homepage

Release becad15

Matthias Geier

2019-07-27

Contents

This is some stuff I wrote/collected:

CHAPTER 1

Getting Started with Git

There are a lot of pages on the web dedicated to Git, this page just shows a few random bits of information which might or might not help you getting started.

Git is a distributed version control system (DVCS) and it's great!

That's about all I'm gonna say about it in general, if you want to read more about it, have a look at the links at the end of this page.

1.1 Cloning a Repository

To obtain the contents of a remote repository, you have to *clone* it. For example, to clone the repository where this very page was created from, do:

```
git clone https://github.com/mgeier/homepage.git
```

This will create a directory named like the repository (in this case `homepage/`).

```
cd homepage
```

1.2 Checking the Status

At any time, you can get the current status of your files with:

```
git status
```

Here you will see if there are changes to any files of the repository and if there are local files which are not yet part of the repository (see below for how to add files).

1.3 Getting Recent Changes from the Server

If the repository on the server changed since you cloned it, you can get up to date with:

```
git pull
```

But be aware that if you made changes to your local files, this may lead to conflicts. It's a good idea to always commit before pulling (see below how to commit changes).

1.4 Initial Setup

Before you make your first commit, you should set up a few things (you have to do this only once):

```
git config --global user.name "Your Name"
git config --global user.email you@example.com
```

You may also want to set your favorite editor:

```
git config --global core.editor "gvim --nofork"
```

To enable colors in `git log` et al.:

```
git config --global color.ui auto
```

And, if you want, you can set a few aliases for your convenience:

```
git config --global alias.s "status --short"
git config --global alias.c "checkout"

git config --global alias.lol "log --oneline --graph --decorate"
git config --global alias.lola "log --oneline --graph --decorate --all"
```

With these aliases, you can use `git s` to get a short status display (one line per file) and you can use `git c mybranch` instead of the significantly longer `git checkout mybranch`. I find myself using *checkout* very often, so `git c` saves a lot of typing.

If you type `git lola`, you'll see a nice and colorful ASCII display of the previous commits and their tree structure.

All these options are stored in `~/.gitconfig` (or somewhere else depending on your operating system). You can also store settings on a per-project basis. Just drop the `--global` option and the settings will be stored for your current Git repository in `.git/config`.

1.5 Command Line Prompt

When working with branches, it is crucial to know the currently active branch. To show the current branch in the command line prompt, put this at the end of your `~/.bashrc` (or wherever else you store settings for your shell):

```
# add git branch to prompt
GIT_PS1_SHOWDIRTYSTATE=1
PS1="${PS1%'\$ '}"'$(__git_ps1 " (%s)")'"\$ "
```

You'll also see an asterisk, e.g. `(master *)`, if your working directory is *dirty*, i.e. if you have local changes which are not yet committed.

1.6 Committing Changes

Any new files have to be added to Git control before you can do anything with them:

```
git add myfile.txt
```

Once you have done some changes, you can make a commit:

```
git commit -a
```

Here, your favorite text editor will be opened and you can (and *should*!) enter a commit message, describing the changes you have made. After you save the file and close the editor, the commit will actually be created.

A commit message should have a short (no more than 50 characters) one-line summary in the first line, then a blank line and then a more detailed description. Have a look at this note about commit messages.

Remember, a commit is a local operation in Git, so nothing was transferred to the server yet.

Todo: add before commit, staging area, local commits

1.7 Pushing Your Changes to the Server

After one or several commits, you can push everything to the server:

```
git push
```

If you created a new online repository and cloned the empty repository, you have to use this command the first time to set up the `master` branch:

```
git push origin master
```

After that, `git push` will suffice.

1.8 Creating and Switching Branches

But you probably don't want to do that yet. You're probably not quite sure yet if your changes are OK and you would like to wait with pushing them to the `master` branch. Probably you would like your colleagues to have a look at your changes first.

That's where *branches* come into the picture.

To see what branches you already have, type:

```
git branch
```

You'll probably get something like this:

```
* master
```

This means you have only one branch which is called `master`. This is typically the default branch and most repositories have it but it is just a branch as any other branch. The asterisk marks the currently active branch. You should also see this in your prompt if you did what I suggested in *Command Line Prompt*.

You can switch between branches with `git checkout`. But you don't have another branch to switch to ... so let's create one:

```
git checkout -b fix-typo
```

The option `-b` combines creating a branch with directly switching to the newly created branch.

Your local files didn't actually change by switching to the new branch because for now, the branches `fix-typo` and `master` are just two different names for the same thing. But if you now start committing changes, these commits will end up in the `fix-typo` branch while the `master` branch will remain unchanged.

Let's check our branch-related situation:

```
git branch
```

Which produces something like this:

```
* fix-typo
  master
```

Now you can actually change something and then commit your changes:

```
git commit -a
```

Todo: more about branches?

1.9 Merging Branches

Todo: more information about merging and potential merge conflicts

Todo: `git mergetool` is really useful!

Setting up Vim + fugitive as `mergetool`:

```
git config --global mergetool.fugitive.cmd 'gvim -f -c "Gdiff" "$MERGED"'
git config --global merge.tool fugitive
```

On *macOS*, you can use *FileMerge* (you need to have *Xcode* installed):

```
git config --global merge.tool opendiff
```

Todo: more advertisement for Vim and fugitive!

1.10 More Aliases

Once you've worked some time with Git, you will realize that there are a few commands that you use very often. It's easy to create aliases that make you type less.

I, for example, often use `git rebase` and afterwards I want to ensure that a *fast forward* merge is done (instead of a separate *merge commit*). Therefore, I have to type `git merge mybranch --ff-only`, which is quite long and tedious to type. With the following alias, I can reduce this to `git ff mybranch`:

```
git config --global alias.ff "merge --ff-only"
```

Sometimes, after a `git fetch` or `git remote update`, I want to fast-forward my local branch to its newly fetched remote branch. With my previous alias, I could do `git ff origin/mybranch`. This is still too long, and Git should be able to automatically figure out which is the correct remote branch. With the following alias, the command is reduced to `git ffu`:

```
git config --global alias.ffu "merge --ff-only @{upstream}"
```

I seldom use `git pull`, because if there are new commits on both upstream and locally, a *merge commit* will be created automatically. And I don't like that. To avoid a merge commit and to only actually merge if a merge commit can be avoided (i.e. if a *fast forward* merge is possible), we can again use the option `--ff-only`. With the following alias, I only have to type `git pff`:

```
git config --global alias.pff "pull --ff-only --prune"
```

The additional `--prune` option is very handy because it removes the remote branches which were deleted on the server (which is not done automatically).

1.11 Ignoring Local Files

Todo: `.gitignore`, global ignore file with `core.excludesfile`, reference to https://github.com/github/gitignore

1.12 Attributes

You can set per-file (or per-path) attributes if you create a file named `.gitattributes`, for example like this:

```
*.bib diff=bibtex
*.cpp diff=cpp
*.h diff=cpp
*.htm diff=html
*.html diff=html
*.java diff=java
*.php diff=php
*.py diff=python
*.rb diff=ruby
*.tex diff=tex
*.pbxproj binary
```

1.13 GUIs for Git

There are many GUIs for Git to choose from; I personally like *gitg* (available as Debian package with the same name) most but there are many more available (see http://git-scm.com/downloads/guis).

1.14 Getting Help

To get help just use:

```
git help
```

You'll get something like this:

```
The most commonly used git commands are:
   add        Add file contents to the index
   bisect     Find by binary search the change that introduced a bug
   branch     List, create, or delete branches
   checkout   Checkout a branch or paths to the working tree
   clone      Clone a repository into a new directory
   commit     Record changes to the repository
   diff       Show changes between commits, commit and working tree, etc
   fetch      Download objects and refs from another repository
   grep       Print lines matching a pattern
   init       Create an empty Git repository or reinitialize an existing one
   log        Show commit logs
   merge      Join two or more development histories together
   mv         Move or rename a file, a directory, or a symlink
   pull       Fetch from and integrate with another repository or a local branch
   push       Update remote refs along with associated objects
   rebase     Forward-port local commits to the updated upstream head
   reset      Reset current HEAD to the specified state
   rm         Remove files from the working tree and from the index
   show       Show various types of objects
   status     Show the working tree status
   tag        Create, list, delete or verify a tag object signed with GPG

See 'git help <command>' or 'git help <concept>' to read about a specific
subcommand or concept.
```

1.15 Git and Subversion (SVN)

See http://git-scm.com/book/en/Git-and-Other-Systems-Git-and-Subversion

1.16 Public Git Hosting Sites

There are several free Git hosting services available, for an overview visit https://git.wiki.kernel.org/index.php/GitHosting

1.17 More Documentation/Links

- The Pro Git Book (CC license): http://book.git-scm.com/
- Understanding Git Conceptually: http://www.eecs.harvard.edu/~cduan/technical/git/
- Git Quick Reference: http://jonas.nitro.dk/git/quick-reference.html
- Git Immersion: http://gitimmersion.com/

- ...

There are many different strategies and methodologies how to use Git, just have a look with your favorite search engine or try this:

- http://nvie.com/posts/a-successful-git-branching-model/
- http://betterexplained.com/articles/aha-moments-when-learning-git/
- http://sethrobertson.github.io/GitBestPractices/
- ...

There are also some nice videos:

- **beginner**
  - http://www.youtube.com/watch?v=4XpnKHJAok8
  - http://www.youtube.com/watch?v=ZDR433b0HJY
  - http://www.youtube.com/watch?v=GYnOwPl8yCE
- **intermediate**
  - http://www.youtube.com/watch?v=Z2ZL14WWEJI
- **advanced**
  - http://blip.tv/scott-chacon/git-tips-4232122

1.18 TODO

I probably should write about these, too:

- pushing and pulling branches
- adding remotes
- merging
- rebasing
- interactive rebasing
- cherry-picking
- `git stash`

CHAPTER 2

Jupyter Notebooks in a Git Repository

It is a very nice feature of Jupyter notebooks that cell outputs (e.g. images, plots, tables with data) can be stored within notebooks. This makes it very easy to share notebooks with other people, who can open the notebooks and can immediately see the results, without having to execute the notebook (which might have some complicated library or data dependencies, or it might simply take a long time to run).

However, those cell outputs can be very annoying when using a version control system like e.g. Git. Whenever a change is made to a code cell, most likely the cell's output will also change. The problem is that both changes will be shown in a "diff" view, but the (often much larger) changes in outputs will distract from the much more interesting changes in the code. This can make it very tedious to work on a notebook with multiple people.

To avoid this problem, it is recommended to strip all outputs from a notebook before committing it to Git.

Todo: clean/smudge filters (https://nbsphinx.readthedocs.io/en/latest/usage.html#Using-Notebooks-with-Git)

There is a catch, though. If you don't commit the cell outputs, other people looking at your repository don't see the outputs (unless they execute the notebooks on their own). But don't worry, the next two sections will show two ways to work with "clean" notebooks but still to be able to share the cell outputs publicly.

2.1 Executing Notebooks on a Server

Todo: https://nbsphinx.readthedocs.io/ and others

Todo: advantages, disadvantages

2.2 Executing Notebooks in a Separate Branch

In this scenario, you and your collaborators mainly work on the `dev` branch (never committing cell outputs), and the `master` branch only contains a single additional commit in which all notebooks are executed.

If you want to use pull requests for collaboration, those should always be based upon the `dev` branch.

Todo: advantages, disadvantages

2.2.1 Getting Started from Scratch

This assumes that you have no Jupyter notebooks in your repository yet or all your notebooks are "clean" (i.e. stored without outputs).

1. Make sure there are no un-committed (and un-pushed) local changes
2. Create a new branch called `dev` (starting at the `master` branch) and switch to the new branch:

```
git checkout -b dev master
```

3. Push the new `dev` branch to the server:

```
git push --set-upstream origin dev
```

That's it!

Now you can continue with the section *Making a Change*.

2.2.2 Getting Started with Pre-executed Notebooks

Don't worry if you have previously committed notebooks without stripping their outputs. There is still a way of getting rid of them retroactively by re-writing your Git history.

1. Make sure there are no un-committed (and un-pushed) local changes
2. Create a new branch called `dev` (starting at the `master` branch) and switch to the new branch:

```
git checkout -b dev master
```

3. Do the steps listed in the section *Cleaning a Whole Repository*
4. Push the changes from the `dev` branch to the server:

```
git push --set-upstream origin dev
```

5. Switch back to the `master` branch and make a backup branch:

```
git checkout master
git branch backup
```

Note: If you think you might need it later (or if you are somewhat paranoid), you can also push the new `backup` branch to the server.

6. Reset the `master` branch to point to the same commit as `dev`:

```
git reset dev --hard
```

Warning: With this step you throw away all your old commits! But you can still use the `backup` branch to get them back.

7. Get the executed version of all notebooks (don't forget the dot!):

```
git checkout backup .
```

8. Create a new commit with a commit message like "Execute notebooks":

```
git commit -m "Execute notebooks"
```

9. If you are satisfied with the result, you can push your changes to the server, but note that you have to use `--force`, because you changed the Git history:

```
git push --force
```

Warning: At this point, you are deleting all your old commits from the server! If you want to keep them, you should also push the `backup` branch.

2.2.3 Making a Change

1. Switch to the `dev` branch:

```
git checkout dev
```

2. Work on your notebooks
3. Create one or more commits with new notebooks or changes to existing ones
4. Push the `dev` branch to the server:

```
git push
```

5. Switch to the `master` branch and re-base it onto `dev`:

```
git checkout master
git rebase -X ours dev
```

Note: The parameter `-X ours` selects a merging strategy where the changes to `dev` are preferred over the changes to `master`.

Special care has to be taken before re-basing when notebooks are removed:

```
git checkout master
git rm the-deleted-notebook.ipynb the-other-deleted-notebook.ipynb
git commit --amend
git rebase -X ours dev
```

6. Manually (re-)run the changed (and any new) notebooks.

   You can execute the notebooks in the Jupyter application, or you can execute them with `nbconvert`:

   ```
   python3 -m nbconvert --execute --inplace my-notebook.ipynb my-other-notebook.ipynb
   ```

   If you have many notebooks, it might be hard to keep in mind which ones you have changed. To get list of changed notebooks (but also other changed files), you can use this command:

   ```
   git diff --name-only dev $(git merge-base dev origin/master)
   ```

7. When all changed notebooks have been executed, you can update the "Execute notebooks" commit:

   ```
   git commit -a --amend
   ```

8. In the end, the changes to `master` have to be force-pushed:

   ```
   git push --force
   ```

   Note: *Normally*, you should *never* use `git push --force` on the `master` branch. However, this is a special case where it's OK, because all actual work will be done on the `dev` branch. This means that you should *never* use `git push --force` on the `dev` branch!

2.3 Executing All Notebooks

To execute all notebooks (whether they have outputs in them or not), you can use:

```
python3 -m nbconvert --execute --inplace *.ipynb **/*.ipynb
```

To disable the timeout, add `--ExecutePreprocessor.timeout=-1` to the command. This should actually be the default, but it's not, see https://github.com/jupyter/nbconvert/issues/791.

Please note the two *globbing* patterns used here. The second pattern (`**/*.ipynb`) is collecting all the notebooks recursively, but it doesn't include the files in the current directory. That's what the first pattern (`*.ipynb`) is used for. If you don't have notebooks in the main directory, you should omit this pattern.

In a future release of `nbconvert` the second pattern might become superfluous.

2.4 Cleaning All Notebooks

Removing outputs from all notebooks should work with this command:

```
python3 -m nbconvert --clear-output *.ipynb **/*.ipynb
```

... except that `--clear-output` is currently broken, see https://github.com/jupyter/nbconvert/issues/822.

It should work with the slightly more verbose:

```
python3 -m nbconvert --ClearOutputPreprocessor.enabled=True --inplace *.ipynb **/*.
↪ipynb
```

2.5 Cleaning a Whole Repository

Make sure you don't have any local changes and no un-committed files!

You might want to create a new branch (and switch to it) before doing this!

Cleaning the whole Git history of the current branch:

```
git filter-branch --tree-filter "python3 -m nbconvert --ClearOutputPreprocessor.
↪enabled=True --inplace *.ipynb **/*.ipynb"
```

If there are some commits without Jupyter notebook in them, you might want to extend the command a bit (to ignore any errors):

```
git filter-branch --tree-filter "python3 -m nbconvert --ClearOutputPreprocessor.
↪enabled=True --inplace *.ipynb **/*.ipynb || true"
```

Depending on the size of your repository and the number of commits, this might take a while ...

CHAPTER 3

Importing Local Python Modules from Jupyter Notebooks

If you re-use local modules a lot, you should consider turning them into proper Python packages which can be installed with Python's package manager `pip`.

The following sections are created from Jupyter notebooks which show multiple ways to import local Python modules, even if they are located in sub-directories.

The file `module-subdirectory/mymodule.py` is used as a dummy example module.

If you know other (reasonable) methods to use local modules, please create an issue or a pull request!

3.1 Use a Symbolic Link

The file mymodule.py in the current directory is a symbolic link to ../module-subdirectory/mymodule.py.

This may not work on all operating systems!

[1]:
```
import mymodule
```

[2]:
```
mymodule.hello()
```

```
Hello, world!
```

3.2 Manipulating `sys.path` in the Notebook

You can add your module's sub-directory to Pythons path like this:

[1]:
```
import os
import sys
sys.path.insert(0, os.path.abspath('../module-subdirectory'))
```

... then you can simply import it:

[2]:
```
import mymodule
```

[3]:
```
mymodule.hello()
```

```
Hello, world!
```

Of course it is quite ugly to show this path manipulation in each notebook. You can also move this to a helper module, see *this notebook*.

3.3 Manipulating `sys.path` in a Helper Module in the Current Directory

Instead of manipulating the path in each notebook (like shown in *this notebook*), we can create a helper module in the current directory which does the path manipulations. For the current notebook, there is a helper module in the same directory: mymodule.py. This helper module does some obscure manipulations that make it look like it's the module from ../module-subdirectory/mymodule.py.

[1]:
```
import mymodule
```

[2]:
```
mymodule.hello()
```

```
Hello, world!
```

Admittedly, the code looks quite scary:

[3]:
```
%cat mymodule.py
```

```
# Import a module with the same name from a different directory.

import importlib
import os
import sys

sys.path.insert(0, os.path.abspath('../module-subdirectory'))

if not hasattr(importlib, 'reload'):
    importlib.reload = reload  # for Python 2 compatibility

# Temporarily hijack __file__ to avoid adding names at module scope;
# __file__ will be overwritten again during the reload() call.
__file__ = {'sys': sys, 'importlib': importlib}

del importlib
del os
del sys

__file__['importlib'].reload(__file__['sys'].modules[__name__])
```

CHAPTER 4

Audio in Python

Look over there!

nbviewer: https://nbviewer.jupyter.org/github/mgeier/python-audio/blob/master/index.ipynb

GitHub: https://github.com/mgeier/python-audio/

Binder: https://mybinder.org/v2/gh/mgeier/python-audio/dev?filepath=index.ipynb

CHAPTER 5

Reproducible Research

Note: This is, much like research itself, and the art of eating spaghetti without soiling yourself, work-in-progress.

This page is not as general as it should be. It is biased towards audio signal processing, audio engineering, spatial audio reproduction and auditory perception. However, many of the ideas presented here can be applied more widely.

Other collections of similar information:

- https://github.com/INRIA/awesome-open-science-software
- https://livingthing.danmackinlay.name/open_notebook_science.html

5.1 Definitions

5.1.1 Openness

Todo: The open definition

http://opendefinition.org/

Definition by Wikipedia:

> **Open science** is the movement to make scientific research, data and dissemination accessible to all levels of an inquiring society, amateur or professional. It encompasses practices such as publishing open research, campaigning for open access, encouraging scientists to practice open notebook science, and generally making it easier to publish and communicate scientific knowledge. [...] In modern times there is debate about the extent to which scientific information should be shared. The conflict is between the desire of scientists to have access to shared resources versus the desire of individual entities to profit when other entities partake of their resources.

Definition by Wikipedia:

> **Open research** is research conducted in the spirit of free and open source software. Much like open source schemes that are built around a source code that is made public, the central theme of open research is to make clear accounts of the methodology freely available via the internet, along with any data or results extracted or derived from them. This permits a massively distributed collaboration, and one in which anyone may participate at any level of the project.
>
> Especially if the research is scientific in nature, it is frequently referred to as *open science*. Open research can also include social sciences, the humanities, mathematics, engineering and medicine.

Definition by Wikipedia:

> **Open data** is the idea that some data should be freely available to everyone to use and republish as they wish, without restrictions from copyright, patents or other mechanisms of control. The goals of the open data movement are similar to those of other "open" movements such as open source, open hardware, open content, and open access.

Definition by Wikipedia:

> **Open science data** is a type of open data focused on publishing observations and results of scientific activities available for anyone to analyze and reuse.

Definition by Wikipedia:

> **Open notebook science** is the practice of making the entire primary record of a research project publicly available online as it is recorded. This involves placing the personal, or laboratory, notebook of the researcher online along with all raw and processed data, and any associated material, as this material is generated. The approach may be summed up by the slogan 'no insider information'. It is the logical extreme of transparent approaches to research and explicitly includes the making available of failed, less significant, and otherwise unpublished experiments; so called 'dark data'.

Definition by Wikipedia:

> **Open access** (**OA**) refers to online research outputs that are free of all restrictions on access (e.g. access tolls) and free of many restrictions on use (e.g. certain copyright and license restrictions). Open access can be applied to all forms of published research output, including peer-reviewed and non peer-reviewed academic journal articles, conference papers, theses, book chapters, and monographs.
>
> Two degrees of open access can be distinguished: gratis open access, which is online access free of charge, and libre open access, which is online access free of charge plus various additional usage rights.

Todo: Reproducible Research vs. Non-Reproducible Research?

Todo: reproducible vs. easily reproducible

Todo: online material as supplement to traditional publications

Todo: http://en.wikipedia.org/wiki/Reproducibility

Todo: http://en.wikipedia.org/wiki/Open_research

Vandewalle et al. distinguish six degrees of reproducibility:

5. The results can be easily reproduced by an independent researcher with at most 15 min of user effort, requiring only standard, freely available tools (C compiler, etc.).
4. The results can be easily reproduced by an independent researcher with at most 15 minutes of user effort, requiring some proprietary source packages (MATLAB, etc.).
3. The results can be reproduced by an independent researcher, requiring considerable effort.
2. The results could be reproduced by an independent researcher, requiring extreme effort.
1. The results cannot seem to be reproduced by an independent researcher.
0. The results cannot be reproduced by an independent researcher.

While I don't agree with all details (especially the over-concrete time specifications and the overly vague effort metrics), I like the general idea.

5.1.2 Replicability vs. Reproducibility

Great overview: Language Log: Replicability vs. reproducibility — or is it the other way around?

Wikipedia thinks it's the same:

> **Reproducibility** is the ability of an entire experiment or study to be duplicated, either by the same researcher or by someone else working independently. Reproducing an experiment is called **replicating** it. Reproducibility is one of the main principles of the scientific method.

Chris Drummond claims they are different:

> Reproducibility requires changes; replicability avoids them. Although reproducibility is desirable, I contend that the impoverished version, replicability, is one not worth having.

Roger D. Peng also claims that they are different, but uses slightly different definitions:

> The replication of scientific findings using independent investigators, methods, data, equipment, and protocols has long been, and will continue to be, the standard by which scientific claims are evaluated. However, in many fields of study there are examples of scientific investigations that cannot be fully replicated because of a lack of time or resources. In such a situation, there is a need for a minimum standard that can fill the void between full replication and nothing. One candidate for this minimum standard is "reproducible research", which requires that data sets and computer code be made available to others for verifying published results and conducting alternative analyses.

Victoria Stodden defines them slightly differently (and throws in a third concept – "repeatability"):

> We can reserve the term "replicability" for the regeneration of published results from author-provided code and data. [...] Reproducibility is a more general term, implying both replication and the regeneration of findings with at least some independence from the code and/or data associated with the original publication. Both refer to the analysis that occurs after publication. A third term, "repeatability," is sometimes used in place of reproducibility, but this is more typically used as a term of art referring to the sensitivity of results when underlying measurements are retaken.

5.2 Guidelines

Here are few guidelines which may (or may not) help to make your work more reproducible:

make everything public (and each step of it) At some point, every aspect of your work should be publicly accessible. And not only the parts which (you think) are most interesting ... every single bit and every single step. This way it will be easiest for others to reproduce your work.

You may not want to publish everything from the very beginning, which leads to the next point ...

release early This is borrowed from the Open Source movement, but it's also applicable here. Even if you feel it's not finished yet, just make it public! Because if you wait too long, you'll probably never release it ...

If you release early, you also give others the chance to comment on your work and to suggest improvements before you think it's "finished" (which may never happen).

make stuff public by default In case of doubt, make it public! Keep things only for yourself if there is a good reason. And even if there is a reason now, you should think about making it public later (e.g. after publication of a related paper).

think about others Don't just think about how great your results are, also think about how you can make it as easy as possible for others to reproduce them.

use tools that others can use, too If you have a choice, prefer tools that are available to other researchers, too.

Of course, often expensive equipment is needed in research, and sometimes only few laboratories have even the theoretical possibility to reproduce your experiments. We have to live with that.

When it comes to software, there is often an alternative to expensive programs, sometimes the free ones are even better. Try to choose software that is accessible to most people, and try to use software that runs on different operating systems.

use open source software TODO: content

specify a license If provide something to the public and don't specify a license, said public may have a hard time using the thing legally. With everything you publish, you should also tell people what they may and may not do with it.

But remember: the more restrictions you impose, the more freedom you take away from people who want to use your work. You can waive all your rights (at least with regard to copyright law), you can request attribution, you can demand that derived works must be published under the same conditions as the original work (a.k.a. *share-alike*), you can forbid commercial use, ...

Try these links to help you choose an appropriate license:

- http://creativecommons.org/choose/
- http://three.org/openart/license_chooser/

For more details, have a look there: https://tldrlegal.com/.

Licensing your research, webinar with Brandon Butler: https://osf.io/6uupa/

bring research and teaching closer together Every research starts from some existing knowledge.

TODO: more arguments

Today's students are tomorrow's researchers.

5.3 What Should be Reproducible?

Short answer: everything!

But let's be a bit more verbose. Ideally, the whole research process should be reproducible. The following list shows things that can (and should!) be made reproducible. There are also some tools mentioned that may help, see below for links to more software and libraries.

All this is of course very much dependent on the research area. Some points may apply to your area, others won't.

collecting ideas Ideas are the core of any research activity. They are also one of the main resources needed by researchers (besides funding). Understandably, many researcher are reluctant to make their ideas public before they reap their fruits themselves.

But at a later time, e.g. after a publication, there may not be a reason anymore to keep the ideas a secret. Also, some researchers (mostly the good ones) have more ideas than they could possibly work on. In this case they should make their "vacant" ideas public for other researchers to work on.

In the era of the world-wide-web there are countless possibilities to share your ideas, no need to give any pointers here, you'll find something.

symbolic derivations In many areas, deriving equations is the daily drill of a researcher. In traditional publications, however, only a limited amount of space can be used for equations, so typically only a few steps of the derivation are shown or even only the final resulting equation.

This can make it very time-consuming for other researchers to reproduce and build on your results. Ideally, for every published equation the complete and detailed derivation should also be publicly available.

You can create nice equations using LaTeX documents, but also some blogging systems support entering math equations. IPython also supports nice-looking equations (using MathJax).

TODO: CASs

numeric calculations, simulations, visualizations, plots TODO: NumPy, SciPy, matplotlib, Mayavi, ...

cluster computing TODO: IPython

measurements TODO: settings, logs, software, pre-/post-processing scripts

experimental apparatus

TODO: detailed description, drawings, photos, detailed list of devices ant the used configuration, ...

TODO: software (ideally open source), scripts, configuration files, data files, ...

statistical evaluation TODO: raw data, all scripts

TODO: pandas, R

5.4 Criticism

Three points from https://en.wikipedia.org/wiki/Open_notebook_science#Drawbacks:

1. data theft
2. not patentable once published
3. data deluge

5.5 Software

The following is a completely subjective selection of open-source software. This is not at all exhaustive, there are a lot of alternatives, both commercial and non-commercial.

5.5.1 Python

Note: Why Python?

The chief reason is that it's just a beautiful programming language. And it's versatile ... so the *two* reasons are its beauty and versatility ... and its extensive standard library, therefore the *three* reasons to use Python are its beauty, versatility and extensive standard library ... and a sheer unimaginably humongous number of third-party libraries and extensions.

Let's just say *amongst* the reasons to choose Python are such diverse elements as beauty, versatility, extremely useful standard library and tons of third-party stuff.

For more information, watch this: http://youtu.be/vt0Y39eMvpI

Scientific Python (SciPy) http://scipy.org/

> This is a collection of many software projects: NumPy, SciPy, matplotlib, IPython, SymPy, pandas, Mayavi, PyTables, and many more ...

See also my introduction to Python, NumPy, IPython, ...

5.5.2 LaTeX

...

Todo: TikZ, gnuplot, beamer

5.5.3 Git

See *Getting Started with Git*.

5.5.4 More Software

There's always more ...

R http://www.r-project.org/

Julia http://julialang.org/

Sage http://sagemath.org/

5.6 Publication Tools

IPython http://ipython.org/

IJulia https://github.com/JuliaLang/IJulia.jl (example notebook)

VisTrails http://www.vistrails.org/index.php/Main_Page

Sweave http://en.wikipedia.org/wiki/Sweave

knitr http://yihui.name/knitr/

Pweave http://mpastell.com/pweave/

ActivePapers

> **http://dirac.cnrs-orleans.fr/plone/software/activepapers/**

- active_papers (JVM): https://bitbucket.org/khinsen/active_papers
- active_papers_py (Python): https://bitbucket.org/khinsen/active_papers_py/wiki/Home

5.7 Online Services

IPython Notebook Viewer http://nbviewer.ipython.org/

Binder (Turn a GitHub repo into a collection of interactive notebooks) http://mybinder.org/

Github http://github.com/

Bitbucket (free unlimited accounts for academic users) http://bitbucket.org/

figshare http://figshare.com/, connecting Github and figshare

zenodo http://zenodo.org/

ORCID http://orcid.org/

crossref http://crossref.org/

DataCite http://www.datacite.org/

my experiment http://www.myexperiment.org/

re3data (Registry of Research Data Repositories) http://www.re3data.org/

RADAR - Research Data Repository http://www.radar-projekt.org/display/RE/Home

Open Science Framework https://osf.io/

DataUp http://dataup.cdlib.org/

Authorea https://authorea.com/

PubPeer (post publication peer review) https://pubpeer.com/

PubMed Commons (post publication peer review) https://www.ncbi.nlm.nih.gov/pubmedcommons/

CKAN (Open Source data portal platform) https://ckan.org/

5.8 Journals

F1000Research (life sciences) http://f1000research.com/

Scientific Data - nature.com (launching in May 2014) http://www.nature.com/scientificdata/

DRYAD http://datadryad.org/

The ReScience Journal http://rescience.github.io/

5.9 Publications

Patrick Vandewalle, Jelena Kovačević, Martin Vetterli, Reproducible Research in Signal Processing, IEEE Signal Processing Magazine Volume 26, Issue 3, 2009.

Robert Gentleman, Duncan Temple Lang, Statistical Analyses and Reproducible Research, Journal of Computational and Graphical Statistics Volume 16, Issue 1, 2007.

Bruce G. Charlton, Peer usage versus peer review, BMJ Volume 335, Issue 7617, 2007.

Arturo Casadevall, Ferric C. Fang, Reproducible Science, Infection and Immunity Volume 78, Issue 12, 2010.

Jonathan B. Buckheit, David L. Donoho, WaveLab and Reproducible Research, in Wavelets and Statistics, Springer, 1995.

Darrel C. Ince, Leslie Hatton, John Graham-Cumming, The Case for Open Computer Programs, Nature Volume 482, 2012.

Nature special Challenges in Irreproducible Research, 2010-2013.

Fernando Pérez, Brian E. Granger, John D. Hunter, Python: An Ecosystem for Scientific Computing, Computing in Science Engineering, Volume 13, Issue 2, 2011.

Peter Suber, Open Access, MIT Press, 2012.

Peter Suber, Gratis and libre open access, SPARC Open Access Newsletter, issue #124, 2008.

John P. A. Ioannidis, Why Most Published Research Findings Are False, PLoS Med 2(8): e124. doi:10.1371/journal.pmed.0020124, 2005.

Detailed comment to the above: http://matthew-brett.github.io/teaching/ioannidis_2005.html

Chris Drummond, Replicability is not Reproducibility: Nor is it Good Science, Proc. of the Evaluation Methods for Machine Learning Workshop at the 26th ICML, 2009.

Ian P. Gent, The Recomputation Manifesto, Unpublished position paper, Version 1.9479, 2013.

Michael Woelfle, Piero Olliaro, Matthew H. Todd, Open science is a research accelerator, Nature Chemistry, Volume 3, Issue 10, 2011.

Radovan Vrana, Open science, open access and open educational resources: Challenges and opportunities, International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), 2015.

Yale Law School Roundtable on Data and Code Sharing, Reproducible Research: Addressing the Need for Data and Code Sharing in Computational Science, Computing in Science & Engineering, Volume 12, Issue 5, 2010.

Toronto International Data Release Workshop Authors, Prepublication Data Sharing, Nature 461, no. 7261, 2009.

Rinze Benedictus, Frank Miedema, and Mark W. J. Ferguson, Fewer Numbers, Better Science, Nature News, Volume 538, Issue 7626, 2016.

J. Wilsdon et al., The Metric Tide: Report of the Independent Review of the Role of Metrics in Research Assessment and Management, 2015.

Barak A. Cohen, Point of View: How should novelty be valued in science?, 2017.

D. Cicchetti, The reliability of peer review for manuscript and grant submissions: A cross-disciplinary investigation, 1991.

J. Bollen et al., From funding agencies to scientific agency, 2014.

J. Bollen et al., An efficient system to fund science: from proposal review to peer-to-peer distributions, 2017.

B. Alberts et al., Self-Correction in Science at Work, Science Vol. 348, Issue 6242, pp. 1420-1422, 2015

B. A. Nosek et al., Promoting an Open Research Culture, Science Vol. 348, Issue 6242, pp. 1422-1425, 2015

5.10 Links

Coursera course about *Reproducible Research* (starting on May 5th 2014) https://www.coursera.org/course/repdata

results may vary (slides for keynote at ISMB/ECCB 2013) http://www.slideshare.net/carolegoble/ismb2013-keynotecleangoble

Reproducibility in Computational Science (slides) http://www.stanford.edu/~vcs/talks/UMN-Oct102013-STODDEN.pdf

The Role of Data Repositories in Reproducible Research: http://isps.yale.edu/news/blog/2013/07/the-role-of-data-repositories-in-reproducible-research

#solo13lego: Research Roles Through Lego http://sophiekershaw.wordpress.com/2013/11/14/research-roles-through-lego/

Reproducibility: An important altmetric http://altmetrics.org/altmetrics12/iorns/

The Truth Wears Off: An odd twist in the scientific method http://www.newyorker.com/reporting/2010/12/13/101213fa_fact_lehrer

Report reveals missteps in Duke cancer trial review http://blogs.nature.com/news/2011/01/report_reveals_missteps_in_ini.html

Reproducible Research in Signal/Image Processing http://reproducibleresearch.net/

European Commission: ***Towards better access to scientific information*** http://www.eesc.europa.eu/?i=portal.en.int-opinions.24976 (PDF)

Preserving Research: The top online archives for storing your unpublished findings http://www.the-scientist.com/?articles.view/articleNo/36695/title/Preserving-Research/

Post-Publication Peer Review Mainstreamed http://www.the-scientist.com/?articles.view/articleNo/37969/title/Post-Publication-Peer-Review-Mainstreamed/

Offene Wissenschaft (de) http://www.offene-wissenschaft.de/

mozilla Science Lab http://mozillascience.org/

Panton Principles http://pantonprinciples.org/

The Open Definition http://opendefinition.org/

Guide to Open Data Licensing http://opendefinition.org/guide/data/

CC0 http://creativecommons.org/publicdomain/zero/1.0/

Joint Declaration of Data Citation Principles https://www.force11.org/node/4769

Madagascar http://reproducibility.org/

Reproducibility Initiative http://reproducibilityinitiative.org/

The Need for Openness in Data Journalism http://nbviewer.ipython.org/github/brianckeegan/Bechdel/blob/master/Bechdel_test.ipynb

Guidelines for Open Educational Resources (OER) in Higher Education http://www.col.org/PublicationDocuments/Guidelines_OER_HE.pdf

10 Simple Rules for the Care and Feeding of Scientific Data https://authorea.com/users/3/articles/3410/_show_article

Scientific Python Lectures: https://github.com/jrjohansson/scientific-python-lectures

Research Objects http://en.wikipedia.org/wiki/Research_Objects

An efficient workflow for reproducible science (SciPy 2013) https://youtu.be/Y-XFNg0QS14

Open Glossary http://blogs.egu.eu/network/palaeoblog/files/2015/02/OpenGlossary1.pdf

Open Access: Berlin Declaration http://openaccess.mpg.de/Berlin-Declaration, Wikipedia article

recomputation.org http://recomputation.org/

Reproducibility in Code and Science http://justingosses.com/reproducibility/

The 7 biggest problems facing science, according to 270 scientists http://www.vox.com/2016/7/14/12016710/science-challeges-research-funding-peer-review-process

Journal of Articles in Support of the Null Hypothesis http://www.jasnh.com/

The Transparency and Openness Promotion Guidelines https://cos.io/top/

épisciences http://episciences.org/

The open archive HAL https://hal.archives-ouvertes.fr/

arXiv.org https://arxiv.org/

Directory of Open Access Journals (DOAJ) https://doaj.org/

Amsterdam Call for Action on Open Science https://english.eu2016.nl/documents/reports/2016/04/04/amsterdam-call-for-action-on-open-science

Reproducibility and reliability of biomedical research https://acmedsci.ac.uk/policy/policy-projects/reproducibility-and-reliability-of-biomedical-research/

Rigor and Reproducibility (NIH guidelines) https://grants.nih.gov/reproducibility/index.htm

Analysis of meta-analyses identifies where sciences' real problems lie https://arstechnica.com/science/2017/03/bias-in-science-small-samples-isolated-scientists-and-dodgy-individuals/

Vienna Principles http://viennaprinciples.org/

sciencecodemanifesto.org https://web.archive.org/web/20160218093215/http://sciencecodemanifesto.org/

Reproducible Research Tools https://www.stat.wisc.edu/reproducible

Peer Reviewers' Openness Initiative https://opennessinitiative.org/

Initiative for Open Citations https://i4oc.org/

Workshop: Reproducible Research using Jupyter Notebooks https://reproducible-science-curriculum.github.io/rr-jupyter-workshop/

ACM Artifact Review and Badging https://www.acm.org/publications/policies/artifact-review-badging

Science is "show me," not "trust me" http://www.bitss.org/2015/12/31/science-is-show-me-not-trust-me/

An unhealthy obsession with p-values is ruining science https://www.vox.com/2016/3/15/11225162/p-value-simple-definition-hacking

The Irreproducibility Crisis of Modern Science: Causes, Consequences, and the Road to Reform https://www.nas.org/projects/irreproducibility_report

https://www.nas.org/images/documents/NAS_irreproducibilityReport.pdf

Why I've lost faith in p values https://lucklab.ucdavis.edu/blog/2018/4/19/why-i-lost-faith-in-p-values

Budapest Open Access Initiative https://www.budapestopenaccessinitiative.org/

FAIR Principles (Findable, Accessible, Interoperable, Re-usable) https://www.nature.com/articles/sdata201618

https://www.go-fair.org/fair-principles/

https://www.force11.org/group/fairgroup/fairprinciples

https://www.force11.org/fairprinciples

CHAPTER 6

Open Education

Note: This is, much like the fabric of space-time, and growing up, work-in-progress.

http://www.oeconsortium.org/

CHAPTER 7

Licensing

Disclaimer

IANAL

7.1 Links

Wikipedia article http://en.wikipedia.org/wiki/Creative_Commons_license

FAQ https://wiki.creativecommons.org/Frequently_Asked_Questions

Open Content - A Practical Guide to Using Creative Commons Licences https://meta.wikimedia.org/wiki/Open_Content_%2D_A_Practical_Guide_to_Using_Creative_Commons_Licences

Logo downloads http://creativecommons.org/about/downloads

CC0 http://creativecommons.org/publicdomain/zero/1.0/

chooser: https://creativecommons.org/choose/zero/

CC license chooser http://creativecommons.org/choose/

Free Cultural Works http://freedomdefined.org/Definition

https://creativecommons.org/freeworks/

The Open Definition http://opendefinition.org/

ShareAlike compatible licenses: https://creativecommons.org/compatiblelicenses/

NonCommercial Consequences, Risks, and side-effects of the license module Non-Commercial – NC http://blog.okfn.org/2013/01/08/consequences-risks-and-side-effects-of-the-license-module-non-commercial-use-only-2/

http://openglam.org/files/2013/01/iRights_CC-NC_Guide_English.pdf

CC and Data https://wiki.creativecommons.org/wiki/Data

Open Data Commons Open Data Commons Public Domain Dedication and License (PDDL) http://opendatacommons.org/licenses/pddl/

Open Data Commons Attribution License http://opendatacommons.org/licenses/by/

Open Data Commons Open Database License (ODbL): http://opendatacommons.org/licenses/odbl/

Guide to Open Data Licensing http://opendefinition.org/guide/data/

Publisher's Guide to Open Data Licensing http://theodi.org/guides/publishers-guide-open-data-licensing

Software Licenses in Plain English https://tldrlegal.com/

Everything an open source maintainer might need to know about open source licensing https://ben.balter.com/2017/11/28/everything-an-open-source-maintainer-might-need-to-know-about-open-source-licensing/

Copyright notices for open source projects https://ben.balter.com/2015/06/03/copyright-notices-for-websites-and-open-source-proje

CHAPTER 8

Re-usable Audio Data

Note: This is, much like social media and freeing Tibet, work-in-progress.

Todo: Some explanations, free as in speech, CC, ...

freesound http://www.freesound.org/

archive.org https://archive.org/details/audio

MUSOPEN https://musopen.org/

Free Music Archive http://freemusicarchive.org/

Jamendo Music http://www.jamendo.com/search/

CC mixter http://ccmixter.org/

CC search http://search.creativecommons.org/

Let's CC http://eng.letscc.net/

SoundCloud https://soundcloud.com/search/sounds

The Audio Commons Initiative http://audiocommons.org/

Public Domain Project http://publicdomainproject.org/

Europeana Music http://www.europeana.eu/portal/collections/music

Openair http://www.openairlib.net/anechoicdb

HSD, ISAVE, stimulusdatenbank https://isave.hs-duesseldorf.de/forschung/projekte/stimulusdatenbank/

Free Firearm/Medieval Weapons Sound Effects https://www.airbornesound.com/downloads/free-firearm-sound-effects-library-expanded-edition/

https://www.airbornesound.com/downloads/free-medieval-weapons-sound-effects-library-curated-edition/

8.1 Multi-Track Recordings

https://archive.org/search.php?query=&and{[}{]}=mediatype%3A%22audio%22&and{[}{]}=subject%3A%22multitrack%22

http://www.soundfieldsynthesis.org/other-resources/#multitracks

http://research.cs.aalto.fi/acoustics/virtual-acoustics/research/acoustic-measurement-and-analysis/85-anechoic-recordings.html

https://www.upf.edu/web/mtg/phenicx-anechoic (Aalto denoised)

8.2 Other Lists with Links

- https://www.diigo.com/list/kvitek/Public-Domain-Audio
- https://guides.library.harvard.edu/Finding_Images/finding_audio
- http://www.publicdomainsherpa.com/public-domain-recordings.html

CHAPTER 9

Creating a Python Module

Note: This is, much like quantum physics, and the universe, work-in-progress.

Here are some very subjective recommendations how to create and publish a Python module.

9.1 Coding Style

Have a look at PEP 8 and PEP 257 and probably also at the Google Python Style Guide.

9.2 Docstrings

Use the NumPy Docstring Standard.

9.3 Testing

py.test

9.4 Coverage

Todo: coverage tool, probably py.test extension

9.5 Online Documentation

Use Sphinx and https://readthedocs.org/.

Use sphinx.ext.autodoc and sphinx.ext.napoleon.

9.6 Installer

setuptools

https://python-packaging-user-guide.readthedocs.org/en/latest/index.html

https://caremad.io/blog/setup-vs-requirement/

https://manikos.github.io/a-tour-on-python-packaging

9.7 License

There are many possibilities, one of them is the MIT license:

```
Copyright (c) <year> <copyright holders>

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in
all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
THE SOFTWARE.
```

If your module consists of only one source file, this can be included in the top as a comment. This makes it easy for others to take just the one file and put it into their own project while still keeping the copyright notice.

If the module consists of several source files, this is not necessary. It's enough to put the copyright notice and the license text into a file *LICENSE* in the main directory (you should have this file in any case).

9.8 Further Reading

https://opensource.guide/

CHAPTER 10

Make Tutorial

Note: This is, much like the internet itself, and democracy, work-in-progress.

...

Todo: `make` runs other programs in the right order, handles intermediate files and generates files only if it's necessary (i.e. if the sources have changed).

Todo: Documentation for GNU make: http://www.gnu.org/software/make/

Note: Do not use `make` for running *LaTeX*!

It may be a reasonably usable tool for this and many people use and recommend it, e.g.

- http://robjhyndman.com/hyndsight/makefiles/
- http://kbroman.org/minimal_make/
- http://sidenote.hu/2012/02/04/makefile-for-large-latex-projects/

But there is a *much* superior tool for that, see *Using Latexmk*!

10.1 `make` Without Makefile

Whenever you encounter a directory with a file called `Makefile` in it, you know you can run `make` to automatically compile/generate/do whatever is specified in the `Makefile`.

But you don't even need such a file. You can also use `make` all by itself. To show an example for that, let's create a file called `hello.cpp` with the following content:

```
#include <iostream>

int main()
{
  std::cout << "Hello, world!" << std::endl;
}
```

Now we can simply run . . .

```
make hello
```

. . . and it will magically be compiled (that is, if you have `make` and a C++ compiler installed).

Note: You can ignore the details for now, but if you are curious, this is the exact command that will be called by `make`:

```
g++     hello.cpp   -o hello
```

That means the C++ compiler is compiling (and linking) the source file `hello.cpp` to produce an executable file called `hello` (which you can run with `./hello` by the way).

The command is automatically generated by this *implicit rule*:

```
%: %.cpp
        $(LINK.cpp) $^ $(LOADLIBES) $(LDLIBS) -o $@
```

This uses the following variable declarations:

```
LINK.cc = $(CXX) $(CXXFLAGS) $(CPPFLAGS) $(LDFLAGS) $(TARGET_ARCH)

LINK.cpp = $(LINK.cc)
```

The variable `CXX` is automatically set to the default C++ compiler (in our case `g++`). The other variables are by default undefined, but you can use them to specify any options you like.

You can find out about all the rules and variables that `make` considers by running `make --print-data-base`, but more about all that later.

10.2 A Simple Makefile

The command `make hello` was pretty simple, wasn't it? But we can make it even simpler, but for that we first have to create a `Makefile` to specify all the necessary details. The most important things to write in there, are so-called rules. Those consist of *targets*, *prerequisites* and a *recipe*, which can have several lines.

```
target: prerequisites
        recipe
        ...
```

A *target* starts at the beginning of a line and ends with a colon. On the right side of the colon, there are zero, one or several *prerequisites* (separated by spaces). *Prerequisites* are sometimes also called *dependencies*. On the following line, after a leading tab character, there can be a *recipe*, which will be used by `make` to update the *target*, if necessary. *Recipes* are sometimes also called *commands*.

But more about that later . . .

Let's create a file called `Makefile` with the following content:

```
hello:
```

That's it. Nothing more. Only one line. Only 6 characters.

This is a rule which contains only a *target*, i.e. the thing we want to get in the end. In our example that's an executable named `hello`. There are no *prerequisites* and no *recipe*. When `make` is called without specifying a target (like we did before with `make hello`), `make` chooses the first target it encounters in the `Makefile`, which is in our case – what a coincidence! – `hello`. With our brand-new `Makefile` in place, we can now simply call:

```
make
```

... and it will do the same thing as before.

Todo: this can happen:

```
make: `hello' is up to date.
```

From now on, we will make our `Makefile` more and more elaborate and add many features, but the call to `make` will mostly stay that simple. That's the beauty of it.

If you run `make`, it looks in the current directory for a file called `Makefile` or `makefile`. That's nice, so we don't have to specify explicitly which file should be used. If we *do* want to specify a different file, however, we can do so with the `-f` option, e.g.:

```
make -f MyOtherMakefile
```

10.3 Cleaning Up

Todo: `make clean`

Todo: clean-target, no prerequisites, one recipe

Todo: $(RM) vs rm

10.4 Adding Options

So using the built-in *implicit rules* is nice and easy, but what if we want to tweak some options?

Well, that's no problem. Let's say we want to add some options for the compiler, e.g. let's enable some warnings. It's always a good idea to enable compiler warnings! We do this by simply adding a variable to our `Makefile`:

```
CXXFLAGS = -Wall -Wextra -pedantic

hello:
```

When we now run `make`, we see that the actual command becomes:

```
g++ -Wall -Wextra -pedantic    hello.cpp   -o hello
```

So it worked, our command line option was put where it belongs.

There are a lot of pre-defined variables . . .

Todo: CXX, CXXFLAGS, CC, CFLAGS, CPPFLAGS, LDLIBS, LDFLAGS, . . .

Now we know how to add options to our `Makefile` permanently, but what if we want to try an option just once?

No problem, we can specify the directly at the command line as environment variables. For example, if we want to try if our program can also be compiled with a different compiler, we can do this:

```
CXX=clang++ make
```

...

```
clang++ -Wall -Wextra -pedantic    hello.cpp   -o hello
```

Let's try something else. Let's run the optimizer on our little program:

```
CXXFLAGS=-O2 make
```

...

```
g++ -Wall -Wextra -pedantic    hello.cpp   -o hello
```

But why? What happened to our `-O2` setting?

Unfortunately, it was overwritten in our `Makefile`. If you want to allow specifying options in environment variables, you have to take care not to overwrite these variables. The easiest way to do this, is to use the += operator in your `Makefile`:

```
CXXFLAGS += -Wall -Wextra

hello:
```

...

```
CXXFLAGS=-O2 make
```

Et voilá:

```
g++ -O2 -Wall -Wextra -pedantic    hello.cpp   -o hello
```

10.5 TODO

...

```
make -p

make -j8
```

- = vs := vs ?= vs +=
- subdirs
- *order-only* prerequisites
- .PHONY
- .DELETE_ON_ERROR
- .NOTPARALLEL
- .SECONDARY
- VPATH, vpath
- $(BLABLA:%old=%new)
- Target/pattern-specific variable values (incl `override`): http://www.gnu.org/software/make/manual/html_node/Target_002dspecific.html http://www.gnu.org/software/make/manual/html_node/Pattern_002dspecific.html

CHAPTER 11

Using Latexmk

If you use cross-references, you often have to run LaTeX more than once, if you use BibTeX for your bibliography or if you want to have a glossary you even need to run external programs in-between.

To avoid all this hassle, you should simply use Latexmk!

Latexmk is a Perl script which you just have to run once and it does everything else for you ... completely automagically.

And the nice thing is: you probably have it already installed on your computer, because it is part of MacTeX and MikTeX and it is bundled with many Linux Distributions.

11.1 Installation

On *Linux*:

- Perl should be already installed.
- You may have to install a package called `latexmk` or similar.

On *macOS* with MacTeX:

- It's probably already installed.
- If not, open "TeX Live Utility", search for "latexmk" and install it.
- If you prefer using the Terminal:

```
sudo tlmgr install latexmk
```

On *Windows* with MikTeX:

- You probably have to install Perl, e.g. from here: http://strawberryperl.com/.
- If it's not installed already, open the MikTeX Package Manager and install the `latexmk` package.

11.2 Running Latexmk

Latexmk is a command line application, see *below* for how to use it with batch files.

In the simplest case you just have to type

```
latexmk
```

This runs LaTeX on all `.tex` files in the current directory using the output format specified by the *configuration files*.

If you want to make sure to get a `.pdf` file as output, just mention it:

```
latexmk -pdf
```

If you want to use `latex` instead of `pdflatex` but still want a `.pdf` file in the end, use

```
latexmk -pdfps
```

If you want to compile only one specific `.tex` file in the current directory, just provide the file name:

```
latexmk myfile.tex
```

If you want to preview the resulting output file(s), just use

```
latexmk -pv
```

And now the *Killer Feature*: If you want Latexmk to continuously check all input files for changes and re-compile the whole thing if needed and always display the result, type

```
latexmk -pvc
```

Then, whenever you change something in *any* of your source files and save your changes, the preview is automagically updated. *But:* This doesn't work with all viewers, especially not with *Adobe Reader*. See the section about *configuration files* below for setting a suitable viewer application.

Of course, options can be combined, e.g.

```
latexmk -pdf -pv myfile.tex
```

11.3 Cleaning Up

After running LaTeX, the current directory is contaminated with a myriad of temporary files; you can get rid of them with

```
latexmk -c
```

This doesn't delete the final `.pdf`/`.ps`/`.dvi` files. If you want to delete those too, use

```
latexmk -C
```

11.4 Running Latexmk with Batch Files

If you are working on *Windows*, you may not be used to typing things at the command line. You may prefer clicking on things.

No problem, just create a file (in the same folder as your `.tex` files) with the extension `.bat` containing

```
latexmk
@pause
```

and double-click on it.

You can of course use all the abovementioned options, don't forget the especially useful ones `-c` and `-pvc`.

11.5 Configuration Files

On *Linux*, you can put your configurations into `$HOME/.latexmkrc`, which could contain something like this:

```
$dvi_previewer = 'start xdvi -watchfile 1.5';
$ps_previewer  = 'start gv --watch';
$pdf_previewer = 'start evince';
```

On *macOS*, you can also use `$HOME/.latexmkrc`, e.g. with this content:

```
$pdf_previewer = 'open -a Skim';
$pdflatex = 'pdflatex -synctex=1 -interaction=nonstopmode';
@generated_exts = (@generated_exts, 'synctex.gz');
```

This uses Skim as preview application, which can be set up to automatically update its display when the PDF file changes by selecting "Preferences" – "Sync" – "Check for file changes". While you are at it, you should also activates the *SyncTeX* feature by selecting you editor right below in the "PDF-TeX Sync support" section. With this selected and with `-synctex=1` in your LaTeX call, you can Shift--click in the preview window and jump directly to the corresponding source text in your editor!

On *Windows*, you can use the system-wide config file `C:\latexmk\LatexMk` (if the file doesn't exist yet, just create a new text file with this name). To choose a PDF viewer, use something like this:

```
$pdf_previewer = 'start gsview32';
```

You'll need *GSview* and *Ghostscript* for that, see http://pages.cs.wisc.edu/~ghost/gsview/.

Some previewers use different methods for updating the viewed PDF file. You can change that with `$pdf_update_method`, like in this example:

```
$pdf_update_method = 4;
$pdf_update_command = 'bla bla bla';
```

Full documentation is available in the manpage.

11.6 Local Configuration Files

You can also put a configuration file in the current directory for settings which only influence files in the current directory. Such a configuration file has to be named `latexmkrc` or `.latexmkrc` and may contain some of the following lines.

To specify if you want PDF or PS output, choose one of those:

```
$pdf_mode = 1;         # tex -> pdf
$pdf_mode = 2;         # tex -> ps -> pdf
$postscript_mode = 1; # tex -> ps
```

If you have your work split up into several parts, you have to specify the main file like this:

```
@default_files = ('main.tex');
```

Or maybe you want to process several files:

```
@default_files = ('file-one.tex', 'file-two.tex');
```

Note: If you don't specify `@default_files`, all `.tex` files in the current directory will be used.

11.7 Advanced Options

Latexmk can also do more crazy stuff.

For example, it can create a nomenclature (you'll have to use the *nomencl* package) like this:

```
@cus_dep_list = (@cus_dep_list, "glo gls 0 makenomenclature");
sub makenomenclature {
   system("makeindex $_[0].glo -s nomencl.ist -o $_[0].gls"); }
@generated_exts = (@generated_exts, 'glo');
```

Or, if you are creating your figures in EPS format but you need them in PDF, you can tell Latexmk to convert them for you:

```
@cus_dep_list = (@cus_dep_list, "eps pdf 0 eps2pdf");
sub eps2pdf {
   system("epstopdf $_[0].eps"); }
```

If you need to enable shell escape for `\write18` (e.g. for on-the-fly figure generation):

```
$latex = 'latex -interaction=nonstopmode -shell-escape';
$pdflatex = 'pdflatex -interaction=nonstopmode -shell-escape';
```

And finally, if `latexmk -c` refuses to remove certain files, you can specify their extensions and next time they'll be gone:

```
$clean_ext = "bbl nav out snm";
```

Have fun!

CHAPTER 12

Getting Started with Sphinx and readthedocs.org

- new github repo
- readthedocs.org account
- install sphinx (debian package?)
- https://read-the-docs.readthedocs.org/en/latest/getting_started.html
- sphinx-quickstart
- edit `index.rst`
- run `make html` to check if it works
- commit and push your files to Github (but without the HTML files!)
- create a new project at readthedocs.org using your Github repo
- go to Github admin section, then to *service hooks*, activate *ReadTheDocs*

12.1 Links

Code: http://sphinx-doc.org/markup/code.html

12.2 TODO

- try *TikZ* extension: https://bitbucket.org/philexander/tikz

CHAPTER 13

Quotes

Hard constraints are the midwife to good design.

—Maciej Cegłowski, http://idlewords.com/talks/web_design_first_100_years.htm

CHAPTER 14

Links

Here are some links that don't fit into the other categories on this website.

Master the command line, in one page https://github.com/jlevy/the-art-of-command-line

Falsehoods Programmers Believe About Names https://www.kalzumeus.com/2010/06/17/falsehoods-programmers-believe-about-names/

https://shinesolutions.com/2018/01/08/falsehoods-programmers-believe-about-names-with-examples/

CHAPTER 15

TODO

As mentioned before, all this is work-in-progress.

There are many things wrong, unclear or simply missing.

If you want to contribute, feel free to do so via Github: https://github.com/mgeier/homepage

15.1 List of TODOs

The following list is automatically created by the Sphinx TODO plugin. If there is no list, either all TODOs are done (very unlikely), or they are disabled with the option `todo_include_todos = False` in the file `conf.py`.

Todo: Some explanations, free as in speech, CC, ...

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/mg/checkouts/latest/free_audio_data.rst, line 8.)

Todo: add before commit, staging area, local commits

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/mg/checkouts/latest/git.rst, line 127.)

Todo: more about branches?

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/mg/checkouts/latest/git.rst, line 195.)

Todo: more information about merging and potential merge conflicts

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/mg/checkouts/latest/git.rst, line 200.)

Todo: `git mergetool` is really useful!

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/mg/checkouts/latest/git.rst, line 202.)

Todo: more advertisement for Vim and fugitive!

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/mg/checkouts/latest/git.rst, line 216.)

Todo: `.gitignore`, global ignore file with `core.excludesfile`, reference to https://github.com/github/gitignore

---

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/mg/checkouts/latest/git.rst, line 257.)

Todo: clean/smudge filters (https://nbsphinx.readthedocs.io/en/latest/usage.html#Using-Notebooks-with-Git)

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/mg/checkouts/latest/git-jupyter.rst, line 31.)

Todo: https://nbsphinx.readthedocs.io/ and others

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/mg/checkouts/latest/git-jupyter.rst, line 48.)

Todo: advantages, disadvantages

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/mg/checkouts/latest/git-jupyter.rst, line 52.)

Todo: advantages, disadvantages

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/mg/checkouts/latest/git-jupyter.rst, line 69.)

Todo: `make` runs other programs in the right order, handles intermediate files and generates files only if it's necessary (i.e. if the sources have changed).

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/mg/checkouts/latest/make_tutorial/make_tutorial.rst, line 15.)

Todo: Documentation for GNU make: http://www.gnu.org/software/make/

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/mg/checkouts/latest/make_tutorial/make_tutorial.rst, line 17.)

Todo: this can happen:

```
make: `hello' is up to date.
```

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/mg/checkouts/latest/make_tutorial/make_tutorial.rst, line 120.)

Todo: `make clean`

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/mg/checkouts/latest/make_tutorial/make_tutorial.rst, line 141.)

Todo: clean-target, no prerequisites, one recipe

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/mg/checkouts/latest/make_tutorial/make_tutorial.rst, line 143.)

Todo: $(RM) vs rm

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/mg/checkouts/latest/make_tutorial/make_tutorial.rst, line 145.)

Todo: CXX, CXXFLAGS, CC, CFLAGS, CPPFLAGS, LDLIBS, LDFLAGS, ...

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/mg/checkouts/latest/make_tutorial/make_tutorial.rst, line 169.)

Todo: coverage tool, probably py.test extension

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/mg/checkouts/latest/python.rst, line 32.)

Todo: The open definition

http://opendefinition.org/

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/mg/checkouts/latest/reproducible_research.rst, line 27.)

Todo: Reproducible Research vs. Non-Reproducible Research?

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/mg/checkouts/latest/reproducible_research.rst, line 97.)

Todo: reproducible vs. easily reproducible

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/mg/checkouts/latest/reproducible_research.rst, line 99.)

Todo: online material as supplement to traditional publications

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/mg/checkouts/latest/reproducible_research.rst, line 101.)

Todo: http://en.wikipedia.org/wiki/Reproducibility

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/mg/checkouts/latest/reproducible_research.rst, line 103.)

Todo: http://en.wikipedia.org/wiki/Open_research

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/mg/checkouts/latest/reproducible_research.rst, line 105.)

Todo: TikZ, gnuplot, beamer

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/mg/checkouts/latest/reproducible_research.rst, line 380.)