a file path like `path/to/file`. Let’s say the path should have only alphanumeric symbols, dots, underscore and slashes as a directory delimiter.
- Implement a `my_mkdir` script that either accepts a directory name as an input parameter or requests it with `read` if no input parameter is given. Script should create a directory if it does not exist with the access permissions 700.
  - (*) Use the `pathvalid` function to add a sanity check before creating a directory.
- (*) Make a `bs` utility and place it to `~/bin` directory. The utility should help to start writing a new BASH script. It must create a file in the current directory with a given name `bs`
script_name with the `#!/bin/bash` header, set +x bit and open that newly created file in
editor of your choice.

• (*) Write a function (add to ~/bin/functions) that validates an IPv4 using =~ matching operator
only. The function should fail incorrect IPs like 0.1.2.3d or 233.204.3.257. The problem should
be solved with the regular expression only. Use return command to exit with the right exit
code.

Loops

Arithmetic

BASH works with integers only (no floating point) but supports wide range of arithmetic operators using arithmetic
expansion `$( expression )`.

• All tokens in the expression undergo parameter and variable expansion, command substitution, and quote re-
moval. The result is treated as the arithmetic expression to be evaluated.

• Arithmetic expansion may be nested.

• Variables inside double parentheses can be without a $ sign.

• BASH has other options to work with the integers, like let, expr, $[], and in older scripts/examples you
may see them.

Available operators:

• n++, n--, ++n, --n increments/decrements

• +, - plus minus

• * * exponent

• *, /, % multiplication, (truncating) division, remainder

• &&, || logical AND, OR

• expr?expr:expr conditional operator (ternary)

• ==, !=, <, >, >=, <= comparison

• =, +=, -=, *=, /=, %= assignment

• () sub-expressions in parentheses are evaluated first

• The full list includes bitwise operators, see man bash section ARITHMETIC EVALUATION.

# without dollar sign value is not returned, though 'n' has been incremented
n=10; ((n++))

# but if we need a value
n=10; m=3; q=$(n**m)

# here we need exit code only
if ((q%2)); then echo odd; fi
if ((n>m)); then ...; fi

# condition ? integer_value_if_true : integer_value_if_false
n=2; m=3; echo ${((n<m?10:100))}

# checking number of input parameters, if $# is zero, then exit

(continues on next page)
# (though the alternative [[ $# == 0 ]] is more often used, and intuitively more clear)
if ! (($#)); then echo Usage: $0 argument; exit1; fi

# sum all numbers from 1..n, where n is a positive integer
# Gauss method, summing pairs
if (($#==1)); then
  n=$1
else
  read -p 'Give me a positive integer ' n
fi
echo Sum from 1..$n is $((n*(n+1)/2))

Left for the exercise: make a summation directly 1+2+3+...+n and compare performance with the above one.

For anything more mathematical than summing integers, one should use something else, one of the option is bc, often installed by default.

# bc -- an arbitrary precision calculator language
# compute Pi number
echo "scale=10; 4*a(1)" | bc -l

For loops

BASH offers several options for iterating over the lists of elements. The options include

- Basic construction for arg in item1 item2 item3 ...
- C-style for loop for ((i=1; i <= LIMIT ; i++))
- while and until constructs

Simple loop over a list of items:

# note that if you put 'list' in quotes it will be considered as one item
for dir in dir1 dir2 dir3/subdir1; do
  echo "Archiving $dir ..."
  tar -caf ${dir///.}.tar.gz $dir && rm -rf $dir
done

If path expansions used (*, ?, [], etc), loop automatically lists current directory:

# example below uses ImageMagick's utility to convert all *.jpg files
# in the current directory to *.png.
# i.e. '*.jpg' similar to 'ls *.jpg'
for f in *.jpg; do
  convert "$f" "$(f/.jpg/.png)" # quotes to avoid issues with the spaces in the name
done

# another command line example renames *.JPG and *.JPEG files to *.jpg
# note: in reality one must check that a newly created *.jpg file does not exist
for f in *.JPG *.JPEG; do mv -i "$f" "$(f/*.jpg)"; done

# do ... done in certain contexts, can be omitted by framing the command block within curly brackets
# and certain for loop can be written in one line as well
for i in {1..10}; { echo i is $i; }
If in list omitted, for loop goes through script/function input parameters $@

```bash
# here is a loop to rename files which names are given as input parameters
# touch file{1..3}; ./newname file1 file2 file3
for old; do
  read -p "old name $old, new name: " new
  mv -i "$old" "$new"
done
```

Note: as side note, while working with the files/directories, you will find lots of examples where loops can be emulated by find ... -print0 | xargs -0 ...

Loop output can be piped or redirected:

```bash
# loop other all Triton users to find out who has logged in within last month
for u in $(getent group triton-users | cut -d: -f4 | tr ',' ' '); do
echo $u: $(last -Rw -n 1 $u | head -1)
done | sort > filename
```

The list can be anything what produces a list, like Brace expansion `{1..10}`, command substitution etc.:

```bash
# on Triton, do something to all pending jobs based on squeue output
for jobid in $(squeue -h -u $USER -t PD -o %A); do
  scontrol update JobId=$jobid StartTime=now+5days
done

# using find to make a list of files to deal with; the benefit here is that you work
# with the filename as a variable, which gives you flexibility as comparing to
# 'find ... -exec {}' or 'find ... print0 | xargs -0 ...'
for f in $(find . -type f -name '*.sh'); do
  if ! bash -n $f &>/dev/null; then
    mv $f ${f/.sh/.fixme.sh}
  fi
done
```

C-style, expressions evaluated according to the arithmetic evaluation rules:

```bash
N=10
for ((i=1; i <= N ; i++)) # LIMIT with no $
do
echo -n "$i "
done
```

Loops can be nested.

**While/until loops**

Other useful loop statement are `while` and `until`. Both execute continuously as long as the condition returns exit status zero/non-zero correspondingly.

```bash
while condition; do
  ...
done

# sum of all numbers 1..n
read -p 'Give a positive integer: ' n
i=1
(continues on next page)```
until ((i > n)); do
  ((s+=i))
  ((i++))
done
echo Sum of 1..$n is $s

# endless loop, note `:`` is a 'no operation' command in BASH, does nothing
# can be run as sort of "deamon", process should be stopped with Ctrl-c or killed
while true; do : ; done

# drop an email every 10 minutes about running jobs on Triton
# can be used in combination with 'screen', and run in background
while true; do
  squeue -t R -u $USER | mail -s 'running jobs' mister.x@aalto.fi
  sleep 600
done

# reads a file passed line by line,
# IFS= variable before read command to prevent leading/trailing whitespace from being trimmed
input=/path/to/txt/file
while IFS= read -r line; do
  echo $line
done < "$input"

# reading file fieldwise
file=/etc/passwd
while IFS=: read -r f1 f2 f3 f4 f5 f6 f7; do
  printf 'Username: %s, Shell: %s, Home Dir: %s\n' "$f1" "$f7" "$f6"
done <"$file"

# reading command output, this will be run in a subshell, and thus all variables used
# inside the loop will die when loop is over
file -b * | while read line; do
  do something with the lines
done

# to avoid above situation, one can use process substitution
while read line; do
  do something with the lines
done < <(file -b *)

All the things mentioned above for for loop applicable to while/until loops.

printf should be familiar to programmers, allows formatted output similar to C printf.⁹

Loop control

Normally for loop iterates until it has processed all its input arguments. while and until loops iterate until the loop control returns a certain status. But if needed, one can terminate loop or jump to a next iteration.

- **break** terminates the loop
- **continue** jump to a new iteration

• **break** *n* will terminate *n* levels of loops if they are nested, otherwise terminated only loop in which it is embedded. Same kind of behaviour for **continue** *n*.

Even though in most of the cases you can design the code to use conditionals or alike, **break** and **continue** certainly add the flexibility.

```bash
# here we expand an earlier example to avoid errors in case $f is missing/not accessible
for f in *.JPG *.JPEG; do
    [[ -r "$f" ]] || ( echo "$f is missing on inaccessible"; continue; )
    mv -i "$f" "${f/.*/.jpg}"
done
```

**Exercise 2.4**

• Write separate scripts that count a sum of any $1+2+3+4+..+n$ sequence, both the Gauss version and direct summation. Accept the *n* on the command line. Benchmark them with **time** for *n=10000* or more.
  
  – (***) For the direct summation one can avoid loops, how? Tip: discover **eval $(echo {1..$n})**

• Write a script or function **days_till** that counts a number of days till a deadline (or vacation/holiday). Script should accept dates suitable to **date -d** like **days_till 2019-6-1**. Tip: investigate **date +%s**.

• Using **for** loop rename all the files with the **.txt** extension to **.fixed.txt**. Tip: combine ‘for’ loop with ‘find’.

• Make script that accepts a list of files and checks if there are files in there with the spaces in the name, and if there are, rename them by replacing spaces with the underscores. Use BASH’s builtin functionality only.
  
  – As a study case, compare it against **find . -depth -name ' * * ' -execdir rename 's/ /_/g' {} \**

• (***) Get familiar with the **getent** and **cut** utilities. Join them with a loop construction to write a **mygetentgroup** script or just a oneliner that generates a list of users and their real names that belong to a given group. Like:

```bash
$ mygetentgroup group_name
meikalaj1: Jussi Meikäläinen
meikalam1: Maija Meikäläinen
...
```

• (***) To Aalto users: on kosh/lyta run **net ads search samaccountname=$USER accountExpires 2>/dev/null** to get your account expiration date. It is a 18-digit timestamp, the number of 100-nanoseconds intervals since Jan 1, 1601 UTC. Implement a function that accept a user name, and if not given uses current user by default, and then converts it to the human readable time format. Tip: [http://meinit.nl/convert-active-directory-lastlogon-time-to-unix-readable-time](http://meinit.nl/convert-active-directory-lastlogon-time-to-unix-readable-time)
  
  – Expand it to handle “Got 0 replies” response, i.e. account name not found.

**Arrays, input, Here Documents**
Aalto scientific computing guide

Arrays
BASH supports both indexed and associative one-dimensional arrays. Indexed array can be declared with declare
-a array_name, or first assignment does it automatically (note: indexed arrays only):
arr=(my very first array)
arr=('my second' array [6]=sure)
arr[5]=234

To access array elements (the curly braces are required, unlike normal variable expansion):
# elements one by one
echo ${arr[0]} ${array[1]}
# array values at once
${arr[@]}
# indexes at once
${!arr[@]}
# number of elements in the array
${#arr[@]}
# length of the element number 2
${#arr[2]}
# to append elements to the end of the array
arr+=(value)
# assign a command output to array
arr=($(command))
# emptying array
arr=()
# sorting array
IFS=$'\n' sorted=($(sort <<<"${arr[*]}"))
# array element inside arithmetic expanssion requires no ${}
((arr[$i]++))
# split a string like 'one two three etc' or 'one,two,three,etc' to an array
# note that IFS=', ' means that separator is either space or comma, not a sequence of
˓→them
IFS=', ' read -r -a arr <<< "$string"
# spliting a word to an array letter by letter
word=qwerty; arr=($(echo $word | grep -o .))

Loops through the indexed array:
for i in ${!arr[@]}; do
echo array[$i] is ${arr[$i]}
done

Negative index counts back from the end of the array, [-1] referencing to the last element.
Quick ways to print array with no loop:

8.1. Training

303


# with keys, as is
declare -p arr

# indexes -- values
echo ${!arr[@]} -- ${arr[@]}

# array elements values one per line
printf "%s\n" "${arr[@]}"

Passing an array to a function as an argument could be the use case when you want to make it local:

```bash
f() {
    local arr=(${!1}) # pass $1 argument as a reference
    # do something to array elements
    echo ${arr[@]}
}

# invoke the function, huom that no changes have been done to the original arr[@]
arr=(....)
f arr[@]
```

BASH associative arrays (this type of array supported in BASH since version 4.2) needs to be declared first (!)
declare -A asarr.

Both indexed arrays and associative can be declared as an array of integers, if all elements values are integers
declare -ia array or declare -iA. This way element values are treated as integers always.

```bash
asarr=('[university]'='Aalto University' ['city']='Espoo ['street address']=Otakaari 1')
asarr['post_index']=02150
```

Addressing is similar to indexed arrays:

```bash
for i in "${!asarr[@]}"; do
    echo asarr[$i] is ${asarr[$i]}
done
```

Even though key can have spaces in it, quoting can be omitted.

```bash
# use case: your command returns list of lines like: 'string1 string2'
# adding them to an associative array like: [string1]=string2
declare -A arr
for i in $(command); do
    arr+=(["${i// /}"]="${i// /}")
done
```

Variable expansions come out in the new light:

```bash
# this will return two elements of the array starting from number 1
${arr[@]:1:2}

# all elements without last one
${arr[@]:0:${#arr[@]}-1}

# parts replacement will be applied to all array elements
declare -A emails=('[Vesa]'='vesa@aalto.fi' '[Kimmo]'='kimmo@helsinki.fi' '[Anna]'='anna@math.
 →tut.fi')
echo ${emails[@]/@*/@gmail.com}
# returns: vesa@gmail.com anna@gmail.com kimmo@gmail.com
```
For a sake of demo: let us count unique users and their occurrences (yes, one can do it with ’uniq -c’ :)

```bash
# declare associative array of integers
declare -A arr

for i in $(w -h | cut -c1-8); do # get list of currently logged users into loop
    for u in ${!arr[@]}; do # check that they are unique
        if [[ $i == $u ]]; then
            ((arr[$i]++))
            continue 2
        fi
    done
    arr[$i]=1 # if new, add a new array element
done

for j in ${!arr[@]}; do # printing out
    echo ${arr[$j]} $j
done
```

Another working demo: script that automates backups or just makes a sync of data to a remote server. Same can be adapted to copy locally, to a USB drive or alike.

```bash
# array of directories to be backuped, to skip one, just comment with #
declare -A dirs
dirs[wlocal]=/l/$USER
dirs[xpproject]=/m/phys/extra/project/xp
dirs[homebin]=$HOME/bin

cmd='/usr/bin/rsync'           # rsync
args="-auvW --delete --progress @" # accept extra args, like '-n' for the dryrun
                             --first
serv='user@server:backups'       # copying to ~/backups that must exist

# array key is used for the remote dir name
for d in ${!dirs[@]}; do
    echo "Syncing ${dirs[$d]}..."
    $cmd $args ${dirs[$d]}/ $serv/$d
done
```

### Exercise 2.5

- make a script/function that produces an array of random numbers, make sure that numbers are unique. Print the array nicely using printf for formatting.
  - one version should use BASH functionality only (Tip: $RANDOM)
  - the other one can use shuf

- (*) Pick up the ipvalid function that we have developed earlier, implement IP matching regular expression as ^((0-9){1,3})\.(0-9){1,3})\.(0-9){1,3})\.(0-9){1,3})$ and work with the ${BASH_REMATCH[*]} array to make sure that all numbers are in the range 0-255

### Working with the input

User input can be given to a script in three ways:

- as command arguments, like ./script.sh arg1 arg2 ...
• interactively from keyboard with read command
• as standard input, like command | ./script

Nothing stops from using a combination of them or all of the approaches in one script. Let us go through the last two first and then get back to command line arguments.

read can do both: read from keyboard or from STDIN

```bash
# the command prints the prompt, waits for the response, and then assigns it
# to variable(s)
read -p 'Your names: ' firstn lastn

# read into array, each word as a new array element ('arr' declared automatically)
read -a arr -p 'Your names: '
```

Given input must be checked (!) with a pattern, especially if script creates directories, removes files, sends emails based on the input.

```bash
# request a new directory name till correct one is given (interrupt with Ctrl-C)
regexp='^[a-zA-Z0-9/_-]+$'
until [[ "$newdir" =~ $regexp ]]; do
  read -p 'New directory: ' newdir
done
```

read selected options

• -a <ARRAY> read the data word-wise into the specified array <ARRAY> instead of normal variables
• -N <NCHARS> reads <NCHARS> characters of input, ignoring any delimiter, then quits
• -p <PROMPT> the prompt string <PROMPT> is output (without a trailing automatic newline) before the read is performed
• -r raw input - disables interpretation of backslash escapes and line-continuation in the read data
• -s secure input - don’t echo input if on a terminal (passwords!)
• -t <TIMEOUT> wait for data <TIMEOUT> seconds, then quit (exit code 1)

read is capable of reading STDIN, case like command | ./script, with while read var it goes through the input line by line:

```bash
# IFS= is empty and echo argument in quotes to make sure we keep the format
# otherwise all spaces and new lines shrunk to one and leading/trailing whitespace
# trimmed
while IFS= read -r line; do
  echo "line is $line"   # do something useful with $line
done
```

Though in general, whatever comes from STDIN can be proceeded as:

```bash
# to check that STDIN is not empty
if [[ -p /dev/stdin ]]; then
  # passing STDIN to a pipeline (/dev/stdin can be omitted)
cat /dev/stdin | cut -d' ' -f 2,3 | sort
fi
```

Other STDIN tricks that one can use in the scripts:
# to read STDIN to a variable, both commands do the same
var=$(</dev/stdin)
var=$(cat)

In the simplest cases like ./script arg1 arg2 ..., you check $# and then assign $1, $2, ... the way your
script requires.

```bash
if (($#==2)); then
    var1=$1 var2=$2
    # ... do something useful
else
    echo 'Wrong amount of arguments'
    echo "Usage: ${0##*/} arg1 arg2"
    exit 1
fi
```

To work with all input arguments at once you have $@:

```bash
if (($#>0)); then
    for i; do
        echo $i
        # ... do something useful with each element of $@
        # note that for loop uses $@ by default if no other list given with 'in list'
    done
fi
```

Often, the above mentioned ways are more than enough for simple scripts. But what if arguments are like ./script
[-f filename] [-z] [-b] or more complex? (common notation: arguments in the square brackets are op-
tional). What if you write a production ready script that will be used by many other as well?

It is were `getopt` offers a more efficient way of handling script’s input options. In the simplest case `getopt`
command (do not get confused with `getopts` built-in BASH function of similar kind) requires two parameters to
work: fist is a list of letters – valid input options – and colons. If letter followed by a colon, the option requires an
argument, if folowed by two colons, argument is optional. For example, the string `getopt "sdf:"` says that the
options -s, -d and -f are valid and -f requires an argument, like -f filename. The second `getopt` argument is a list of
input parameters, often just $@.

```bash
# here is the whole trick: getopt validates the input parameters, returns the correct
# ones
# then they are reassigned back to $@ with 'set --'
opts=$(getopt "sdf:" "$@") || exit 1  # instead of exit, can be 'usage' message/
# function
set -- $opts

# note: in one line one can do it like, though ugly
#set -- $(getopt "sdf:" "$@" || kill -HUP $$)
# $( ... || exit) does not work, since exit from inside a subshell, closes the
# subshell only

# since script input parameters have been validated and structured, we can go through
# them
# we start an endless while and go through $@ with 'case' one by one
# 'shift' makes another trick, every time it is invoked, it shifts down $@ params,
# $2 becomes $1, $2 becomes $3, etc while old $1 is unset
# getopt adds -- to $@ which separates valid options and the rest that did not qualify
while :; do
    case ${1} in
```
getopt can do way more, go for man getopt for details, as an example:

```bash
# here is getopt sets name with '-n' used while reporting errors: our script name
# accepts long options like '--filename myfile' along with '-f myfile'
getopt -n $(basename $0) -o "hac::f:" --long "help,filename:,compress::" -- "$@"
```

If you implement a script that can accept both STDIN and positional parameters, you have to check both.

**Exercise 2.6**

- Make a `getemail` script that asks for the user Aalto email, check that given email is correct (`^.*@aalto\.fi$` or alike is enough) and if not, requests it again till correct one is given or the user has pressed Ctrl-C.
- Make the same script as above but accept STDIN like `echo email@address | getemail`
- Make the same script but accept the command line arguments, like `getemail -h` would return help info, `getemail -e email@domain` would accept an email.
- (*) Improve the `getemail` script:
  - join all three approaches in to one script, priority should go like: command line argument, STDIN, interactive request. Thus if email is given as an argument other two possibilities skipped.
  - for the interactive part, let it fail with the error message if wrong email is given for three times
  - make regular expression more robust, let us say email supposed to have at least eight alphanumeric character, dots can be used as a delimiter

**Here Document, placeholders**

A here document takes the lines following and sends them to standard input. It’s a way to send larger blocks to stdin.

```bash
command <<SomeLimitString
Here comes text with $var and even $() substitutions
and more just text
which finally ends on a new line with the:
SomeLimitString
```

Often used for messaging, be it an email or dumping bunch of text to file.:
mail -s 'Account expiration' $EMAIL<<END-OF-EMAIL
Dear $NAME $SURNAME,

your account is about to expire in $DAYS days.

$(date)

Best Regards,
Aalto ITS
END-OF-EMAIL

Or just outputting to a file (same can be done with echo commands):

cat <<EOF >filename
... text
EOF

One trick that is particularly useful is using this to make a long comment:

```
: <<\COMMENTS
here come text that is seen nowhere
there is no need to comment every single line with #
COMMENTS
```

**Hint** <<\LimitString to turn off substitutions and place text as is with $ marks etc

In case you have a template file which contains variables as placeholders, replacing them:

```
# 'template' file like:
The name is $NAME, the email is $EMAIL

# command to substitute the placeholders and redirect to 'output' file
# the original 'template' file remains as is
NAME=Jussi EMAIL=jussi@gmail.com
cat template | while IFS= read -r line; do eval echo $line; done > output
# resulting file: The name is Jussi, the email is jussi@gmail.com
```

**Traps, debugging, profiling**

**Catching kill signals: trap**

What if your script generates temp file and you’d like to keep it clean even if script gets interrupted at the execution time?

The built-in `trap` command lets you tell the shell what to do if your script received signal to exit. It can catch all, but here listed most common by their numbers. Note that signals are one of the common ways of communicating with running processes in UNIX: you see these same numbers and names in programs like `kill`.

- 0 EXIT exit command
- 1 HUP when session disconnected
- 2 INT interrupt - often Ctrl-c
- 3 QUIT quit - often Ctrl-
- 9 KILL real kill command, it can’t be caught
• **15 TERM** termination, by **kill** command

```bash
# 'trap' catches listed signals only, others it silently ignores
# Usage: trap group_of_commands/function list_of_signals
trap 'echo Do something on exit' EXIT
```

Expanding the backup script from the Arrays section, this can be added to the very beginning:

```bash
interrupted() {
    echo 'Seems that backup has been interrupted in the middle'
    echo 'Rerun the script later to let rsync to finish its job'
    exit 1
}

trap interrupted 1 2 15
# ... the rest of the script
```

In other situation, instead of `echo`, one can come up with something else: removing temp files, put something to the log file or output a valuable error message to the screen.

**Hint** About signals see *Standard signals* section at man 7 signal. Like Ctrl-c is **INT** (aka **SIGINT**).

### Debugging and profiling

**BASH** has no a debugger, but there are several ways to help with the debugging

Check for syntax errors without actual running it `bash -n script.sh`

Or echos each command and its results with `bash -xv script.sh`, or even adding options directly to the script. `-x` enables tracing during the execution, `-v` makes bash to be verbose. Both can be set directly from the command line as above or with `set -xv` inside the script.

```bash
#!/bin/bash -xv
```

To enable debugging for some parts of the code only:

```bash
set +x
... some code
set -x
```

If you want to check quickly a few commands, with respect to how variables or other substitutions look like, use `DEBUG` variable set to `echo`.

```bash
#!/bin/bash
$DEBUG command1 $arguments
command2
# call this script like 'DEBUG=echo ./script.sh' to see how *command1* looks like
# otherwise the script can be run as is.
```

One can also `trap` at the **EXIT**, this should be the very first lines in the script:

```bash
end() { echo Variable Listing: a = $a  b = $b; }
trap end EXIT # will execute end() function on exit
```
For a sake of profiling one can use PS4 and date (GNU version that deals with nanoseconds). PS4 is a built in BASH variable which is printed before each command bash displays during an execution trace. The first character of PS4 is replicated multiple times, as necessary, to indicate multiple levels of indirection. The default is +. Add the lines below right after ‘#!/bin/bash’

```
# this will give you execution time of each command and its line number
# \\
PS4='+\\
exec 5> ${0##*/}.$$x && BASH_XTRACEFD='5' && set -x
```

Optionally, if you want tracing output to be in a separate file:

```
PS4='+\\
exec 5> ${0##*/}.$$x && BASH_XTRACEFD='5' && set -x
```

Or to get your script looking more professional, one can enable DEBUG, i.e. tracing only happens when you run as DEBUG=profile ./script.sh:

```
case $DEBUG in
  profile|PROFILE|p|P)
    PS4='+\\
    exec 5> ${0##*/}.$$x && BASH_XTRACEFD='5' && set -x ;;
esac
```

For the larger scripts with loops and functions tracing output with the date stamps and line numbers can be summarized. For further discussion please take a look at

```
Parallel, crontab, perl/awk/sed

Running in parallel with BASH

The shell doesn’t do parallelzation in the HPC way (threads, MPI), but can run some simple commands at the same time without interaction.

The simplest way of parallelization is sending processes to a background and waiting in the script till they are done.:

```
# in the script body one may run several processes, like
command1 &
command2 &
command3 &
```

Here is an example that can be run as time script to demonstrate that execution takes 5 seconds, that is the timing of the longest chunk, and all the processes are run in parallel and finished before script’s exit.:

```
# trap is optional, just to be on the safe side
# at the beginning of the script, to get child processes down on exit
trap 'killall $(jobs -p) 2>/dev/null' EXIT

# dummy sleep commands grouped with echo and sent to the background
for i in 1 3 5; do
  { sleep $i; echo sleep for $i s is over; } &
done

# 'wait' makes sure jobs are done before script is finished
```

(continues on next page)

10 https://stackoverflow.com/questions/5014823/how-to-profile-a-bash-shell-script-slow-startup
# try to comment it to see the difference
wait
echo THE END

Putting `wait` at very end of the script makes it to wait till all the child processes are over and only then exit. Having `trap` at very beginning makes sure we kill all the process whatever happens to the script. Otherwise they may stay running on their own even if script has exited.

Another way to run in parallel yet avoiding sending to the background is using parallel. This utility runs the specified command, passing it a single one of the specified arguments. This is repeated for each argument. Jobs may be run in parallel. The default is to run one job per CPU. If no command is specified before the `--`, the commands after it are instead run in parallel.

```bash
# normally the command is passed the argument at the end of its command line. With -i option, any instances of "()" in the command are replaced with the argument.
parallel command {} -- arguments_list

# example of making a backup with parallel rsync
parallel -i rsync -auvW {} /user@server:{}.backup -- dir1 dir2 dir3

# in case you want to run a command, say ten times, the arguments can be any dummy list
parallel -i date -- {1..10}

# if no command is specified before the --, the commands after it are instead run in parallel,
parallel -- ls df "echo hi"
```

On Triton we have installed Tollef Fog Heen’s version of parallel from `moreutils-parallel` CentOS’ RPM. GNU project has its own though, with different syntax, but of exactly the same name, so do not get confused.

**Crontab**

Allows run tasks automatically in the background. Users may set their own crontabs. Once crontab task is set, it will run independently on whether you are logged in to the system or not.

Run `crontab -l` to list all your current cron jobs, `crontab -e` to start editor. When in, you may add one or several lines, then save what you have added and exit the editor normally.

```bash
# run 'script' daily at 23:30
30 23 * * * $HOME/bin/backup_script > /home/user/log/backup.log 2>&1

# every two hours on Mon-Wed,Fri
0 /2 * * 1-3,5 rm /path/to/my/tmp/dir/* >/dev/null 2>&1
```

The executable script could be a normal command, but crontab’s shell has quite limited functionality, in case of anything more sophisticated than just a single command and a redirection you end up writing a separate script.

The first five positions corresponds to:

- minute (0-59)
- hour (0-23)
• day (1-31)
• month (1-12)
• day of week (0-7, 0 or 7 is Sunday)

Possible values are:
• * any value
• , value list separator
• – range of values
• / steps

You set your favorite editor: `export EDITOR=vim` (can be a part of ~/.bashrc).

As part of the crontab file you may set several environment variables, like `MAILTO=name.surname@aalto.fi` to receive an output from the script or any possible errors. If `MAILTO` is defined but empty (`MAILTO=""`), no mail is sent.

**Perl, awk, sed**

Powerful onliners. Please consult corresponding man pages and other docs for the details, here we provide some examples. As it was stated at very beginning of the course, shell, with all its functionality is only a glue in between all kind of utilities, like `grep`, `find`, etc. Perl, awk and sed are what makes terminal even more powerful. Even though Perl can do everything what can awk and sed, one still may find tons of examples with the later ones. Here we provide some of them.

Python is yet another alternative.

```
# set delimiter to : and prints the first field of each line of passwd file (user→name)
awk -F: '{print $1}' /etc/passwd

# sort lines by length, several ways to do it
cat file | perl -e 'print sort { length($a) <=> length($b) } <>'
cat file | awk '{ print length, $0 }' | sort -n -s | cut -d" " -f2-

# placeholders replacement example above could be
NAME=Jussi EMAIL=jussi@gmail.com; sed -e "s/\$NAME/\$NAME/" -e "s/\$EMAIL/\$EMAIL/" →template

# inline word replacing in all files at once
perl -i -p -e "s/TKK/Aalto/g" *.html
```

**About homework assignments**

Available on Triton. See details in the `$course_directory`.

**References**

**To continue: course development ideas/topics**

**Additional topics:**
- select command
• revise coreutils section, expand the examples and explanations, make it clear that BASH is about getting those small utilities to work together
• benchmark: C-code vs BASH, Python vs BASH, Perl vs BASH

Ideas for exercises

• function to find all broken links
• (homework?) Implement a profiler, that summarizes PS4/date output mentioned above
• (homework?) Script that makes ‘pe1 pe2 … gpu32’ out of ‘pe[1-16],gpu11,gpu32’

In general, there could one script that one starts building from the first line up to a parallelization. Like backup script with rsync.

Git usage?

Bonus material

Parts that did not fit.

[FIXME: should be moved to another tutorial SSH: beyond login]

SSH keys and proxy (bonus section)

• SSH is the standard for connecting to remote computers: it is both powerful and secure.
• It is highly configurable, and doing some configuration will make your life much easier.

SSH keys and proxy jumping makes life way easier. For example, logging on to Triton from your Linux workstation or from kosh/lyta.

For PuTTY (Windows) SSH keys generation, please consult section “Using public keys for SSH authentication” at

On Linux/Mac: generate a key on the client machine

```
ssh-keygen -o  # you will be prompted for a location to save the keys, and a passphrase for the keys. Make sure passphrase is strong (!)
ssh-copy-id aalto_login@triton.aalto.fi  # transfer file to a Triton, or/and any other host you want to login to
```

From now on you should be able to login with the SSH key instead of password. When SSH key added to the ssh-agent (once during the login to workstation), one can login automatically, passwordless.

Note that same key can be used on multiple different computers.

SSH proxy is yet another trick to make life easier: allows to jump through a node (in OpenSSH version 7.2 and earlier -J option is not supported yet, here is an old recipe that works on Ubuntu 16.04). By using this, you can directly connect to a system (Triton) through a jump host (kosh): On the client side, add to ~/.ssh/config file (create it if does not exists and make it readable by you only):

```
Host triton triton.aalto.fi
  Hostname triton.aalto.fi
  ProxyCommand ssh YOUR_AALTO_LOGIN@kosh.aalto.fi -W %h:%p
```

Now try:

7 https://the.earth.li/~sgtatham/putty/0.70/htmldoc/
8.1.2 Python for Scientific Computing

Abstract

Python is a modern, object-oriented programming language, which has become popular in several areas of software development. This course discusses how Python can be utilized in scientific computing. The course starts by introducing the main Python package for numerical computing, NumPy, and discusses then SciPy toolbox for various scientific computing tasks as well as visualization with the Matplotlib package.

Motivation

Why Python

Python has become popular, largely due to good reasons. It’s very easy to get started, there’s lots of educational material, a huge amount of libraries for doing everything imaginable. Particularly in the scientific computing space, there is the Numpy, Scipy, and matplotlib libraries which form the basis of almost everything. Numpy and Scipy are excellent examples of using Python as a glue language, meaning to glue together battle-tested and well performing code and present them with an easy to use interface. Also machine learning and deep learning frameworks have embraced python as the glue language of choice. And finally, Python is open source, meaning that anybody can download and install it on their computer, without having to bother with acquiring a license or such. This makes it easier to distribute your code e.g. to collaborators in different universities.

Why not Python for Scientific Computing

While Python is extremely popular in scientific computing today, there are certainly things better left to other tools.

- Implementing performance-critical kernels. Python is a very slow language, which often doesn’t matter if you can offload the heavy lifting to fast compiled code, e.g. by using Numpy array operations. But if what you’re trying to do isn’t vectorizable then you’re out of luck. An alternative to Python, albeit much less mature and with a smaller ecosystem, but which provides very fast generated code, is Julia.

- Creating libraries that can be called from other languages. In this case you’ll often want to create a library with a C interface, which can then be called from most languages. Suitable languages for this sort of task, depending on what you are doing, could be Rust, C, C++, or Fortran.

- You really like static typing, or functional programming approaches. Haskell might be what you’re looking for.

Python 2 vs Python 3

There are two slightly incompatible versions of Python being used today, 2 and 3. Most large projects have supported 3 for a long time already, and have announced dropping Python 2 support for future versions (or have already done so), so at this point you should use version 3 unless you’re working on an existing project that for some reason hasn’t yet been ported to version 3. Accordingly, in this course we will use Python 3. For more info, see Python 3 statement by many other the major projects.
Practical details

The instructor will use the anaconda3/latest module available on triton. However, if you have Python 3 with the usual scientific libraries installed locally on your laptop, you should be able to use that as well, if you prefer.

For interactively testing things in Python, you can use a Jupyter notebook, or the ipython shell. For writing Python code you will need a text editor or IDE; Jupyter Lab does have one, if you prefer to work in a browser based environment. Popular free programming text editors or IDE’s with good Python support include:

- Emacs
- Vim
- VS Code
- Spyder
- Eclipse + PyDev
- PyCharm

You’re not expected to know much Python at the start of the course, but a little bit of programming proficiency is needed as a prerequisite.

Although not necessary for this course, knowledge of a version control system is useful when programming (or writing papers with LaTeX or other text-based formats). The most common and powerful version control system today is git. To get started with git, see our list of Git tutorials.

The course focuses on hands-on demonstrations and exercises rather than lectures.

Introduction to Python

If you are not familiar with Python, a very short introduction; first, the builtin scalar and collection types:

Scalars

Scalar types, that is, single elements of various types:

```python
i = 42  # integer
i = 2**77  # Integers are arbitrary precision
g = 3.14  # floating point number
c = 2 - 3j  # Complex number
b = True  # boolean
s = "Hello!"  # String (Unicode)
q = b'Hello'  # bytes (8-bit values)
```

Collections

Collections are data structures capable of storing multiple values.

```python
l = [1, 2, 3]  # list
l[1]  # lists are indexed by int
l[1] = True  # list elements can be any type
d = {"Janne": 123, "Richard": 456}  # dictionary
d["Janne"]
s = set()  # Set of unique values
```
Control structures

Python has the usual control structures, that is conditional statements and loops:

```python
x = 2
if x == 3:
    print('x is 3')
elif x == 2:
    print('x is 2')
else:
    print('x is something else')
```

While loops loop until some condition is met:

```python
x = 0
while x < 42:
    print('x is ', x)
    x += 0.2
```

For loops loop over some collection of values:

```python
xs = [1, 2, 3, 4]
for x in xs:
    print(x)
```

Often you want to loop over a sequence of integers, in that case the `range` function is useful:

```python
for x in range(9):
    print(x)
```

Another common need is to iterate over a collection, but at the same time also have an index number. For this there is the `enumerate` function:

```python
xs = ['1', 'hello', 'world']
for ii, x in enumerate(xs):
    print(ii, x)
```

Functions and classes

Python functions are defined by the `def` keyword. They take a number of arguments, and return a number of return values.

```python
def hello(name):
    """Say hello to the person given by the argument""
    print('Hello', name)
    return 'Hello ' + name
```

Classes are defined by the `class` keyword:

```python
class Hello:
    def __init__(self, name):
        self._name = name
    def say(self):
        print('Hello', self._name)
```
Python type system

Python is strongly and dynamically typed.

Strong here means, roughly, that it’s not possible to circumvent the type system (at least, not easily, and not without invoking undefined behavior).

```python
x = 42
type(x)
x + "hello"
```

Dynamic typing means that types are determined at runtime, and a variable can be redefined to refer to an instance of another type:

```python
x = 42
x = "hello"
```

*Jargon:* Types are associated with rvalues, not lvalues. In statically typed language, types are associated with lvalues, and are (typically) reified during compilation.

Organizing Python code

Start Python scripts with

```bash
#!/usr/bin/env python3
```

This ensures you get the correct python3 for the environment you are using.

In general, don’t put executable statements directly into the top level scope in your files (modules), as this code is then run if you try to import the module.

Instead, use this common idiom:

```python
if __name__ == '__main__':
    # your code goes here
```

When developing code it’s often convenient to be able to reload a module into your IPython (or IPython notebook) session without having to restart the entire session. This can be done with the `reload` function:

```python
from importlib import reload
import foo
foo.bar()
# Edit foo.py
reload(foo)
foo.bar()
```

Exercise 1.1

Who needs numpy anyway? Implement matrix multiplication with nested lists as your matrix representation. *Hint for beginners:* Create one function

```python
def creatematrix(n, m):
    # ...
```

which creates an NxM matrix filled with random values (e.g. `random.random()`). Then create another function
def matrixmult(a, b):
    # ...

which multiplies together two matrices a and b.

Exercises 1.2

Let's continue with the previous example, and add some object oriented scaffolding around our matrix code. Create a Matrix class with a constructor to create the random matrix, and overload the '*' operator to multiply two Matrix instances. Reuse the code from the previous exercise.

Exercise 1.3

The essence of science is experiment and measurement. So let's measure our matrix multiplication implementation, and calculate how fast it can multiply matrices, in terms of “Gflops/s” (Giga floating point operations per second). Hint: A “flop” is a floating point multiply or addition/subtraction. First figure out of many flops are needed to multiply two matrices. Then you need to time it; for this you can use the IPython magic %timeit command. And finally, equipped with this information, you can calculate a Gflops/s score for your multiplication method.

Exercise 1.4

Basic file I/O. Run the following python snippet to create a file pangrams.txt:

```python
with open('pangrams.txt', 'w') as f:
    f.write("The quick brown fox jumps over the lazy dog
Sphinx of black quartz, judge my vow
The dog ate my homework
Pack my box with five dozen liquor jugs"
"")
```

Next, create Python code to read that file, and check each line whether it's a pangram. A pangram is a sentence to use all the letters of the alphabet.

Enter NumPy

Introduction

The NumPy package provides a N-dimensional array type, and syntax and utility functions for working with these arrays.

In contrast to a python list, a numpy array can only hold elements of the same type. The element type can be seen via the 'dtype' attribute.

```python
import numpy as np
da = np.array(([1, 2, 3], [4, 5, 6]))
da.dtype
da[0, 0] = "hello" # error!
da[0, 0] = 2**100 # error!
```
What these restrictions buy you is that the memory layout of a numpy array is very efficient, similar to what you see in low level languages like C or Fortran. This means operating on these arrays is very efficient; in fact, much of the speed advantage of numpy comes from the fact that array syntax is implemented in fast C code.

Due to the memory layout of numpy being compatible with C and Fortran, numpy arrays allows one to use functionality written in these other languages. Much of the SciPy ecosystem (NumPy, SciPy, etc.) consist of python wrappers around widely used and battle-tested numerical libraries written in C or Fortran such as LAPACK and BLAS.

The Python list

```python
a_list = [1, "hello", 1.2]
```

has roughly the following layout in memory:

In contrast, the NumPy array

```python
n = np.array((1,2,3))
```

has the memory layout like

**Exercise 2.1**

1. In the example above we saw that \(2^{100}\) was too large. What is the default datatype of a numpy integer array if we don’t explicitly specify some type, and what is the largest possible integer we can store in such an element.

2. What is the smallest negative element (that is, the largest absolute value of a negative number)? Is it different from the largest positive number, and if so, why?

3. What is the absolute value of the smallest negative element? Why?

**Other ways of creating NumPy arrays**

There are many different ways to create NumPy arrays, here’s a few of the most common ones:

```python
np.zeros((2, 3))  # 2x3 array with all elements 0
np.ones((3, 2), bool)  # 3x2 boolean array
np.arange(3)  # Evenly spaced values in an interval
np.linspace(...)  # similar to above
```

**NumPy array slicing syntax**

NumPy provides a convenient array syntax to reference subarrays, similar to MATLAB for Fortran.

```python
a[low:high:step]
```

returns the array elements in the range \([low, high)\) with a stride of \(step\). Equivalently for multidimensional arrays. For multidimensional arrays NumPy by default stores arrays in row-major order, like C. Note that this is in contrast to e.g. Fortran, MATLAB or Julia that use a column-major layout.

Using array syntax efficiently is key to using NumPy in a fashion that leads to short as well as efficient code. NumPy also provides so-called advanced indexing, where you can select elements with a list of indices.
Views vs. copies

When slicing an array, you DO NOT get a copy of those elements, but rather a view. That is, the data elements are the same as in the original array.

Views rather than copies is more efficient, particularly for large arrays, but they can sometimes be confusing. Be careful!

If you do need a copy, NumPy arrays have a copy method to create a copy rather than getting a view.

NOTE With advanced indexing, you always get a copy!

Array shape and size

NumPy arrays have a shape and size attribute.

We can modify the shape of an array with the reshape or resize methods. Or for the special case of flattening an array to a 1D array, ravel.

Combining, splitting and rolling arrays

For combining multiple arrays into a larger array, see the concatenate, stack, block, and the more specialized variants hstack, vstack, dstack.

Similarly, for splitting an array into multiple parts, there’s split, hsplit, vsplit.

To roll an array, that is shift the elements along a give axis, use roll.

Exercise 2.2

Create an array $x$ of 100 evenly spaced numbers in the range $[-2\pi, 2\pi]$.

Next, create an array $y$, where each element is the sin of each element in the previously created array.

Then, figure out the indices where the array $y$ changes sign. What are the $x$ values for these indices?

NumPy I/O

NumPy has functionality for saving and loading NumPy arrays from files. For reading/writing textfiles there is loadtxt and savetxt. See also genfromtxt with more sophisticated handling of missing values etc.
For large arrays, it’s faster to use a binary format. For these NumPy defines a .npy format. Loading and saving these files can be done with the load and save methods. There’s also the .npz format, which is a zip archive containing several numpy ndarrays in one file. .npz format files can be read/written with load, savez and savez_compressed methods. This is a good choice for temporary or intermediate files such as checkpoints etc. Note that the format is Numpy-specific, and other languages might not easily be able to read it. Similarly, for long-term archiving other formats might be a better choice.

**Random Numbers in NumPy**

The numpy.random module contains functionality to create pseudorandom numbers following different distributions.

**Linear algebra in Numpy**

The dot method provides a generalized dot product. It can compute dot products of 1D vectors, matrix-vector products as well as matrix-matrix products. It is an interface to the famous BLAS library, of which multiple highly optimized versions exist. The numpy.linalg module contains interfaces to the most common linear algebra operations, such as calculating eigenvalues, Cholesky and singular value decompositions, solving linear systems, least squares, (pseudo)inverse. This module is an interface to the LAPACK library (which in turn builds on top of BLAS).

**Exercise 2.3**

Remember our first exercise, implementing matrix multiplication? Now do the same, but use NumPy arrays and the dot method. Compare performance to the code you wrote yourself earlier, using the IPython %timeit macro.

**Exercise 2.4**

Here’s a number of quick numpy exercises to get you a feel of numpy functionality, index manipulation etc.

1. Reverse a vector. Given a vector, reverse it such that the last element becomes the first, e.g. [1, 2, 3] => [3, 2, 1]
2. Create an identity matrix of size 4x4.
3. Create a 2D array with zeros on the borders and 1 inside.
4. Create a random array with elements [0, 1), then add 10 to all elements in the range [0.2, 0.7).
5. What is np.round(0.5)? What is np.round(1.5)? Why?
6. In addition to np.round, explore np.ceil, np.floor, np.trunc. In particular, take note of how they behave with negative numbers.
7. Recall the identity \( \sin^2(x) + \cos^2(x) = 1 \). Create a random 4x4 array with values in the range [0, 10). Now test the equality with np.equal. What result do you get with np.allclose instead of ``np.equal``?
8. Create a 1D array with 10 random elements. Sort it.
9. What’s the difference between np_array.sort() and np.sort(np_array)?
10. For the random array in question 8, instead of sorting it, perform an indirect sort. That is, return the list of indices which would index the array in sorted order.
11. Create a 4x4 array of zeros, and another 4x4 array of ones. Next combine them into a single 8x4 array with the content of the zeros array on top and the ones on the bottom. Finally, do the same, but create a 4x8 array with the zeros on the left and the ones on the right.
More quick NumPy exercises like this one over here.

Exercise 2.5

The topic of this exercise is `np.einsum` which implements the ‘Einstein summation convention’. The Einstein summation convention is a commonly used when working with tensors, but can also be useful for succinctly representing array expressions.

For instance, matrix multiplication can be expressed as

\[ C^i_k = A^i_j B^j_k \]

Implement this with the help of `np.einsum`.

Einstein notation is also available in pytorch and tensorflow. For more information about Einstein notation in NumPy see

- A basic introduction to NumPy’s einsum
- Einste Summation in NumPy
- Einsum is all you need - Einstein summation in deep learning

SciPy

SciPy is a library that builds on top of NumPy. It contains a lot of interfaces to battle-tested numerical routines written in Fortran or C, as well as python implementations of many common algorithms. Briefly, it contains functionality for

- Special functions (Bessel, Gamma, etc.)
- Numerical integration
- Optimization
- Interpolation
- Fast Fourier Transform (FFT)
- Linear algebra (more complete than in NumPy)
- Sparse matrices
- Statistics
- More I/O routine, e.g. Matrix Market format for sparse matrices, MATLAB files (.mat), etc.

Exercise 3.1

Using scipy, calculate the integral of the function \( \sin \) in the interval \([0, \pi]\), and compare with the analytical result.

Exercise 3.2

Use the SciPy sparse matrix functionality to create a random sparse matrix with a probability of non-zero elements of 0.05 and size 10000 x 10000. The use the SciPy sparse linear algebra support to calculate the matrix-vector product of the sparse matrix you just created and a random vector. Use the `%timeit` macro to measure how long it takes. Does the optional `format` argument when you create the sparse matrix make a difference?
Then, compare to how long it takes if you’d instead first convert the sparse matrix to a normal NumPy dense array, and use the NumPy `dot` method to calculate the matrix-vector product.

Can you figure out a quick rule of thumb when it’s worth using a sparse matrix representation vs. a dense representation?

**Matplotlib**

Matplotlib is the ‘standard’ Python plotting library. It is quite full-featured, and provides a MATLAB-like plotting API.

To use it, typically you start with

```python
import matplotlib.pyplot as plt
# ...
x = linspace(-4, 4)
plt.plot(x, np.sin(x))
plt.show()
```

When using Jupyter notebooks, use the magic

```python
%matplotlib inline
```

which will cause matplotlib plots to appear inline in the notebooks. Very convenient for quick analysis!

Matplotlib has two slightly different interfaces, a state machine interface similar to MATLAB and an object based interface. The state machine interface is quick and easy to get started, but since it’s based on hidden global state behind the scenes, for more complex stuff it might get confusing. Below is an example using the state machine interface.

```python
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(0.0, 3.0)
y1 = np.cos(2 * np.pi * x) * np.exp(-x)
y2 = np.cos(2 * np.pi * x)
plt.subplot(2, 1, 1)
plt.plot(x, y1, 'o-')
plt.title('A tale of 2 subplots')
plt.ylabel('Damped oscillation')
plt.subplot(2, 1, 2)
plt.plot(x, y2, '.-')
plt.xlabel('time (s)')
plt.ylabel('Undamped')
plt.show()
```

And here is the same thing, but using the object-based interface

```python
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(0.0, 3.0)
```

(continues on next page)
Exercise 4.1

Try to recreate the figure below:

Exercise 4.2

Create 1000 normally distributed numbers with $\mu = 0$ and $\sigma = 10$. Then create a histogram plot with 50 bins.

Exercise 4.3

Often it’s useful to be able to plot things on a logarithmic scale. Create a plot with 4 subplots, one with a linear scale, one with logarithmic scale on the x-axis, one with logarithmic scale on the y-axis. Then create 4 functions, such that each will produce a straight line in one of the plots, and plot them.

Image and pseudocolor plots

Matplotlib can also plot 2D data such as images. A common type of 2D plot is the pseudocolor plot, where you want to convert a scalar value in some range into a color value. This means that you must map the range of values into a colormap. Things to think about when selecting a colormap:

- The map should be perceptually uniform. From https://bids.github.io/colormap/ : A “perceptually uniform” colormap is one for which the “perceptual deltas” plot makes a simple horizontal line. (This is essentially the derivative of the colormap in perceptual space with respect to the data. We want our colormap to have the property that if your data goes from 0.1 to 0.2, this should create about the same perceptual change as if your data goes from 0.8 to 0.9. For color geeks: we’re using CAM02-UCS as our model of perceptual distance.)
- It should look good when rendered in gray-scale, e.g. if someone prints the picture on a black-and-white printer. Or sometimes per-page costs in journals are lower if everything is BW.
- It should make sense to people with the most common type of color blindness (red-green). In practice this means the color maps shouldn’t use both green and red colors, so that they are not confused.

Bad news: The commonly used rainbow (or “jet” as it’s often called) is very bad when comparing against these criteria! NEVER USE IT! It was the default in matplotlib < 2.0, and in MATLAB for a long time.
**Good news:** Matplotlib >= 2.0 has sane defaults here! See changes in default styles for matplotlib 2.0. In particular, compare matplotlib < 2.0 default colormap and the matplotlib >= 2.0 default colormap. As an aside, the 2.0 default colormap (“viridis”) is similar to the default colormap in current versions of MATLAB (“parula”).

One case where you should NOT use viridis is if your data has some “natural” zero point. In that case it’s better to use a “cool-warm” style colormaps, see “Diverging Colormaps” at the matplotlib colormap reference.

```python
N = M = 200
X, Y = np.ogrid[0:20:N*1j, 0:20:M*1j]
data = np.sin(np.pi * X*2 / 20) * np.cos(np.pi * Y*2 / 20)

fig, (ax2, ax1) = plt.subplots(1, 2, figsize=(7, 3))
im = ax1.imshow(data, extent=[0, 200, 0, 200])
ax1.set_title("v2.0: 'viridis'")
fig.colorbar(im, ax=ax1, shrink=0.8)
im2 = ax2.imshow(data, extent=[0, 200, 0, 200], cmap='jet')
fig.colorbar(im2, ax=ax2, shrink=0.8)
ax2.set_title("classic: 'jet'")
fig.tight_layout()
```

**Exercise 4.4**

Find an image on the internet or already on the machine you’re working on, load it into a NumPy array (you can use `imageio.imread()` for this), and see if you can create an Andy Warhol-like print by using a suitable colormap.

**Demo application**

To demonstrate how to make a simple simulation program, here the lecturer will ‘live-code’ a small simulation program. For a suitable model, let’s choose a suitably fascinating problem that can be simulated with a relatively simple model. First, some background.

**Topological phase transitions**

Historically, for a long time we believed there were two, and only two, kinds of phase transitions in nature. So-called discontinuous, or first-order, transitions which are characterized by the presence of a latent heat (mathematically, a discontinuity in the first derivative of the free energy with respect to some thermodynamic parameter), whereas continuous phase transitions are characterized by a discontinuity in the second or higher derivative of the free energy.

However, in the 1970’ies, some experiments on ultrathin films of superfluid Helium-3 were made which produced data that existing theories could not describe. Eventually Kosterlitz and Thouless (and independently Berezinskii in the then Soviet Union) were able to describe what was happening. What they had discovered was an entirely new kind of phase transition which defied the existing classification schemes. Namely, there is NO discontinuity in any free energy derivative. So in a way, it’s an infinite-order phase transition.

What is happening is that topological defects (vortices in this case) in the system change how they interact with each other at the critical temperature. At low temperatures below the transition temperature the correlation function between spins decays as a power law, whereas above the transition temperature the correlation decays exponentially. This results in vortex-antivortex pairs at low temperature, and a vortex unbinding transition at the transition temperature with free vortices at higher temperatures.

This work eventually resulted in the 2016 Nobel Prize in Physics. See the scientific background for the 2016 physics prize.
The XY model

Topological phase transitions can be studied with a XY model (also called the planar model, or rotor model). Take a lattice with spins rotating in the plane. Each spin interacts with its neighbors, and the configuration energy of the system is given by

$$E = -J \sum_{i \neq j} s_i s_j,$$

where the sum is over nearest neighbor spins.

In this case we can ignore the constant $J$ which determines the interaction strength. Also, since the spin vectors are all of equal lengths the dot product can be simplified, so we have

$$E = -\sum_{i \neq j} \cos(\theta_i - \theta_j).$$

The Metropolis-Hastings Monte Carlo algorithm

The Metropolis-Hastings algorithm is a Markov chain Monte Carlo method that can be used for sampling a probability distribution. In this case, the basic idea is that for each spin $s$ we do a trial move, to change the spin. We then calculate a random trial spin $s'$, and calculate an acceptance probability

$$A = \min(1, \frac{P(s')}{P(s)}).$$

In this case the probability density is the Boltzmann distribution

$$P(s) = \frac{1}{Z} \exp(-\beta E(s)),$$

where $\beta$ is the thermodynamic beta, or

$$\beta = \frac{1}{k_B T},$$

where $k_B$ is the Boltzmann constant. For this simulation we can set it to 1 and ignore it hereafter. $\beta$ is thus just the inverse of the temperature.

Thus the quotient

$$\frac{P(s')}{P(s)}$$

can be calculated as

$$\exp(-\beta(E' - E)).$$

Then finally, calculate a uniform random number $r$ in the interval $[0,1)$. If $r \leq A$ the new state is accepted. Repeating this for all the spins constitutes a single Monte Carlo step in the algorithm.

Other useful Scientific Python libraries

A few other widely used libraries in the Scientific Python ecosystem:

- **Pandas**: Python Data Analysis library. Pandas gives Python a dataframe type, similar to data frames in R, which is useful for representing tabular data where every column can be of a different type. If you’re interested in this topic, see the *Practical R and Python Data Analysis* course by Aalto Science-IT.
• scikit-learn: Machine Learning library. Implementations of the most common ML algorithms such as SVM, random forest, k-means, etc.

• Seaborn: Statistical data visualization. Plotting library that builds on top of matplotlib, providing a higher level interface aimed at visualizing statistical data.

• Cython: C-extensions for Python. Write fast C code in an extended subset of Python syntax.

• Numba: JIT compiler that can accelerate (some) loops with NumPy expressions.

• MPI for Python (mpi4py): Python bindings for the Message Passing Interface (MPI) standard for creating parallel applications using Python.

• SymPy: Symbolic mathematics in Python.

Homework: 2D Ising model

This homework exercise shares many similarities with the XY model studied above. The main difference is that in the 2D Ising model, the spins are perpendicular to the plane, and can take only two values, +1 and −1. This model can be used to study the ferromagnetic phase transition. Below the critical temperature ferromagnetic domains, where the spins are aligned, form. Above the critical temperature this order breaks down. In the Ising model the configuration energy is defined as

\[ E = -J \sum_{i \neq j} \sigma_i \sigma_j - \mu H \sum_j \sigma_j, \]

where J is the exchange energy, \( \mu \) is the magnetic moment of the spins, and H is the external magnetic field in the direction perpendicular to the plane. To simplify, you can set J and \( \mu \) to 1.

Implement a simulation program simulating the 2D Ising model. Use the Metropolis-Hastings Monte Carlo algorithm. Visualize the results with matplotlib. Run the simulation at different temperatures and with different starting configurations (random vs. ordered), and see if you can find the critical temperature by observing your visualizations.

If you find the above too easy, a few topics for further exploration. Not needed to pass the course.

• Implement the Wolff algorithm, which flips whole clusters at a time instead of individual spins. This helps avoid a phenomena called critical slowing down close to the critical temperature, which is problematic for algorithms such as the Metropolis algorithm that flip one spin at a time.

• Calculate and plot the net magnetization, the magnetic susceptibility, and the heat capacity of the system as a function of the temperature. How do they behave around the critical temperature?

For course announcements at Aalto, see the Scientific Computing in Practice courses page.

8.1.3 A: Basics

<table>
<thead>
<tr>
<th>A01 University IT systems</th>
<th>This covers the basics of research facilities at Aalto and how to use them.</th>
<th>For now see Welcome, researchers!</th>
</tr>
</thead>
<tbody>
<tr>
<td>A10 Configuring Mac for scientific work</td>
<td>Getting your Mac computer set up for scientific computing tasks. After this, you can follow most of the other instructions below which assume a Linux-like system.</td>
<td></td>
</tr>
<tr>
<td>A11 Configuring Windows for scientific work</td>
<td>Like A10, but for Windows. (Why isn’t there a Linux course? Because these are to get you close enough to Linux to have the power you need for computing.)</td>
<td></td>
</tr>
</tbody>
</table>
8.1.4 B: Scientific computing

Core courses:

<table>
<thead>
<tr>
<th>Course</th>
<th>Description</th>
<th>Reading</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td>B10 Basic shell</td>
<td>Let’s face it: the Linux command line is the basis of most data science if you are doing more than running other people’s programs.</td>
<td>Software Carpentry shell-novice sections 1-4. Our shell course covers this at the beginning, too.</td>
<td></td>
</tr>
<tr>
<td>B14 Data management</td>
<td>If you do the obvious thing, your data will turn into a huge mess and you won’t be able to work anymore. This course gives some practical hints.</td>
<td>(For now, check out the data section)</td>
<td></td>
</tr>
<tr>
<td>B23 Text editors and IDEs</td>
<td>Your best friend is a good text editor - sometimes you just need to edit things quickly on some remote system.</td>
<td>Software Carpentry shell-novice, part of section 3.</td>
<td></td>
</tr>
<tr>
<td>B20 Shell scripting</td>
<td>If you can do it on the Linux shell, you can automate it.</td>
<td>Continue with the Science-IT Linux shell tutorial.</td>
<td></td>
</tr>
<tr>
<td>B21 Version control for you</td>
<td>Version control lets you track changes, go back in time, and collaborate on code and papers: an absolute requirement for scientific computing.</td>
<td>CodeRefinery Introduction to version control</td>
<td></td>
</tr>
</tbody>
</table>

Other courses:

<table>
<thead>
<tr>
<th>Course</th>
<th>Description</th>
<th>Reading</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td>B30 Makefiles</td>
<td>Makefiles are like smart shell scripts. We learn some about them and in the process, become ever more efficient.</td>
<td>Software Carpentry make-novice.</td>
<td></td>
</tr>
<tr>
<td>B50 Version control for teams</td>
<td>Previously, you learned only the basics. Now for the real stuff.</td>
<td>CodeRefinery collaborative distributed version control lesson</td>
<td></td>
</tr>
<tr>
<td>B51 Jupyter Notebooks</td>
<td>Notebooks are an efficient way to make self-documenting code and scripts and do data science well.</td>
<td>CodeRefinery Jupyter course.</td>
<td></td>
</tr>
</tbody>
</table>

**Software development track:** Do you do programming? These courses are for you. This does not teach you how to program: you need to find your own course for that, but this will make sure you can do scientific programming well.
8.1.5 C: High performance computing

When your own computer is not enough, you need more power. For that, high-performance computing is your next step. Level C is about using HPC, level D is about programming it yourself.

Core courses:

<table>
<thead>
<tr>
<th>Course Code</th>
<th>Course Name</th>
<th>Reading</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td>C01</td>
<td>What is HPC?</td>
<td>Before you can use larger resources, you need to understand the difference from your own computers</td>
<td>training by Science-IT, About Science-IT and Triton</td>
</tr>
<tr>
<td>C20</td>
<td>Modules and software</td>
<td>Using and installing software on a cluster is different from your own computer, because hundreds of people are sharing it. Modules are the solution.</td>
<td>Scientific Computing in Practice training by Science-IT or Software Modules</td>
</tr>
<tr>
<td>C21</td>
<td>Slurm</td>
<td>On a cluster, you have to share resources with others. Slurm is one batch queuing system that makes it possible.</td>
<td>See Scientific Computing in Practice training by Science-IT or interactive, serial, array</td>
</tr>
<tr>
<td>C22</td>
<td>HPC Storage</td>
<td>Storage turns out to be just as important as computing power. There are different places available, each with different advantages.</td>
<td>See Scientific Computing in Practice training by Science-IT or storage basics, lustre, local storage, small files</td>
</tr>
<tr>
<td>C23</td>
<td>Parallel computing</td>
<td>The point of a cluster is to run things in parallel. How does this work?</td>
<td>See Scientific Computing in Practice training by Science-IT, and Parallel computing.</td>
</tr>
<tr>
<td>C24</td>
<td>Advanced shell scripting and automation</td>
<td>Hands-on shell scripting, putting everything together to automate large computations on the cluster.</td>
<td>Various courses, finishing the linux shell tutorial is a good start.</td>
</tr>
</tbody>
</table>
8.1.6 D: Advanced high performance computing

<table>
<thead>
<tr>
<th>Dxx Parallel programming computers</th>
<th>This is an academic course taught in the CS department. It mainly covers OpenMP and CUDA. Usually taught in 5th period (Apr-May), search MyCourses/Oodi for CS-E4580.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dxx GPU Programming</td>
<td>This was an advanced guest course, useful if you want to know how to program GPU applications.</td>
</tr>
<tr>
<td>Dxx MPI Programming</td>
<td>This was an advanced guest course, useful if you want to know internals of MPI or program MPI applications.</td>
</tr>
<tr>
<td>Dxx HTCondor</td>
<td>Condor allows you to use many workstations as a high throughput cluster, ideal for mid-range embarrassingly parallel problems.</td>
</tr>
</tbody>
</table>

Also see the Science-IT training archive for more level D courses.

8.1.7 Recommended programming courses

Need to learn programming? We will include some recommended online programming courses here.
These docs are open source: all content is licensed under CC-BY 4.0 and all examples under CC0 (public domain). Additionally, this is an open project and we strongly encourage anyone to contribute. For information, see the About these docs and the Github links at the top of every page. Either make Github issues, pull requests, or ask for direct commit access. Be bold: the biggest problem is missing information, and mistakes can always be fixed.

9.1 About these docs

These docs originally came from the Triton User Guide, but now serves as a general Aalto scientific computing guide. The intention is a good central resources for researchers, kept up to date by the whole community.

9.1.1 Contributing

This documentation is Open Source (CC-BY 4.0), and we welcome contributions from the Aalto community. The project is run on Github (https://github.com/AaltoScienceIT/triton-docs).

To contribute, you can always use the normal Github contribution mechanisms: make a pull request or comments. If you are at Aalto, you can also get direct write access. Make a github issue, then contact us in person/by email for us to confirm.

The worst contribution is one that isn’t made. Don’t worry about making things perfect: since this is in version control, we track all changes and will just fix anything that’s not perfect. This is also true for formatting errors - if you can’t do ReStructuredText perfectly, just do your best (and pretend it’s markdown because all the basics are similar). Contributing gives consent to use content under the licenses (CC-BY 4.0 or CC0 for examples).

9.1.2 Requirements and building

The only software needed is Sphinx: Debian package python-sphinx, : PyPI: python-sphinx. It is already installed on Aalto workstations.

To build the docs, run make html.
HTML output is in _build/html/index.html, and other output formats are available as well.

### 9.1.3 Editing

Look at examples and copy. To add sections, add a new page in a subfolder. Link it from the main Table of Contents (toctree) in index.rst to have the document appear and be cross-referenced.

You can see a complete example from UiT: source and compiled HTML.

### 9.1.4 ReStructured text

ReStructured Text is similar to markdown for basics, but has a more strictly defined syntax and more higher level structure. This allows more semantic markup, more power to compile into different formats (since there isn’t embedded HTML), and advanced things like indexing, permanent references, etc.

Restructured text home and quick reference.

Note: Literal inline text uses `:` instead of a single `'` (second works but gives warning).

A very quick guide is below.

---

**Inline code/monospace, emphasis, strong emphasis**

```
``Inline code/monospace```, `emphasis`, `**strong emphasis**`


```
Block quote
Block quote
```

```
::
  Block quote
  Block quote
```

Block quotes can also start with paragraph ending in double colon, like this:

```
Block quote
```

Block quotes can also start with paragraph ending in double colon, like this:

```
Block quotes can also start with paragraph ending in double colon, like this::
  Block quote
```

**Inline link, or anonymous, or separate, or different text links. Trailing underscores indicate links.**

```
Inline `link <http://python.org>`_, or anonymous__, or separate_, or `different text <separate_>`_ links. Trailing underscores indicate links.
```

(continues on next page)
Linking to the web. If possible use a permanent reference (next section), but you can also refer to specific files by name. Note, that for internal links there are no trailing underscores:

```rst
.. _doc:`../tut/interactive.rst` (recommended)
`../tut/interactive.rst` (short, no warning if link breaks)
```

With different text:

```rst
.. _doc:`Text <../tut/interactive.rst>` (recommended)
`Text <../tut/interactive.rst>` (short, no warning if link breaks)
```

Internal links. Permanent references across files

Label things this way (note only one colon):

```rst
.. _label-name:
```

Reference them this way:

```rst
.. _ref:`label-name` (recommended)
`label-name` (short, no warning if link breaks)
`Text <label-name>` (short, no warning if link breaks)
```

Notes, warnings, etc.

**Note:** This is a note

**Warning:** This is a warning

```rst
.. note::
   This is a note
   .. warning::
   This is a warning
```

- genindex
- search