EasyBuild Documentation

Release 20160713.01

Ghent University

Mon, 01 Aug 2016 13:51:52

Contents

Welcome to the documentation of EasyBuild, a software build and installation framework that allows you to manage (scientific) software on High Performance Computing (HPC) systems in an efficient way.

This documentation is intended for EasyBuild version 2.8.2, and was last rebuilt on Mon, 01 Aug 2016 13:51:43.

CHAPTER 1

Introductory topics

1.1 What is EasyBuild?

EasyBuild is a software build and installation framework that allows you to manage (scientific) software on High Performance Computing (HPC) systems in an efficient way. It is motivated by the need for a tool that combines the following features:

- a **flexible framework** for building/installing (scientific) software
- fully **automates** software builds
- divert from the standard `configure` / `make` / `make install` with custom procedures
- allows for easily **reproducing** previous builds
- keep the software build recipes/specifications **simple and human-readable**
- supports **co-existence of versions/builds** via dedicated installation prefix and module files
- enables **sharing** with the HPC community (win-win situation)
- automagic **dependency resolution**
- **retain logs** for traceability of the build processes

Some key features of EasyBuild:

- build & install (scientific) software **fully autonomously**
  - also interactive installers, code patching, generating module file, ...
- easily *configurable*: config file/environment/command line
  - including aspects like module naming scheme
- thorough logging and archiving (see Understanding EasyBuild logs)
  - entire build process is logged thoroughly, logs are stored in install directory;
  - easyconfig file used for build is archived (install directory + file/svn/git repo)
- automatic **dependency resolution** (see *Enabling dependency resolution, –robot / -r and –robot-paths*)
  - build entire software stack with a single command, using `--robot`
- building software in **parallel**
- robust and thoroughly tested code base, fully unit-tested before each release
- thriving, growing **community**

Take a look at our HUST'14 workshop paper *Modern Scientific Software Management Using EasyBuild and Lmod* (download PDF here) and use that as a reference in case you present academic work mentioning EasyBuild.

1.2 Concepts and terminology

EasyBuild consists of a collection of Python modules and packages that interact with each other, dynamically picking up additional Python modules as needed for building and installing a (stack of) software package(s) specified via simple specification files.

Or, in EasyBuild terminology: the *EasyBuild framework* leverages *easyblocks* to automatically build and install software using particular compiler *toolchains*, as specified by one or multiple *easyconfig files*.

1.2.1 EasyBuild framework

The EasyBuild **framework** embodies the core of the tool, providing functionality commonly needed when installing scientific software on HPC systems. For example, it deals with downloading, unpacking and patching of sources, loading module files for dependencies, setting up the build environment, autonomously running (interactive) shell commands, creating module files that match the specification files, etc.

Included in the framework is an *abstract* implementation of a software build and install procedure, which is split up into different *steps*:

- unpacking sources
- configuration
- build
- installation
- module generation
- etc.

Most of these steps, i.e., the ones which are generally more-or-less analogous across different software packages, have appropriate (default) implementations. The only exceptions are the configuration, build and installation steps that are purposely left unimplemented (since there is no common procedure for them).

Each of the steps can be tweaked and steered via different parameters known to the framework, for which values are either obtained from the provided specification files or set to reasonable default values. See *Easyconfig files*.

In EasyBuild version 2.8.2 the framework source code consists of about 19000 lines of code, organized across about 125 Python modules in roughly a dozen Python package directories, next to almost 7000 lines of code for tests. This provides some notion of the size of the EasyBuild framework and the amount of supporting functionality it has to offer.

1.2.2 Easyblocks

The implementation of a particular software build and install procedure is done in a Python module, which is aptly referred to as an **easyblock**.

Each easyblock ties in with the framework API by defining (or extending/replacing) one or more of the step functions that are part of the abstract procedure used by the EasyBuild framework. Easyblocks typically heavily rely on the supporting functionality provided by the framework, for example for (autonomously) executing (interactive) shell commands and obtaining the command's output and exit code.

A distinction is made between **software-specific** and **generic** easyblocks. Software-specific easyblocks implement a build and install procedure which is entirely custom to one particular software package (e.g., WRF), while generic easyblocks implement a procedure using standard tools (e.g., CMake). Since easyblocks are implemented in an object-oriented scheme, the step methods implemented by a particular easyblock can be reused in others via inheritance, enabling code reuse across build procedure implementations.

For each software package being built, the EasyBuild framework will determine which easyblock should be used, based on the name of the software package or the value of the `easyblock` specification parameter (see *Easyblock specification*). Since EasyBuild v2.0, an easyblock *must* be specified in case no matching easyblock is found based on the software name (cfr. *Automagic fallback to ConfigureMake*).

EasyBuild version 2.4.0 includes 154 software-specific easyblocks and 28 generic easyblocks (see also *List of easyblocks*), providing support for automatically installing a wide range of software packages. Examples range from fairly easy-to-build programs like gzip, other basic tools like compilers, various MPI stacks and commonly used libraries, primarily for x86_64 architecture systems, to large scientific software packages that are notorious for their involved and tedious install procedures, such as: *CP2K*, *NWChem*, *OpenFOAM*, *QuantumESPRESSO*, *WRF*.

1.2.3 Toolchains

EasyBuild employs so-called **compiler toolchains** or, simply *toolchains* for short, which are a major concept in handling the build and installation processes.

A typical toolchain consists of one or more compilers, usually put together with some libraries for specific functionality, e.g., for using an MPI stack for distributed computing, or which provide optimized routines for commonly used math operations, e.g., the well-known BLAS/LAPACK APIs for linear algebra routines.

For each software package being built, the toolchain to be used must be specified in some way.

The EasyBuild framework prepares the *build environment* for the different toolchain components, by loading their respective modules and defining environment variables to specify compiler commands (e.g., via `$F90`), compiler and linker options (e.g., via `$CFLAGS` and `$LDFLAGS`), the list of library names to supply to the linker (via `$LIBS`), etc. This enables making easyblocks largely *toolchain-agnostic* since they can simply rely on these environment variables; that is, unless they need to be aware of, for example, the particular compiler being used to determine the build configuration options.

Recent releases of EasyBuild include out-of-the-box toolchain support for:

- various compilers, including GCC, Intel, Clang, CUDA
- common MPI libraries, such as Intel MPI, MPICH, MVAPICH2, OpenMPI
- various numerical libraries, including ATLAS, Intel MKL, OpenBLAS, ScalaPACK, FFTW

`dummy` toolchain

The `dummy` toolchain is a special case. It is an *empty* toolchain, i.e. a toolchain without any components, and corresponds to using the readily available compilers and libraries (e.g., the ones provided by the OS, or by modules which were loaded before issuing the `eb` command).

When the `dummy` toolchain is used, a corresponding `dummy` module file is not required/loaded and no build environment is being defined.

When the toolchain version is also specified as `dummy`, no (build) dependencies will be loaded when the build is performed. If the toolchain version is specified as an empty string, the listed dependencies will be loaded (as is done with other toolchains).

1.2.4 Easyconfig files

The specification files that are supplied to EasyBuild are referred to as **easyconfig files** (or simply *easyconfigs*), which are basically plain text files containing (mostly) key-value assignments for build parameters supported by the framework, also referred to as **easyconfig parameters** (see Writing easyconfig files: the basics for more information).

Note that easyconfig files only provide the bits of information required to determine the corresponding module name; the module name itself is computed by EasyBuild framework by querying the module naming scheme being used. The complete list of supported easyconfig parameters can be easily obtained via the EasyBuild command line using `eb -a` (see also *All available easyconfig parameters, –avail-easyconfig-params / -a*).

As such, each easyconfig file provides a complete specification of which particular software package should be installed, and which settings should be used for building it. After completing an installation, EasyBuild copies

the used easyconfig file to the install directory, as a template, and also supports maintaining an easyconfig archive which is updated on every successful installation. Therefore, reproducing installations becomes trivial.

EasyBuild version 2.8.2 includes support for over 511 different software packages, spread over 2800 easyconfig files describing distinct builds.

1.2.5 Extensions

Some software packages support installing additional add-ons alongside the 'main' software, either in the same installation prefix, or in a separate location.

In EasyBuild, we use the neutral term '**extensions**' to refer these add-ons.

Well-known examples include:

- Perl modules (http://www.cpan.org/modules/)
- Python packages (https://pypi.python.org/pypi)
- R libraries (http://cran.r-project.org/web/packages/)
- Ruby gems (http://guides.rubygems.org/what-is-a-gem/)

1.3 Typical workflow example: building and installing WRF

This section shows an example case of building Weather Research and Forecasting (WRF) scientific software application, which is a notoriously complex software application to build and install. With EasyBuild however, WRF can be installed quite easily and here is how.

First, you search which easyconfigs are available for WRF, using `--search` (see *Searching for easyconfigs, –search / -S*) and you select one based on the software version, toolchain, etc.

Using the selected easyconfig file, you can get an overview of the planned installations using `--dry-run` (see *Get an overview of planned installations –dry-run / -D*).

Finally, building and installing WRF is done by specifying the matching easyconfig file in the eb command line, and using `--robot` (see *Enabling dependency resolution, –robot / -r and –robot-paths*) to enable dependency resolution. That way WRF and all of its dependencies are installed with *a single command*!

1.3.1 Searching for available easyconfigs files

Searching for build specification for a particular software package can be done using the `--search`/`-S` command line options (see *Searching for easyconfigs, –search / -S*); for example, to get a list of available easyconfig files for WRF:

```
$ eb -S WRF
== temporary log file in case of crash /tmp/easybuild-MdAp7p/easybuild-zEBJMk.log
== Searching (case-insensitive) for 'WRF' in /home/example/.local/easybuild/software/EasyBuild/1.
CFGS1=/home/example/.local/easybuild/software/EasyBuild/1.15.2/lib/python2.7/site-packages/easybu
 * $CFGS1/WRF-3.3.1-goalf-1.1.0-no-OFED-dmpar.eb
 * $CFGS1/WRF-3.3.1-goolf-1.4.10-dmpar.eb
 [...]
 * $CFGS1/WRF-3.5-goolf-1.4.10-dmpar.eb
 * $CFGS1/WRF-3.5-ictce-4.1.13-dmpar.eb
 * $CFGS1/WRF-3.5-ictce-5.3.0-dmpar.eb
 * $CFGS1/WRF-3.5.1-goolf-1.4.10-dmpar.eb
 * $CFGS1/WRF-3.5.1-goolf-1.5.14-dmpar.eb
 [...]
== temporary log file /tmp/easybuild-MdAp7p/easybuild-zEBJMk.log has been removed.
== temporary directory /tmp/easybuild-MdAp7p has been removed.
```

Various easyconfig files are found: for different versions of WRF (e.g., v3.5 and v3.5.1), for different (versions of) compiler toolchains (e.g., goolf v1.4.10, goolf v1.5.14, ictce), etc.

For the remainder of this example, we will use the available `WRF-3.5.1-goolf-1.4.10-dmpar.eb` easyconfig file to specify to EasyBuild to build and install WRF v3.5.1 using version 1.4.10 of the `goolf` toolchain.

See *List of known toolchains* for a list of available toolchains. The `goolf` toolchain stands for `GCC, OpenMPI, OpenBLAS, ScaLAPACK and FFTW`.

1.3.2 Getting an overview of planned installations

To get an overview of the software that EasyBuild is going to build and install we can use the `--dry-run`/`-D` (see *Get an overview of planned installations –dry-run / -D*) command line option. This will show a list of easyconfig files that will be used, together with the module files that will be installed, as well as their current availability (`[x]` marks available modules).

Note that EasyBuild will take care of all of the dependencies of WRF as well, and can even install the compiler toolchain as well if the corresponding modules are not available yet:

```
$ eb WRF-3.5.1-goolf-1.4.10-dmpar.eb -Dr
== temporary log file in case of crash /tmp/easybuild-HqpcAZ/easybuild-uNzmpk.log
Dry run: printing build status of easyconfigs and dependencies
CFGS=/home/example/.local/easybuild/software/EasyBuild/1.15.2/lib/python2.7/site-packages/easybuil
 * [x] $CFGS/g/GCC/GCC-4.7.2.eb (module: GCC/4.7.2)
 * [x] $CFGS/h/hwloc/hwloc-1.6.2-GCC-4.7.2.eb (module: hwloc/1.6.2-GCC-4.7.2)
 * [x] $CFGS/o/OpenMPI/OpenMPI-1.6.4-GCC-4.7.2.eb (module: OpenMPI/1.6.4-GCC-4.7.2)
 * [x] $CFGS/g/gompi/gompi-1.4.10.eb (module: gompi/1.4.10)
 * [x] $CFGS/o/OpenBLAS/OpenBLAS-0.2.6-gompi-1.4.10-LAPACK-3.4.2.eb (module: OpenBLAS/0.2.6-gompi-
 * [x] $CFGS/f/FFTW/FFTW-3.3.3-gompi-1.4.10.eb (module: FFTW/3.3.3-gompi-1.4.10)
 * [x] $CFGS/s/ScaLAPACK/ScaLAPACK-2.0.2-gompi-1.4.10-OpenBLAS-0.2.6-LAPACK-3.4.2.eb (module: Scal
 * [x] $CFGS/g/goolf/goolf-1.4.10.eb (module: goolf/1.4.10)
 * [ ] $CFGS/s/Szip/Szip-2.1-goolf-1.4.10.eb (module: Szip/2.1-goolf-1.4.10)
 * [ ] $CFGS/f/flex/flex-2.5.37-goolf-1.4.10.eb (module: flex/2.5.37-goolf-1.4.10)
 * [ ] $CFGS/n/ncurses/ncurses-5.9-goolf-1.4.10.eb (module: ncurses/5.9-goolf-1.4.10)
 * [ ] $CFGS/m/M4/M4-1.4.16-goolf-1.4.10.eb (module: M4/1.4.16-goolf-1.4.10)
 * [ ] $CFGS/j/JasPer/JasPer-1.900.1-goolf-1.4.10.eb (module: JasPer/1.900.1-goolf-1.4.10)
 * [ ] $CFGS/z/zlib/zlib-1.2.7-goolf-1.4.10.eb (module: zlib/1.2.7-goolf-1.4.10)
 * [ ] $CFGS/t/tcsh/tcsh-6.18.01-goolf-1.4.10.eb (module: tcsh/6.18.01-goolf-1.4.10)
 * [ ] $CFGS/b/Bison/Bison-2.7-goolf-1.4.10.eb (module: Bison/2.7-goolf-1.4.10)
 * [ ] $CFGS/h/HDF5/HDF5-1.8.10-patch1-goolf-1.4.10.eb (module: HDF5/1.8.10-patch1-goolf-1.4.10)
 * [ ] $CFGS/d/Doxygen/Doxygen-1.8.3.1-goolf-1.4.10.eb (module: Doxygen/1.8.3.1-goolf-1.4.10)
 * [ ] $CFGS/n/netCDF/netCDF-4.2.1.1-goolf-1.4.10.eb (module: netCDF/4.2.1.1-goolf-1.4.10)
 * [ ] $CFGS/n/netCDF-Fortran/netCDF-Fortran-4.2-goolf-1.4.10.eb (module: netCDF-Fortran/4.2-goolf
 * [ ] $CFGS/w/WRF/WRF-3.5.1-goolf-1.4.10-dmpar.eb (module: WRF/3.5.1-goolf-1.4.10-dmpar)
== temporary log file /tmp/easybuild-HqpcAZ/easybuild-uNzmpk.log has been removed.
== temporary directory /tmp/easybuild-HqpcAZ has been removed.
```

1.3.3 Installing a software stack

To make EasyBuild build and install WRF, including all of its dependencies, a **single command** is sufficient.

By using the `--robot`/`-r` (see *Enabling dependency resolution, –robot / -r and –robot-paths*) command line option, we enable dependency resolution such that the entire software stack is handled:

```
$ eb WRF-3.5.1-goolf-1.4.10-dmpar.eb --robot
[...]
== building and installing zlib/1.2.7-goolf-1.4.10...
[...]
== building and installing Szip/2.1-goolf-1.4.10...
[...]
== building and installing ncurses/5.9-goolf-1.4.10...
```

```
[...]
== building and installing flex/2.5.37-goolf-1.4.10...
[...]
== building and installing M4/1.4.16-goolf-1.4.10...
[...]
== building and installing JasPer/1.900.1-goolf-1.4.10...
[...]
== building and installing HDF5/1.8.10-patch1-goolf-1.4.10...
[...]
== building and installing tcsh/6.18.01-goolf-1.4.10...
[...]
== building and installing Bison/2.7-goolf-1.4.10...
[...]
== building and installing Doxygen/1.8.3.1-goolf-1.4.10...
[...]
== building and installing netCDF/4.2.1.1-goolf-1.4.10...
[...]
== building and installing netCDF-Fortran/4.2-goolf-1.4.10...
[...]
== building and installing WRF/3.5.1-goolf-1.4.10-dmpar...
[...]
== Build succeeded for 13 out of 13
```

Once the installation has succeeded, modules will be available for WRF and all of its dependencies:

```
$ module load WRF
$ module list
Currently Loaded Modulefiles:
  1) GCC/4.7.2                                                 9) JasPer/1.900.1-goolf-1.4.10
  2) hwloc/1.6.2-GCC-4.7.2                                    10) zlib/1.2.7-goolf-1.4.10
  3) OpenMPI/1.6.4-GCC-4.7.2                                  11) Szip/2.1-goolf-1.4.10
  4) gompi/1.4.10                                             12) HDF5/1.8.10-patch1-goolf-1.4.1
  5) OpenBLAS/0.2.6-gompi-1.4.10-LAPACK-3.4.2                 13) netCDF/4.2.1.1-goolf-1.4.10
  6) FFTW/3.3.3-gompi-1.4.10                                  14) netCDF-Fortran/4.2-goolf-1.4.1
  7) ScaLAPACK/2.0.2-gompi-1.4.10-OpenBLAS-0.2.6-LAPACK-3.4.2 15) WRF/3.5.1-goolf-1.4.10-dmpar
  8) goolf/1.4.10
```

For more information, see the other topics discussed in the documentation (see *Introductory topics*).

CHAPTER 2

Getting started

2.1 Installing EasyBuild

EasyBuild is Python software, so there are a couple of ways to install it.

We recommend using the **bootstrap** install procedure described at *Bootstrapping EasyBuild*, because of various issues with the different installation tools that are available for Python packages.

Do take into account the required and optional dependencies (see *Requirements* and *Dependencies*).

For advanced options, see *Advanced bootstrapping options*.

Notes on other ways of installing EasyBuild are available under section *Alternative installation methods*.

Contents

2.1.1 Requirements

The only strict requirements are:

- Linux (or OS X)
- **Python version 2.6**, or a more recent 2.x version + `setuptools`, `vsc-install` & `vsc-base`
  - see also *Required Python packages*

- a **modules tool**: Tcl(/C) environment modules or Lmod
  - the actual module command/script (`modulecmd`, `modulecmd.tcl` or `lmod`) *must* be available via `$PATH`
  - see *Required modules tool* for more details

For more information on (optional) dependencies, see *Dependencies*.

2.1.2 Bootstrapping EasyBuild

Installing any Python package can be a real pain, and since EasyBuild is basically a set of Python packages glued together, installing EasyBuild may (ironically) cause some headaches.

To resolve this, we have created a bootstrap script that installs the latest EasyBuild version for you together with an environment module for it - and yes, we use EasyBuild for doing so.

Bootstrapping procedure

The easiest way (by far) to install EasyBuild is by bootstrapping it, i.e., installing the latest EasyBuild release (obtained from PyPI) using EasyBuild itself.

To bootstrap EasyBuild:

- download the bootstrap script from https://raw.githubusercontent.com/hpcugent/easybuild-framework/develop/easybuild/scripts/bootstrap_eb.py
- execute it, and specify an installation prefix as an argument

Yes, it's that easy!

The bootstrap script will perform a 3-stage bootstrap procedure:

- *stage 0:* download and install a specific version of the `distribute` Python package, which provides the `easy_install` tool for installing Python software into a temporary directory
- *stage 1:* download and install the most recent version of EasyBuild from PyPI into a temporary location, using the `easy_install` tool from stage 0
- *stage 2:* install the most recent version of EasyBuild into the specified installation prefix, using the temporary EasyBuild installation from stage 1 (inception!)

This should result in an `EasyBuild` module that you can load to start using EasyBuild, after making sure the module is available by updating `$MODULEPATH`. More specifically, you need to include the `modules/all` subdirectory of the specified installation prefix into `$MODULEPATH`.

For example:

```
# pick an installation prefix to install EasyBuild to (change this to your liking)
EASYBUILD_PREFIX=$HOME/.local/easybuild

# download script
curl -O https://raw.githubusercontent.com/hpcugent/easybuild-framework/develop/easybuild/scripts/

# bootstrap EasyBuild
python bootstrap_eb.py $EASYBUILD_PREFIX

# update $MODULEPATH, and load the EasyBuild module
module use $EASYBUILD_PREFIX/modules/all
module load EasyBuild
```

Note: The path you specify to the bootstrap script is where EasyBuild should be installed. If you also want software that is built/installed using EasyBuild to be located there, you will need to configure EasyBuild accord-

ingly (see *Configuring EasyBuild*), for example by putting the definition for `$EASYBUILD_PREFIX` in your `.bashrc`.

See also *Configuring EasyBuild*.

Tip: The bootstrap script will only succeed if command `module --version` reports a sufficiently recent version (e.g., environments-modules-c >=v3.2.10 or Lmod >= 5.6.3), because modules are applied throughout, e.g., to resolve dependencies and detect already installed software.

Normally, only when the above fails to work for you for some reason, should you resort to one of the alternative approaches documented at *Alternative installation methods* (these are more involved but also they may give more control).

Sanity check

Compare the version of `eb`, the main EasyBuild command, with the version of the EasyBuild module that was installed. For example:

```
$ module load EasyBuild
$ module list

Currently Loaded Modules:
  1) EasyBuild/1.16.1

$ eb --version
This is EasyBuild 1.16.1 (framework: 1.16.1, easyblocks: 1.16.1) on host example.local
```

Tip: The Tcl-based or Lmod implementations of environment modules do their default sorting differently. The former will normally sort in the lexicographic order, while Lmod follows an approach that is closer to Python's construct `LooseVersion` way of ordering. Such aspects may make a big difference, if you have installed both versions 1.9.0 and 1.15.2, with respect to what is the version being loaded by default.

Running unit tests

After completion of the bootstrap procedure and loading the `EasyBuild` module, try running the EasyBuild unit tests:

```
# specify modules tool to use: EnvironmentModulesC (default), EnvironmentModulesTcl, Lmod)
# see also http://easybuild.readthedocs.org/en/latest/Configuration.html#modules-tool-modules-too
export TEST_EASYBUILD_MODULES_TOOL=Lmod

# run full unit test suite for EasyBuild framework
python -m test.framework.suite
```

Keep in mind that this is just an example, more details about the EasyBuild unit tests are available at *Unit tests*.

If this does not complete successfully, please open an issue to report it.

Example bootstrap run

Example output for bootstrapping EasyBuild v1.16.1:

```
[[INFO]] Found module command 'lmod' (Lmod), so using it.
[[INFO]]

+++ STAGE 0: installing distribute via included (patched) distribute_setup.py...
```

```
Downloading http://pypi.python.org/packages/source/d/distribute/distribute-0.6.34.tar.gz
Extracting in /tmp/tmpz0zyAG
Now working in /tmp/tmpz0zyAG/distribute-0.6.34
Installing Distribute
[[INFO]]

+++ STAGE 1: installing EasyBuild in temporary dir with easy_install...


Installing with setuptools.setup...
Installing version 1.16.1
warning: install_lib: 'build/lib' does not exist -- no Python modules to install

zip_safe flag not set; analyzing archive contents...
Installing with setuptools.setup...
Installing version 1.16.1 (API version 1)
Installing with setuptools.setup...
Installing version 1.16.1 (required versions: API >= 1)
Installing with setuptools.setup...
Installing version 1.16.1.0 (required versions: API >= 1, easyblocks >= 1.16)
warning: install_lib: 'build/lib' does not exist -- no Python modules to install

[[INFO]]

+++ STAGE 2: installing EasyBuild in /home/example/.local/easybuild with EasyBuild from stage 1..


Couldn't import dot_parser, loading of dot files will not be possible.
== temporary log file in case of crash /tmp/easybuild-zql_Ft/easybuild-peQ8GA.log
== processing EasyBuild easyconfig /tmp/tmp_gzHPM/EasyBuild-1.16.1.eb
== building and installing EasyBuild/1.16.1...
== fetching files...
== creating build dir, resetting environment...
== unpacking...
== patching...
== preparing...
== configuring...
== building...
== testing...
== installing...
== taking care of extensions...
== packaging...
== postprocessing...
== sanity checking...
== cleaning up...
== creating module...
== COMPLETED: Installation ended successfully
== Results of the build can be found in the log file /home/example/.local/easybuild/software/Easy
== Build succeeded for 1 out of 1
== temporary log file /tmp/easybuild-zql_Ft/easybuild-peQ8GA.log has been removed.
== temporary directory /tmp/easybuild-zql_Ft has been removed.
[[INFO]] Done!
[[INFO]]
[[INFO]] EasyBuild v1.16.1 was installed to /home/example/.local/easybuild, so make sure your $MO
[[INFO]]
[[INFO]] Run 'module load EasyBuild', and run 'eb --help' to get help on using EasyBuild.
[[INFO]] Set $EASYBUILD_MODULES_TOOL to 'Lmod' to use the same modules tool as was used now.
[[INFO]]
[[INFO]] By default, EasyBuild will install software to $HOME/.local/easybuild.
[[INFO]] To install software with EasyBuild to /home/example/.local/easybuild, make sure $EASYBUI
[[INFO]] See http://easybuild.readthedocs.org/en/latest/Configuration.html for details on configu
```

After the bootstrap completes, the installed `EasyBuild` module can be loaded:

```
$ module use $HOME/.local/easybuild/modules/all
$ module av
------------------------ /home/example/.local/easybuild/modules/all -------------------------
EasyBuild/1.16.1

$ module load EasyBuild
$ module list
Currently Loaded Modulefiles:
  1) EasyBuild/1.16.1

$ which eb
/home/example/.local/easybuild/software/EasyBuild/1.16.1/bin/eb

$ eb --version
This is EasyBuild 1.16.1 (framework: 1.16.1, easyblocks: 1.16.1) on host example.local.
```

Now, enjoy!

2.1.3 Advanced bootstrapping options

To use these advanced options, make sure you are using the latest version of the bootstrap script, available at https://raw.githubusercontent.com/hpcugent/easybuild-framework/develop/easybuild/scripts/bootstrap_eb.py .

Skipping the installation of `easy_install` (stage 0)

The first stage of the bootstrap procedure consists of installing a specific version of the `distribute` Python package, which provides the `easy_install` installation tool for Python software, in a temporary location. The bootstrap script then tries to ensure this particular installation is used during the other bootstrap stages.

If you already have a version of `easy_install` on your system, and if you are confident that it behaves (in particular, that it complies to the installation prefix specified via `--prefix`), you can skip stage 0 of the bootstrap procedure.

To do so, simply define the `EASYBUILD_BOOTSTRAP_SKIP_STAGE0` environment variable (the value doesn't matter):

```
$ export EASYBUILD_BOOTSTRAP_SKIP_STAGE0=1
$ python bootstrap_eb.py $HOME/eb/test_nostage0
...
[[INFO]] Skipping stage0, using local distribute/setuptools providing easy_install
...

+++ STAGE 1: installing EasyBuild in temporary dir with easy_install...

...
```

Bootstrapping using supplied source tarballs

By default, the bootstrap script will download the most recent (stable) EasyBuild version from PyPI, the Python Package Index (https://pypi.python.org/pypi).

Recent versions of the bootstrap script also allow to supply source tarballs for the different EasyBuild components (framework, easyblocks, easyconfigs), and (optionally) the vsc-base library EasyBuild depends on.

The source tarball filenames must match a pattern like `<pkg>*.tar.gz`, where `<pkg>` denotes the name of the respective EasyBuild component:

- `vsc-base*.tar.gz`

- `easybuild-framework*.tar.gz`
- `easybuild-easyblocks*.tar.gz`
- `easybuild-easyconfigs*.tar.gz`

The location of the source tarballs can be specified using the `$EASYBUILD_BOOTSTRAP_SOURCEPATH` environment variable.

Example usage, with the relevant output at the start of stage 1 of the bootstrap process:

```
$ export EASYBUILD_BOOTSTRAP_SOURCEPATH=/tmp/$USER
$ python bootstrap_eb.py $HOME/eb/test_tarballs

+++ STAGE 0: installing distribute via included (patched) distribute_setup.py...

...

+++ STAGE 1: installing EasyBuild in temporary dir with easy_install...

[[INFO]] Fetching sources from /tmp/example...
[[INFO]] Found /tmp/example/vsc-base-2.0.2.tar.gz for vsc-base package
[[INFO]] Found /tmp/example/easybuild-framework-v2.0.0dev.tar.gz for easybuild-framework package
[[INFO]] Found /tmp/example/easybuild-easyblocks.tar.gz for easybuild-easyblocks package
[[INFO]] Found /tmp/example/easybuild-easyconfigs.tar.gz for easybuild-easyconfigs package
...
```

Note: Providing a source tarball for `vsc-base` is *optional*. If not specified, the most recent version available on PyPI will be downloaded and installed automatically when the `easybuild-framework` package is installed. Source tarballs for all three EasyBuild components *must* be provided when `$EASYBUILD_BOOTSTRAP_SOURCEPATH` is defined, however.

2.1.4 Updating an existing EasyBuild installation

To upgrade to a newer EasyBuild version (say, 2.8.2) than the one currently installed there are several options:

1. (re)bootstrap EasyBuild to obtain an `EasyBuild` module for version 2.8.2, using the instructions above, see *Bootstrapping EasyBuild*.
2. install EasyBuild version 2.8.2 with a previous version of EasyBuild, using the easyconfig file available on the develop branch at Github
3. install EasyBuild version 2.8.2 from PyPI, using one of the standard Python installation tools (`easy_install`, `pip`, ...), see also *Installing EasyBuild without admin rights*
4. update the `master` branch of your Git working copies of the different EasyBuild repositories

2.1.5 Dependencies

EasyBuild has a couple of dependencies, some are optional.

Required dependencies

- **Linux** (or OSX) operating system
  - some common shell tools are expected to be available, see *Required shell tools*
- Python 2.6, or a more recent 2.x version
  - some additional non-standard Python packages are required, see *Required Python packages*
- a **modules tool**: Tcl(/C) environment modules or Lmod

  - the actual modules tool *must* be available via `$PATH`, see *Required modules tool*
- a C/C++ compiler (e.g., `gcc` and `g++`)
  - only required to build and install GCC with, or as a dependency for the Intel compilers, for example

Required shell tools

A couple of shell tools may be required, depending on the particular use case (in relative order of importance):

- shell builtin commands:
  - `type`, for inspecting the `module` function (if defined)
  - `ulimit`, for quering user limits
- tools for unpacking (source) archives:
  - commonly required: `tar`, `gunzip`, `bunzip2`
  - occasionally required: `unzip`, `unxz`
- `patch`, for applying patch files to unpacked sources (relatively common)
- `rpm` or `dpkg`, for quering OS dependencies (only needed occasionally)
- `locate`, only as a (poor mans) fallback to `rpm`/`dpkg` (rarely needed)
- `sysctl`, for quering system characteristics (only required on non-Linux systems)

Required modules tool

EasyBuild not only generates module files to be used along with the software it installs, it also depends on the generated modules, mainly for resolving dependencies. Hence, a modules tool must be available to consume module files with.

Supported module tools:

- Tcl/C environment-modules (version >= 3.2.10)
- Tcl-only variant of environment modules
- Lmod (version >= 5.6.3), *highly recommended*

Note: The path to the actual modules tool binary/script used *must* be included in `$PATH`, to make it readily available to EasyBuild.

- for Tcl/C environment modules: `modulecmd`
- for Tcl-only environment modules: `modulecmd.tcl`
- for Lmod: `lmod`

The path where the modules tool binary/script is located can be determined via the definition of the `module` function; for example, using `type module` or `type -f module`.

Note: For Lmod specifically, EasyBuild will try to fall back to finding the `lmod` binary via the `$LMOD_CMD` environment variable, in case `lmod` is not availabe in `$PATH`.

In EasyBuild versions *prior* to 2.1.1, the path specified by `$LMOD_CMD` was (erroneously) preferred over the (first) `lmod` binary available via `$PATH`.

Additional notes:

- Tcl(/C) environment-modules requires Tcl to be installed (with header files and development libraries)

- Lmod requires Lua and a couple of non-standard Lua libraries (`lua-posix`, `lua-filesystem`) to be available
  - Tcl (`tclsh`) must also be available for Lmod to support module files in `Tcl` syntax
- a guide to installing Tcl/C environment modules without having root permissions is available at *Installing environment modules without root permissions*.
- a guide to installing Lmod without having root permissions is available at *Installing Lmod without root permissions*.

Required Python packages

- `setuptools`: used to define the `easybuild` namespace across different directories
  - available at https://pypi.python.org/pypi/setuptools
  - must be version 0.6 or more recent
  - strictly required since EasyBuild v2.7.0
- `vsc-install`: provides setuptools functions and support for unit test suites for Python tools
  - also required to install `vsc-base` (see below)
  - available at https://pypi.python.org/pypi/vsc-install
  - the required version depends primarily on the `vsc-base` version
- `vsc-base`: a Python library providing the `fancylogger` and `generaloption` Python modules
  - available at https://pypi.python.org/pypi/vsc-base and https://github.com/hpcugent/vsc-base
  - the required version of `vsc-base` depends on the EasyBuild version

---

Note: `vsc-base` is installed automatically along with EasyBuild, if an installation procedure is used that consumes the `setup.py` script that comes with the EasyBuild framework (e.g., EasyBuild or the EasyBuild bootstrap script, `pip`, `easy_install`, ...)

Other Python packages are optional dependencies, see *Optional Python packages*.

Optional dependencies

Some dependencies are optional and are only required to support certain features.

Optional Python packages

- GitPython, only needed if EasyBuild is hosted in a git repository or if you're using a git repository for easyconfig files (.eb)
- pysvn, only needed if you're using an SVN repository for easyconfig files (.eb)
- python-graph-dot, only needed for building nice-looking dependency graphs using `--dep-graph *.dot`.
- **graphviz for Python,** only needed for building nice-looking dependency graphs using `--dep-graph *.pdf / *.png`.

2.1.6 Sources

EasyBuild is split up into three different packages, which are available from the Python Package Index (PyPi):

- easybuild-framework - the EasyBuild framework, which includes the easybuild.framework and easybuild.tools Python packages that provide general support for building and installing software
- easybuild-easyblocks - a collection of easyblocks that implement support for building and installing (collections of) software packages
- easybuild-easyconfigs - a collection of example easyconfig files that specify which software to build, and using which build options; these easyconfigs will be well tested with the latest compatible versions of the easybuild-framework and easybuild-easyblocks packages

Next to these packages, a meta-package named easybuild is also available on PyPi, in order to easily install the full EasyBuild distribution.

The source code for these packages is also available on GitHub:

- easybuild-framework git repository
- easybuild-easyblocks git repository
- easybuild-easyconfigs git repository
- the main EasyBuild repository mainly hosts *this* EasyBuild documentation

2.1.7 In case of installation issues...

Should the installation of EasyBuild fail for you, please open an issue to report the problems you're running into.

How to collect info in case sanity checks fail or there is another issue

In order to get a better understanding in which kind of environment you are using the bootstrap script, please copy-paste the commands below and provide the output in your problem report. **Do not worry if some of these commands fail or spit out error messages.**

```
python -V
type module
type -f module
module --version
module av EasyBuild
which -a eb
eb --version
```

2.2 Configuring EasyBuild

This page discusses the recommended style of configuring EasyBuild, which is supported since EasyBuild v1.3.0.

Contents

2.2.1 Supported configuration types

Configuring EasyBuild can be done by:

- using `eb` with **command line arguments**
- setting **environment variables** (`$EASYBUILD_...`)
- providing one or more **configuration files**

Of course, combining any of these types of configuration works too (and is even fairly common).

The order of preference for the different configuration types is as listed above, that is:

- environment variables override the corresponding entries in the configuration file
- command line arguments in turn override the corresponding environment variables *and* matching entries in the configuration file

Consistentency across supported configuration types

Note that the various available configuration options are handled **consistently** across the supported configuration types.

For example: to configure EasyBuild to use Lmod as modules tool, the following alternatives are available:

- configuration file entry (key-value assignment):

```
[config]
modules-tool = Lmod
```

- environment variable (upper case, `EASYBUILD_` prefix, -'s becomes _'s):

```
$ export EASYBUILD_MODULES_TOOL=Lmod
```

- command line argument (long options preceded by `--` and (optionally) using `=`):

```
$ eb --modules-tool=Lmod
```

or

```
$ eb --modules-tool Lmod
```

For more details w.r.t. each of the supported configuration types, see below.

Configuration file(s)

List of used configuration files

The list of configuration files that will be used by EasyBuild is determined in the following order of preference:

- the path(s) specified via the **command line argument** `--configfiles`
- the path(s) specified via the `$EASYBUILD_CONFIGFILES` **environment variable**
- the **default paths** for EasyBuild configuration files

Default configuration files By default, EasyBuild will use existing configuration files at the following paths:

- `$dir/easybuild.d/*.cfg`, for each directory `$dir` listed in `$XDG_CONFIG_DIRS` (where `$XDG_CONFIG_DIRS` defaults to `/etc`)
- `$XDG_CONFIG_HOME/easybuild/config.cfg` (where `$XDG_CONFIG_HOME` defaults to `$HOME/.config`)

Hence, if `$XDG_CONFIG_HOME` and `$XDG_CONFIG_DIRS` are not defined, EasyBuild will only consider default configuration files at `/etc/easybuild.d/*.cfg` and `$HOME/.config/easybuild/config.cfg`.

The configuration file located in `$XDG_CONFIG_HOME` will be listed *after* the ones obtained via `$XDG_CONFIG_DIRS`, such that user-defined configuration settings have preference over system defaults.

A detailed overview of the list of default configuration files is available via `eb --show-default-configfiles` (available since EasyBuild v2.1.0). For example:

```
$ XDG_CONFIG_DIRS=/tmp/etc:/tmp/moreetc eb --show-default-configfiles
Default list of configuration files:

[with $XDG_CONFIG_HOME: (not set), $XDG_CONFIG_DIRS: /tmp/etc:/tmp/moreetc]

* user-level: ${XDG_CONFIG_HOME:-$HOME/.config}/easybuild/config.cfg
  -> /home/example/.config/easybuild/config.cfg => found
* system-level: ${XDG_CONFIG_DIRS:-/etc}/easybuild.d/*.cfg
  -> {/tmp/etc, /tmp/moreetc}/easybuild.d/*.cfg => /tmp/etc/easybuild.d/config.cfg, /tmp/moreetc/e

Default list of existing configuration files (4): /tmp/etc/easybuild.d/config.cfg, /tmp/moreetc/ea
```

Multiple configuration files If multiple configuration files are listed via a mechanism listed above, the settings in the last configuration file have preference over the others.

Each available configuration file will be used, and the configuration settings specified in these files will be retained according to the order of preference as indicated above: settings in configuration files specfied via `--configfiles` override those in configuration files specified via `$EASYBUILD_CONFIGFILES`, which in turns override settings in default configuration files.

Ignored configuration files On top of this, the `--ignoreconfigfiles` configuration option allows to specify configuration files that should be *ignored* by EasyBuild (regardless of whether they are specified via any of the options above).

Configuration file format

The EasyBuild configuration file follows the default Python configuration format as parsed by the `configparser` module (see http://docs.python.org/2/library/configparser.html).

Configuration files are organized in sections, the section name for a particular configuration setting is indicated in the output of `eb --help`. Some examples sections are: `MAIN`, `basic`, `config`, `informative`, `override`, `regtest`, `software`, `unittest`, etc.

Sections are indicated by specifying the section name in square brackets on a dedicated line, e.g., `[basic]`.

Configuration settings are specified in a `key = value` or `key: value` format, **without using quotes for string-like values**. For boolean configuration settings, values that evaluated to `True` (e.g., `true`, `1`, ...) are all equivalent to enabling the setting.

Comment lines start with a hash character `#` (just like in Python code).

An example configuration file that should make everything clear is shown below.

```
[basic]
# always enable logging to stdout
logtostdout = true
[config]
# use Lmod as modules tool
modules-tool: Lmod
# use different default installation path
prefix=/home/you/work/easybuild/
```

Templates and constants supported in configuration files

Two types of template values `%(...)s` are supported in configuration files:

- for configuration options defined in the configuration file (and only those)
  - *syntax:* `%(opt)s`, i.e., using the (lowercase) name of the configuration option
- for the default value of selected configuration options (see `eb --avail-cfgfile-constants`)
  - *syntax:* `%(DEFAULT_OPT)s`, i.e., using the uppercase name of the configuration option and prefixed with `DEFAULT_`

Note: These template values are only supported in configuration files, *not* in environment variable values or command line option values.

Note: Using an unknown template value, i.e. either one for a configuration option that was not defined in the configuration file, or a non-existing one for a particular default value, will result in an error like: `ConfigParser.InterpolationMissingOptionError: Bad value substitution`.

Example To include both the (custom) location for the easyconfigs archive repository and the default list of robot search paths in the active robot search path, the following configuration file entry can be used, featuring the template for the `repositorypath` configuration option and the provided `DEFAULT_ROBOT_PATHS` constant:

```
repositorypath = /home/example/easybuild/easyconfigs_archive
robot-paths = %(repositorypath)s:%(DEFAULT_ROBOT_PATHS)s
```

See also *Controlling the robot search path*.

Generating a template configuration file

Since EasyBuild v1.10, a command line option `--confighelp` is available that prints out the help text as an annotated configuration file. This can be used as an empty template configuration file:

```
$ mkdir -p $HOME/.easybuild
$ eb --confighelp > $HOME/.easybuild/config.cfg
```

```
$ head $HOME/.easybuild/config.cfg
[MAIN]
# Enable debug log mode (def False)
#debug=
# Enable info log mode (def False)
#info=
# Enable info quiet/warning mode (def False)
#quiet=

[basic]
# Print build overview incl. dependencies (full paths) (def False)
```

Environment variables

All configuration settings listed as long options in `eb --help` can also be specified via `EASYBUILD_`-prefixed environment variables.

Configuration settings specified this way always override the corresponding setting specified in a configuration file.

For example, to enable debug logging using an environment variable:

```
$ export EASYBUILD_DEBUG=1
```

More examples of using environment variables to configure EasyBuild are shown in the sections below.

Tip: Any configuration option of EasyBuild which can be tuned by command line or via the configuration file, can also be tuned via a corresponding environment variable.

Note: If any `$EASYBUILD`-prefixed environment variables are defined that do not correspond to a known configuration option, EasyBuild will report an error message and exit.

Command line arguments

The configuration type with the highest precedence are the `eb` command line arguments, which override settings specified through environment variables or in configuration files.

For some configuration options, both short and long command line arguments are available (see `eb --help`); the long options indicate how the configuration setting should be specified in a configuration file or via an environment variable (`$EASYBUILD_<LONGOPTION>`).

For boolean configuration settings, both the `--<option>` and `--disable-<option>` variants are always available.

Examples (more below):

- enable debug logging (long option) and logging to stdout (short option)

```
$ eb --debug -l ...
```

- use `/dev/shm` as build path, install to temporary install path, disable debug logging

```
$ eb --buildpath=/dev/shm --installpath=/tmp/$USER --disable-debug ...
```

2.2.2 Overview of current configuration (`--show-config`, `--show-full-config`)

To get an overview of the current EasyBuild configuration across all configuration types, you can use `eb --show-config`.

The output will specify:

- any configuration setting for which the current value is different from the default value
- a couple of selected important configuration settings (even if they are still set to the default value), i.e.:
  - build path (see *Build path (–buildpath)*)
  - install path (see *Software and modules install path (–installpath, –installpath-software, –installpath-modules)*)
  - path to easyconfigs repository (see *Easyconfigs repository (–repository, –repositorypath)*)
  - the robot search path (see *Searching for easyconfigs: the robot search path*)
  - source path (see *Source path (–sourcepath)*)
- through which configuration type each setting was defined
  - i.e., default value, configuration file, environment variable or command line argument

Example output:

```
$ cat $HOME/.config/easybuild/config.cfg
[config]
buildpath = /tmp/eb-build

$ export EASYBUILD_MODULES_TOOL=Lmod
$ export EASYBUILD_OPTARCH=''

$ eb --show-config --installpath=$HOME/apps --job-cores=4
#
# Current EasyBuild configuration
# (C: command line argument, D: default value, E: environment variable, F: configuration file)
#
buildpath      (F) = /tmp/eb-build
installpath    (C) = /Users/example/apps
job-cores      (C) = 4
modules-tool   (E) = Lmod
optarch        (E) = ''
repositorypath (D) = /Users/example/.local/easybuild/ebfiles_repo
robot-paths    (D) = /Users/example/easybuild-easyconfigs/easybuild/easyconfigs
sourcepath     (D) = /Users/example/.local/easybuild/sources
```

For a full overview of the current configuration, including *all* configuration settings, see `eb --show-full-config`.

2.2.3 Available configuration settings

To obtain a full and up-to-date list of available configuration settings, see `eb --help`. We refrain from listing all available configuration settings here, to avoid outdated documentation.

A couple of selected configuration settings are discussed below, in particular the mandatory settings.

Mandatory configuration settings

A handful of configuration settings are **mandatory**, and should be provided using one of the supported configuration types.

The following configuration settings are currently mandatory (more details in the sections below):

- *Source path (–sourcepath)*
- *Build path (–buildpath)*
- *Software and modules install path (–installpath, –installpath-software, –installpath-modules)*
- *Easyconfigs repository (–repository, –repositorypath)*
- *Logfile format (–logfile-format)*

If any of these configuration settings is not provided in one way or another, EasyBuild will complain and exit.

In practice, all of these have reasonable defaults (see `eb --help` for the default settings).

Note: The mandatory path-related options can be tweaked collectively via `--prefix`, see *Overall prefix path (–prefix)* for more information.

Source path (`--sourcepath`)

default: `$HOME/.local/easybuild/sources/` (determined via *Overall prefix path (–prefix)*)

The `sourcepath` configuration setting specifies the parent path of the directory in which EasyBuild looks for software source and install files.

Looking for the files specified via the `sources` parameter in the .eb easyconfig file is done in the following order of preference:

- `<sourcepath>/<name>`: a subdirectory determined by the name of the software package
- `<sourcepath>/<letter>/<name>`: in the style of the `easyblocks`/`easyconfigs` directories: in a subdirectory determined by the first letter (in lower case) of the software package and by its full `name`
- `<sourcepath>`: directly in the source path

Note that these locations are also used when EasyBuild looks for patch files in addition to the various `easybuild/easyconfigs` directories that are listed in the $PYTHONPATH.

Build path (`--buildpath`)

default: `$HOME/.local/easybuild/build/` (determined via *Overall prefix path (–prefix)*)

The `buildpath` configuration setting specifies the parent path of the (temporary) directories in which EasyBuild builds its software packages.

Each software package is (by default) built in a subdirectory of the specified `buildpath` under `<name>/<version>/<toolchain><versionsuffix>`.

Note that the build directories are emptied and removed by EasyBuild when the installation is completed (by default).

Tip: Using `/dev/shm` as build path can significantly speed up builds, if it is available and provides a sufficient amount of space. Setting up the variable `EASYBUILD_BUILDPATH` in your shell startup files makes this default. However be aware that, fi., two parallel GCC builds may fill up `/dev/shm` !

Software and modules install path (`--installpath`, `--installpath-software`, `--installpath-modules`)

defaults:

- *software install path:* `$HOME/.local/easybuild/software` (determined via *Overall prefix path (–prefix)* and `--subdir-software`)
- *modules install path:* `$HOME/.local/easybuild/modules/all` (determined via *Overall prefix path (–prefix)*, `--subdir-modules` and `--suffix-modules-path`)

There are several ways in which the software and modules install path used by EasyBuild can be configured:

- using the direct configuration options `--installpath-software` and `--installpath-modules` (see below)
- via the parent install path configuration option `--installpath` (see below)
- via the overall prefix path configuration option `--prefix` (see *Overall prefix path (–prefix)*)

Direct options: `--installpath-software` and `--installpath-modules` *default*: *(no default specified)*

The `--installpath-software` and `--installpath-modules` configuration options (available since EasyBuild v2.1.0) allow to directly specify the software and modules install paths, respectively.

These configuration options have precedence over all of the other configuration options that relate to specifying the install path for software and/or modules (see below).

Parent install path: `--installpath` *default*: *(no default specified)*

The `--installpath` configuration option specifies the *parent* path of the directories in which EasyBuild should install software packages and the corresponding module files.

The install path for software and modules specifically is determined by combining `--installpath` with `--subdir-software`, and combining `--installpath` with `--subdir-modules` and `--suffix-modules-path`, respectively.

For more information on these companion configuration options, see *Software and modules install path subdirectories (–subdir-software, –subdir-modules, –suffix-modules-path)*.

Full install path for software and module file The full software and module install paths for a particular software package are determined by the active module naming scheme along with the general software and modules install paths specified by the EasyBuild configuration.

Both the software itself and the corresponding module file will be installed in a subdirectory of the corresponding install path named according to the active module naming scheme (default format: `<name>/<version>-<toolchain><versionsuffix>`). Additionally, symlinks to the actual module file are installed in a subdirectory of the modules install path named according to the value of the `moduleclass` easyconfig parameter.

For more information on the module naming scheme used by EasyBuild, see *Active module naming scheme (–module-naming-scheme)*.

Updating `$MODULEPATH` To make the modules generated by EasyBuild available, the `$MODULEPATH` environment variable must be updated to include the modules install path.

The recommended way to do this is to use the `module use` command. For example:

```
$ eb --installpath=$HOME/easybuild
$ module use $HOME/easybuild/modules/all
```

It is probably a good idea to add this to your (favourite) shell `.rc` file, e.g., `~/.bashrc`, and/or the `~/.profile` login scripts, so you do not need to adjust `$MODULEPATH` every time you start a new session.

Note: Updating `$MODULEPATH` is not required for EasyBuild itself, since `eb` updates `$MODULEPATH` itself at runtime according to the modules install path it is configured with.

Easyconfigs repository (`--repository`, `--repositorypath`)

default: `FileRepository` at `$HOME/.local/easybuild/ebfiles_repo` (determined via *Overall prefix path (–prefix)*)

EasyBuild has support for archiving (tested) `.eb` easyconfig files. After successfully installing a software package using EasyBuild, the corresponding `.eb` file is uploaded to a repository defined by the `repository` and `repositorypath` configuration settings.

Currently, EasyBuild supports the following repository types (see also `eb --avail-repositories`):

- `FileRepository('path', 'subdir')`: a plain flat file repository; `path` is the path where files will be stored, `subdir` is an *optional* subdirectory of that path where the files should be stored
- `GitRepository('path', 'subdir/in/repo'`: a *non-empty* **bare** git repository (created with `git init --bare` or `git clone --bare`); `path` is the path to the git repository (can also be a URL); `subdir/in/repo` is optional, and specifies a subdirectory of the repository where files should be stored in
- `SvnRepository('path', 'subdir/in/repo')`: an SVN repository; `path` contains the subversion repository location (directory or URL), the optional second value specifies a subdirectory in the repository

You need to set the `repository` setting inside a configuration file like this:

```
[config]
repository = FileRepository
repositorypath = <path>
```

Or, optionally an extra argument representing a subdirectory can be specified, e.g.:

```
$ export EASYBUILD_REPOSITORY=GitRepository
$ export EASYBUILD_REPOSITORYPATH=<path>,<subdir>
```

You do not have to worry about importing these classes, EasyBuild will make them available to the configuration file.

Using `git` requires the `GitPython` Python modules, using `svn` requires the `pysvn` Python module (see *Dependencies*).

If access to the easyconfigs repository fails for some reason (e.g., no network or a missing required Python module), EasyBuild will issue a warning. The software package will still be installed, but the (successful) easyconfig will not be automatically added to the archive (i.e., it is not considered a fatal error).

Logfile format (`--logfile-format`)

default: `easybuild, easybuild-%(name)s-%(version)s-%(date)s.%(time)s.log`

The *logfile format* configuration setting contains a tuple specifying a log directory name and a template log file name. In both of these values, using the following string templates is supported:

- `%(name)s`: the name of the software package to install
- `%(version)s`: the version of the software package to install
- `%(date)s`: the date on which the installation was performed (in `YYYYMMDD` format, e.g. `20120324`)
- `%(time)s`: the time at which the installation was started (in `HHMMSS` format, e.g. `214359`)

Note: Because templating is supported in configuration files themselves (see *Templates and constants supported in configuration files*), the '%' character in these template values must be escaped when used in a configuration file (and only then), e.g., '%%(name)s'. Without escaping, an error like `InterpolationMissingOptionError: Bad value substitution` will be thrown by `ConfigParser`.

For example, configuring EasyBuild to generate a log file mentioning only the software name in a directory named `easybuild` can be done via the `--logfile-format` command line option:

```
eb --logfile-format="easybuild,easybuild-%(name)s.log" ...
```

or the `$EASYBUILD_LOGFILE_FORMAT` environment variable:

```
export EASYBUILD_LOGFILE_FORMAT="easybuild,easybuild-%(name)s.log"
```

or by including the following in an EasyBuild configuration file (note the use of '%%' to escape the `name` template value here):

```
logfile-format = easybuild,easybuild-%%(name)s.log
```

Optional configuration settings

The subsections below discuss a couple of commonly used optional configuration settings.

Overall prefix path (`--prefix`)

default: `$HOME/.local/easybuild`

The overall prefix path used by EasyBuild can be specified using the `--prefix` configuration option.

This affects the default value of several configuration options:

- source path (see *Source path (–sourcepath)*)
- build path (see *Build path (–buildpath)*)
- software and modules install path (see *Software and modules install path (–installpath, –installpath-software, –installpath-modules)*)
- easyconfigs repository path (see *Easyconfigs repository (–repository, –repositorypath)*)

Software and modules install path subdirectories (`--subdir-software`, `--subdir-modules`, `--suffix-modules-path`)

defaults:

- *software install path subdirectory* (`--subdir-software`): `software`
- *modules install path subdirectory* (`--subdir-modules`): `modules`
- *modules install path suffix* (`--suffix-modules-path`): `all`

The subdirectories for the software and modules install paths (relative to `--installpath`, see *Software and modules install path (–installpath, –installpath-software, –installpath-modules)*) can be specified using the corresponding dedicated configuration options (available since EasyBuild v1.14.0).

For example:

```
$ export EASYBUILD_SUBDIR_SOFTWARE=installs
$ eb --installpath=$HOME/easybuild --subdir-modules=module_files ...
```

Modules tool (`--modules-tool`)

default: `EnvironmentModulesC`

Specifying the modules tool that should be used by EasyBuild can be done using the `modules-tool` configuration setting. A list of supported modules tools can be obtained using `eb --avail-modules-tools`.

Currently, the following modules tools are supported:

- `EnvironmentModulesC`: Tcl/C version of environment modules (`modulecmd`)
- `EnvironmentModulesTcl`: Tcl-only version of environment modules (`modulecmd.tcl`)
- `Lmod`: Lmod, an modern alternative to environment modules, written in Lua (`lmod`)

You can determine which modules tool you are using by checking the output of `type -f module` (in a `bash` shell), or `alias module` (in a `tcsh` shell).

The actual module command (i.e., `modulecmd`, `modulecmd.tcl`, `lmod`, ...) must be available via `$PATH` (which is not standard).

For example, to indicate that EasyBuild should be using `Lmod` as modules tool:

```
$ eb --modules-tool=Lmod ...
```

Active module naming scheme (`--module-naming-scheme`)

default: `EasyBuildModuleNamingScheme`

The module naming scheme that should be used by EasyBuild can be specified using the `module-naming-scheme` configuration setting.

```
$ eb --module-naming-scheme=HierarchicalMNS ...
```

For more details, see the dedicated page: https://github.com/hpcugent/easybuild/wiki/Using-a-custom-module-naming-scheme .

Module files syntax (`--module-syntax`)

default: `Tcl`

supported since: EasyBuild v2.1

The syntax to use for generated module files can be specified using the `--module-syntax` configuration setting.

Possible values are:

- `Lua`: generate module files in Lua syntax
  - this requires the use of Lmod as a modules tool to consume the module files (see *Modules tool (–modules-tool)*)
  - module file names will have the `.lua` extension
- `Tcl`: generate module files in Tcl syntax
  - Tcl module files can be consumed by all supported modules tools
  - module files will contain a header string `#%Module` indicating that they are composed in Tcl syntax

Note: Lmod is able to deal with having module files in place in both Tcl and Lua syntax. When a module file in Lua syntax (i.e., with a `.lua` file name extension) is available, a Tcl module file with the same name will be

ignored. The Tcl-based environment modules tool will simply ignore module files in Lua syntax, since they do not contain the header string that is included in Tcl module files.

Note: Using module files in Lua syntax has the advantage that Lmod does not need to translate from Lua to Tcl internally when processing the module files, which benefits responsiveness of Lmod when used interactively by users. In terms of Lmod-specific aspects of module files, the syntax of the module file does *not* matter; Lmod-specific statements supported by EasyBuild can be included in Tcl module files as well, by guarding them by a condition that only evaluates positively when Lmod is consuming the module file, i.e. ‘`if { [ string match "*tcl2lua.tcl" $env(_) ] } { ... }`‘. Only conditional load statements like ‘`load(atleast("gcc","4.8"))`‘ can only be used in Lua module files.

CHAPTER 3

Basic usage topics

3.1 Using the EasyBuild command line

Basic usage of EasyBuild is described in the following sections, covering the most important range of topics if you are new to EasyBuild.

`eb` is EasyBuild's main command line tool, to interact with the EasyBuild framework and hereby the most common command line options are being documented.

3.1.1 Specifying builds

To instruct EasyBuild which software packages it should build/install and which build parameters it should use, one or more *easyconfig files* (see *Easyconfig files*) must be specified.

This can be done in a number of ways:

- *By providing a single easyconfig file*
- *Via command line options*
- *By providing a set of easyconfig files*
- from_pr

Whenever EasyBuild searches for *explicitely specified* easyconfig files, it considers a couple of locations, i.e. (in order of preference):

1. the local working directory
2. the robot search path (see *Searching for easyconfigs: the robot search path*)
3. the path to easyconfig files that are part of the active EasyBuild installation (which is included in the default robot search path)

These locations are only considered for easyconfig files that are specified only by filename or using a relative path, *not* for easyconfig files that are specified via an absolute path. The dependencies are resolved using the robot search path (see *Searching for easyconfigs: the robot search path*).

Note: For easyconfig files specified on the `eb` command line, the *full* robot search path is only considered since EasyBuild v2.0.0. Earlier versions only considered the local working directory and the easyconfig files that are part of the active EasyBuild installation for *explicitely specified* easyconfig files.

By providing a single easyconfig file

The most basic usage is to simply provide the name of an easyconfig file to the `eb` command. EasyBuild will (try and) locate the easyconfig file, and perform the installation as specified by that easyconfig file.

For example, to build and install `HPL` using the `goolf` toolchain:

```
$ eb HPL-2.0-goolf-1.4.10.eb --robot
[...]
== building and installing GCC/4.7.2...
[...]
== building and installing goolf/1.4.10...
[...]
== building and installing HPL/2.0-goolf-1.4.10...
== fetching files...
== creating build dir, resetting environment...
== unpacking...
== patching...
== preparing...
== configuring...
== building...
== testing...
== installing...
== taking care of extensions...
== packaging...
== postprocessing...
== sanity checking...
== cleaning up...
== creating module...
== COMPLETED: Installation ended successfully
== Results of the build can be found in the log file /home/example/.local/easybuild/software/HPL/
== Build succeeded for 9 out of 9
== temporary log file /tmp/easybuild-UOEWix/easybuild-NiswcV.log has been removed.
== temporary directory /tmp/easybuild-UOEWix has been removed.
```

Then, we can actually load the freshly installed HPL module:

```
$ module load HPL/2.0-goolf-1.4.10
$ which xhpl
/home/example/.local/easybuild/software/HPL/2.0-goolf-1.4.10/bin/xhpl
```

All easyconfig file names' suffixes are `.eb` and follow format:

```
``<name>-<version>-<toolchain>-<versionsuffix>``
```

This is a crucial design aspect, since the dependency resolution mechanism (see *Enabling dependency resolution, –robot / -r and –robot-paths*) relies upon this convention.

Tip: You may wish to modify the installation prefix (e.g., using `--prefix` or by defining `$EASYBUILD_PREFIX`), in order to redefine the build/install/source path prefix to be used; default value is: `$HOME/.local/easybuild`.

Via command line options

An alternative approach is to only use command line options to specify which software to build. Refer to the `Software search and build options` section in the `eb --help` output for an overview of the available command line options for this purpose (cfr. basic_usage_help).

Here is how to build and install last version of HPCG (that EasyBuild is aware of) using the `goolf/1.4.10` toolchain:

```
$ eb --software-name=HPCG --toolchain=goolf,1.4.10
[...]
== building and installing HPCG/2.1-goolf-1.4.10...
[...]
```

```
== COMPLETED: Installation ended successfully
[...]
```

At this point, a module `HPCG/2.1-goolf-1.4.10` should have been installed.

By providing a set of easyconfig files

Multiple easyconfig files can be provided as well, either directly or by specifying a directory that contains easyconfig files.

For example, to build and install both HPCG and GCC with a single command, simply list the easyconfigs for both on the `eb` command line (note that HPCG is not being reinstalled, since a matching module is already available):

```
$ eb HPCG-2.1-goolf-1.4.10.eb GCC-4.8.3.eb
[...]
== HPCG/2.1-goolf-1.4.10 is already installed (module found), skipping
[...]
== building and installing GCC/4.8.3...
[...]
== Build succeeded for 1 out of 1
[...]
```

When one or more directories are provided, EasyBuild will (recursively) traverse them to find easyconfig files. For example:

```
$ find set_of_easyconfigs/ -type f
set_of_easyconfigs/GCC-4.8.3.eb
set_of_easyconfigs/foo.txt
set_of_easyconfigs/tools/HPCG-2.1-goolf-1.4.10.eb
```

```
$ eb set_of_easyconfigs/
== temporary log file in case of crash /tmp/easybuild-1yxCvv/easybuild-NeNmZr.log
== HPCG/2.1-goolf-1.4.10 is already installed (module found), skipping
== GCC/4.8.3 is already installed (module found), skipping
== No easyconfigs left to be built.
== Build succeeded for 0 out of 0
== temporary log file /tmp/easybuild-1yxCvv/easybuild-NeNmZr.log has been removed.
== temporary directory /tmp/easybuild-1yxCvv has been removed.
```

Note: EasyBuild will only pick up the files which end with `.eb` ; anything else will be ignored.

Tip: Calling EasyBuild is designed as an *idempotent* operation; if a module is available that matches with an provided easyconfig file, the installation will simply be skipped.

3.1.2 Commonly used command line options

Command line help, `--help` / `-H`

Detailed information about the usage of the eb command is available via the `--help`, `-H`, `-h` help options.

Refer to page basic_usage_help for more detailed information.

Note: `--help` / `-H` spit out the long help info (i.e. including long option names), `-h` only includes short option names.

Tip: This is the best way to query for certain information, esp. recent features, since this is in sync with the actual EasyBuild version being used.

Report version, `--version`

You can query which EasyBuild version you are using with `--version`:

```
$ eb --version
This is EasyBuild 1.15.2 (framework: 1.15.2, easyblocks: 1.15.2) on host example.local.
```

Tip: Asking EasyBuild to print own its version is a quick way to ensure that `$PYTHONPATH` is set up correctly, so that the entire EasyBuild installation (framework and easyblocks) is available.

List of known toolchains, `--list-toolchains`

For an overview of known toolchains, use `eb --list-toolchains`.

Toolchains have brief mnemonic names, for example:

- `goolf` stands for `GCC, OpenMPI, OpenBLAS/LAPACK, FFTW and ScaLAPACK`
- `iimpi` stands for `icc/ifort, impi`
- `cgmvolf` stands for `Clang/GCC, MVAPICH2, OpenBLAS/LAPACK, FFTW`

The complete table of available toolchains is available at *List of known toolchains*.

List of available easyblocks, `--list-easyblocks`

You can obtain a list of available *Easyblocks* via `--list-easyblocks`.

The `--list-easyblocks` command line option prints the easyblocks in a hierarchical way, showing the inheritance patterns, with the "base" easyblock class `EasyBlock` on top.

Software-specific easyblocks have a name that starts with `EB_`; the ones that do not are generic easyblocks. (cfr. *Easyblocks* for the distinction between both types of easyblocks).

For example, a list of easyblocks can be obtained with:

```
$ eb --list-easyblocks
```

To see more details about the available easyblocks, i.e., in which Python module the class is defined, and where it is located, use `--list-easyblocks=detailed`.

Refer to page *List of easyblocks* for more information.

All available easyconfig parameters, `--avail-easyconfig-params` / `-a`

EasyBuild provides a significant amount of easyconfig parameters. An overview of all available easyconfig parameters can be obtained via `eb --avail-easyconfig-params`, or `eb -a` for short.

Refer to page easyconfigs_parameters for more information, the possible parameters are a very rich set.

Combine -a with `--easyblock/-e` to include parameters that are specific to a particular easyblock. For example:

```
$ eb -a -e EB_WRF
```

If you want to see the full output of running this command, look at Available easyconfig parameters for EB_WRF.

Enable debug logging, `--debug` / `-d`

Use `eb --debug/-d` to enable debug logging, to include all details of how EasyBuild performed a build in the log file:

```
$ eb HPCG-2.1-goolf-1.4.10.eb -d
```

Tip: You may enable this by default via adding `debug = True` in your EasyBuild configuration file

Note: Debug log files are significantly larger than non-debug logs, so be aware.

Forced reinstallation, `--force` / `-f`

Use `eb --force/-f` to force the reinstallation of a given easyconfig/module.

Warning: Use with care, since the reinstallation of existing modules will be done without requesting confirmation first!

Tip: Combine `--force` with `--dry-run` to get a good view on which installations will be forced. (cfr. *Get an overview of planned installations –dry-run / -D*)

Searching for easyconfigs, `--search` / `-S`

Searching for available easyconfig files can be done using the `--search` (long output) and `-S` (short output) command line options. All easyconfig files available in the robot search path are considered (see *Searching for easyconfigs: the robot search path*), and searching is done *case-insensitive*.

For example, to see which easyconfig files are available for the software package named *Mesquite*:

```
$ eb --search mesquite
== temporary log file in case of crash /tmp/eb-_TYdTf/easybuild-iRJ2vb.log
== Searching (case-insensitive) for 'mesquite' in /home/example/easybuild-easyconfigs/easybuild/ea
 * /Users/kehoste/work/easybuild-easyconfigs/easybuild/easyconfigs/m/Mesquite/Mesquite-2.3.0-gool
 * /Users/kehoste/work/easybuild-easyconfigs/easybuild/easyconfigs/m/Mesquite/Mesquite-2.3.0-ictce
 * /Users/kehoste/work/easybuild-easyconfigs/easybuild/easyconfigs/m/Mesquite/Mesquite-2.3.0-ictce
== temporary log file(s) /tmp/eb-_TYdTf/easybuild-iRJ2vb.log* have been removed.
== temporary directory /tmp/eb-_TYdTf has been removed.
```

The same query with `-S` is more readable, when there is a joint path that can be collapsed to a variable like `$CFGS1`:

```
$ eb -S mesquite
== temporary log file in case of crash /tmp/eb-5diJjn/easybuild-nUXlkj.log
== Searching (case-insensitive) for 'mesquite' in /home/example/easybuild-easyconfigs/easybuild/ea
CFGS1=/home/example/easybuild-easyconfigs/easybuild/easyconfigs/m/Mesquite
 * $CFGS1/Mesquite-2.3.0-goolf-1.4.10.eb
 * $CFGS1/Mesquite-2.3.0-ictce-4.1.13.eb
 * $CFGS1/Mesquite-2.3.0-ictce-5.3.0.eb
== temporary log file(s) /tmp/eb-5diJjn/easybuild-nUXlkj.log* have been removed.
== temporary directory /tmp/eb-5diJjn has been removed.
```

For more specific searching, a regular expression pattern can be supplied (since EasyBuild v2.1.1).

For example, to search which easyconfig files are available for GCC v4.6.x, without listing easyconfig files that use GCC v4.6.x as a toolchain:

```
$ eb -S ^GCC-4.6
== temporary log file in case of crash /tmp/eb-PpwTwm/easybuild-b8yrOG.log
== Searching (case-insensitive) for '^GCC-4.6' in /home/example/easybuild-easyconfigs/easybuild/ea
CFGS1=/home/example/easybuild-easyconfigs/easybuild/easyconfigs/g/GCC
 * $CFGS1/GCC-4.6.3-CLooG-PPL.eb
 * $CFGS1/GCC-4.6.3.eb
 * $CFGS1/GCC-4.6.4.eb
== temporary log file(s) /tmp/eb-PpwTwm/easybuild-b8yrOG.log* have been removed.
== temporary directory /tmp/eb-PpwTwm has been removed.
```

Or, to find all easyconfig files for Python versions 2.7.8 and 2.7.9 that use the `intel` toolchain:

```
$ eb -S '^Python-2.7.[89].*intel'
== temporary log file in case of crash /tmp/eb-Dv5LEJ/easybuild-xpGGSF.log
== Searching (case-insensitive) for '^Python-2.7.[89].*intel' in /home/example/easybuild-easyconfi
CFGS1=/home/example/easybuild-easyconfigs/easybuild/easyconfigs/p/Python
 * $CFGS1/Python-2.7.8-intel-2014.06.eb
 * $CFGS1/Python-2.7.8-intel-2014b.eb
 * $CFGS1/Python-2.7.8-intel-2015a.eb
 * $CFGS1/Python-2.7.9-intel-2015a-bare.eb
 * $CFGS1/Python-2.7.9-intel-2015a.eb
== temporary log file(s) /tmp/eb-Dv5LEJ/easybuild-xpGGSF.log* have been removed.
== temporary directory /tmp/eb-Dv5LEJ has been removed.
```

Note: Prior to EasyBuild v2.1.1, the full path to easyconfig files was considered when matching the search pattern. Starting with EasyBuild v2.1.1, only the filename of the easyconfig file itself is taken into account.

Enabling dependency resolution, `--robot` / `-r` and `--robot-paths`

EasyBuild supports installing an entire software stack, including the required toolchain if needed, with a single `eb` invocation.

To enable dependency resolution, use the `--robot` command line option (or `-r` for short):

```
$ eb mpiBLAST-1.6.0-goolf-1.4.10.eb --robot
[...]
== building and installing GCC/4.7.2...
[...]
== building and installing hwloc/1.6.2-GCC-4.7.2...
[...]
== building and installing OpenMPI/1.6.4-GCC-4.7.2...
[...]
== building and installing gompi/1.4.10...
[...]
== building and installing OpenBLAS/0.2.6-gompi-1.4.10-LAPACK-3.4.2...
[...]
== building and installing FFTW/3.3.3-gompi-1.4.10...
[...]
== building and installing ScaLAPACK/2.0.2-gompi-1.4.10-OpenBLAS-0.2.6-LAPACK-3.4.2...
[...]
== building and installing goolf/1.4.10...
[...]
== building and installing mpiBLAST/1.6.0-goolf-1.4.10...
[...]
== Build succeeded for 9 out of 9
```

The dependency resolution mechanism will construct a full dependency graph for the software package(s) being installed, after which a list of dependencies is composed for which no module is available yet. Each of the retained dependencies will then be built and installed, in the required order as indicated by the dependency graph.

Tip: Using `--robot` is particularly useful for software packages that have an extensive list of dependencies, or when reinstalling software using a different compiler toolchain (you can use the `--try-toolchain` command line option in combination with `--robot`).

Note: Unless dependency resolution is enabled, EasyBuid requires that modules are available for every dependency. If `--robot` is not used and one or more modules are missing, `eb` will exit with an error stating that a module for a particular dependency could not be found. For example:

```
add_dependencies: no module 'GCC/4.7.2' found for dependency {...}
```

Searching for easyconfigs: the robot search path

For each dependency that does not have a matching module installed yet, EasyBuild requires a corresponding easyconfig file. If no such easyconfig file was specified on the `eb` command line, the dependency resolution mechanism will try to locate one in the *robot search path*.

Searching for easyconfigs is done based on filename (see also *What is an easyconfig (file)?*), with filenames being derived from the dependency specification (i.e. software name/version, toolchain and version suffix). For each entry in the robot search path, a couple of different filepaths are considered, mostly determined by the software name.

For example, when looking for an easyconfig for `OpenMPI` version `1.6.4` and version suffix `-test` with toolchain `GCC/4.7.2`, the following filepaths are considered (relative to each entry in the robot search path):

- `OpenMPI/1.6.4-GCC-4.7.2-test.eb`
- `OpenMPI/OpenMPI-1.6.4-GCC-4.7.2-test.eb`
- `o/OpenMPI/OpenMPI-1.6.4-GCC-4.7.2-test.eb`
- `OpenMPI-1.6.4-GCC-4.7.2-test.eb`

Note: Sometimes easyconfig files are needed even when the modules for the dependencies are already available, i.e., whenever the information provided by the dependency specification (software name/version, toolchain and version suffix) is not sufficient. This is the case when using `--dry-run` to construct the complete dependency graph, or when the active module naming scheme requires some additional information (e.g., the `moduleclass`).

Note: If EasyBuild is unable to locate required easyconfigs, an appropriate error message will be shown. For example:

```
Irresolvable dependencies encountered: GCC/4.7.2
```

or:

```
Failed to find easyconfig file 'GCC-4.7.2.eb' when determining module name for: {...}
```

Default robot search path By default, EasyBuild will only include the collection of easyconfig files that is part of the EasyBuild installation in the robot search path. More specifically, only directories listed in the *Python search path* (partially specified by the `$PYTHONPATH` environment variable) that contain a subdirectory named `easybuild/easyconfigs` are considered part of the robot search path (in the order they are encountered).

Controlling the robot search path

To control the robot search path, you can specify a (colon-separated list of) path(s) to `--robot`/`-r` and/or `--robot-paths` (or, equivalently, `$EASYBUILD_ROBOT` and/or `$EASYBUILD_ROBOT_PATHS`):

```
eb --robot=$PWD:$HOME ...
```

These two configuration options each serve a particular purpose, and together define the robot search path:

- `--robot`, `-r`:
  - intended to be used (only) as a command line option to `eb` (although it can also be defined through another configuration type)
  - enables the dependency resolution mechanism (disabled by default)
  - optionally a list of paths can be specified, which is included *first* in the robot search path
  - by default, the corresponding list of paths is *empty*
- `--robot-paths`:
  - intended to be defined in an EasyBuild configuration file, or via `$EASYBUILD_ROBOT_PATHS`
  - does *not* enable the dependency resolution mechanism
  - the specified list of paths is included *last* in the robot search path
  - by default, only the path to the easyconfig files that are part of the EasyBuild installation is listed
  - **note**: setting this configuration option implies redefining the default robot search path, unless a prepending/appending list of paths is specified, see *Prepending and/or appending to the default robot search path*

For both options, the list of paths should be specified as a colon-separated (`:`) list.

By combining `--robot` and `--robot-paths` using the different configuration types (see also *Supported configuration types*), you have full control over the robot search path: which paths are included, the order of those paths, whether or not the easyconfig files that are part of the EasyBuild installation should be considered, etc.

A constant named `DEFAULT_ROBOT_PATHS` is available that can be used (only) in EasyBuild configuration files to refer to the default robot search path, i.e. the path to the easyconfigs that are part of the EasyBuild installation. For more information on using constants in EasyBuild configuration files, see *Templates and constants supported in configuration files*.

Tip: Only use `--robot` to enable the dependency resolution mechanism; define `robot-paths` in your EasyBuild configuration file or via `$EASYBUILD_ROBOT_PATHS` to specify which sets of easyconfig files EasyBuild should consider, and in which order. By means of exception, a path can be specified to `--robot` to give a specific set of easyconfig files precedence over others, for example when testing modified easyconfig files.

Note: The paths specified on the configuration type with the highest order of preference *replace* any paths specified otherwise, i.e. values are not retained across configuration types. That is: `--robot` *overrides* the value in `$EASYBUILD_ROBOT`, `$EASYBUILD_ROBOT_PATHS` *overrides* the `robot-paths` specification in an EasyBuild configuration file, etc. Of course, the value specified for `--robot` does not directly affect the value specified for `--robot-paths`, on any configuration level, and vice versa. For more information on the relation between the different configuration types, see *Supported configuration types*.

Prepending and/or appending to the default robot search path Prepending or appending to the default robot search path is supported via the `--robot-paths` configuration option.

To *prepend* one or more paths, a list of paths followed by a '`:`' should be specified.

Analogously, to *append* one or more paths, a list of paths preceded by a '`:`' should be specified.

For example:

- `export EASYBUILD_ROBOT_PATHS=/tmp/$USER`: specifies to prepend `/tmp/$USER` to the robot search path
- `--robot-paths :$HOME/eb:$HOME/test` specifies to append `$HOME/eb` and `$HOME/test` to the robot search path (in that order)
- `--robot-paths=/tmp/$USER::$HOME/test` specifies to prepend `/tmp/$USER` *and* append `$HOME/test` to the robot search path

Example use case For example, say we want to configure EasyBuild to behave as follows w.r.t. the robot search path:

- (always) prefer easyconfig files in the archive/repository over the ones that are included in the EasyBuild installation (i)
- consider easyconfig files located in the current directory or home directory first (in that order), before any others (ii)

Matching setup:

- satisfy (i) using `robot-paths` in one of the active EasyBuild configuration files (see also *List of used configuration files*):

```
robot-paths = %(repositorypath)s:%(DEFAULT_ROBOT_PATHS)s
```

- satisfy (ii) via `--robot` on the `eb` command line:

```
eb mpiBLAST-1.6.0-goolf-1.4.10.eb --robot $PWD:$HOME
```

Get an overview of planned installations `--dry-run` / `-D`

You can do a "dry-run" overview by supplying `-D`/`--dry-run` (typically combined with `--robot`, in the form of `-Dr`):

```
$ eb mpiBLAST-1.6.0-goolf-1.4.10.eb -Dr
== temporary log file in case of crash /tmp/easybuild-vyNQhw/easybuild-pO8EJv.log
Dry run: printing build status of easyconfigs and dependencies
CFGS=/home/example/.local/easybuild/software/EasyBuild/1.15.2/lib/python2.7/site-packages/easybui
 * [*] $CFGS/g/GCC/GCC-4.7.2.eb (module: GCC/4.7.2)
 * [*] $CFGS/h/hwloc/hwloc-1.6.2-GCC-4.7.2.eb (module: hwloc/1.6.2-GCC-4.7.2)
 * [*] $CFGS/o/OpenMPI/OpenMPI-1.6.4-GCC-4.7.2.eb (module: OpenMPI/1.6.4-GCC-4.7.2)
 * [*] $CFGS/g/gompi/gompi-1.4.10.eb (module: gompi/1.4.10)
 * [ ] $CFGS/o/OpenBLAS/OpenBLAS-0.2.6-gompi-1.4.10-LAPACK-3.4.2.eb (module: OpenBLAS/0.2.6-gompi
 * [ ] $CFGS/f/FFTW/FFTW-3.3.3-gompi-1.4.10.eb (module: FFTW/3.3.3-gompi-1.4.10)
 * [ ] $CFGS/s/ScaLAPACK/ScaLAPACK-2.0.2-gompi-1.4.10-OpenBLAS-0.2.6-LAPACK-3.4.2.eb (module: Scal
 * [ ] $CFGS/g/goolf/goolf-1.4.10.eb (module: goolf/1.4.10)
 * [ ] $CFGS/m/mpiBLAST/mpiBLAST-1.6.0-goolf-1.4.10.eb (module: mpiBLAST/1.6.0-goolf-1.4.10)
== temporary log file /tmp/easybuild-vyNQhw/easybuild-pO8EJv.log has been removed.
== temporary directory /tmp/easybuild-vyNQhw has been removed.
```

Note how the different status symbols denote distinct handling states by EasyBuild:

- `[ ]` The build is not available, EasyBuild will deliver it
- `[x]` The build is available, EasyBuild will skip building this module
- `[F]` The build is available, however EasyBuild has been asked to force a rebuild and will do so

Note: Since EasyBuild v2.4.0, a detailed overview of the build and install procedure that EasyBuild will be execute can be obtained using `--extended-dry-run` or `-x`, see *Extended dry run*.

Tweaking existing easyconfig files, using `--try-*`

Making minor changes to existing easyconfig files can be done straight from the `eb` command line. This way, having to manually copying and editing easyconfig files can be avoided.

Tweaking existing easyconfig files can be done using the **–try-*** command line options. For each of the software build options that can be used as an alternative to specifying easyconfig file names, a matching `--try-X` command line options is available:

- `--try-toolchain` to try using the toolchain with the given name and version
  - format: `--try-toolchain=<name>,<version>`
  - `--try-toolchain-name` to try using the latest toolchain version of a toolchain
  - `--try-toolchain-version` to try using a different toolchain version
- `--try-software-version` to try building a different software version
- `--try-amend` to try tweaking a different easyconfig parameter
  - format: `--try-amend=<param>=<value>`
  - only supports string and list-of-strings value types

For example, to build and install WRF and its dependencies with a different toolchain version:

```
$ eb WRF-3.5.1-goolf-1.4.10-dmpar.eb --try-toolchain-version=1.5.14 -Dr
== temporary log file in case of crash /tmp/easybuild-Y9WApt/easybuild-VmPiOH.log
Dry run: printing build status of easyconfigs and dependencies
 * [x] /home/example/work/easybuild-easyconfigs/easybuild/easyconfigs/g/GCC/GCC-4.8.2.eb (module:
 * [x] /home/example/work/easybuild-easyconfigs/easybuild/easyconfigs/h/hwloc/hwloc-1.8.1-GCC-4.8
 * [x] /home/example/work/easybuild-easyconfigs/easybuild/easyconfigs/o/OpenMPI/OpenMPI-1.6.5-GCC-
 * [x] /home/example/work/easybuild-easyconfigs/easybuild/easyconfigs/g/gompi/gompi-1.5.14.eb (mo
 * [x] /home/example/work/easybuild-easyconfigs/easybuild/easyconfigs/o/OpenBLAS/OpenBLAS-0.2.8-g
 * [x] /home/example/work/easybuild-easyconfigs/easybuild/easyconfigs/f/FFTW/FFTW-3.3.4-gompi-1.5
 * [x] /home/example/work/easybuild-easyconfigs/easybuild/easyconfigs/s/ScaLAPACK/ScaLAPACK-2.0.2
 * [x] /home/example/work/easybuild-easyconfigs/easybuild/easyconfigs/g/goolf/goolf-1.5.14.eb (mo
 * [ ] /tmp/easybuild-Y9WApt/tweaked_easyconfigs/zlib-1.2.7-goolf-1.5.14.eb (module: zlib/1.2.7-g
 * [ ] /tmp/easybuild-Y9WApt/tweaked_easyconfigs/Szip-2.1-goolf-1.5.14.eb (module: Szip/2.1-goolf
 * [ ] /tmp/easybuild-Y9WApt/tweaked_easyconfigs/ncurses-5.9-goolf-1.5.14.eb (module: ncurses/5.9
 * [ ] /tmp/easybuild-Y9WApt/tweaked_easyconfigs/flex-2.5.37-goolf-1.5.14.eb (module: flex/2.5.37
 * [ ] /tmp/easybuild-Y9WApt/tweaked_easyconfigs/M4-1.4.16-goolf-1.5.14.eb (module: M4/1.4.16-goo
 * [ ] /tmp/easybuild-Y9WApt/tweaked_easyconfigs/JasPer-1.900.1-goolf-1.5.14.eb (module: JasPer/1
 * [ ] /tmp/easybuild-Y9WApt/tweaked_easyconfigs/HDF5-1.8.10-patch1-goolf-1.5.14.eb (module: HDF5
 * [ ] /tmp/easybuild-Y9WApt/tweaked_easyconfigs/tcsh-6.18.01-goolf-1.5.14.eb (module: tcsh/6.18.
 * [ ] /tmp/easybuild-Y9WApt/tweaked_easyconfigs/Bison-2.7-goolf-1.5.14.eb (module: Bison/2.7-goo
 * [ ] /tmp/easybuild-Y9WApt/tweaked_easyconfigs/Doxygen-1.8.3.1-goolf-1.5.14.eb (module: Doxygen
 * [ ] /tmp/easybuild-Y9WApt/tweaked_easyconfigs/netCDF-4.2.1.1-goolf-1.5.14.eb (module: netCDF/4
 * [ ] /tmp/easybuild-Y9WApt/tweaked_easyconfigs/netCDF-Fortran-4.2-goolf-1.5.14.eb (module: netC
 * [ ] /tmp/easybuild-Y9WApt/tweaked_easyconfigs/WRF-3.5.1-goolf-1.5.14-dmpar.eb (module: WRF/3.5
== temporary log file /tmp/easybuild-Y9WApt/easybuild-VmPiOH.log has been removed.
== temporary directory /tmp/easybuild-Y9WApt has been removed.
```

Note: The `--try-*` command line options behave as expected when combined with `--robot`. For example: a modified toolchain specified via `--try-toolchain` only trickles down until the toolchain level (not deeper). This makes for a particularly powerful combo for rebuilding entire software stacks using a different toolchain.

Note: Modifying the software version does **not** trickle down the entire software stack, even when combined with `--robot`, since a software version is tied to a particular software package.

3.2 Writing easyconfig files: the basics

This page explains all the basic information about how to write easyconfig files.

For software builds that follow established build patterns, an easyconfig is all that you need to create in order to build and install the software and the corresponding module file.

Luckily, the majority of software delivery mechanisms are being designed around either autotools or CMake or, perhaps, some simple file extraction/copy pattern. Yet, in case the software build calls for more elaborate steps (scientific software never fails to surprise us in this regard), an *easyblock* may be needed, which is the subject of other part of this documentation.

3.2.1 What is an easyconfig (file)?

An easyconfig file serves as a *build specification* for EasyBuild.

It consists of a plain text file (in Python syntax) with mostly *key-value* assignment to define **easyconfig parameters**.

Easyconfigs typically follow a (fixed) strict naming scheme, i.e. `<name>-<version>[-<toolchain>][<versionsuffix>`

The `-<toolchain>` label (which includes the toolchain name and version) is omitted when a *dummy toolchain* is used. The `<versionsuffix>` label is omitted when the version suffix is empty.

Note: the filename of an easyconfig is only important w.r.t. dependency resolution (`--robot`), see *Enabling dependency resolution, –robot / -r and –robot-paths*.

Example:

```
# easyconfig file for GCC v4.8.3
name = 'GCC'
version = '4.8.3'
...
```

Tip: Comments can be included in easyconfig files using the hash (`#`) character (just like in Python code).

3.2.2 Available easyconfig parameters

About 50 different (generic) easyconfig parameters are supported currently. An overview of all available easyconfig parameters is available via the `-a` command line option.

By default, the parameters specific to generic (default) easyblock `ConfigureMake` are included; using `--easyblock`/`-e` parameters that are specific to a particular easyblock can be consulted.

See *All available easyconfig parameters, –avail-easyconfig-params / -a* for more details.

Example:

```
$ eb -a -e Binary
Available easyconfig parameters (* indicates specific for the Binary EasyBlock)
MANDATORY
---------
[..]
name:            Name of software (default: None)
[...]
EASYBLOCK-SPECIFIC
------------------
install_cmd(*):     Install command to be used. (default: None)
[...]
```

3.2.3 Mandatory easyconfig parameters

A handful of easyconfig parameters are *mandatory*:

- **name, version**: specify what software (version) to build
- **homepage, description**: metadata (used for module help)
- **toolchain**: specifies name and version of compiler toolchain to use
  - format: dictionary with name/version keys, e.g., `{'name': 'foo', 'version': '1.2.3'}`
  - a list of supported toolchains can be found at toolchain_table

Remarks:

- some others are planned to be required in the future
  - *docurls, software license, software license urls*

Example:

```
name = 'HPL'
version = '2.0'

homepage = 'http://www.netlib.org/benchmark/hpl/'
description = "High Performance Computing Linpack Benchmark"

toolchain = {'name': 'goolf', 'version': '1.4.10'}
[...]
```

3.2.4 Common easyconfig parameters

This section includes an overview of some commonly used (optional) easyconfig parameters.

Source files and patches

- **sources**: list of source files (filenames only)
- **source urls**: list of URLs where sources can be downloaded
- **patches**: list of patch files to be applied (`.patch` extension)

Remarks:

- sources are downloaded (best effort), unless already available
- proxy settings are taken into account, since the urllib2 Python package is used for downloading (since EasyBuild v2.0)
- patches need to be EasyBuild-compatible
  - unified diff format (`diff -ru`)
  - patched locations relative to unpacked sources

Example:

```
name = 'HPL'
[...]
source_urls = ['http://www.netlib.org/benchmark/hpl']
sources = ['hpl-2.0.tar.gz']

# fix Make dependencies, so parallel build also works
patches = ['HPL_parallel-make.patch']
[...]
```

Note: Rather than hardcoding the version (and name) in the list of sources, a string template *%(version)s* can be used, see also *Dynamic values for easyconfig parameters*.

Dependencies

- **dependencies**: build/runtime dependencies
- **builddependencies**: build-only dependencies (not in module)
- **hiddendependencies**: dependencies via hidden modules (see also *Installing dependencies as hidden modules using –hide-deps*)
- **osdependencies**: system dependencies (package names)

Remarks:

- modules must exist for all (non-system) dependencies
- (non-system) dependencies can be resolved via `--robot`
- format: `(<name>, <version>[, <versionsuffix>[, <toolchain>]])`

Example:

```
name = 'GTI'
...
toolchain = {'name': 'goolf', 'version': '1.5.14'}
dependencies = [('PnMPI', '1.2.0')]
builddependencies = [('CMake', '2.8.12', '', ('GCC', '4.8.2')]
```

For each of the specified (build) dependencies, the corresponding module will be loaded in the build environment defined by EasyBuild. For the *runtime* dependencies, `module load` statements will be included in the generated module file.

Note: By default, EasyBuild will try to resolve dependencies using the same toolchain as specified for the software being installed.

A different toolchain can be specified on a per-dependency level (cfr. the `CMake` build dependency in the example above).

Alternatively, you can instruct EasyBuild to use the most minimal (sub)toolchain when resolving dependencies, see *Using minimal toolchains for dependencies*.

Loading of modules for dependencies with a `dummy` toolchain

When a *dummy toolchain* is used, EasyBuild will only load the modules for each of the (build) dependencies when an *empty* string is used as a toolchain version, i.e.

```
toolchain = {'name': 'dummy', 'version': ''}
```

When specifying a non-empty string as version for the *dummy toolchain* (e.g., `dummy`), modules for the (build) dependencies will *not* be loaded in the build environment as defined by EasyBuild. Load statements for the runtime dependencies will still be included in the generated module file, however.

Specifying dependencies using `dummy` toolchain

To make EasyBuild resolve a dependency using the `dummy` toolchain, either specify '`dummy`' as toolchain name in the tuple representing the dependency specification, or simply use `True` as 4th value in the tuple.

For example, to specify PnMPI version 1.2.0 built with the `dummy` toolchain as a (runtime) dependency:

```
dependencies = [('PnMPI', '1.2.0', '', ('dummy', ''))]
```

which is equivalent to:

```
dependencies = [('PnMPI', '1.2.0', '', True)]
```

Using external modules as dependencies

Since EasyBuild v2.1, specifying modules that are not provided via EasyBuild as dependencies is also supported. See *Using external modules* for more information.

Configure/build/install command options

- **configopts**: options for configure command
- **preconfigopts**: options used as prefix for configure command

In analogy to *configure*, also *build* and *install* commands are tuneable:

- **buildopts, prebuildopts**: options for build command
- **installopts, preinstallopts**: options for install command

Example:

```
easyblock = 'ConfigureMake'
...
# configure with: ./autogen.sh && ./configure CC="$CC" CFLAGS="$CFLAGS"
preconfigopts = "./autogen.sh && "
buildopts = 'CC="$CC" CFLAGS="$CFLAGS"'
# install with: make install PREFIX=<installation prefix>
installopts = 'PREFIX=%(installdir)s'
```

Note: For more details w.r.t. use of string templates like `%(installdir)s`, see *Dynamic values for easyconfig parameters*.

Sanity check

Custom paths and commands to be used in the sanity check step can be specified using the respective parameters. These are used to make sure that an installation didn't (partly) fail unnoticed.

- **sanity_check_paths**: files/directories that must get installed
- **sanity_check_commands**: (simple) commands that must work when the installed module is loaded

Remarks:

- format: Python dictionary with (*only*) `files`/`dirs` keys
- values must be lists of (tuples of) strings, one of both **must** be non-empty
  - paths are *relative* to installation directory
  - for a path specified as a tuple, only one of the specified paths must be available
- default values:
  - paths: non-empty `bin` and `lib` or `lib64` directories
  - commands: none

Example:

```
sanity_check_paths = {
    'files': ["bin/xhpl"],
    'dirs': [],
}
```

Easyblock specification

To make EasyBuild use a specific (usually generic) easyblock the **easyblock** parameter can be used.

By default, EasyBuild will assume that the easyblock to use can be derived from the software name. For example: for `GCC`, EasyBuild will look for an easyblock class named `EB_GCC` in the Python module `easybuild.easyblocks.gcc`.

A list of available easyblocks is available via `--list-easyblocks` (see also *List of available easyblocks, –list-easyblocks*); generic easyblocks are the ones for which the name does *not* start with `EB_`.

Example:

```
easyblock = 'CMakeMake'
name = 'GTI'
version = '1.2.0'
...
```

Tip: It is highly recommended to use existing (generic) easyblocks, where applicable. This avoids the need for creating (and maintaining) new easyblocks. Typically, generic easyblocks support several custom easyconfig parameters which allow to steer their behavior (see also *All available easyconfig parameters, –avail-easyconfig-params / -a*).

Example:

```
easyblock = 'Binary'
[...]
install_cmd = "./install.bin"
[...]
```

Module class

The category to which the software belongs to can be specified using the **moduleclass** easyconfig parameter. By default, the `base` module class is used (which should be replaced with a more appropriate category).

EasyBuild enforces that only known module classes can be specified (to avoid misclassification due to typos).

The default list of module classes is available via `--show-default-moduleclasses`; additional module classes can be defined via the `--moduleclasses` configure option.

Example:

```
name = 'GCC'
[...]
moduleclass = 'compiler'
```

Note: By default, EasyBuild will create a symlink to the generated module file in a module class-specific path. This behavior is configurable through the module naming scheme being used.

Tip: The module class may play a significant role in other aspects. For example, the alternative (hierarchical) module naming scheme `HierarchicalMNS` heavily relies on the `moduleclass` parameter for discriminating

compilers and MPI libraries.

3.2.5 Tweaking existing easyconfig files

The ability to modify easyconfig files on the fly with EasyBuild, provides a very powerful and flexible feature to describe builds, without having to manually create all the input files.

Tweaking existing easyconfigs can be done using the `--try-*` command lines options. See *Tweaking existing easyconfig files, using –try-** for more details.

Example:

- GCC version update:

```
eb GCC-4.9.0.eb --try-software-version=4.9.1
```

- install WRF + its dozen dependencies with a different toolchain (!):

```
eb WRF-3.5.1-ictce-5.3.0-dmpar.eb --try-toolchain=intel,2014b -r
```

3.2.6 Dynamic values for easyconfig parameters

String templates are completed using the value of particular easyconfig parameters, typically `name` and/or `version`. These help to avoid hardcoding values in multiple locations.

A list of available string templates can be obtained using `--avail-easyconfig-templates`.

Additionally, constants that can be used in easyconfig files are available via `--avail-easyconfig-constants`.

Example:

```
name = 'GCC'
version = '4.8.3'
...
source_urls = [
    # http://ftpmirror.gnu.org/gcc/gcc-4.8.3
    'http://ftpmirror.gnu.org/%(namelower)s/%(namelower)s-%(version)s',
]
sources = [SOURCELOWER_TAR_GZ]  # gcc-4.8.3.tar.gz
...
```

Note: Proper use of string templates is important, in particular to avoid hardcoding the software version in multiple locations of an easyconfig file; this is critical to make `--try-software-version` behave as expected (see also *Tweaking existing easyconfig files, using –try-**).

3.2.7 Contributing back

Contribute back your working easyconfig files!

Share your expertise with the community, avoid duplicate work, especially if:

- the software package is not supported yet
- an existing easyconfig needs (non-trivial) changes for a different version/toolchain
- it is a frequently used software package (compilers, MPI, etc.)

Notes:

- over 25% of easyconfigs are provided by contributors outside of HPC-UGent

- contributing back does require a limited amount of knowledge on Git/GitHub
- contributions are reviewed & thoroughly tested before inclusion
  - see https://github.com/hpcugent/easybuild/wiki/Contributing-back for a step-by-step walkthrough

3.3 Understanding EasyBuild logs

EasyBuild thoroughly keeps track of the executed build and install procedures. This page details some of the specifics, to help you making sense of them.

3.3.1 Basic information

During an invocation of the `eb` command, a temporary log file is provided. This log can be consulted in case any problems occur during the process. Right before completing successfully, EasyBuild will clean up this temporary log file.

A separate log file is created for each build and install procedure that is performed. After each successful installation, this application log file is copied to the install directory for future reference.

By default, the application log file is copied to a subdirectory of the installation prefix named `easybuild`, and has a filename like `easybuild-HPL-2.0-20141103.104412.log` for example, which corresponds to the filename template `easybuild-%(name)s-%(version)s-%(date)s.%(time)s.log`. This aspect can be tweaked via the `--logfile-format` configuration option.

Example:

```
$ eb HPL-2.0-goolf-1.4.10.eb
== temporary log file in case of crash /tmp/easybuild-rHHgBu/easybuild-XD0Ae_.log
[...]
== building and installing HPL/2.0-goolf-1.4.10...
[...]
== COMPLETED: Installation ended successfully
== Results of the build can be found in the log file /home/example/.local/easybuild/software/HPL/
== Build succeeded for 1 out of 1
== temporary log file /tmp/easybuild-rHHgBu/easybuild-XD0Ae_.log has been removed.
== temporary directory /tmp/easybuild-rHHgBu has been removed.
```

Note: Enabling debug mode using the `--debug` or `-d` command line option ensures that all details of the executed build and installation procedure are included in the log file, but will also result is significantly bigger and more verbose logs.

Tip: Always include a reference to a log file (even if partial) when reporting a potential bug in EasyBuild. A particularly useful way of doing so is by creating a Gist (https://gist.github.com/), and sharing the corresponding URL. This is much better than sending a lengthy log file via email, since it can be easily shared across different communication channels (mailing list, IRC, IM, etc.).

3.3.2 Navigating log files

Extracting the information you're interested in from an EasyBuild log file may be a daunting task, especially for debug logs. The information and guidelines in this section should make navigating logs less scary.

Log message format

Each log message as emitted by EasyBuild follows a well-defined format. Example:

```
== 2014-11-03 13:34:31,906 main.EB_HPL INFO This is EasyBuild 1.15.2 (framework: 1.15.2, easyblock
```

Each log line consists of the following parts:

- a prefix label ==, which is useful to discriminate between EasyBuild log messages and the output of executed shell commands;
- date and time information (e.g., `2014-11-03 13:34:31,906`);
- the Python module/class/function that is responsible for the log message (e.g., `main.EB_HPL`);
- the log level (e.g., `INFO`);
- and a string with the actual log message at the end

Useful handles in log files

Next to looking for a particular search pattern (e.g., `[Ee]rror`), there are a couple of handles that can be used to jump around in log files.

Step markers

For each step performed in the build and installation process, corresponding log messages is emitted. For example:

```
== 2014-11-03 13:34:48,816 main.EB_HPL INFO configuring...
== 2014-11-03 13:34:48,817 main.EB_HPL INFO Starting configure step
[...]
== 2014-11-03 13:34:48,823 main.EB_HPL INFO Running method configure_step part of step configure
```

This allows you to navigate a log file step by step, for example using the `_step` search pattern.

Executed shell commands

For each executed shell command, log messages are included with the full command line, the location where the command was executed and the command's output and exit code. For example:

```
== 2014-11-03 13:34:48,823 main.run DEBUG run_cmd: running cmd /bin/bash make_generic (in /tmp/use
== 2014-11-03 13:34:48,823 main.run DEBUG run_cmd: Command output will be logged to /tmp/easybuild
== 2014-11-03 13:34:48,849 main.run INFO cmd "/bin/bash make_generic" exited with exitcode 0 and
```

If you are primarily interested in the different commands as they were executed by EasyBuild, you can use `INFO cmd` (or `run_cmd`, in debug logs) as a search pattern.

Note: Next to the configure/build/install commands, EasyBuild also runs a couple of other commands to obtain system information, or to query the modules tool. Typically, a single invocation of `eb` involves executing a dozen or so different shell commands, minimally.

CHAPTER 4

Advanced usage topics

4.1 Controlling compiler optimization flags

This page provides an overview on the different ways in which compiler optimization flags used by EasyBuild can be controlled.

Contents

4.1.1 Controlling target architecture specific optimizations via `--optarch`

Default behaviour

By default, EasyBuild optimizes builds for the CPU architecture of the build host, by instructing the compiler to generate instructions for the highest instruction set supported by the process architecture of the build host processor.

This is done by including specific compiler flags in `$CFLAGS`, `$CXXFLAGS`, `$FFLAGS`, `$F90FLAGS`, etc.

For example:

- for toolchains using the GCC compilers, `--march=native` will be used (see https://gcc.gnu.org/onlinedocs/gcc-4.9.0/gcc/i386-and-x86-64-Options.html)
- for toolchains using the Intel compilers, `-xHost` will be used (https://software.intel.com/en-us/node/522846)

Caveats

Heterogeneous clusters

Optimizing for the processor architecture of the build host is usually what you want in an HPC cluster, but it has some implications if your cluster is heterogeneous (i.e., has different processor generations), or if you want to execute your applications in a machine with a processor architecture that is different from the one of the build host.

For example, if you compile your application optimized for an Intel Haswell processor (i.e. using AVX2 instructions), it will not run on a system with an older Intel Nehalem processor.

One possible workaround for heterogeneous HPC clusters is to build as many copies of the software stack as you have processor generations in your cluster, and to configure your system so each compute node uses the right software stack matching its processor architecture type. Details for one way of doing this, using automounter/autofs are available at http://hpcugent.github.io/easybuild/files/sciCORE-software-management_20150611.pdf .

Another solution is to configure EasyBuild to not optimize for the processor architecture of the build host via `--optarch`, see below.

Build environment vs hardcoding in build scripts

Be aware that that using `--optarch` as described below does not provide hard guarantees that the build will be executed using the intended compiler flags.

EasyBuild will define the appropriate environment variables (`$CFLAGS` and co) to use the compiler flags as specified, but some MakeFiles or build systems could have hardcoded values that have not been dealt with yet (for example, via a patch file or by specifying options to the `make` command).

For example, the OpenBLAS build system will autodetect the processor architecture in the build host, and will optimize for that processor architecture by default.

If you want a generic OpenBLAS build you will need to tweak the OpenBLAS easyconfig file to define the desired `TARGET` to use. For this you will need to modify the `buildopts` easyconfig parameter, for example:

```
buildopts = 'TARGET=PRESCOTT BINARY=64 ' + threading + ' CC="$CC" FC="$F77"'
```

See these links for more details w.r.t. OpenBLAS:

- https://github.com/xianyi/OpenBLAS/blob/develop/TargetList.txt
- https://github.com/xianyi/OpenBLAS/issues/685

Specifying target architecture specific optimization flags to use via `--optarch=<flags>`

Using the `--optarch` EasyBuild configuration option, specific compiler flags can be provided that EasyBuild should use, rather than the ones used by default (depending on the compiler in the toolchain being used).

Like any other configuration setting, this can also be specified via `$EASYBUILD_OPTARCH`, or by defining `optarch` in an EasyBuild configuration file (cfr. *Consistentency across supported configuration types*).

For example, by specifying `--optarch=march=core2`, EasyBuild will use `-march=core2` rather than the default flag `--march=native` (when using GCC compilers).

Likewise, to avoid using the default `-xHost` flag with the Intel compilers and using `-xSSSE3` instead, you can define `$EASYBUILD_OPTARCH` to be equal to `xSSSE3`.

Note: The first dash (`-`) is added automagically to the value specified to `--optarch`, because of technicalities with the current implementation.

The `--optarch` configuration option gives you flexibility to define the specific target architecture optimization flags you want, but requires that you take care of specifying different flags for different compilers and choose the right flags depending on your specific processor architecture.

Optimizing for a generic processor architecture via `--optarch=GENERIC`

To make EasyBuild optimize for a *generic* processor architecture, `--optarch` can be set to '`GENERIC`'.

When this is the case, EasyBuild will use the right compiler flags to optimize for a generic processor architecture, i.e. avoid using machine instructions that are only supported by very recent processors.

The `GENERIC` keyword for `--optarch` is recognized since EasyBuild v2.5.0, and is supported for GCC and Intel compilers on x86-64 systems (Intel or AMD). For other compilers that can be used in toolchains and other system architectures, the necessary compiler flags will be defined in later EasyBuild versions.

Currently, using `--optarch=GENERIC` will result in the following target architecture optimization flags being used:

- for toolchains using GCC compilers: `-march=x86-64 -mtune=generic`
- for toolchains using Intel compilers: `-xSSE2`

4.2 Experimental features

First introduced in EasyBuild v2.1.0 (see *EasyBuild v2.1.0 Release Notes*), experimental features can only be used by enabling the `--experimental` configuration option.

An experimental feature indicates to users that these features may change significantly in a future release and should be used only for testing, not (yet) for production usage.

Currently enabled experimental features include:

- Cray support (see https://github.com/hpcugent/easybuild/wiki/EasyBuild-on-Cray)
- support for easyconfig files in YAML syntax (see easyconfig_yeb_format)
- support for using minimal toolchains for dependencies (see *Using minimal toolchains for dependencies*)

4.3 Extended dry run

Using `--extended-dry-run` or `-x` (supported since EasyBuild v2.4.0, see release notes for *v2.4.0 (November 10th 2015)*), a detailed overview of the build and install procedure that EasyBuild is going to execute can be obtained almost instantly.

All time-consuming operations, including executing commands to configure/build/install the software, are only *reported* rather than being actually performed.

Example output is available at extended_dry_run_examples.

Contents

4.3.1 Important notes

There are a couple of things you should be aware of when using `--extended-dry-run` and interpreting the output it produces.

Build/install procedure reported by dry run may be (slightly) different

The actual build and install procedure may (slightly) differ from the one reported by `--extended-dry-run`, due to conditional checks in the easyblock being used.

For example, expressions that are conditional on the presence of certain files or directories in the build directory will always be false, since the build directory is never actually populated.

Errors are ignored (by default) during dry run

Any errors that occur are ignored, and are reported with a clear warning message. This is done because it is possible that these errors occur because of the dry run mechanism.

For example, the install step could assume that certain files created by a previous step will be present, but they will not be there since the commands that are supposed to produce them were not actually performed in dry run mode.

Errors are ignored *on a per-step basis*. When an error is ignored in a particular step, that step is aborted, which may result in partial dry run output for that particular step. Subsequent steps will still be run (in dry run mode), however.

Since it's possible that these errors occur due to a bug in the easyblock being used, it's important to pay attention to these ignored errors.

Ignored errors are reported as follows, for example:

```
== testing... [DRY RUN]

[test_step method]
!!!
```

```
!!! WARNING: ignoring error "[Errno 2] No such file or directory: 'test'"
!!!
```

At the end of dry run output, anonother warning message is shown if any ignored errors occurred:

```
== COMPLETED: Installation ended successfully

!!!
!!! WARNING: One or more errors were ignored, see warnings above
!!!
```

Disabling ignoring errors during dry run

Ignoring errors that occur during a dry run is enabled by default; it can be disabled using the configuration option that is available for it, i.e. by:

- the `--disable-extended-dry-run-ignore-errors` command line option
- by defining the `$EASYBUILD_DISABLE_EXTENDED_DRY_RUN_IGNORE_ERRORS` environment variable
- or by defining `disable-extended-dry-run-ignore-errors` in an EasyBuild configuration file

(see also *Configuring EasyBuild*)

4.3.2 Overview of dry run mechanism

During an extended dry run, several operations are not performed, or are only simulated.

The sections below give a detailed overview of the dry run mechanism.

Temporary directories as build/install directories

To make very sure that EasyBuild does not touch any files or directories during the dry run, the build and (software/module) install directories are replaced by subdirectories of the temporary directory used by that particular EasyBuild session.

In the background, the values for `self.builddir`, `self.installdir` and `self.installdir_mod` are changed in the `EasyBlock` instance(s) being used; this also affects the use of the `%(builddir)s` and `$(installdir)s` values in easyconfig files.

Although the build and install directories are effectively temporary directories during a dry run (under a prefix like `/tmp/eb-aD_yNu/__ROOT__`), this is not visible in the dry run output: the 'fake' build and install directories are replaced by the corresponding original value in the dry run output. For example:

```
[extract_step method]
  running command "tar xzf /home/example/easybuild/sources/b/bzip2/bzip2-1.0.6.tar.gz"
  (in /tmp/example/eb_build/bzip2/1.0.6/GCC-4.9.2)
```

Note on build directory in dry run mode

The build (sub)directory used during an actual (non-dry run) EasyBuild session may be different than the one mentioned in the dry run output.

This is because during a dry run, EasyBuild will *guess* the name of the subdirectory that is created by unpacking the first source file in the build directory as being `<name>-<version>`. Although this is a common pattern, it is not always 100% correct.

For example, you may see this in the dry run output for WRF (for which a build-in-installdir procedure is used):

```
[build_step method]
  running command "tcsh ./compile -j 4 wrf"
  (in /home/example/eb/software/WRF/3.6.1-intel-2015a-dmpar/WRF-3.6.1)
```

The actual build (and install) subdirectory is slightly different while not in dry run mode however, i.e.: `/home/example/eb/software/WRF/3.6.1-intel-2015a-dmpar/WRFV3`.

No downloading of missing source files/patches

Required files (source files/patches) are not downloaded during a dry run if they are not available yet.

The dry run output will specify whether files are found (and if so, at which path) or not; the exact output for files that were not found depends on whether or not source URLs are available.

For example: if the required source file for `bzip2` is not available yet, it is indicated where EasyBuild will try to download it to:

```
[fetch_step method]
Available download URLs for sources/patches:
  * http://www.bzip.org/1.0.6/$source

List of sources:
  * bzip2-1.0.6.tar.gz downloaded to /home/example/easybuild/sources/b/bzip2/bzip2-1.0.6.tar.gz

List of patches:
(none)
```

If the source file is already available in the source path that EasyBuild was configured with, it is indicated as such:

```
List of sources:
  * bzip2-1.0.6.tar.gz found at /home/example/easybuild/sources/b/bzip2/bzip2-1.0.6.tar.gz
```

In case no source URLs are available and required files are missing, they are simply marked as such:

```
Available download URLs for sources/patches:
(none)

List of sources:
  * bzip2-1.0.6.tar.bz2 (MISSING)
```

However, since the dry run mechanism never actually uses the source files/patches, this does not affect the remainder of the output of `--extended-dry-run`/`-x`.

Checksum verification is skipped

Computing checksums of sources files/patches, and verifying them against specified checksums (if available) is *skipped* during a dry run, because it is considered potentially too time-consuming. In addition, source files/patches may not be available anyway.

If checksums are available they are only reported, for example (for GCC v4.9.3):

```
[checksum_step method]
* expected checksum for gcc-4.9.3.tar.bz2: 6f831b4d251872736e8e9cc09746f327
* expected checksum for gmp-6.0.0a.tar.bz2: b7ff2d88cae7f8085bd5006096eed470
* expected checksum for mpfr-3.1.2.tar.gz: 181aa7bb0e452c409f2788a4a7f38476
* expected checksum for mpc-1.0.2.tar.gz: 68fadff3358fb3e7976c7a398a0af4c3
* expected checksum for mpfr-3.1.2-allpatches-20141204.patch: 58aec98d15982f9744a043d2f1c5af82
```

Source files are not unpacked

Source files are *not* unpacked, since this may require too much time (in case of large source files). Additionally, source files may not be available anyway.

This has a number of implications:

- files or directories that may be expected to be there are not, which may lead to (ignored) errors if the used easyblock does not take this into account (see also *Errors are ignored (by default) during dry run*)
- the build directory in which commands are executed may be incorrect in the dry run output (see also *Note on build directory in dry run mode*)

The extraction command is mentioned in the dry run output however, for example:

```
[extract_step method]
  running command "tar xjf bzip2-1.0.6.tar.bz2"
  (in /tmp/example/eb_build/bzip2/1.0.6/GCC-4.9.2)
```

Patch files are not applied, no runtime patching

Since source files are not unpacked, patch files can not applied either.

The dry run output does provide an overview of patch files, together with where they are found and how they are applied:

```
[patch_step method]
* applying patch file WRF_parallel_build_fix.patch
  running command "patch -b -p<derived> -i /home/example/easybuild/sources/w/WRF/WRF_parallel_bui
  (in /home/example/easybuild/easybuild/software/WRF/3.6.1-intel-2015a-dmpar)
* applying patch file WRF-3.6.1_known_problems.patch
  running command "patch -b -p<derived> -i /home/example/easybuild/sources/w/WRF/WRF-3.6.1_known_
  (in /home/example/easybuild/easybuild/software/WRF/3.6.1-intel-2015a-dmpar)
```

Likewise, runtime patching performed by the easyblock itself can not work either. If the `apply_regex_substitutions` function (available from `easybuild.tools.filetools`) is used, a clear overview is included in the dry run output (see also *Runtime patching of files: apply_regex_substitutions*).

For example, in the `configure` step of the WRF easyblock when using the Intel compilers, this yields:

```
[configure_step method]
...
applying regex substitutions to file configure.wrf
  * regex pattern '^(DM_FC\s*=\s*).*$', replacement string '\1 mpif90'
  * regex pattern '^(DM_CC\s*=\s*).*$', replacement string '\1 mpicc -DMPI2_SUPPORT'
```

If the `apply_regex_substitutions` function provided for runtime patching is not used (and `fileinput` is used directly, for example), runtime patching performed by the easyblock will most likely result in an error, leading to the step in which it is being performed being aborted (see *Errors are ignored (by default) during dry run*).

Module `load` statements are executed or simulated

`module load` statements are either effectively executed or simulated, dependending on whether the corresponding module files are available or not.

Available modules are loaded

`module load` statements are fairly light-weight, so they are effectively executed if the module being loaded is available.

The dry run output includes an overview of the modules being loaded. In addition an overview of all loaded modules, including the ones that were loaded indirectly, is shown.

For example:

```
[prepare_step method]
Defining build environment, based on toolchain (options) and specified dependencies...

Loading toolchain module...

module load GCC/4.9.2

Loading modules for dependencies...

module load M4/1.4.17-GCC-4.9.2

Full list of loaded modules:
  1) GCC/4.8.2
  2) M4/1.4.17-GCC-4.9.2
```

Loading of non-available modules is simulated

If the module file required to execute a particular `module load` statement is not available, the dry run mechanism will *simulate* the loading of the module.

The `module load` statements that were simulated rather than actually performed are clearly indicated using `[SIMULATED]` in the dry run output, for example:

```
[prepare_step method]
Defining build environment, based on toolchain (options) and specified dependencies...

Loading toolchain module...

module load intel/2015a

Loading modules for dependencies...

module load JasPer/1.900.1-intel-2015a
module load netCDF/4.3.2-intel-2015a [SIMULATED]
module load netCDF-Fortran/4.4.0-intel-2015a [SIMULATED]
module load tcsh/6.18.01-intel-2015a
```

Only modules that were effectively loaded will appear in the full list of modules being printed; modules for which the load was simulated will not be included.

Simulated loading of non-available *dependency* modules For dependencies, simulating a `module load` statement basically (only) entails defining the `$EBROOT*` and `$EBVERSION*` environment variables (the full variable names are determined by the software name), which are picked up by resp. the `get_software_root` and `get_software_version` functions often used in easyblocks.

The `$EBVERSION*` environment variable is defined with the actual software version of the dependency.

For the `$EBROOT*` environment variable, the name of the environment variable itself prefixed with a '`$`' is used as a dummy value, rather than using an fake installation software prefix. For example, when simulating the load statement for a `GCC` module, the environment variable `$EBROOTGCC` is defined as the string value `'$EBROOTGCC'` (literally).

This results in sensible output when this value is picked up via `get_software_root` by the easyblock.

For example, for netCDF used as a dependency for WRF the following is included in the module file contents included in the dry run output:

```
setenv  NETCDF          "$EBROOTNETCDF"
setenv  NETCDFF         "$EBROOTNETCDFMINFORTRAN"
```

Simulated loading of non-available *toolchain* module When the module that corresponds to the toolchain being used is not available, the dry run mechanism will also simulate the `module load` statements for the individual toolchain components, to ensure that version checks on the toolchain components can work as expected.

For example, if the toolchain module `intel/2015a` is not available, the loading of the `icc`, `ifort`, `impi` and `imkl` modules that would be loaded by the `intel` module is also simulated:

```
[prepare_step method]
Defining build environment, based on toolchain (options) and specified dependencies...

Loading toolchain module...

module load icc/2015.1.133-GCC-4.9.2 [SIMULATED]
module load ifort/2015.1.133-GCC-4.9.2 [SIMULATED]
module load impi/5.0.2.044-iccifort-2015.1.133-GCC-4.9.2 [SIMULATED]
module load imkl/11.2.1.133-iimpi-7.2.3-GCC-4.9.2 [SIMULATED]
module load intel/2015a [SIMULATED]
```

Build environment is reported

The build environment that is set up based on the toolchain (and toolchain options) being used, and the dependencies being loaded is reported as a part of the dry run output.

For example, when `GCC` is used as a toolchain something like this will be included in the `prepare_step` part of the dry run output:

```
Defining build environment...

  export CC="gcc"
  export CFLAGS="-O2"
  export CXX="g++"
  export CXXFLAGS="-O2"
  export F77="gfortran"
  export F90="gfortran"
  export F90FLAGS="-O2"
  export FFLAGS="-O2"
  export FLIBS="-lgfortran"
  export LDFLAGS="-L/home/example/eb/software/GCC/4.8.2/lib"
  export LIBS="-lm -lpthread"
  export OPTFLAGS="-O2"
  export PRECFLAGS=""
```

This is particularly useful as an overview of which environment variables that are defined by the toolchain mechanism, and to assess the effect of changing toolchain options.

The output is deliberately formatted such that is can be easily copy-pasted, which can be useful to mimic the environment in which EasyBuild will perform the build and install procedure.

Shell commands are not executed

Any shell commands that are executed via the `run_cmd` and `run_cmd_qa` functions that are provided by the EasyBuild framework via the `easybuild.tools.run` are *not* executed, only reported (see also *Executing commands: run_cmd and run_cmd_qa*).

This typically includes the commands that are defined in the easyblock to be run as a part of the configure/build/install steps.

For example:

```
configuring... [DRY RUN]

[configure_step method]
  running command " ./configure --prefix=/home/example/eb/software/make/3.82-GCC-4.8.2 "
  (in /home/example/eb/build/make/3.82/GCC-4.8.2/make-3.82)

building... [DRY RUN]

[build_step method]
  running command " make -j 4 "
  (in /home/example/eb/build/make/3.82/GCC-4.8.2/make-3.82)

...

installing... [DRY RUN]

[stage_install_step method]

[make_installdir method]

[install_step method]
  running command " make install "
  (in /home/example/eb/build/make/3.82/GCC-4.8.2/make-3.82)
```

There are a couple of minor exceptions though. Some (light-weight) commands are always run by the EasyBuild framework, even in dry run mode, and an easyblock can specify that particular commands *must* always be run (see also *Executing commands: run_cmd and run_cmd_qa*).

Sanity check paths/commands are not checked

Since nothing is actually being installed during a dry run, the sanity check paths/commands can not be checked.

Instead, the dry run mechanism will produce a clear overview of which paths are expected to be found in the installation directory, and which commands are expected to work (if any).

For example:

```
sanity checking... [DRY RUN]

[sanity_check_step method]
Sanity check paths - file ['files']
  * WRFV3/main/ideal.exe
  * WRFV3/main/libwrflib.a
  * WRFV3/main/ndown.exe
  * WRFV3/main/nup.exe
  * WRFV3/main/real.exe
  * WRFV3/main/tc.exe
  * WRFV3/main/wrf.exe
Sanity check paths - (non-empty) directory ['dirs']
  * WRFV3/main
  * WRFV3/run
Sanity check commands
  (none)
```

Module file is incomplete and only printed

During a dry run, the contents of the module file that would be installed is still generated, but only printed; it is not actually written to file.

More importantly however, the module file being reported is bound to be **incomplete**, since the module generator only includes certain statements conditionally, for example only if the files/directories to which they relate actually

exist. This typically affects `prepend-path` statements, e.g. for `$PATH`, `$LD_LIBRARY_PATH`, etc.

For example, the reported module file for make v3.82 built with `GCC/4.8.2` may look something like:

```
creating module... [DRY RUN]

[make_module_step method]
Generating module file /home/example/eb/modules/all/make/3.82-GCC-4.8.2, with contents:

    #%Module
    proc ModulesHelp { } {
        puts stderr { make-3.82: GNU version of make utility - Homepage: http://www.gnu.org/softwa
        }
    }

    module-whatis {Description: make-3.82: GNU version of make utility - Homepage: http://www.gnu

    set root /home/example/eb/software/make/3.82-GCC-4.8.2

    conflict make

    if { ![ is-loaded GCC/4.8.2 ] } {
        module load GCC/4.8.2
    }

    setenv  EBROOTMAKE              "$root"
    setenv  EBVERSIONMAKE           "3.82"
    setenv  EBDEVELMAKE             "$root/easybuild/make-3.82-GCC-4.8.2-easybuild-devel"

    # Built with EasyBuild version 2.4.0
```

Note that there is no `prepend-path PATH` statement for the `bin` subdirectory, for example.

4.3.3 Guidelines for easyblocks

To ensure useful output under `--extended-dry-run`, easyblocks should be implemented keeping in mind that some operations are possible not performed, to avoid generating errors in dry run mode.

Although errors are just ignored by the dry run mechanism on a per-step basis, they may hide subsequent operations and useful information for the remainder of the step (see also *Errors are ignored (by default) during dry run*).

Detecting dry run mode and enhancing the dry run output

To detect whether an easyblock is being used in dry run mode, it suffices to check the `self.dry_run` class variable.

Additional messages can be included in the dry run output using the `self.dry_run_msg` method.

For example:

```
class Example(EasyBlock):

    def configure_step(self):

        if self.dry_run:
            self.dry_run_msg("Dry run mode detected, not reading template configuration files")
            ...
```

Check whether files/directories exist before accessing them

Rather than assuming that particular files or directories will be there, easyblocks should take into that they may not be, for example because EasyBuild is being run in dry run mode.

For example, instead of simply assuming that a directory named ‘`test`‘ will be there, the existence should be checked first. If not, an appropriate error should be produced, but only when the easyblock is *not* being used in dry run mode.

Bad example:

```
# *BAD* example: maybe the 'test' directory is not there (e.g., because we're in dry run mode)!
try:
    testcases = os.listdir('test')
except OSError as err:
    raise EasyBuildError("Unexpected error when determining list of test cases: %s", err)
```

Good example:

```
# make sure the 'test' directory is there before trying to access it
if os.path.exists('test'):
    try:
        testcases = os.listdir('test')
    except OSError as err:
        raise EasyBuildError("Unexpected error when determining list of test cases: %s", err)

# only raise an error if we're not in dry run mode
elif not self.dry_run:
    raise EasyBuildError("Test directory not found, failed to determine list of test cases")
```

Easyblocks that do not take this into account are likely to result in ignored errors during a dry run (see also *Errors are ignored (by default) during dry run*). For example, for the bad example shown above:

```
!!!
!!! WARNING: ignoring error "Unexpected error when determining list of test cases: [Errno 2] No su
!!!
```

Use functions provided by the EasyBuild framework

The EasyBuild framework provides a bunch of functions that are “*dry run-aware*”, and which can significantly help in keeping easyblocks free from conditional statements checking `self.dry_run`:

- *Defining environment variables: setvar*
- *Writing or appending to files: write_file*
- *Runtime patching of files: apply_regex_substitutions*
- *Executing commands: run_cmd and run_cmd_qa*

Defining environment variables: `setvar`

For defining environment variables, the `setvar` function available in the `easybuild.tools.environment` module should be used.

For example, from the WRF easyblock:

```
jasper = get_software_root('JasPer')
if jasper:
    env.setvar('JASPERINC', os.path.join(jasper, 'include'))
```

When triggered in dry run mode, this will result in a clear dry run message like:

```
export JASPERINC="$EBROOTJASPER/include"
```

The actual output depends on whether or not the required module for `JasPer` is available (see *Simulated loading of non-available dependency modules*).

Silently defining environment variables The `setvar` function also supports defining environment variables *silently*, i.e. without producing a corresponding dry run message, via the named argument `verbose`.

This is used in a couple of places in the EasyBuild framework, to avoid some environment variables being defined cluttering the dry run output without added value. It can be used for similar reasons in easyblocks.

For example, the `PythonPackage` uses it in the *install* step, to modify `$PYTHONPATH` as required by the `python setup.py install` procedure (which is considered not relevant to include in the dry run output, since it's a technicality):

```
env.setvar('PYTHONPATH', new_pythonpath, verbose=False)
```

Writing or appending to files: `write_file`

For writing and appending to files, the EasyBuild framework provides the `write_file` function (available from the `easybuild.tools.filetools` module).

Using it is straightforward, for example:

```
write_file('example.txt', "Contents for the example file")
```

To append to an existing file, `write_file` support a named argument `append`.

When used in dry run mode, `write_file` does not actually (attempt to) write to the file; instead, it just produces an appropriate dry run message and returns.

For example:

```
file written: /tmp/eb-ksVC07/tmp.conf
```

Runtime patching of files: `apply_regex_substitutions`

To make runtime patching of files in easyblocks easier, and to do it with taking the possibility of being in dry run module into account, the EasyBuild framework provides the `apply_regex_substitutions` function (available from the `easybuild.tools.filetools` module, since EasyBuild v2.4.0).

This function takes two arguments: a path to the file that should be patched, and a list of tuples specifying the regular expression pattern to match on, and the string value that should be used as replacement text.

For example (simple fictional example):

```
# replace value for C++ compiler
apply_regex_substitutions('config.mk', [('^(CPLUSPLUS\s*=).*', '\1 %s' % os.environ['CXX'])])
```

When used in dry run mode, it will produce a message like:

```
applying regex substitutions to file config.mk
  * regex pattern '^(CPLUSPLUS\s*=\s).*', replacement string '\1 g++'
```

Executing commands: `run_cmd` and `run_cmd_qa`

To execute shell commands, the `run_cmd` and `run_cmd_qa` functions are provided by the EasyBuild framework in the `easybuild.tools.run` module, with the latter providing support for running interactive commands.

In their simplest form, they simply take the command to execute as a string. For example:

```
run_cmd("tcsh ./compile -j %s wrf" % self.cfg['parallel'])
```

In dry run mode, these functions just produce a dry run message, rather than actually executing the specified command. For example:

```
running command "tcsh ./compile -j 4 wrf"
(in /home/example/eb/software/WRF/3.6.1-intel-2015a-dmpar/WRF-3.6.1)
```

Take into account that the directory included in the message may not be 100% accurate, see *Note on build directory in dry run mode*.

Silently executing commands The `verbose` named argument supported by the `run_cmd` function allows to execute a particular command silently, i.e. without producing a dry run message.

For example:

```
# only run for debugging purposes
run_cmd("ulimit -v", verbose=False)
```

Forced execution of particular commands Sometimes, it can be required that specific (light-weight) commands are *always* executed, because they have side-effects that are assumed to have taken place later in the easyblock.

For this, the `run_cmd` function support another named argument, i.e. `force_in_dry_run`. When set to `True`, the specified command will always be executed, even when in dry run mode.

This is mainly intended for use in the EasyBuild framework itself, where commands that verify certain things must be executed, but it can also be useful for easyblocks (if used correctly).

For example:

```
out, exit_code = run_cmd("type module", simple=False, force_in_dry_run=True)
```

4.3.4 Example output

Output examples for `eb --extended-dry-run`/`eb -x`:

- extended_dry_run_examples_make382_GCC482
- extended_dry_run_examples_WRF361_intel2015a

4.4 Including additional Python modules (`--include-*`)

EasyBuild's capabilities can be extended easily, by including additional Python modules that implement support for building and installing software that is not supported (yet), define additional module naming schemes, or introduce additional toolchains, (optionally) with support for additional compilers, MPI libraries, linear algebra libraries, etc.

Since EasyBuild v2.2.0, dedicated configuration options are available that make it straightforward to get EasyBuild to pick up additional Python modules, and get them registered in the appropriate `easybuild` subnamespace.

- *Including additional easyblocks (–include-easyblocks)*
- *Including additional module naming schemes (–include-module-naming-schemes)*
- *Including additional toolchains (–include-toolchains)*

4.4.1 General aspects of `--include-*` options

Configuration types

The `--include-*` options can be specified via the `eb` command line, using an environment variable (`$EASYBUILD_INCLUDE_*`) or by defining the corresponding `include-*` parameters in an EasyBuild configuration file, just like all other configuration options (see also *Consistentency across supported configuration types*).

Format

The `--include-*` options accept a comma-separated list of paths to Python modules.

These paths can be absolute or relative paths, or so-called *glob patterns*, i.e., paths containing wildcard characters like `*` or `?`. The latter can be used to include several Python modules at once.

For example, to include all Python modules located in the directory `$HOME/myeb`, a path pattern like `$HOME/myeb/*.py` can be specified to the appropriate `--include-*` option.

Note: Shell expansion can get in the way of specifying paths to `eb` that contains wildcards. To avoid problems simply wrap the path in single quotes, or escape the wildcard characters using a backslash. Keep in mind that using single quotes also prevents environment variables (e.g., `$HOME`) from being expanded.

Examples of correct path specifications containing wildcards:

- in a configuration file (no escaping of wildcards required): `include-easyblocks = /home/example/myeb/*.py`
- using an environment variable: `export EASYBUILD_INCLUDE_EASYBLOCKS="$HOME/myeb\*.py"`
- on the command line: `eb --include-easyblocks='/home/example/myeb/*.py' ...`

How it works

For each of the `--include-*` options, EasyBuild will set up a temporary directory providing the corresponding Python package. In this directory, symlinks will be put in place to each of the included Python modules. The parent path is then injected in the Python search path to make the included Python modules available as required.

Order of preference

Python modules that are included via `--include-*` get preference over other Python modules available in the Python search path (e.g., the one that are part of the EasyBuild installation you are using). This may be useful when testing modifications to particular components of EasyBuild, for example easyblocks.

Note: It is recommended to only override existing components during testing. Future EasyBuild versions may include important updates like bug fixes, which may be missed if customised implementations of components were put in place.

4.4.2 Including additional easyblocks (`--include-easyblocks`)

Adding support for building and installing additional software packages can be done by specifying the location of Python modules that implement easyblocks via `--include-easyblocks`.

Generic easyblocks are expected to be located in a directory named `generic`.

To verify that the easyblocks you included are indeed being picked up, `--list-easyblocks=detailed` can be used (see also *List of available easyblocks, –list-easyblocks*).

Example

The example below shows how all self-implemented easyblocks (both software-specific and generic) located in the `$HOME/myeasyblocks` directory can be included:

```
$ export EASYBUILD_INCLUDE_EASYBLOCKS=$HOME/myeasyblocks/\*.py,$HOME/myeasyblocks/generic/\*.py
$ eb --list-easyblocks=detailed
...
|-- EB_mytest (easybuild.easyblocks.mytest @ /tmp/example/eb-Bk3zxb/included-easyblocks/easybuild/
...
|-- foo (easybuild.easyblocks.generic.foo @ /tmp/example/eb-Bk3zxb/included-easyblocks/easybuild/e
...
```

4.4.3 Including additional module naming schemes (`--include-module-naming-schemes`)

To make EasyBuild aware of one or more custom module naming schemes, the path to the corresponding Python modules can be specified via `--include-module-naming-schemes`.

To verify that EasyBuild is aware of the additional module naming schemes, the `--avail-module-naming-schemes` option can be used.

Example

The example below shows how all custom module naming schemes located in the `$HOME/myebmns` can be included:

```
$ eb --include-module-naming-schemes=$HOME/myebmns/\*.py --avail-module-naming-schemes
List of supported module naming schemes:
    ...
    MyCustomMNS
    MyOtherCustomMNS
    ...
```

4.4.4 Including additional toolchains (`--include-toolchains`)

Plugging in Python modules that add support for additional toolchains, optionally including additional toolchain components (compilers, MPI libraries, BLAS/LAPACK/FFT libraries, ...) can be done via `--include-toolchains`.

EasyBuild will determine whether the Python module is a *toolchain definition* or implements support for an *additional toolchain component* based on the name of the directory in which it is located. Implementations of toolchain components are expeced to be located in a directory named according to the type of component (`compiler`, `mpi`, `linalg` or `fft`).

To verify that EasyBuild is aware of the included toolchains, `--list-toolchains` can be used.

Example

The example below shows how the support for additional toolchains and the required additional compiler/MPI toolchain components implemented by the Python modules located in the directory `$HOME/myebtcs` can be included:

```
$ export EASYBUILD_INCLUDE_TOOLCHAINS=$HOME/myebtcs/\*.py,$HOME/myebtcs/compiler/\*.py,$HOME/myebt
$ eb --list-toolchains
List of known toolchains (toolchainname: module[,module...]):
    ...
    mytoolchain: MyCompiler, MyMPI
    ...
```

4.5 Integration with GitHub

EasyBuild provides several features that integrate with GitHub, where the different EasyBuild repositories are located.

From the EasyBuild command line `eb` several options are available to reach out to GitHub, which are documented below.

Contents

4.5.1 Requirements

Depending on which GitHub integration features you want to use, there are a couple of requirements:

- **a GitHub account**
  - see https://github.com; creating an account is free but requires registration
- **a GitHub user name**
  - only required for authenticated access to the GitHub API, which can help to avoid rate limitations
  - *not* strictly necessary for read-only operations
    - i.e. *not* required for *Using easyconfigs from pull requests (–from-pr)* and *Reviewing easyconfig pull requests (–review-pr)* (but it can help)
  - see *Providing a GitHub username (–github-user)*
- **a GitHub token** + `keyring` **Python package**

  - allows accessing the GitHub API with authentication
  - only strictly required for features that require GitHub 'write' permissions
    - i.e. for *Uploading test reports (–upload-test-report)* and *Submitting pull requests (–new-pr)*
  - see *Installing a GitHub token (–install-github-token)*
- `git` **command /** `GitPython` **Python package**
  - only required when local `git` commands need to be executed, e.g. to manipulate a Git repository
    - i.e. for *Submitting pull requests (–new-pr)* and *Updating existing pull requests (–update-pr)*
- **SSH public key registered on GitHub**
  - only required when `push` access to Git repositories that reside on GitHub is required
    - i.e. for *Submitting pull requests (–new-pr)* and *Updating existing pull requests (–update-pr)*
  - see https://github.com/settings/ssh
- **fork of the EasyBuild repositories on GitHub**
  - only required for submitting/updating pull requests (*Submitting pull requests (–new-pr)* and *Updating existing pull requests (–update-pr)*)
  - see `Fork` button (top right) at https://github.com/hpcugent/easybuild-easyconfigs (for example)

See also *Checking status of GitHub integration (–check-github)*.

4.5.2 Configuration

The following sections discuss the EasyBuild configuration options relevant to the GitHub integration features.

Providing a GitHub username (`--github-user`)

To specify your GitHub username, do one of the following:

- use the `--github-user` configuration option on the `eb` command line
- define the `$EASYBUILD_GITHUB_USER` environment variable
- specify `github-user` in your EasyBuild configuraton file

(see also *Configuring EasyBuild*)

Installing a GitHub token (`--install-github-token`)

Note: *requires*: GitHub username + `keyring` Python package

A GitHub token is a string of 40 hexidecimal (lowercase) characters that is tied to your GitHub account, allowing you to access the GitHub API authenticated.

Using a GitHub token is beneficial with respect to rate limitations, and enables write permissions on GitHub (e.g., posting comments, creating gists, opening pull requests, etc.).

To obtain a GitHub token:

- visit https://github.com/settings/tokens/new and log in with your GitHub account
- enter a token description, for example: "`EasyBuild`"
- make sure (only) the `gist` and `repo` scopes are fully enabled
- click `Generate token`

- *copy-paste* the generated token

Note: You will only be able to copy-paste the generated token right after you have created it. The value corresponding to an existing token can *not* be retrieved later through the GitHub interface.

Please keep your token secret at all times; it allows fully authenticated access to your GitHub account!

You can install the GitHub token in your keyring using EasyBuild, so it can pick it up when it needs to, using `eb --install-github-token`:

```
$ eb --github-user example --install-github-token
Token: <copy-paste-your-40-character-token-here>
Validating token...
Token seems to be valid, installing it.
Token 'e3a..0c2' installed!
```

EasyBuild will validate the provided token, to check that authenticated access to your GitHub account works as expected.

Note: EasyBuild will never print the full token value, to avoid leaking it. For debugging purposes, only the first and last 3 characters will be shown.

Specify location of working directories (`--git-working-dirs-path`)

You can specify the location of your Git working directories using `--git-working-dirs-path`.

The provided path should be the *parent* directory of the location of the working directories (i.e. clones) of the EasyBuild repositories (`easybuild-easyconfigs`, etc.); the assumption is that you keep them all in a single parent directory.

Although not strictly required, this is useful for speeding up `--new-pr` and `--update-pr`, since it allows that the repository can be copied & updated, rather than being cloned from scratch.

4.5.3 Checking status of GitHub integration (`--check-github`)

To check the status of your setup w.r.t. GitHub integration, the `--check-github` command line option can be used.

Using this will trigger EasyBuild to perform a number of checks, and report back on what the test results mean for the different GitHub integration features.

If all requirements are taken care of in your setup, you should see output like this:

```
$ eb --check-github

== temporary log file in case of crash /tmp/eb-xWCpWl/easybuild-hGnKS5.log

Checking status of GitHub integration...

Making sure we're online... OK

* GitHub user... example => OK
* GitHub token... e3f..0c8 (len: 40) => OK (validated)
* git command... OK ("git version 2.7.4 (Apple Git-66); ")
* GitPython module... OK
* push access to example/easybuild-easyconfigs repo @ GitHub... OK
* creating gists... OK
* location to Git working dirs...  OK (/home/example/git-working-dirs)
```

```
All checks PASSed!

Status of GitHub integration:
* --from-pr: OK
* --new-pr: OK
* --review-pr: OK
* --update-pr: OK
* --upload-test-report: OK
```

Note: Checking whether push access to GitHub works may take some time, since a recent clone of the easybuild-easyconfigs GitHub repository will be created in the process (at a temporary location).

See also *Requirements*.

4.5.4 Using easyconfigs from pull requests (`--from-pr`)

(supported since EasyBuild v1.13.0)

Via the `--from-pr` command line option (available since EasyBuild v1.13.0), easyconfig files that are added or modified by a particular pull request to the easybuild-easyconfigs GitHub repository can be used (regardless of whether the pull request is merged or not).

This can be useful to employ easyconfig files that are not available yet in the active EasyBuild installation, or to test new contributions by combining `--from-pr` with `-upload-test-report` (see *Uploading test reports (–upload-test-report)*).

When `--from-pr` is used, EasyBuild will download all modified files (easyconfig files and patches) to a temporary directory before processing them.

For example, to use the GCC v4.9.2 easyconfigs contributed via easyconfigs pull request #1177:

```
$ eb --from-pr 1177 --dry-run
== temporary log file in case of crash /tmp/eb-88quZc/easybuild-62fFdo.log
Dry run: printing build status of easyconfigs and dependencies
 * [ ] /tmp/eb-88quZc/files_pr1177/GCC-4.9.2-CLooG-multilib.eb (module: GCC/4.9.2-CLooG-multilib)
 * [ ] /tmp/eb-88quZc/files_pr1177/GCC-4.9.2-CLooG.eb (module: GCC/4.9.2-CLooG)
 * [ ] /tmp/eb-88quZc/files_pr1177/GCC-4.9.2.eb (module: GCC/4.9.2)
== temporary log file /tmp/eb-88quZc/easybuild-62fFdo.log has been removed.
== temporary directory /tmp/eb-88quZc has been removed.
```

Note: To avoid GitHub rate limiting, let EasyBuild know which GitHub account should be used to query the GitHub API, and provide a matching GitHub token; see also *Installing a GitHub token (–install-github-token)*.

Synergy with `--robot`

Since EasyBuild v1.15.0, the temporary directory containing the easyconfigs (and patch files) from the specified pull request is *prepended* to the robot search path, to ensure that easyconfigs that were modified in the respective pull request are picked up via `--robot` when they are required. Thus, for easyconfig files that are both available in the pull request and are available locally, the ones from the specified pull request will be preferred.

For example, to build and install `HPL` with the `intel/2015a` toolchain, both of which are contributed via easyconfigs pull request #1238:

```
$ eb --from-pr 1238 --dry-run --robot $HOME/easyconfigs
== temporary log file in case of crash /tmp/eb-A1fRvw/easybuild-Eqc8Oi.log
Dry run: printing build status of easyconfigs and dependencies
 * [x] /home/example/easyconfigs/g/GCC/GCC-4.9.2.eb (module: GCC/4.9.2)
 * [x] /home/example/easyconfigs/i/icc/icc-2015.1.133-GCC-4.9.2.eb (module: icc/2015.1.133-GCC-4.
```

```
 * [x] /home/example/easyconfigs/i/ifort/ifort-2015.1.133-GCC-4.9.2.eb (module: ifort/2015.1.133-G
 * [x] /home/example/easyconfigs/i/iccifort/iccifort-2015.1.133-GCC-4.9.2.eb (module: iccifort/201
 * [x] /home/example/easyconfigs/i/impi/impi-5.0.2.044-iccifort-2015.1.133-GCC-4.9.2.eb (module: i
 * [x] /home/example/easyconfigs/i/iimpi/iimpi-7.2.3-GCC-4.9.2.eb (module: iimpi/7.2.3-GCC-4.9.2)
 * [x] /home/example/easyconfigs/i/imkl/imkl-11.2.1.133-iimpi-7.2.3-GCC-4.9.2.eb (module: imkl/11
 * [ ] /tmp/eb-A1fRvw/files_pr1238/intel-2015a.eb (module: intel/2015a)
 * [ ] /tmp/eb-A1fRvw/files_pr1238/HPL-2.1-intel-2015a.eb (module: HPL/2.1-intel-2015a)
== temporary log file /tmp/eb-A1fRvw/easybuild-Eqc8Oi.log has been removed.
== temporary directory /tmp/eb-A1fRvw has been removed.
```

Note that the easyconfigs that are required to resolve dependencies and are available locally in `$HOME/easyconfigs` are being picked up as needed.

Specifying particular easyconfig files

Since EasyBuid v2.0.0 the particular easyconfigs to be used can be specified, rather than using all easyconfigs that are touched by the pull request (which is the default if no easyconfigs are specified alongside `--from-pr`).

For example, to only use `CMake-3.0.0-intel-2015a.eb` from easyconfigs pull request #1330, and ignore the other easyconfigs being contributed in that same pull request for netCDF, WRF, ...:

```
$ eb --from-pr 1330 CMake-3.0.0-intel-2015a.eb --dry-run --robot $HOME/easyconfigs
== temporary log file in case of crash /tmp/eb-QhM_qc/easybuild-TPvMkJ.log
Dry run: printing build status of easyconfigs and dependencies
 * [x] /home/example/easyconfigs/g/GCC/GCC-4.9.2.eb (module: GCC/4.9.2)
 * [x] /home/example/easyconfigs/i/icc/icc-2015.1.133-GCC-4.9.2.eb (module: icc/2015.1.133-GCC-4.9
 * [x] /home/example/easyconfigs/i/ifort/ifort-2015.1.133-GCC-4.9.2.eb (module: ifort/2015.1.133-G
 * [x] /home/example/easyconfigs/i/iccifort/iccifort-2015.1.133-GCC-4.9.2.eb (module: iccifort/201
 * [x] /home/example/easyconfigs/i/impi/impi-5.0.2.044-iccifort-2015.1.133-GCC-4.9.2.eb (module: i
 * [x] /home/example/easyconfigs/i/iimpi/iimpi-7.2.3-GCC-4.9.2.eb (module: iimpi/7.2.3-GCC-4.9.2)
 * [x] /home/example/easyconfigs/i/imkl/imkl-11.2.1.133-iimpi-7.2.3-GCC-4.9.2.eb (module: imkl/11
 * [x] /home/example/easyconfigs/i/intel/intel-2015a.eb (module: intel/2015a)
 * [x] /home/example/easyconfigs/n/ncurses/ncurses-5.9-intel-2015a.eb (module: ncurses/5.9-intel-2
 * [ ] /tmp/eb-QhM_qc/files_pr1330/CMake-3.0.0-intel-2015a.eb (module: CMake/3.0.0-intel-2015a)
== temporary log file /tmp/eb-QhM_qc/easybuild-TPvMkJ.log has been removed.
== temporary directory /tmp/eb-QhM_qc has been removed.
```

Again, note that locally available easyconfigs that are required to resolve dependencies are being picked up as needed.

4.5.5 Uploading test reports (`--upload-test-report`)

(supported since EasyBuild v1.13.0)

Note: requires that a GitHub token was required `gist` permissions is available, cfr. *Installing a GitHub token (–install-github-token)*

For every installation performed with EasyBuild, a test report is generated. By default, the test report is copied in the installation directory, right next to the log file (see also *Understanding EasyBuild logs*).

Using `--upload-test-report`, the test report can also be pushed to GitHub (as a *gist*, cfr. https://gist.github.com) to share it with others.

Each test report includes:

- an overview of the easyconfigs being processed
- time & date
- the exact `eb` command line that was used

- the full EasyBuild configuration that was in place
- information about the system on which EasyBuild was used (hostname, OS, architecture, etc.)
- the list of modules that was loaded
- the full environment of the session in which `eb` was run (note: can be filtered, see *Filtering the environment details (–test-report-env-filter)*)

For each easyconfig that *failed* to install a partial log will be uploaded as a separate gist, and a link to this gist will be included in the test report.

If `--upload-test-report` is combined with `--from-pr`, a comment referring to the test report (incl. a brief summary) will be placed in the respective pull request. This makes it a very powerful tool when testing contributions.

Note: If you want to easily access a test report without uploading it to GitHub, use `--dump-test-report`.

Example:

```
$ eb --from-pr 3153 --force --upload-test-report
== temporary log file in case of crash /tmp/eb-aqk20q/easybuild-wuyZBV.log
== processing EasyBuild easyconfig /tmp/eb-aqk20q/files_pr3153/EasyBuild/EasyBuild-2.8.1.eb
== building and installing EasyBuild/2.8.1...
...
== COMPLETED: Installation ended successfully
== Results of the build can be found in the log file /home/example/software/EasyBuild/2.8.1/easybu
== Test report uploaded to https://gist.github.com/1cb2db8a2913a1b8ddbf1c6fee3ff83c and mentioned
== Build succeeded for 1 out of 1
== Temporary log file(s) /tmp/eb-aqk20q/easybuild-wuyZBV.log* have been removed.
== Temporary directory /tmp/eb-aqk20q has been removed.
```

The resulting test report can be viewed at https://gist.github.com/1cb2db8a2913a1b8ddbf1c6fee3ff83c .

Note: It is common to use `--force` in combination with `--upload-test-report`, to ensure that all easyconfigs in the pull request are rebuilt, resulting in a complete test report.

Filtering the environment details (`--test-report-env-filter`)

Since the environment of the session in which `eb` was used may contain sensitive information, it can be filtered through `--test-report-env-filter`.

This configuration option takes a regular expression that is used to determine which environment variables can be included in the test report (based on their name). Environment variables for which the name *matches* the specified regular expression will *not* be included in the test report.

An example of a typical setting:

```
export EASYBUILD_TEST_REPORT_ENV_FILTER='^SSH|USER|HOSTNAME|UID|.*COOKIE.*'
```

4.5.6 Reviewing easyconfig pull requests (`--review-pr`)

A useful tool when reviewing pull requests for the easybuild-easyconfigs repository that add new or update existing easyconfig files is `--review-pr`.

The 'files' tab in the GitHub interface shows the changes being made to existing files; using `--review-pr` the differences with one or more other *similar* easyconfig files, for example the one(s) with the same toolchain (version) and/or software version, can also be evaluated.

This is very useful to quickly see how easyconfig files in pull requests differ from existing easyconfig files, and to maintain consistency across easyconfig files where desired.

The `--review-pr` output consists of a 'multidiff' view per easyconfig file that is being touched by the specified pull request. The exact format of the output depends on whether EasyBuild is configured to allow colored output (enabled by default, see `--color`).

Search criteria for similar easyconfigs

The set of existing similar easyconfig files is determined by specific search criteria; the first one that results in a non-empty set of easyconfigs is retained.

The search criteria consists of a combination of the *software version criterion* with additional restrictions.

The software version criterion is one of the criterions below (considered in order), with `x.y.z` the software version of the easyconfig file from the pull request:

- exact same software version
- same major/minor software version (same `x` and `y`)
- same major software version (same `x`)
- no (partial) version match (so consider any version)

The addition restrictions are the following (also considered in order):

- matching versionsuffix and toolchain name/version
- matching versionsuffix and toolchain name (any toolchain version)
- matching versionsuffix (any toolchain name/version)
- matching toolchain name/version (any versionsuffix)
- matching toolchain name (any versionsuffix, toolchain version)
- no extra requirements (any versionsuffix, toolchain name/version)

4.5.7 Submitting new and updating pull requests (`--new-pr`, `--update-pr`)

(supported since EasyBuild v2.6.0)

EasyBuild provides two simple yet powerful features that make contributing back to the central EasyBuild repositories significantly easier and less error-prone, especially for people who are not very familiar with `git` and/or GitHub yet:

- `--new-pr` to create new pull requests
- `--update-pr` to update existing pull requests

Note: Both `--new-pr` and `--update-pr` are still **experimental**, meaning that their behaviour may change in future releases and that they require having `--experimental` enabled; see also *Experimental features*.

Submitting pull requests (`--new-pr`)

To create a new pull request, the `--new-pr` command line option can be used, provided that the necessary requirements are fullfilled (see *Requirements*).

In its simplest form, you just provide the location of the file(s) that you want to include in the pull request:

```
$ eb --new-pr test.eb --experimental
```

This takes care of all the steps required to make a contribution, i.e.:

- set up a working copy of the relevant EasyBuild repository (e.g., `easybuild-easyconfigs`)
- create a new 'feature' branch, starting from the up-to-date `develop` branch
- renaming easyconfig files according to their `name`, `version`, `versionsuffix` and `toolchain`
- moving easyconfig files to the right location in the repository (e.g. `easybuild/easyconfigs/e/EasyBuild/`)
- staging and committing the files in the feature branch
- pushing the feature branch to your fork of the relevant EasyBuild repository on GitHub
- creating the pull request, targetting the `develop` branch of the central EasyBuild repository (e.g. `hpcugent/easybuild-easyconfigs`)

It should be clear that automating this whole procedure with a single simple `eb` command greatly lowers the bar for contributing back, especially since it even alleviates the need for interacting directly with `git` entirely!

The working copy of the EasyBuild repository is created in a temporary location, and cleaned up once the pull request has been created. EasyBuild does *not* make changes to an existing working copy you may have in place already (cfr. *Specify location of working directories (–git-working-dirs-path)*).

Example

For example, to create a pull request for a new version of, let's say, EasyBuild:

```
$ eb --new-pr ~/WIP/newEB.eb --experimental
== temporary log file in case of crash /tmp/eb-mWKR9u/easybuild-cTpf2W.log
== copying /home/example/git-working-dirs/easybuild-easyconfigs...
== fetching branch 'develop' from https://github.com/hpcugent/easybuild-easyconfigs.git...

Opening pull request
* target: hpcugent/easybuild-easyconfigs:develop
* from: boegel/easybuild-easyconfigs:20160530131447_new_pr_EasyBuild281
* title: "{tools}[dummy/dummy] EasyBuild v2.8.1"
* description:
"""
(created using `eb --new-pr`)

"""
* overview of changes:
 .../easyconfigs/e/EasyBuild/EasyBuild-2.8.1.eb     | 35 ++++++++++++++++++++++++
 1 file changed, 35 insertions(+)

Opened pull request: https://github.com/hpcugent/easybuild-easyconfigs/pull/3153
```

Yes, it's that easy!

Updating existing pull requests (`--update-pr`)

Similarly to creating new pull requests, existing pull requests can be easily updated using `eb --update-pr` (regardless of whether or not they were created with `--new-pr`).

The usage is equally simple, for example to update pull request `#1234` just list the changed/new file(s):

```
$ eb --update-pr 1234 ~/WIP/changed.eb --experimental
```

Again, this take care of the whole procedure required to update an existing pull request:

- set up a working copy of the relevant EasyBuild repository (e.g., `easybuild-easyconfigs`)
- determining the branch corresponding to the pull request, which should be updated by pushing a new commit to it

- checking out that branch
- renaming easyconfig files according to their `name`, `version`, `versionsuffix` and `toolchain`
- moving easyconfig files to the right location in the repository (e.g. `easybuild/easyconfigs/e/EasyBuild/`)
- staging and committing the (changed/new) files
- pushing the updated branch to GitHub

Again, not a single `git` command to be executed; the only thing that is required is the ID of the pull request that should be updated.

Just like with `--new-pr`, this is done in a temporary working copy of the repository, no changes are made to a possible existing working copy.

Example

For example, to update pull request #3153 with a changed easyconfig file:

```
eb --update-pr 3153 ~/WIP/newEB.eb --experimental
== temporary log file in case of crash /tmp/eb-gO2wJu/easybuild-37Oo2z.log
== Determined branch name corresponding to hpcugent/easybuild-easyconfigs PR #3153: 20160530131447
== copying /home/example/git-working-dirs/easybuild-easyconfigs...
== fetching branch '20160530131447_new_pr_EasyBuild281' from https://github.com/boegel/easybuild-e
Overview of changes:
 easybuild/easyconfigs/e/EasyBuild/EasyBuild-2.8.1.eb | 3 +++
 1 file changed, 3 insertions(+)

Updated hpcugent/easybuild-easyconfigs PR #3159 by pushing to branch boegel/20160530131447_new_pr
```

Controlling pull request metadata

You can control the metadata for pull requests using the following configuration options:

- `--pr-branch-name`: branch name for new pull requests
- `--pr-commit-msg`: commit message to use when creating new or updating existing pull requests
- `--pr-descr`: pull request description
- `--pr-title`: pull request title

EasyBuild will use sensible defaults for each of these, see below.

Default branch name for new pull requests

The branch name for new pull requests will be composed from:

- a timestamp, down to the second in an attempt to make it unique
  - example: `20160513141133` for a pull request created on May 13th 2016, 2:11:33 PM
- a label `new_pr`
- the software name and version of the first easyconfig file, with some filtering (e.g. remove `.`'s)
  - example: `GCC530` for GCC v5.3.0

Full example: `20160513141133_new_pr_GCC530`

Although there is usually no reason to change this default, it can be done if desired using `--pr-branch-name` when opening a new pull request with `--new-pr`.

Default commit message

The default commit message is very simple: it specifies for each easyconfig file whether it was a new file being added, or an existing file being modifed, for example "`add easyconfig GCC-5.3.0.eb, modify easyconfig GCC-4.9.3.eb`".

It is highly recommended to provide a more descriptive commit message via `--pr-commit-msg` *whenever existing easyconfig files are being modified, both with* `--new-pr` *and* `--update-pr`.

Default pull request description

By default, the pull description will only contain the following text:

```
(created using eb --new-pr)
```

It is generally advised to provide more descriptive information, although the changes made by the pull request may be self-explanatory (e.g. when only adding new easyconfig files).

To change this default text, you can either use `--pr-descr` or edit the description via the GitHub interface after the pull request has been opened.

Particularly useful information to specify here is dependencies on other pull requests, by copy-pasting the respective URLs with a short descriptive message like '`depends on PR <URL>`'.

Default pull request title

The pull request title is derived from the easyconfig files being changed/added, taking into account the recommendation for easyconfig pull requests to clearly specify module class, toolchain, software name/version, as follows: `{<module_class>}[<toolchain>] <software_name> v<software_version>`.

For example, when opening a pull request for an easyconfig for Python 2.7.11 with the `intel/2016a` toolchain, the default pull request title will be something like: `{lang}[intel/2016a] Python v2.7.11`.

If multiple easyconfig files are provided, the respective software names/versions will be included separated by a `,`, up until the first 3 easyconfig files (to avoid excessively lenghty pull request titles).

In case (only) existing easyconfig files are being changed, it's advisable to provide a more descriptive title using `--pr-title`.

Configuring `--new-pr` and `--update-pr`

By default, `--new-pr` and `--update-pr` affect pull requests to the central `hpcugent/easybuild-easyconfigs` repository.

However, this can be changed with the following configurations options:

- `--pr-target-account` (default: `hpcugent`): target GitHub account for new pull requests
- `--pr-target-branch` (default: `develop`): target branch for new pull requests
- `--pr-target-repo` (default: `easybuild-easyconfigs`): target repository for new pull requests

Synergy with `--dry-run/-D` and `--extended-dry-run/-x`

Both `--new-pr` and `--update-pr` are 'dry run-aware', in the sense that you can combine them with either `--dry-run/-D-` or `--extended-dry-run/-x` to preview the pull request they would create/update without actually doing so.

For example:

```
$ eb --new-pr EasyBuild-2.9.0.eb --experimental -D
== temporary log file in case of crash /tmp/eb-1ny69k/easybuild-UR1Wr4.log
== copying /home/example/git-working-dirs/easybuild-easyconfigs...
== fetching branch 'develop' from https://github.com/hpcugent/easybuild-easyconfigs.git...

Opening pull request [DRY RUN]
* target: hpcugent/easybuild-easyconfigs:develop
* from: boegel/easybuild-easyconfigs:20160603105641_new_pr_EasyBuild290
* title: "{tools}[dummy/dummy] EasyBuild v2.9.0"
* description:
"""
(created using `eb --new-pr`)

"""
* overview of changes:
 .../easyconfigs/e/EasyBuild/EasyBuild-2.9.0.eb     | 35 ++++++++++++++++++++++++
 1 file changed, 35 insertions(+)
```

The only difference between using `--dry-run` and `--extended-dry-run` is that the latter will show the full diff of the changes (equivalent to `git diff`), while the former will only show a summary of the changes (equivalent to `git diff --stat`, see example above).

4.6 Manipulating dependencies

A couple of different ways are available to manipulate the list of dependencies that are specified for the software packages being installed.

Contents

4.6.1 Filtering out dependencies using `--filter-deps`

To avoid that certain dependencies are being installed, a list of software names can be specified to `--filter-deps`. Any time a dependency with a name from this list is specified, it will be simply filtered out by EasyBuild, and thus disregarded when resolving dependencies, loading modules for the dependencies in the build environment, and including '`module load`' statements in the generated module files.

This can be useful when particular tools and libraries are already provided by OS packages (or in some other way), and should not be reinstalled as modules by EasyBuild.

For example:

- overview of dependencies of HDF5:

```
$ eb HDF5-1.8.13-intel-2015a.eb -D
 ...
 * [ ] $CFGS/i/intel/intel-2015a.eb (module: intel/2015a)
 * [ ] $CFGS/z/zlib/zlib-1.2.8-intel-2015a.eb (module: zlib/1.2.8-intel-2015a)
 * [ ] $CFGS/s/Szip/Szip-2.1-intel-2015a.eb (module: Szip/2.1-intel-2015a)
 * [ ] $CFGS/h/HDF5/HDF5-1.8.13-intel-2015a.eb (module: HDF5/1.8.13-intel-2015a)
```

- overview of dependencies of HDF5, with zlib and Szip excluded:

```
$ eb HDF5-1.8.13-intel-2015a.eb --filter-deps=zlib,Szip -D
 ...
 * [ ] $CFGS/i/intel/intel-2015a.eb (module: intel/2015a)
 * [ ] $CFGS/h/HDF5/HDF5-1.8.13-intel-2015a.eb (module: HDF5/1.8.13-intel-2015a)
```

4.6.2 Installing dependencies as hidden modules using `--hide-deps`

Selected software packages can be marked to be installed as hidden modules (i.e., modules that are not visible via '`module avail`', but can be loaded) whenever they are included as a dependency, via the `--hide-deps` configuration option.

For example (note the preceding '`.`' in the last part of the module names for zlib and Szip):

```
$ eb HDF5-1.8.13-intel-2015a.eb --hide-deps=zlib,Szip -D
 ...
 * [ ] $CFGS/i/intel/intel-2015a.eb (module: intel/2015a)
 * [ ] $CFGS/z/zlib/zlib-1.2.8-intel-2015a.eb (module: zlib/.1.2.8-intel-2015a)
 * [ ] $CFGS/s/Szip/Szip-2.1-intel-2015a.eb (module: Szip/.2.1-intel-2015a)
 * [ ] $CFGS/h/HDF5/HDF5-1.8.13-intel-2015a.eb (module: HDF5/1.8.13-intel-2015a)
```

Note: Using Lmod (version >= 5.7.5), hidden modules can be made visible in the output of '`module avail`' using the `--show-hidden` option.

For example:

```
$ module avail bzip2

Use "module spider" to find all possible modules.
Use "module keyword key1 key2 ..." to search for all possible modules matching any of the "keys".

$ module --show-hidden avail bzip2
----- /home/example/.local/easybuild/modules/all -----
bzip2/.1.0.6

Use "module spider" to find all possible modules.
Use "module keyword key1 key2 ..." to search for all possible modules matching any of the "keys".
```

4.6.3 Using minimal toolchains for dependencies

Note: This is an **experimental** feature, see *Experimental features*.

By default, EasyBuild will try to resolve dependencies using the same toolchain as the one that is used for the software being installed, unless a specific toolchain is specified for the dependency itself (see *Dependencies*).

Using the `--minimal-toolchains` configuration option, you can instruct EasyBuild to consider subtoolchains as well for dependencies. This can be useful to refrain from having to frequently hardcode specific toolchains in order to avoid having the same dependency version installed with multiple toolchains that are compatible with each other. Although hardcoding toolchain for dependencies will work fine, it severly limits the power of other EasyBuild features, like `--try-toolchain` for example.

When instructed to use minimal toolchains, EasyBuild will check whether an easyconfig file is available (in the robot search path, see *Searching for easyconfigs: the robot search path*) for that dependency using the different subtoolchains of the toolchain specified for the 'parent' software. Subtoolchains are considered 'bottom-up', i.e. starting with the most minimal subtoolchain (typically a compiler-only toolchain), and then climbing up towards the toolchain that is specified for the software being installed.

Note that if a specific toolchain is specified for a particular dependency, EasyBuild will stick to using it, even when instructed to use minimal toolchains.

Considering `dummy` as minimal toolchain

The *dummy toolchain* is only considered as the most minimal subtoolchain if the `--add-dummy-to-minimal-toolchains` configuration option is enabled. By default, this configuration option is *disabled*.

Taking existing modules into account

You can instruct EasyBuild to take existing modules into account when determining which subtoolchain should be used for each of the dependencies, using the `--use-existing-modules` configuration option.

By default existing modules are ignored, meaning that the EasyBuild dependency resolution mechanism will pick a minimal toolchain for each dependency solely based on the available easyconfig files (if the `--minimal-toolchains` configuration option is enabled, that is).

With `--use-existing-modules` enabled, EasyBuild will first check whether modules exist for the dependencies that were built with any of the subtoolchains. If so, the module using the most minimal toolchain will determine the toolchain being used. If not, the toolchain to use will be determined based on the available easyconfig files.

Example

Consider the following (partial) easyconfig file for Python v2.7.9 with the `foss/2015b` toolchain:

```
name = 'Python'
version = '2.7.9'

toolchain = {'name': 'foss', 'version': '2015b'}

dependencies = [
    ('zlib', '1.2.8'),
]
```

When the `--minimal-toolchains` configuration option is enabled, EasyBuild will also consider the subtoolchains `GCC/4.9.3` and `gompi/2015b` of the `foss/2015b` toolchain (in that order) as potential minimal toolchains when determining the toolchain to use for dependencies.

So, for the zlib v1.2.8 dependency included in the example above, the following scenarios are possible:

- without the use of `--minimal-toolchains`, EasyBuild will only consider the `foss/2015b` toolchain for zlib, even if other zlib easyconfigs using a compatible toolchain are available
- if (only) `--minimal-toolchains` is enabled, EasyBuild will search for an easyconfig file for zlib v1.2.8 using the `GCC/4.9.3` toolchain; if no such easyconfig file is found, it will continue searching using the `gompi/2015b` toolchain, and finally the `foss/2015b` toolchain
- if `--add-dummy-to-minimal-toolchains` is also enabled, EasyBuild will try locating an easyconfig file for zlib v1.2.8 that uses the `dummy` toolchain prior to consider the `GCC/4.9.3` toolchain
- additionally, with `--use-existing-modules` enabled, EasyBuild will first check whether a `zlib` module for version 1.2.8 built with the (sub)toolchains being considered exists; if not, it will search for an easyconfig file for zlib as outlined above

4.7 Packaging support

Contents

Note: Packaging support was added as an experimental feature in EasyBuild v2.2.0 (cfr. *Experimental features*). Since EasyBuild v2.5.0, it is considered stable.

4.7.1 Prerequisites

EasyBuild leverages FPM to create binary packages (RPMs, Debian files, etc.).

Hence, FPM must be available in some way or another. One way is via EasyBuild, for example by installing a module for FPM using once of the available easyconfig files.

EasyBuild will also take care of installing Ruby for you (which is required for FPM itself):

```
$ export EASYBUILD_PREFIX=/home/example

$ eb FPM-1.3.3-Ruby-2.1.6.eb --robot
[...]
== building and installing Ruby/2.1.6...
[...]
== COMPLETED: Installation ended successfully
[...]
== building and installing FPM/1.3.3-Ruby-2.1.6...
[...]
== COMPLETED: Installation ended successfully
== Results of the build can be found in the log file /home/example/software/FPM/1.3.3-Ruby-2.1.6/e
== Build succeeded for 2 out of 2

$ module load FPM/1.3.3-Ruby-2.1.6

$ fpm --version
1.3.3
```

4.7.2 Configuration options

Several configuration options related to packaging support are available.

- `--package`:
  - enables packaging; other options will be void unless this option is enabled
- `--package-tool=<tool>`:
  - specifies which tool you wish to package with; for now, only `fpm` is supported (and is set as default)
- `--package-type=<type>`:
  - specifies which type of package you wish to build, which is passed through to `fpm` (as target type); examples include: `rpm` (default), `deb`, ... (see https://github.com/jordansissel/fpm/wiki#overview)
- `--package-naming-scheme=<PNS>`:
  - specifies which package naming scheme to use; default: `EasyBuildPNS`

- `--packagepath`:
  - specifies destination path of packages being built
- `--package-release`:
  - specifies the package release (default: `1`); typically, this should be an integer value

Note: Changing the package naming scheme should be done with caution. For example, RPM will only allow one package of a particular *name* to be installed, so if you wish multiple versions of a package to be installed at the same time you need to ensure variables like the software version are included in the package name.

4.7.3 Usage

To make EasyBuild generate packages, just use `--package`. By default, this will make EasyBuild leverage FPM to create RPMs:

```
$ export EASYBUILD_PREFIX=/home/example
$ eb --package Perl-5.20.1-GCC-4.9.2-bare.eb --robot
[...]
== building and installing Perl/5.20.1-GCC-4.9.2-bare...
== fetching files...
== creating build dir, resetting environment...
== unpacking...
== patching...
== preparing...
== configuring...
== building...
== testing...
== installing...
== taking care of extensions...
== postprocessing...
== sanity checking...
== cleaning up...
== creating module...
== packaging...
== COMPLETED: Installation ended successfully
== Results of the build can be found in the log file /home/example/software/Perl/5.20.1-GCC-4.9.2
== Build succeeded for 1 out of 1
```

Packages will be located in the directory indicated by the `--packagepath` configuration option; by default, this corresponds to `$prefix/packages`.

By defauilt, the package will have the following properties:

```
$ rpm -qip --requires --provides /home/example/packages/Perl-5.20.1-GCC-4.9.2-bare.eb2.2.0-1.x86_
Name        : Perl-5.20.1-GCC-4.9.2-bare
Version     : eb2.2.0
Release     : 1
Architecture: x86_64
Install Date: (not installed)
Group       : default
Size        : 64539427
License     : unknown
Signature   : (none)
Source RPM  : Perl-5.20.1-GCC-4.9.2-bare.eb2.2.0-1.x86_64.src.rpm
Build Date  : Tue 07 Jul 2015 11:27:54 PM EDT
Build Host  : 59e46bbf1cd0
Relocations : /
Packager    : <easybuild@59e46bbf1cd0>
Vendor      : easybuild@59e46bbf1cd0
URL         : http://example.com/no-uri-given
```

```
Summary     : no description given
Description :
no description given
GCC-4.9.2-dummy-dummy
rpmlib(PartialHardlinkSets) <= 4.0.4-1
rpmlib(PayloadFilesHavePrefix) <= 4.0-1
rpmlib(CompressedFileNames) <= 3.0.4-1
Perl-5.20.1-GCC-4.9.2-bare
Perl-5.20.1-GCC-4.9.2-bare = eb2.2.0-1
Perl-5.20.1-GCC-4.9.2-bare(x86-64) = eb2.2.0-1
```

4.7.4 Packaging existing installations

To create packages for existing software installations (performed using EasyBuild), combine `--package` with `--skip`:

```
$ eb --package --skip Perl-5.20.1-GCC-4.9.2-bare.eb --robot
[...]
== building and installing Perl/5.20.1-GCC-4.9.2-bare...
== fetching files...
== creating build dir, resetting environment...
== unpacking [skipped]
== patching [skipped]
== preparing...
== configuring [skipped]
== building [skipped]
== testing [skipped]
== installing [skipped]
== taking care of extensions...
== postprocessing [skipped]
== sanity checking...
== cleaning up...
== creating module...
== packaging...
== COMPLETED: Installation ended successfully
== Results of the build can be found in the log file /home/example/software/Perl/5.20.1-GCC-4.9.2
== Build succeeded for 1 out of 1
```

4.8 Partial installations

Several ways of performing partial installations are supported. These may be useful when debugging a particular issue with the installation procedure being performed by EasyBuild, updating existing software installations or module files, or after changing the EasyBuild configuration (e.g., switching to module files in Lua syntax or a different module naming scheme).

- *Stopping the installation procedure after a step using -s/–stop*
- *Installing additional extensions using -k/-skip*
- *Only (re)generating (additional) module files using –module-only*
  - *Only (re)generating (existing) module file*
  - *Generating additional module files*

4.8.1 Stopping the installation procedure *after* a step using `-s`/`--stop`

To stop the installation procedure *after* a specific step in the installation procedure, the `-s`/`--stop` command line option can be used; the name of the step must be supplied as an argument.

The following step names are recognized (listed in execution order): `fetch`, `ready`, `source`, `patch`, `prepare`, `configure`, `build`, `test`, `install`, `extensions`, `package`, `postproc`, `sanitycheck`, `cleanup`, `module`, `testcases`.

Example usage:

```
$ eb GCC-4.9.2.eb --stop configure
== temporary log file in case of crash /tmp/eb-X2Z0b7/easybuild-mGxmNb.log
== processing EasyBuild easyconfig /home/example/GCC-4.9.2.eb
== building and installing GCC/4.9.2...
== fetching files...
== creating build dir, resetting environment...
== unpacking...
== patching...
== preparing...
== configuring...
== COMPLETED: Installation STOPPED successfully
== Results of the build can be found in the log file /dev/shm/example/GCC/4.9.2/dummy-dummy/easybu
== Build succeeded for 1 out of 1
== temporary log file(s) /tmp/eb-X2Z0b7/easybuild-mGxmNb.log* have been removed.
== temporary directory /tmp/eb-X2Z0b7 has been removed.
```

4.8.2 Installing additional extensions using `-k`/`--skip`

For software applications that may include *Extensions*, it is often required to install one or more additional extensions without having to reinstall the software package (and all extensions) from scratch. For this purpose, the `-k`/`--skip` command line option is available.

To actually skip an existing software installation and all installed extensions, a corresponding module must be available already; if not, the installation procedure will be performed from scratch. To trigger the installation of missing extensions, `--force` must be used as well; without it, the installation procedure will be skipped as a whole (since the module is already available).

When `--skip` is combined with `--force`, EasyBuild will:

1. ensure that all (extension) sources are available (and try to fetch them if needed);
2. prepare the build environment;
3. check which extensions have not been installed yet;
4. install the missing extensions;
5. run the sanity check (which includes checking that all extensions are available)
6. regenerate the module file (since it contains a list of installed extensions)

Example usage:

```
$ eb Python-2.7.9-intel-2015a.eb --skip
...
== Python/2.7.9-intel-2015a is already installed (module found), skipping
== No easyconfigs left to be built.
== Build succeeded for 0 out of 0
```

```
$ eb Python-2.7.9-intel-2015a.eb --skip --force
...
== building and installing Python/2.7.9-intel-2015a...
...
== configuring [skipped]
== building [skipped]
== testing [skipped]
== installing [skipped]
== taking care of extensions...
...
```

```
== sanity checking...
== cleaning up...
== creating module...
== COMPLETED: Installation ended successfully
```

Note: Upgrading of extensions to a newer version does not work (yet) using `--skip`, because the way in which extensions are checked for availability, i.e. the extensions filter, is (usually) version-agnogstic.

Note: The 'skipsteps' easyconfig parameter has a different purpose, i.e. to specify which installation steps should *always* be skipped when the installation of a particular software package is performed, no matter whether the software or corresponding module is already available or not.

4.8.3 Only (re)generating (additional) module files using `--module-only`

Since EasyBuild v2.1, it is possible to only (re)generate the module file that matches the specifications in the easyconfig file, using `--module-only`. Depending on the use case, additional options should be supplied.

Usually `--force` is also required, either to ignore the existing module file (if the module is available under the same name as the one being (re)generated), or to skip the sanity check that verifies the software installation (if no software installation is available).

Combining `--module-only` with `--installpath-modules` is also a common use case, to generate the module file in a (test) location other than the software installation prefix (see *Software and modules install path (–installpath, –installpath-software, –installpath-modules)*).

Note: Although `--module-only` was already supported in EasyBuild v2.1.0, we strongly recommend to use EasyBuild v2.1.1 or a more recent version, because of some critical bug fixes with respect to `--module-only` (see *v2.1.1 (May 18th 2015)*).

Use cases:

- *Only (re)generating (existing) module file*
- *Generating additional module files*

Only (re)generating (existing) module file

To only generate a module file (i.e., skip actually building and installing the software), or to regenerate an existing module file, `--module-only` can be used.

In the former case, enabling `--force` is required because the sanity check step that verifies whether the installation produced the expected files and/or directories is not skipped unless forced. In the latter case, `--force` must be used to make EasyBuild ignore that the module is already available according to the modules tool.

Example usage:

- only generate module file:

```
$ module avail GCC
--------------------------- /home/example/.local/modules/all ----------------------------
GCC/4.8.2

$ eb GCC-5.1.0.eb --module-only --force
...
== building and installing GCC/5.1.0...
== fetching files [skipped]
```

```
...
== configuring [skipped]
== building [skipped]
== testing [skipped]
== installing [skipped]
...
== sanity checking [skipped]
== cleaning up [skipped]
== creating module...
== COMPLETED: Installation ended successfully
...

$ module avail GCC

---------------------------- /home/example/.local/modules/all -----------------------------
GCC/4.8.2     GCC/5.1.0
```

- regenerate existing module file:

```
$ module avail GCC/4.8.2

---------------------------- /home/example/.local/modules/all -----------------------------
GCC/4.8.2

$ ls -l /home/example/.local/modules/all/GCC/4.8.2
-rw-rw-r-- 1 example example 1002 Jan 11 17:19 /home/example/.local/modules/all/GCC/4.8.2

$ eb GCC-4.8.2.eb --module-only --force
...
== building and installing GCC/4.8.2...
...
== sanity checking [skipped]
== creating module...
== COMPLETED: Installation ended successfully
...

$ ls -l /home/example/.local/modules/all/GCC/4.8.2
-rw-rw-r-- 1 example example 1064 Apr 30 10:54 /home/example/.local/modules/all/GCC/4.8.2
```

Generating additional module files

Generating an additional module file, next to the one(s) already available, is also supported. This can achieved by combining `--module-only` with additional configuration options that apply to the module generation.

Examples:

- to generate a module file in Lua syntax, next to an already existing module file in Tcl syntax, `--module-only --module-syntax=Lua` can be used:

```
$ module avail GCC/4.8.2

---------------------------- /home/example/.local/modules/all -----------------------------
GCC/4.8.2

$ ls -l /home/example/.local/modules/all/GCC/4.8.2*
-rw-rw-r-- 1 example example 1064 Apr 30 10:54 /home/example/.local/modules/all/GCC/4.8.2

$ eb GCC-4.8.2.eb --modules-tool=Lmod --module-only --module-syntax=Lua --force
...
== building and installing GCC/4.8.2...
...
== sanity checking [skipped]
== creating module...
```

```
== COMPLETED: Installation ended successfully
...

$ ls -l /home/example/.local/modules/all/GCC/4.8.2*
-rw-rw-r-- 1 example example 1064 Apr 30 10:54 /home/example/.local/modules/all/GCC/4.8.2
-rw-rw-r-- 1 example example 1249 Apr 30 11:56 /home/example/.local/modules/all/GCC/4.8.2.lua
```

Note: Since only Lmod can consume module files in Lua syntax, it must be used as a modules tool; see also *Module files syntax (–module-syntax)*.

Only changing the syntax of the module file does not affect the module name, so Lmod will report the module as being available. Hence, `--force` must be used here as well.

- to generate a module file using a different naming scheme, `--module-only` can be combined with `--module-naming-scheme`:

```
$ eb --installpath-modules=$HOME/test/modules --module-only --module-naming-scheme=Hierarchic
...
== building and installing Core/GCC/4.8.2...
...
== sanity checking [skipped]
== creating module...
== COMPLETED: Installation ended successfully

$ module unuse $HOME/.local/modules/all
$ module use $HOME/test/modules/all
$ module avail

---------------------------- /home/example/test/modules/all -----------------------------
Core/GCC/4.8.2
```

Note: Modules that are generated used different module naming schemes should *never* be mixed, hence the use of `--installpath-modules`, see also *Direct options: –installpath-software and –installpath-modules*.

Note: The modules files generated using the specified module naming scheme will most likely **not** be tied to an existing software installation in this case (unless the software installation was already there somehow), since the name of the subdirectory of the software installation prefix is also governed by the active module naming scheme. This is also why `--force` must be used in the example above (to skip the sanity check that verifies the software installation).

Thus, this is only useful to assess how the module tree would look like under a particular module naming scheme; the modules themselves are useless since they point to empty installation directories.

To tie a module file generated using to an existing software installation that was performed under a different module naming scheme, a simple module naming scheme can be implemented that mixes two modules naming schemes, by providing the name of the software installation subdirectory using one scheme, and the module names (and other metadata for module files) with the other.

An example of such a module naming scheme is `MigrateFromEBToHMNS`, which allows to generate module files using the hierarchical module naming scheme implemented by `HierarchicalMNS` for the software installed in subdirectories following the default EasyBuild module naming scheme `EasyBuildMNS`. The `MigrateFromEBToHMNS` module naming scheme is available since EasyBuild v2.2.0.

4.9 Submitting jobs using `--job`

Topics:

- *Quick introduction to –job*
- *Configuring –job*
- *Usage of –job*
- *Examples*

4.9.1 Quick introduction to `--job`

Using the `--job` command line option, you can instruct EasyBuild to submit jobs for the installations that should be performed, rather than performing the installations locally on the system you are on.

If dependency resolution is enabled using `--robot` (see also *Enabling dependency resolution, –robot / -r and –robot-paths*), EasyBuild will submit separate jobs and set dependencies between them to ensure they are run in the order dictated by the software dependency graph(s).

4.9.2 Configuring `--job`

Selecting the job backend (`--job-backend`)

The job backend to be used can be specified using the `--job-backend` EasyBuild configuration option.

Since EasyBuild 2.2.0, two backends are supported:

- `PbsPython` *(default)*
  - `pbs_python` version 4.1.0 (or more recent) required (see https://oss.trac.surfsara.nl/pbs_python)
  - **note**: requires TORQUE resource manager (see http://www.adaptivecomputing.com/products/open-source/torque/)
- `GC3Pie` *(recommended)*
  - `GC3Pie` version 2.4.0 (or more recent) required (https://gc3pie.readthedocs.org)
  - works with different resource managers and job schedulers, including TORQUE, SLURM, etc.
  - **note**: requires that a GC3Pie configuration file is provided, see *Configuring the job backend (–job-backend-config)*

For historical reasons, `PbsPython` is still the default job backend in EasyBuild version 2.x.

Configuring the job backend (`--job-backend-config`)

To configure the job backend, the path to a configuration file must be specified via `--job-backend-config`.

- for `PbsPython` backend: *(irrelevant, no configuration file required)*
- for `GC3Pie` backend: see https://gc3pie.readthedocs.org/en/latest/users/configuration.html
  - example configuration files are available at *Example configurations for GC3Pie job backend*

Number of requested cores per job (`--job-cores`)

The number of cores that should be requested for each job that is submitted can be specified using `--job-cores` (default: *not specified*).

The mechanism for determining the number of cores to request in case `--job-cores` was *not* specified depends on which job backend is being used:

- if the `PbsPython` job backend is used, the (most common) number of available cores per workernode in the target resource is determined; this usually results in jobs requesting full workernodes (at least in terms of cores) by default
- if the `GC3Pie` job backend is used, the requested number of cores is left unspecified, which results in falling back to the default mechanism used by GC3Pie to pick a number of cores; most likely, this results in single-core jobs to be submitted by default

Maximum walltime of jobs (`--job-max-walltime`)

An integer value specifying the maximum walltime for jobs (in hours) can be specified via `--job-max-walltime` (default: 24).

For easyconfigs for which a reference required walltime is available via the `build_stats` parameter in a matching easyconfig file from the easyconfig repository (see *Easyconfigs repository (–repository, –repositorypath)*), EasyBuild will set the walltime of the corresponding job to twice that value (unless the resulting value is higher than the maximum walltime for jobs).

If no such reference walltime is available, the maximum walltime is used.

Job output directory (`--job-output-dir`)

The directory where job log files should be placed can be specified via `--job-output-dir` (default: current directory).

Job polling interval (`--job-polling-interval`)

The frequency with which the status of submitted jobs should be checked can be specified via `--job-polling-interval`, using a floating-point value representing the number of seconds between two checks (default: 30 seconds).

Note: This setting is currently only relevant to GC3Pie; see also *Submitting jobs to a GC3Pie backend*.

Target resource for job backend (`--job-target-resource`)

The target resource that should be used by the job backend can be specified using `--job-target-resource`.

- for `PbsPython` backend: hostname of TORQUE PBS server to submit jobs to (default: `$PBS_DEFAULT`)
- for `GC3Pie` backend: name of resource to submit jobs to (default: none, which implies weighted round-robin submission across all available resources)

4.9.3 Usage of `--job`

To make EasyBuild submit jobs to the job backend rather than performing the installations directly, the `--job` command line option can be used.

This following assumes that the required configuration settings w.r.t. the job backend to use are in place, see *Configuring –job*.

Submitting jobs to a `PbsPython` backend

When using the `PbsPython` backend, EasyBuild will submit separate jobs for each installation to be performed to TORQUE, and then exit reporting a list of submitted jobs.

To ensure that the installations are performed in the order dictated by the software dependency graph, dependencies between installations are specified via *job dependencies*, more specifically using the `afterany` dependency relation (see http://docs.adaptivecomputing.com/mwm/Content/topics/jobAdministration/jobdependencies.html for more information).

See also *Example: submitting installations to TORQUE via pbs_python*.

Note: Submitted jobs will be put on hold until all jobs have been submitted. This is required to ensure that the dependencies between jobs can be specified correctly; if a job would run to completion before other jobs that depend on it were submitted, the submission process would fail.

Submitting jobs to a `GC3Pie` backend

When using the `GC3Pie` backend, EasyBuild will create separate tasks for each installation to be performed and supply them to GC3Pie, which will then take over and pass the installations through as jobs to the available resource(s) (see also *Configuring the job backend (–job-backend-config)*).

To ensure that the installations are performed in the order dictated by the software dependency graph, dependencies between installations are specified to GC3Pie as inter-task dependencies. GC3Pie will then gradually feed the installations to its available resources as their dependencies have been satisfied.

Any log messages produced by GC3Pie are included in the EasyBuild log file, and are tagged with `gc3pie`.

See also *Example: submitting installations to SLURM via GC3Pie*.

Note: The `eb` process will not exit until the full set of tasks that GC3Pie was provided with has been processed. An overall progress report will be printed regularly (see also *Job polling interval (–job-polling-interval)*). As such, it is advised to run the `eb` process in a screen/tmux session when using the GC3Pie backend for `--job`.

4.9.4 Examples

Example configurations for GC3Pie job backend

When using GC3Pie as a job backend, a configuration file must be provided via `--job-backend-config`. This section includes a couple of examples of GC3Pie configuration files (see also https://gc3pie.readthedocs.org/en/latest/users/configuration.html).

Example GC3Pie configuraton for local system

```
[resource/localhost]
enabled = yes
type = shellcmd
frontend = localhost
transport = local
max_memory_per_core = 10GiB
max_walltime = 100 hours
# max # jobs ~= max_cores / max_cores_per_job
max_cores_per_job = 1
max_cores = 4
architecture = x86_64
auth = none
```

```
override = no
resourcedir = /tmp/gc3pie
```

Example GC3Pie configuration for PBS/TORQUE

```
[resource/pbs]
enabled = yes
type = pbs

# use settings below when running GC3Pie on the cluster front-end node
frontend = localhost
transport = local
auth = none

max_walltime = 2 days
# max # jobs ~= max_cores / max_cores_per_job
max_cores_per_job = 16
max_cores = 1024
max_memory_per_core = 2 GiB
architecture = x86_64

# to add non-std options or use PBS/TORQUE tools located outside of
# the default PATH, use the following:
#qsub = /usr/local/bin/qsub -q my-special-queue
```

Example GC3Pie configuration for SLURM

```
[resource/slurm]
enabled = yes
type = slurm

# use settings below when running GC3Pie on the cluster front-end node
frontend = localhost
transport = local
auth = none

max_walltime = 2 days
# max # jobs ~= max_cores / max_cores_per_job
max_cores_per_job = 16
max_cores = 1024
max_memory_per_core = 2 GiB
architecture = x86_64

# to add non-std options or use SLURM tools located outside of
# the default PATH, use the following:
#sbatch = /usr/bin/sbatch --mail-type=ALL
```

Example: submitting installations to SLURM via GC3Pie

When submitting jobs to the `GC3Pie` job backend, the `eb` process will not exit until all tasks have been completed. A job overview will be printed every N seconds (see *Job polling interval (–job-polling-interval)*).

Jobs are only submitted to the resource manager (SLURM, in this case) when all task dependencies have been resolved.

```
$ export EASYBUILD_JOB_BACKEND=GC3Pie
$ export EASYBUILD_JOB_BACKEND_CONFIG=$PWD/gc3pie.cfg
$ eb GCC-4.6.0.eb OpenMPI-1.8.4-GCC-4.9.2.eb --robot --job --job-cores=16 --job-max-walltime=10
```

```
== temporary log file in case of crash /tmp/eb-ivAiwD/easybuild-PCgmCB.log
== resolving dependencies ...
== GC3Pie job overview: 2 submitted (total: 9)
== GC3Pie job overview: 2 running (total: 9)
== GC3Pie job overview: 2 running (total: 9)
...
== GC3Pie job overview: 4 terminated, 4 ok, 1 submitted (total: 9)
== GC3Pie job overview: 4 terminated, 4 ok, 1 running (total: 9)
...
== GC3Pie job overview: 8 terminated, 8 ok, 1 running (total: 9)
== GC3Pie job overview: 9 terminated, 9 ok (total: 9)
== GC3Pie job overview: 9 terminated, 9 ok (total: 9)
== Done processing jobs
== GC3Pie job overview: 9 terminated, 9 ok (total: 9)
== Submitted parallel build jobs, exiting now
== temporary log file(s) /tmp/eb-ivAiwD/easybuild-PCgmCB.log* have been removed.
== temporary directory /tmp/eb-ivAiwD has been removed.
```

Checking which jobs have been submitted to SLURM at regular intervals reveals that indeed only tasks for which all dependencies have been processed are actually submitted as jobs:

```
$ squeue -u $USER
JOBID       USER       ACCOUNT           NAME     REASON   START_TIME     END_TIME  TIME_LEFT NODE
6161545     easybuild  example      GCC-4.9.2       None 2015-07-01T1 2015-07-01T2    9:58:55
6161546     easybuild  example      GCC-4.6.0       None 2015-07-01T1 2015-07-01T2    9:58:55

$ squeue -u $USER
JOBID       USER       ACCOUNT           NAME     REASON   START_TIME     END_TIME  TIME_LEFT NODE
6174527     easybuild  example Automake-1.15-  Resources          N/A          N/A   10:00:00

$ squeue -u $USER
JOBID       USER       ACCOUNT           NAME     REASON   START_TIME     END_TIME  TIME_LEFT NODE
6174533     easybuild  example OpenMPI-1.8.4-       None 2015-07-03T0 2015-07-03T1    9:55:59
```

Example: submitting installations to TORQUE via pbs_python

Using the `PbsPython` job backend, `eb` submits jobs directly to Torque for processing, and exits as soon as all jobs have been submitted:

```
$ eb GCC-4.6.0.eb OpenMPI-1.8.4-GCC-4.9.2.eb --robot --job
== temporary log file in case of crash /tmp/eb-OMNQAV/easybuild-9fTuJA.log
== resolving dependencies ...
== List of submitted jobs (9): GCC-4.6.0 (GCC/4.6.0): 508023.example.pbs; GCC-4.9.2 (GCC/4.9.2): 5
libtool-2.4.2-GCC-4.9.2 (libtool/2.4.2-GCC-4.9.2): 508025.example.pbs; M4-1.4.17-GCC-4.9.2 (M4/1.4
8026.example.pbs; Autoconf-2.69-GCC-4.9.2 (Autoconf/2.69-GCC-4.9.2): 508027.example.pbs; Automake-
tomake/1.15-GCC-4.9.2): 508028.example.pbs; numactl-2.0.10-GCC-4.9.2 (numactl/2.0.10-GCC-4.9.2): 5
hwloc-1.10.0-GCC-4.9.2 (hwloc/1.10.0-GCC-4.9.2): 508030.example.pbs; OpenMPI-1.8.4-GCC-4.9.2 (Open
2): 508031.example.pbs
== Submitted parallel build jobs, exiting now
== temporary log file(s) /tmp/eb-OMNQAV/easybuild-9fTuJA.log* have been removed.
== temporary directory /tmp/eb-OMNQAV has been removed.

$ qstat -a

example.pbs:
                                                                  Req'd    Req'd
Job ID               Username    Queue    Jobname          SessID  NDS   TSK   Memory   Time    S
-------------------- ----------- -------- ---------------- ------ ----- ------ ------ --------- -
508023.example.pbs  easybuild   batch    GCC-4.6.0           --      1     16    --   24:00:00 R
508024.example.pbs  easybuild   batch    GCC-4.9.2           --      1     16    --   24:00:00 Q
508025.example.pbs  easybuild   batch    libtool-2.4.2-GC    --      1     16    --   24:00:00 H
508026.example.pbs  easybuild   batch    M4-1.4.17-GCC-4.    --      1     16    --   24:00:00 H
```

```
508027.example.pbs  easybuild   batch    Autoconf-2.69-GC    --       1      16     --   24:00:00 H
508028.example.pbs  easybuild   batch    Automake-1.15-GC    --       1      16     --   24:00:00 H
508029.example.pbs  easybuild   batch    numactl-2.0.10-G    --       1      16     --   24:00:00 H
508030.example.pbs  easybuild   batch    hwloc-1.10.0-GCC    --       1      16     --   24:00:00 H
508031.example.pbs  easybuild   batch    OpenMPI-1.8.4-GC    --       1      16     --   24:00:00 H
```

Holds are put in place to ensure that the jobs run in the order dictated by the dependency graph(s). These holds are released by the TORQUE server as soon as they jobs on which they depend have completed:

```
$ qstat -a

example.pbs:
                                                                                  Req'd    Req'd
Job ID               Username    Queue    Jobname          SessID  NDS   TSK    Memory   Time    S
-------------------- ----------- -------- ---------------- ------ ----- ------ ------ --------- -
508025.example.pbs  easybuild   batch    libtool-2.4.2-GC    --       1      16     --   24:00:00 Q
508026.example.pbs  easybuild   batch    M4-1.4.17-GCC-4.    --       1      16     --   24:00:00 Q
508027.example.pbs  easybuild   batch    Autoconf-2.69-GC    --       1      16     --   24:00:00 H
508028.example.pbs  easybuild   batch    Automake-1.15-GC    --       1      16     --   24:00:00 H
508029.example.pbs  easybuild   batch    numactl-2.0.10-G    --       1      16     --   24:00:00 H
508030.example.pbs  easybuild   batch    hwloc-1.10.0-GCC    --       1      16     --   24:00:00 H
508031.example.pbs  easybuild   batch    OpenMPI-1.8.4-GC    --       1      16     --   24:00:00 H

...

$ qstat -a

example.pbs:
                                                                                  Req'd    Req'd
Job ID               Username    Queue    Jobname          SessID  NDS   TSK    Memory   Time    S
-------------------- ----------- -------- ---------------- ------ ----- ------ ------ --------- -
508031.example.pbs  easybuild   batch    OpenMPI-1.8.4-GC    --       1      16     --   24:00:00 R
```

4.10 Using external modules

Since EasyBuild v2.1, support for using modules that were not installed via EasyBuild is available. We refer to such modules as *external modules*.

This can be useful to reuse vendor-supplied modules, for example on Cray systems.

Contents

4.10.1 Using external modules as dependencies

External modules can be used as dependencies, by including the module name in the `dependencies` list (see *Dependencies*), along with the `EXTERNAL_MODULE` constant marker.

For example, to specify the readily available module `fftw/3.3.4.2` as a dependency:

```
dependencies = [('fftw/3.3.4.2', EXTERNAL_MODULE)]
```

For such dependencies, EasyBuild will:

- load the module before initiating the software build and install procedure
- include a '`module load`' statement in the generated module file (for non-build dependencies)

If the specified module is not available, EasyBuild will exit with an error message stating that the dependency can not be resolved because the module could not be found. It will *not* search for a matching easyconfig file in order to try and install the module to resolve the dependency.

4.10.2 Metadata for external modules

Since very little information is available for external modules based on the dependency specification alone (i.e., only the module name), metadata can be supplied to EasyBuild for external modules.

Using the `--external-modules-metadata` configuration option, the location of one or more files can be specified that provide such metadata. The files are expected to be in INI format, with a section per module name and key-value assignments denoting the metadata specific to that module.

Format:

```
[modulename]
key1 = value1
key2 = value2, value3
```

Default

Up until EasyBuild v2.6.0, no metadata for external modules was available by default.

Since EasyBuild v2.7.0, a file providing metadata for Cray-provided modules on Cray systems is included, and is active by default (i.e., unless other files are specified via `--external-modules-metadata`); see https://github.com/hpcugent/easybuild-framework/blob/master/etc/cray_external_modules_metadata.cfg.

Example

For example, the following file provides metadata for a handful of modules that may be provided on Cray systems:

```
[cray-hdf5/1.8.13]
name = HDF5
version = 1.8.13
prefix = HDF5_DIR

[cray-hdf5-parallel/1.8.13]
name = HDF5
version = 1.8.13
prefix = /opt/cray/hdf5-parallel/1.8.13/GNU/49

[cray-netcdf/4.3.2]
name = netCDF, netCDF-Fortran
version = 4.3.2, 4.3.2
prefix = NETCDF_DIR

[fftw/3.3.4.5]
name = FFTW
version = 3.3.4.5
prefix = FFTW_INC/..
```

Supported metadata values

The following keys are supported:

- `name`: specifies the software name(s) that is (are) provided by the module
- `version`: specifies the software version(s) that is (are) provided by the module
- `prefix`: specifies the installation prefix of the software that is provided by the module

For `prefix`, a couple of different options are avaialble, i.e.:

- specifying an absolute path (cfr. `cray-hdf5-parallel/1.8.13` in the example above)
- specifying the environment variable that is defined by the module and provides the installation prefix (cfr. `cray-netcdf/4.3.2` in the example above)
  - this can be optionally combined with a relative path that serves as a 'correction' (cfr. `fftw/3.3.4.5` in the example above) *[supported since EasyBuild v2.7.0]*

Any other keys are simply ignored.

Note: When both `name` and `version` are specified, they must provide an *equal number of values* (see for example the `cray-netcdf` example above).

Handling of provided metadata

Using the provided metadata, EasyBuild will define environment variables that are also defined by modules that are generated by EasyBuild itself, if an external module for which metadata is available is loaded as a dependency.

In particular, for each software name that is specified:

- the corresponding environment variable `$EBROOT<NAME>` is defined to the specified `prefix` value (if any)
- the corresponding environment variable `$EBVERSION<NAME>` is defined to the corresponding `version` value (if any)

For example, for the external modules for which metadata is provided in the example above, the following environment variables are set in the build environment when the module is used as a dependency:

- for `cray-hdf5/1.8.1.13`:
  - `$EBROOTHDF5` = `$HDF5_DIR`
  - `$EBVERSIONHDF5` = `1.8.13`
- for `cray-hdf5-parallel/1.8.13`:
  - `$EBROOTHDF5` = `/opt/cray/hdf5-parallel/1.8.13/GNU/49`
  - `$EBVERSIONHDF5` = `1.8.13`
- for `cray-netcdf/4.3.2`:
  - `$EBROOTNETCDF` = `$NETCDF_DIR`
  - `$EBROOTNETCDFMINFORTRAN` = `$NETCDF_DIR`
  - `$EBVERSIONNETCDF` = `4.3.2`
  - `$EBVERSIONNETCDFMINFORTRAN` = `4.3.2`
- for `fftw/3.3.4.5`:
  - `$EBROOTFFTW` = `$FFTW_INC/../`
  - `$EBVERSIONFFTW` = `3.3.4.5`

The `get_software_root` and `get_software_version` functions that are commonly used occasionally in easyblocks pick up the `$EBROOT*` and `$EBVERSION*` environment variables, respectively.

CHAPTER 5

Other topics

5.1 Code style

The code style we follow in the EasyBuild code repository is mainly dictated by the Python standard PEP8.

Highlighted PEP8 code style rules:

- **use 4 spaces for indentation, do not use tabs** for example, use `:set tabstop 4` and `:set expandtab` in Vim
- **indent items in a list at an extra 4 spaces** nested lists can be indented at the same indentation as the first item in the list if it is on the first line, closing brackets must match visual indentation
- use *optional underscores*, not camelCase, for variable, function and method names (i.e. `poll.get_unique_voters()`, **not** `poll.getUniqueVoters`)
- use `InitialCaps` for class names
- in docstrings, don't use "action words"

The only (major) exception to PEP8 is our preference for longer line lengths: line lengths **must be limited to 120 characters**, and should by preference be *shorter than 100 characters* (as opposed to the 80-character limit in PEP8).

5.1.1 Notes

Style guides that go a step beyond PEP8:

- http://www.gramps-project.org/wiki/index.php?title=Programming_guidelines
- http://code.google.com/p/volatility/wiki/StyleGuide

Automatic rewriting of Python code: http://pypi.python.org/pypi/PythonTidy/1.22

`pep8` might be a useful tool to check PEP8 compliance: https://github.com/jcrocholl/pep8

5.2 Unit tests

Since EasyBuild v1.0, an extensive suite of unit tests has been implemented. The unit tests have become an indispensable factor in keeping EasyBuild stable and backward-compatible during development.

We refer to the available tests as unit tests, even though they may not be *unit* tests in the strict sense of the word. Some tests are actually end-to-end tests or integration tests, see also http://en.wikipedia.org/wiki/Software_testing#Testing_levels.

Following the test-driven development paradigm, additional unit tests are implemented when new features are added or when bugs are uncovered (and fixed).

5.2.1 What the unit tests are *not*

Each of the EasyBuild unit tests aim to test a specific characteristic of EasyBuild, e.g., a configuration option, a particular function or method in the EasyBuild framework, some specific feature, how EasyBuild handles a particular type of input, etc.

The unit tests do *not* test the build and installation process of particular supported software packages (other than a handful of toy ones included in the tests themselves), let alone test the software installations obtained using EasyBuild themselves.

Each stable EasyBuild release is subjected to a (time- and resource-consuming) *regression test*, however, which is out of scope here.

5.2.2 Available unit test suites

Three different unit test suites are available for EasyBuild, each of which tied to one of the EasyBuild code repositories on GitHub: *EasyBuild framework*, *easyblocks*, *easyconfigs*.

EasyBuild framework unit tests

The unit test suite for the EasyBuild framework is by far the most extensive one, in terms of code size, coverage and the amount of effort that is put into it.

These tests probe the functionality provided by the EasyBuild framework, ranging from standard operation (building and installing software, and generating module files) to specific configuration options, selected functional aspects, optional features and edge cases.

At the time of writing (EasyBuild v2.0), more than 250 tests were implemented, organised in 28 subgroups.

Running the full EasyBuild framework unit test suite takes about 15-40 minutes, depending on your system resources and testing configuration (see for example https://jenkins1.ugent.be/view/EasyBuild/).

easyblocks unit tests

The easyblocks unit test suite consists of a coupe of light-weight tests that are run per easyblock:

- initialising the easyblock, to check for Python syntax errors and faulty imports
- checking for the use of deprecated (or no longer supported) functionality

Running the full easyblocks unit test suite takes less than one minute.

easyconfigs unit tests

The easyconfigs unit test suite consists a couple of tests for each of the available easyconfig files, followed by two large-scale overall tests.

For each of the available easyconfig files, the following aspects are tested:

- parsing the easyconfig file, to make sure no syntax errors are present
- verifying that the filename matches the contents of the easyconfig file (software name/version, version suffix and toolchain name/version)
- creating an `EasyBlock` instance using the parsed easyconfig, to check whether mandatory easyconfig parameters are defined
- ensuring that all required patch files are available (right next to the easyconfig file)
- making sure that the specified sanity check paths adher to the requirements, i.e. only (and both) the `files`/`dirs` keys are listed, with the value for either one being non-empty
- checking for the use of deprecated (or no longer supported) functionality

If these tests pass for each and every available easyconfig file, two additional overall tests are run, i.e.:

- ensuring that an easyconfig file is available for each specified dependency
- checking whether any version conflicts occur in the dependency graph for each easyconfig file

Running the full easyconfigs unit test suite should only take a couple of minutes.

5.2.3 Applications

The unit test suites are automatically triggered by Jenkins for:

- each pull request (update), see https://jenkins1.ugent.be/view/pull-request-builder/
- each (set of) change(s) that is merged in (usually via a pull request)

The status of the different unit test suites is tracked separately for the `master` and `develop` branches of the EasyBuild code repositories:

- `master` (stable, latest EasyBuild release): https://jenkins1.ugent.be/view/EasyBuild/
- `develop` (development, potentially unstable): https://jenkins1.ugent.be/view/EasyBuild%20(develop)/

We strictly adher to the policy of only merging pull requests for which the corresponding (latest) run of the *full* unit test suite passes successfully.

5.2.4 Usage

Using the unit tests is deliberately kept very simple.

Configuration

Since EasyBuild v2.0, the unit test suites are fully isolated from any (system- or user-level) EasyBuild configuration which is in place in the environment where the tests are being run.

The easyblocks and easyconfigs unit test suite are oblivious to any defined configuration options.

For the EasyBuild framework unit tests, all configuration files and `EASYBUILD_`-prefixed environment variables are ignored (see also configuration_types:).

To enable running the EasyBuild framework unit test suite under a specific configuration that differs from the default, environment variables can be defined for each of the configuration options supported by EasyBuild.

Before starting a set of EasyBuild framework tests, all defined environment variables for which the name is prefixed by `TEST_EASYBUILD_` will be injected into the test environment as environment variables prefixed with `EASYBUILD_` instead. Thus, to set a particular configuration option `--foo`, you should define the environment variable `$TEST_EASYBUILD_FOO`.

Modules tool to use for running tests

One particular configuration option worth mentioning explicitely is the modules tool that is to be used by the EasyBuild framework, which is by default the traditional Tcl/C environment modules, referred to as `EnvironmentModulesC` in EasyBuild configuration terms (see `eb --help` and `eb --avail-modules-tools`).

To run the EasyBuild framework unit tests with a particular modules tool, simply define the `$TEST_EASYBUILD_MODULES_TOOL` environment variable with the corresponding value.

Just like for EasyBuild itself, the EasyBuild framework unit test suite expects that the modules tool binary/script (`modulecmd`, `modulecmd.tcl` or `lmod`) is available via `$PATH`, see *Required modules tool*.

Installing a GitHub token for the unit tests

Some of the EasyBuild framework unit tests require that a GitHub token is in place for the `easybuild_test` user, in a non-encrypted keyring (so it can be obtained without having to provide a password).

This can be done as follows (copy-paste the GitHub token at the `Password:` prompt):

```
$ python
>>> import getpass, keyring
>>> keyring.set_keyring(keyring.backends.file.PlaintextKeyring())
>>> keyring.set_password('github_token', 'easybuild_test', getpass.getpass())
Password:
>>> exit()
```

More details about obtaining and installing a GitHub token in your keyring are available at https://github.com/hpcugent/easybuild/wiki/Review-process-for-contributions#setting-things-up.

Running

To run a full unit test suite, simply run the respective `suite` Python module.

- EasyBuild framework: `python -m test.framework.suite`
- easyblocks: `python -m test.easyblocks.suite`
- easyconfigs: `python -m test.easyconfigs.suite`

For the EasyBuild framework unit tests, each of the test subgroups can be run separately via a dedicated Python module other than `suite`, to focus on testing a particular aspect. See https://github.com/hpcugent/easybuild-framework/tree/master/test/framework for an overview of the available Python modules corresponding to subgroups of tests (note: `__init__.py` and `utilities.py` are *not* such modules).

For example, to run the full EasyBuild framework unit test suite using Lmod as a modules tool:

```
$ export TEST_EASYBUILD_MODULES_TOOL=Lmod
$ python -m test.framework.suite
```

To only run the subgroup of tests for `filetools`:

```
$ python -m test.framework.filetools
```

Since EasyBuild v2.8.2, tests can be *filtered* by name. Simply add the string to filter with to the test command.

For example, to run only the tests containing the word `load` in the subgroup `modules`, run:

```
$ python -m test.framework.modules load

Filtered ModulesTest tests using 'load', retained 2/19 tests: test_load, test_load_in_hierarchy
..
----------------------------------------------------------------------
Ran 2 tests in 2.688s

OK
```

This works with as many filter words as you want to use. For example, to run every test method in `modules` containing the words `load` or `bash`:

```
$ python -m test.framework.modules load bash
```

Results

The test results will be printed as the unit test suite progresses, potentially producing a lot of information for failing tests to help determine the cause of the failure. It may thus be useful to capture the output for later inspection, for example:

```
python -m test.framework.suite 2>&1 | tee eb_test.log
```

Note: Some tests will be skipped deliberately, because of missing optional dependencies or other provisions, for example a GitHub token. The output of the unit tests will clearly indicate which tests were skipped.

Examples

A successful run of the EasyBuild framework test suite, without skipped tests:

```
$ python -m test.framework.suite
Running tests...
----------------------------------------------------------------------
.....................................................................................................
----------------------------------------------------------------------
Ran 250 tests in 1404.897s

OK
```

A run with a couple of deliberately skipped tests and a single failed test (denoted by `F`), along with the corresponding traceback:

```
$ python -m test.framework.suite
Running tests...
----------------------------------------------------------------------
...............Skipping test_from_pr, no GitHub token available?
.Skipping test_from_pr, no GitHub token available?
......F.............(skipping GitRepository test)
..(skipping SvnRepository test)
.....................................................................................................
.Skipping test_read, no GitHub token available?
.Skipping test_read_api, no GitHub token available?
.Skipping test_walk, no GitHub token available?
.....................................................................................
======================================================================
FAIL: Test listing easyblock hierarchy.
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/tmp/example/easybuild-framework/test/framework/options.py", line 544, in test_list_easybl
    self.assertTrue(re.search(pat, outtxt), "Pattern '%s' is found in output of --list-easyblocks
AssertionError: Pattern 'EasyBlock\n' is found in output of --list-easyblocks:

----------------------------------------------------------------------
Ran 250 tests in 2641.200s

FAILED (failures=1)
ERROR: Not all tests were successful.
Log available at /tmp/example/easybuild-dy2ZTx/easybuild-tests-l0doQ2.log
```

5.3 Useful scripts

A couple of useful stand-alone scripts are provided along with the EasyBuild framework.

The latest stable version of these scripts is available in the `easybuild-framework` GitHub repository at https://github.com/hpcugent/easybuild-framework/tree/master/easybuild/scripts.

This documentation provides some information on how to use these scripts.

5.3.1 fix_broken_easyconfigs.py

download from: https://github.com/hpcugent/easybuild-framework/blob/master/easybuild/scripts/fix_broken_easyconfigs.py

To fix easyconfig files that are broken due to relying on functionality that is no longer supported in EasyBuild v2.x, the `fix_broken_easyconfigs.py` script can be used.

This script rewrites easyconfig files that are broken because they still:

- rely on the automagic fallback to the generic `ConfigureMake` easyblock (see *Automagic fallback to ConfigureMake*)
- define the `premakeopts` and/or `makeopts` eaysconfig parameters (see *Options for build command*)
- use the `shared_lib_ext` 'constant' (see *Shared library extension*)
- incorrectly use the `license` easyconfig parameter (see *Software license*)

The script accepts a list of easyconfig files or directories containing easyconfig files (`.eb`) as arguments, and will consider all easyconfig files it can find.

Only easyconfig files that are considered broken (according to one or more of the aspects listed above) are patched; other easyconfigs will be left untouched.

To determine whether `easyblock = 'ConfigureMake'` should be added in an easyconfig file that does not include any `easyblock` specification yet, the easyblocks available in the active Python search path (i.e., the ones listed in the output of `eb --list-easyblocks`, see also *List of available easyblocks, –list-easyblocks*) are taken into account.

A backup copy is created for each easyconfig file that is being patched.

Example usage:

```
$ python easybuild/scripts/fix_broken_easyconfigs.py broken.eb myeasyconfigs GCC-4.9.2.eb
== 2015-03-05 17:02:22,438 fix_broken_easyconfigs.FIX_BROKEN_EASYCONFIGS INFO Processing 3 easycon
== 2015-03-05 17:02:22,454 fix_broken_easyconfigs.FIX_BROKEN_EASYCONFIGS INFO Backed up broken.eb
== 2015-03-05 17:02:22,454 fix_broken_easyconfigs.FIX_BROKEN_EASYCONFIGS INFO broken.eb: fixed
== 2015-03-05 17:02:22,458 fix_broken_easyconfigs.FIX_BROKEN_EASYCONFIGS INFO /home/example/myeasy
== 2015-03-05 17:02:22,461 fix_broken_easyconfigs.FIX_BROKEN_EASYCONFIGS INFO GCC-4.9.2.eb: nothi
```

This results in `broken.eb` being rewritten/fixed, after creating a backup copy `broken.eb.bk`:

```
$ diff -u broken.eb.bk broken.eb
--- broken.eb.bk    2015-02-25 14:45:52.000000000 +0100
+++ broken.eb       2015-03-05 16:51:44.000000000 +0100
@@ -1,4 +1,6 @@
 # licenseheader
+easyblock = 'ConfigureMake'
+
 name = 'foo'
 version = '1.2.3'

@@ -7,7 +9,7 @@

 toolchain = {'name': 'bar', 'version': '3.2.1'}

-premakeopts = 'FOO=libfoo.%s' % shared_lib_ext
-makeopts = 'CC=gcc'
+prebuildopts = 'FOO=libfoo.%s' % SHLIB_EXT
+buildopts = 'CC=gcc'

-license = 'foo.lic'
+license_file = 'foo.lic'
```

5.3.2 `install-EasyBuild-develop.sh`

download from: https://github.com/hpcugent/easybuild-framework/blob/master/easybuild/scripts/install-EasyBuild-develop.sh

A script to set up an EasyBuild development environment. For more information, see *Installation of latest development version using provided script*.

5.3.3 `clean_gists.py`

download from: https://github.com/hpcugent/easybuild-framework/blob/master/easybuild/scripts/clean_gists.py

When using `--from-pr` in combination with `-upload-test-report` (see https://github.com/hpcugent/easybuild/wiki/Review-process-for-contributions#automated-testing-of-easyconfigs-pull-requests), you can end up with a bunch of gists in your GitHub account containing test reports, that may no longer be relevant.

To help with that the `clean_gists.py` script is available, to clean up gists containing test reports:

- `clean_gists.py -p`: delete all gists from closed pull requests (default action if no other action is specified)
- `clean_gists.py -a`: delete all gists generated by Easybuild
- `clean_gists.py -o`: delete all gists without a matching pull request (created by using `-upload-test-report` without `--from-pr`)

By default, the script will use the same GitHub account that Easybuild uses (see `--github-user`); to specify a different GitHub account, use `-g`.

The script expects that a valid GitHub token for the used GitHub account username is available, see *Installing a GitHub token (–install-github-token)*.

5.4 Deprecated functionality

Some of the functionality that was available in previous EasyBuild versions is *deprecated*, and should no longer be used.

This functionality will be removed in a future version of EasyBuild (see *Removed functionality* for more information about functionality that has been removed already).

This page:

- provides an *overview of currently deprecated functionality* together with available alternatives
- describes the *EasyBuild deprecation policy*
- explains how to easily *check whether you are still relying on deprecated functionality*

5.4.1 Overview of deprecated functionality in EasyBuild version 2.8.2

The different section below document the functionality that is deprecated in EasyBuild version 2.8.2, for which support will be removed in EasyBuild version 3.0.

For EasyBuild users:

(nothing yet)

For authors of easyconfig files:

(nothing yet)

For developers of easyblocks:

- *Report errors by raising EasyBuildError rather than using log methods*

For EasyBuild framework developers:

- *Report errors by raising EasyBuildError rather than using log methods*

Report errors by raising `EasyBuildError` rather than using log methods

Use of the `error()` and `exception()` log methods is deprecated.

- *deprecated since:* EasyBuild v2.1.0 (April'15)
- *removed in:* EasyBuild v3.0
- *alternative(s)*: **use `raise EasyBuildError(...)` instead**

The `error()` and `exception()` log methods defined by EasyBuild (in the `easybuild.tools.build_log` module) do not match the semantics of the standard Python log methods, in the sense that they also raise an exception next to logging messages.

This may cause problems when 3rd party libraries (e.g., gc3pie) are being used by EasyBuild, since they may be using these log methods without expecting an exception being raised.

The custom definitions for the `error()` and `exception()` log methods will be removed in EasyBuild v3.0.

Hence, these log methods should no longer be used to report errors since they will not raise an exception anymore once their custom definitions are removed. Note that this applies both to the EasyBuild framework and to (custom) easyblocks.

To report errors, an `EasyBuildError` should be raised instead. For example:

```
# make sure config.sh script is there
if not os.path.exists(os.path.join(self.builddir, 'config.sh')):
    raise EasyBuildError("config.sh script is missing in %s", self.builddir)
```

5.4.2 Deprecation policy

With every EasyBuild release, we try very hard to maintain *backward compatibility*. That is, EasyBuild version `X.Y` should still build software packages that could be built with EasyBuild version `X.(Y-1)`, without requiring modifications to the used easyconfig file or easyblock.

However, every now and then a breaking change needs to be made to EasyBuild, because of design decisions or to resolve mistakes that were made in the past. These changes involve *deprecating* the behaviour or functionality we want to get rid of, together with supporting a better alternative.

Deprecating functionaliy:

- using a `log.deprecated("msg", 'X.Y')` statement in EasyBuild version `X.(Y-n)` a certain block of code is tagged as *deprecated*, indicating that the corresponding functionality will no longer be supported in EasyBuild version `X.Y`; for example, to deprecate the use of the `makeopts` easyconfig parameter with EasyBuild v2.0:

```
    if self.cfg['makeopts']:
        self.log.deprecated("Easyconfig parameter 'makeopts' is deprecated, use 'buildopts' inste
```

- until EasyBuild version `X.Y`, the deprecation log message will manifest itself as a *warning*, highlighting the use of deprecated functionality; for example:

```
    == 2014-12-16 16:29:07,906 main.easyconfig.easyconfig WARNING Deprecated functionality, will
    Easyconfig parameter 'makeopts' is deprecated, use 'buildopts' instead;
    see http://easybuild.readthedocs.org/en/latest/Deprecated-functionality.html for more informa
```

Removing support for deprecated behavior:

- starting with EasyBuild version `X.Y`, the deprecation log message will result in an *error*, indicating that the deprecated behavior is no longer supported; for example:

```
ERROR: EasyBuild encountered an exception (at easybuild/framework/easyconfig/easyconfig.py:93
Failed to process easyconfig /home/example/gzip-1.5-goolf-1.4.10.eb:
DEPRECATED (since v2.0) functionality used: Easyconfig parameter 'makeopts' is deprecated, us
see http://easybuild.readthedocs.org/en/latest/Deprecated-functionality.html for more informa
```

- the code supporting the deprecated functionality is *removed* in EasyBuild version `X.(Y+1)` (i.e., the first non-bugfix-only release after version `X.Y`), see also *Removed functionality*
- until EasyBuild version `X.(Y+1)`, the code supporting the deprecated functionality will still be available; using the `--deprecated` command line option (or, equivalently, the `$EASYBUILD_DEPRECATED` environment variable), the deprecated functionality can be reactivated by specifying a *lower* version; for example, to avoid running into an error with EasyBuild v2.0 for functionality that was deprecated for EasyBuild v2.0:

```
eb gzip-1.5-goolf-1.4.10.eb --deprecated=1.0
```

5.4.3 How to check for use of deprecated functionality

Since EasyBuild v1.16.0, the `--deprecated` command line option can be used to check whether deprecated behavior is still being triggered in your EasyBuild setup.

For example, using `--deprecated=2.0` in EasyBuild v1.x will transform any deprecation warning for functionality that will no longer be supported in EasyBuild v2.0 into an error message. For example:

```
$ eb OpenMPI-1.8.1-GCC-4.8.3.eb --deprecated=2.0
== temporary log file in case of crash /tmp/easybuild-WWalWX/easybuild-aoL9Nd.log
ERROR: EasyBuild encountered an exception (at easybuild/framework/easyconfig/easyconfig.py:945 in
Failed to process easyconfig /home/example/work/easybuild-easyconfigs/easybuild/easyconfigs/o/Oper
DEPRECATED (since v2.0) functionality used: Magic 'global' easyconfigs variables like shared_lib_e
be used; see http://easybuild.readthedocs.org/en/latest/Deprecated-functionality.html for more in
```

Tip: Define `deprecated` to the next major EasyBuild version in one of your EasyBuild configuration files (see *Configuration file(s)*) or by defining `$EASYBUILD_DEPRECATED=2.0`, to ensure you are made aware of deprecated behavior as early as possible.

You can (temporarily) still rely on the deprecated functionality by specifying a *lower* version via `--deprecated` to overrule that setting, until the functionality is actually disabled.

5.5 Removed functionality

Some of the functionality that was available in previous EasyBuild versions is now *removed*, after it was deprecated first in an earlier EasyBuild version (see *Deprecation policy*).

5.5.1 Overview of removed functionality since EasyBuild v2.0

In EasyBuild v2.0, some intrusive changes were made that break backward compatibility with EasyBuild v1.x.

For EasyBuild users:

- *Python version compatibility*
- *EasyBuild configuration*
- *$SOFTX environment variables in generated module files*

For authors of easyconfig files:

- *Automagic fallback to ConfigureMake*
- *Easyconfig parameters*
- *BEAGLE dependency in MrBayes easyblock replaced by beagle-lib*

For developers of easyblocks:

- *Easyblocks API (EasyBlock class from easybuild.framework.easyblock)*
- *Renamed/relocated functions*
- *Changes in (generic) easyblocks*

For EasyBuild framework developers:

- *easybuild.tools.modules Python module*

Note: A script `fix-broken-easyconfigs.py` is provided to fix easyconfig files that were broken by the backward-incompatible changes documented at *Automagic fallback to ConfigureMake* and *Easyconfig parameters*. See *fix_broken_easyconfigs.py* for more information.

Python version compatibility

Compatibility with Python 2.4 is removed.

- *deprecated since:* EasyBuild v1.14.0 (July'14)
- *removed in:* EasyBuild v2.0
- *alternative(s)*: **upgrade to Python v2.6.x or v2.7.x**

Ever since EasyBuild v1.0, the codebase has been Python 2.4 compatible. One reason for this is that EasyBuild was being used on a daily basis on Scientific Linux 5, in which the Python 2.4.x is the system default.

Starting with EasyBuild v2.0 support for Python 2.4 is removed, and only ensure compatibility with Python 2.6.x or a more recent Python 2.x.

This will enable us to gradually also make the codebase compatible with Python 3.x, which is difficult to do without removing support for Python 2.4.

EasyBuild configuration

Old-style EasyBuild configuration is removed.

- *deprecated since:* EasyBuild v1.3.0 (Apr'13)
- *removed in*: EasyBuild v2.0
- *alternatives:* **new-style configuration** (see *Configuring EasyBuild*)

Early versions of EasyBuild v1.x provided support for configuring EasyBuild via a *Python module* that was automagically executed when available.

Since EasyBuild v1.3 a safer and more consistent way of configuring EasyBuild is supported, which aligns the EasyBuild command line options, `$EASYBUILD_X` environment variables and key-value style configuration files.

More information about the new(er) and recommended configuration style is available *here*.

For detailed information with respect to porting from the old to the new configuration style, see *Configuration Legacy*.

Location of default configuration file

The default configuration file location `$HOME/.easybuild/config.cfg` is no longer considered.

- *deprecated since:* EasyBuild v1.11.0 (Feb'14)
- *removed in*: EasyBuild v2.0
- *alternatives:* `$XDG_CONFIG_HOME/easybuild/config.cfg` (equivalent to `$HOME/.config/easybuild/config.cfg`)

The default path for the new-style configuration path is `$XDG_CONFIG_HOME/easybuild/config.cfg` (or `$HOME/.config/easybuild/config.cfg` if `$XDG_CONFIG_HOME` is not set), see *List of used configuration files*.

The previous default path `$HOME/.easybuild/config.cfg` that was in place since EasyBuild v1.3.0 is no longer considered.

Automagic fallback to `ConfigureMake`

The automagic fallback to the `ConfigureMake` easyblock is removed.

- *deprecated since:* EasyBuild v1.16.0 (Dec'14)
- *removed in:* EasyBuild v2.0
- *alternative(s)*: specify `easyblock = 'ConfigureMake'` in easyconfig file

If the `easyblock` easyconfig was not specified, EasyBuild tries to find a matching easyblock based on the software name. In EasyBuild v1.x, the generic `ConfigureMake` easyblock was used if no matching easyblock could be found.

This behavior is now removed; instead, easyconfigs that require using the `ConfigureMake` easyblock *must* include the following:

```
easyblock = 'ConfigureMake'
```

Easyconfig parameters

Some easyconfig parameters are removed.

Note: A script is available to fix easyconfig files that are broken because they still rely on this functionality, see *fix_broken_easyconfigs.py*.

Options for build command

The `premakeopts` and `makeopts` easyconfig parameters are removed.

- *deprecated since:* EasyBuild v1.13.0 (May'14)
- *removed in:* EasyBuild v2.0
- *alternative(s)*: use `prebuildopts`/`buildopts` instead

For consistency in terminology, the `premakeopts` and `makeopts` generic easyconfig parameters are removed, in favor of their alternative parameters, `prebuildopts` and `buildopts`, resp.

(see also *Configure/build/install command options*)

Note: Since EasyBuild v1.13.0, `buildopts` is automatically defined with the value of `makeopts`, unless `buildopts` was specified by itself. When both values are specified, `buildopts` takes precedence of `makeopts` (analogous for `prebuildopts`/`premakeopts`).

Shared library extension

The `shared_lib_ext` 'constant' in easyconfigs is no longer defined.

- *deprecated since:* EasyBuild v1.5.0 (June'13)
- *removed in:* EasyBuild v2.0
- *alternative(s)*: use `SHLIB_EXT` instead

The `shared_lib_ext` "magic" variable representing the extension for shared libraries (`.so` on Linux, `.dylib` on OS X) is no longer defined; the easyconfig constant `SHLIB_EXT` should be using instead.

Software license

The `license` easyconfig parameter is removed.

- *deprecated since:* EasyBuild v1.11.0 (Feb'14)
- *removed in:* EasyBuild v2.0
- *alternative(s)*: use `license_file` or `software_license` instead

The `license` easyconfig parameter, which was specific to the `IntelBase` generic easyblock and thus relevant for Intel tools, is removed. The generic `license_file` easyconfig parameter should be used instead, to specify the location of the license file (or server).

This change was made to avoid confusion with the `software_license` generic easyconfig parameter, which can be used to specify the license under which the software was released (e.g., GPLv2, BSD, etc.). Here, the specified value *must* be a known license type (see `eb --avail-easyconfig-licenses`).

Note: The *software_license* easyconfig parameter will become **mandatory** at some point.

`BEAGLE` dependency in `MrBayes` easyblock replaced by `beagle-lib`

The `MrBayes` easyblock no longer considers `BEAGLE` as a valid dependency.

- *deprecated since:* EasyBuild v1.6.0 (Jul'14)
- *removed in:* EasyBuild v2.0
- *alternative(s)*: use `beagle-lib` instead

Due to a misnomer in the easyconfig files for `beagle-lib` (formerly named `BEAGLE`), the custom easyblock for `MrBayes` now no longer considers `BEAGLE` as a dependency.

The library required by `MrBayes` must now be provided as a dependency named `beagle-lib`.

EasyBuild API changes

Some changes in the EasyBuild API were made, which potentiallty affects easyblocks and the EasyBuild framework itself.

Easyblocks API (`EasyBlock` class from `easybuild.framework.easyblock`)

The API for easyblocks was modified slightly, to correct for a couple of historic mistakes.

Return type of `extra_options` method **The list-of-tuples return type of the `extra_options` method must now be a `dict` instead.**

- *deprecated since:* EasyBuild v1.12.0 (Apr'14)
- *removed in:* EasyBuild v2.0
- *alternative(s)*: ensure/assume `dict` return type

The return type of the `extra_options` static method in the `EasyBlock` class has been changed to a *dictionary* (`dict`), rather than a list of key-value tuples.

Custom easyconfig parameters should be added via a *dict*-typed value to the `extra_options` function of parent easyblock.

For example (taken from the generic easyblock `Binary`):

```
@staticmethod
def extra_options(extra_vars=None):
    """Extra easyconfig parameters specific to Binary easyblock."""
    extra_vars = EasyBlock.extra_options(extra_vars)
    extra_vars.update({
        'install_cmd': [None, "Install command to be used.", CUSTOM],
    })
    return extra_vars
```

Extension filter template **The `name` and `version` templates in `exts_filter` are removed.**

- *deprecated since:* EasyBuild v1.2.0 (Feb'13)
- *removed in:* EasyBuild v2.0
- *alternative(s)*: use `ext_name` and `ext_version` instead

Only the `ext_name`, `ext_version` and `src` template strings can be used in the `exts_filter` extension filter easyconfig parameter; the `name` and `version` template strings are removed.

For example (default extension filter for Python packages):

```
exts_filter = ("python -c 'import %(ext_name)s'", "")
```

Module path of default class for extensions **Specifying the module path in `exts_defaultclass` is no longer possible.**

- *deprecated since:* EasyBuild v0.5 (Apr'12)
- *removed in:* EasyBuild v2.0
- *alternative(s)*: *(none required, module path is derived from specified class name)*

Explicitely specifying the module path for the default class to use for extensions (via `exts_defaultclass`) is no longer possible. Only the class name should be specified, the corresponding module path is derived from it.

Module path for easyblocks **Deriving the module path for easyblocks from the software name is removed.**

- *deprecated since:* EasyBuild v1.4.0 (May'13)
- *removed in:* EasyBuild v2.0
- *alternative(s)*: use easyblock class name according to encoding scheme (e.g., `EB_Foo`)

Determining the *location* of Python modules representing easyblocks based on the software name (`name`) is removed.

EasyBuild *must* be able to determine the easyblock module path solely based on the name of the easyblock Python class.

Easyblocks with a class name that is already honoring the encoding scheme implemented by the `encode_class_name` function will not be affected.

`easybuild.tools.modules` Python module

The API of the `easybuild.tools.modules` module has been updated, certain aspects of the old API are removed.

- *deprecated since:* EasyBuild v1.8.0 (Oct'13) & v1.15.0 (Sept'15)
- *removed in:* EasyBuild v2.0
- *alternative(s)*: use equivalents available in new API (see below)

The API of the `easybuild.tools.modules` Python module has been changed extensively when implementing support for alternative module naming schemes:

- the `modules` class variable and the `add_module`/`remove_module` methods are removed; modules should be (un)loaded using the `load` and `unload` methods instead
- the `mod_paths` and `modulePath` named arguments for the `run_module` method aare removed; the class instance should be created with a specific list of module paths instead
- the `Modules` class to obtain a class instance representing a modules tool interface is removed; the `modules_tool` function should be used instead

Additionally, the `exists` method which only takes a single module name is removed; it is replaced by the `exist` method, which takes a list of module names *(since EasyBuild v1.15.0 (Sept'15))*.

Easyblocks should not be using `easybuild.tools.modules` directly, and hence should be unaffected.

`$SOFTX` environment variables in generated module files

`$SOFTX` environment variables set by module files generated with EasyBuild v0.x will no longer be taken into account.

- *deprecated since:* EasyBuild v1.3.0 (Apr'13)
- *removed in:* EasyBuild v2.0
- *alternative(s)*: reinstall (ancient) module files which are only defining the `$SOFTX` environment variables

The `get_software_root` and `get_software_version` functions will only take `$EBROOTFOO` and `$EBVERSIONFOO` environment variables into account, as opposed to also considering the `$SOFTROOTFOO` and `$SOFTVERSIONFOO` environment variables (which were set in modules generated by EasyBuild v0.x). Likewise, adhering to the `$SOFTDEVELFOO` environment variables is removed.

This is only relevant to early adopters who are still using module files generated by EasyBuild v0.x.

Renamed/relocated functions

Some functions/methods have been renamed or relocated, their equivalents under a previous location/name are removed.

- *deprecated since: (depends on function/method, see below)*
- *removed in:* EasyBuild v2.0
- *alternative(s)*: use new location/name

A number of functions and methods that are part of the EasyBuild framework API have been renamed, mainly for consistency reasons.

- the `moduleGenerator` handle to the `ModuleGenerator` object instance has been renamed to `module_generator`; hence, easyblock should be using `self.module_generator` rather than `self.moduleGenerator` *(since EasyBuild v1.16.0 (Dec'14))*
- `source_paths()` (in `easybuild.tools.config`) replaces the removed `source_path()` *(since EasyBuild v1.8.0 (Oct'13))*
- `get_avail_core_count()` (in `easybuild.tools.systemtools`) replaces the removed `get_core_count()` *(since EasyBuild v1.9.0 (Nov'13))*
- `get_os_type()` (in `easybuild.tools.systemtools`) replaces the removed `get_kernel_name` *(since EasyBuild v1.3.0 (Apr'13))*
- the `det_full_ec_version` function available from `easybuild.tools.module_generator` replaces the removed `det_installversion` function that was available from `easybuild.framework.easyconfig.*` *(since EasyBuild v1.8.0 (Oct'13))*

Some functions have moved to a different location:

- the `read_environment` function is now provided by the `easybuild.tools.environment` module, rather than by `easybuild.tools.config` or `easybuild.tools.utilities` *(since EasyBuild v1.7.0 (Sept'13))*
- the `modify_env` function is now provided by the `easybuild.tools.environment` module, rather than by `easybuild.tools.filetools` *(since EasyBuild v1.7.0 (Sep'13))*
- the `run_cmd`, `run_cmd_qa` and `parse_log_for_error` functions are now provided by the `easybuild.tools.run` module, rather than by `easybuild.tools.filetools` *(since EasyBuild v1.11.0 (Feb'14))*

The `get_log` function provided by the `easybuild.tools.build_log` module has been removed entirely, no alternatives are provided (since none are needed). *(since EasyBuild v1.3.0 (Apr'13))*

Changes in (generic) easyblocks

`srcdir` replaces `builddir` as named argument in `CMakeMake.configure_step` **The named argument `builddir` in the `configure_step` method of the generic `CMakeMake` easyblock was replaced by `srcdir`.**

- *deprecated since:* EasyBuild v1.4.0 (May'13)
- *removed in:* EasyBuild v2.0
- *alternative(s)*: equivalent `srcdir` named argument

Since the `builddir` named argument in the `configure_step` method of the generic `CMakeMake` easyblock was a misnomer (it specifies the location of the *source* directory that should be provided to `cmake`), it was replaced with an equivalent named argument `srcdir`.

`VersionIndependentPythonPackage` replaces `VersionIndependendPythonPackage` **The generic easyblock `VersionIndependendPythonPackage` was replaced with the equivalent generic easyblock `VersionIndependentPythonPackage`.**

- *deprecated since:* EasyBuild v1.11.0 (Feb'14)
- *removed in:* EasyBuild v2.0
- *alternative(s)*: `VersionIndependentPythonPackage`

Because of to a typo in the name, the `VersionIndependendPythonPackage` generic easyblock was replaced by the equivalent `VersionIndependentPythonPackage` generic easyblock.

`get_sitearch_suffix` function in `Perl` easyblock is removed **The `get_sitearch_suffix` function in the `Perl` easyblock was replaced in favor of the more generic `get_site_suffix` function.**

- *deprecated since:* EasyBuild v1.7.0 (Sept'13)
- *removed in:* EasyBuild v2.0
- *alternative(s)*: `get_site_suffix('sitearch')`

The `get_sitearch_suffix` function provided by the `Perl` easyblock, which can be (and is) imported in/used by other easyblocks, has been replaced by the more generic `get_site_suffix` function.

To obtain the same functionality as was provided by `get_sitearch_suffix`, use `get_site_suffix('sitearch')` instead.

CHAPTER 6

Getting help

Having trouble? We'd like to help!

- Search this documentation collection
- Search for information in the archives of the easybuild@lists.ugent.be mailing list or subscribe to post a question.
- Did you try `eb --help`?
- Ask a question in the #easybuild IRC channel on the Freenode network
- Consider participating to an EasyBuild conference call
- Report issues with EasyBuild framework in our framework ticket tracker.
- Report issues with EasyBuild easyblocks in our easyblocks ticket tracker.
- Report issues with EasyBuild easyconfigs in our easyconfigs ticket tracker.
- Report issues with EasyBuild documentation or other aspects in our general ticket tracker.

CHAPTER 7

Lists and tables

As of version EasyBuild version 2.8.2:

- The complete table of available toolchains is visible at *List of known toolchains*
- The list of available easyblocks is visible at *List of easyblocks*
- The list of available (generic) easyconfig parameters is visible at easyconfigs_parameters

CHAPTER 8

Appendices

- *Documentation changelog*
- *EasyBuild release notes*
- Search
- *Useful Links*

8.1 Changelog for EasyBuild documentation

- **release 20160713.01** (*July 13th 2016*): update release notes for EasyBuild v2.8.2 (see *v2.8.2 (July 13th 2016)*)
- **release 20160613.01** (*June 13th 2016*): clarify required dependencies (setuptools, vsc-install) (see *Required Python packages*)
- **release 20160607.01** (*June 7th 2016*): update/complete documentation on GitHub integration (see *Integration with GitHub*)
- **release 20160530.01** (*May 30th 2016*): update release notes for EasyBuild v2.8.1 (see *v2.8.1 (May 30th 2016)*)
- **release 20160518.01** (*May 18th 2016*): update release notes for EasyBuild v2.8.0 (see *v2.8.0 (May 18th 2016)*)
- **release 20160429.01** (*April 29th 2016*): add section on updating EasyBuild, see *Updating an existing EasyBuild installation*
- **release 20160320.01** (*March 20th 2016*): update release notes for EasyBuild v2.7.0 (see *v2.7.0 (March 20th 2016)*)
- **release 20160228.01** (*February 28th 2016*):
  - update documentation on external modules metadata (see *Metadata for external modules*)
- **release 20160214.01** (*February 14th 2016*):
- add section on `--show-config` (see *Overview of current configuration (–show-config, –show-full-config)*)
- **release 20160126.02** (*January 26th 2016*): packaging support is stable since EasyBuild v2.5.0 (see *Packaging support*)
- **release 20160126.01** (*January 26th 2016*): update release notes for EasyBuild v2.6.0 (see *v2.6.0 (January 26th 2016)*)
- **release 20151217.01** (*December 17th 2015*): update release notes for EasyBuild v2.5.0 (see *v2.5.0 (December 17th 2015)*)
- **release 20151209.01** (*December 9th 2015*):

  - add documentation on controlling compiler optimizations flags, see *Controlling compiler optimization flags*

- **release 20151110.01** (*November 10th 2015*): update release notes for EasyBuild v2.4.0 (see *v2.4.0 (November 10th 2015)*)
- **release 20151108.01** (*November 8th 2015*):
  - document (experimental) support for using minimal toolchains (see *Using minimal toolchains for dependencies*)
- **release 20151028.01** (*October 28th 2015*): document extended dry run mechanism (see *Extended dry run*)
- **release 20151021.01** (*October 21st 2015*):
  - include initial documentation on experimental support for easyconfig files in YAML syntax (`.yeb`), see easyconfig_yeb_format
- **release 20150902.01** (*September 2nd 2015*): update release notes for EasyBuild v2.3.0 (see *v2.3.0 (September 2nd 2015)*)
- **release 20150715.01** (*July 15th 2015*): update release notes for EasyBuild v2.2.0 (see *v2.2.0 (July 15th 2015)*)
- **release 20150714.01** (*July 14th 2015*): add documentation on *Packaging support*
- **release 20150709.01** (*July 9th 2015*): add documentation on *Submitting jobs using –job*
- **release 20150708.01** (*July 8th 2015*):
  - add documentation on `--include-*` options (see *Including additional Python modules (–include-*)*)
- **release 20150703.01** (*July 3rd 2015*):
  - fix outdated documentation on `easyblock` parameter (see *Easyblock specification*)
- **release 20150624.01 (*June 24th 2015*): mention `MigrateFromEBToHMNS` module naming scheme in section on** `--module-only` (see *Generating additional module files*)
- **release 20150610.01** (*June 10th 2015*): update *Installing Lmod without root permissions* for Lmod v6.0
- **release 20150518.01** (*May 18th 2015*):
  - update section on `--search`: better examples + highlight ability to search via regular expression (see *Searching for easyconfigs, –search / -S*)
  - update release notes for EasyBuild v2.1.1 (see *v2.1.1 (May 18th 2015)*)
- **release 20150506.01** (*May 6th 2015*): updated documentation for EasyBuild v2.1.1
  - add note on `$LMOD_CMD` fallback to find full path to `lmod` binary (see *Required modules tool*)
- **release 20150430.01** (*Apr 30th 2015*): updated documentation for EasyBuild v2.1.0
  - also cover extensions in page on concepts and terminology (see *Extensions*)
  - add documentation on *Partial installations*, covering `--stop`, `--skip` and `--module-only`
  - add documentation on *Manipulating dependencies*, covering `--filter-deps` and `--hide-deps`
  - document `-module-syntax` configuration option (see *Module files syntax (–module-syntax)*)
  - add note on detection of unknown `$EASYBUILD`-prefixed environment variables (see *Environment variables*)
  - mention support for prepending/appending to `--robot-paths` (see *Prepending and/or appending to the default robot search path*)
  - update release notes for EasyBuild v2.1.0 (see *EasyBuild release notes*)
- **release 20150425.01** (*Apr 25th 2015*):
  - add documentation on *Using external modules*

- **release 20150407.01** (*Apr 7th 2015*):
  - add link to *Unit tests* page in dedicated section at *Installing EasyBuild* page (see *Running unit tests*)
  - clarify relation between `--installpath`, `--prefix`, `-subdir-*` and `--installpath-*` configuration options (see *Software and modules install path (–installpath, –installpath-software, –installpath-modules)*)
  - mention `--show-default-configfiles` command line option in relevant section (see *Default configuration files*)
- **release 20150327.01** (*Mar 27th 2015*):
  - documented deprecated functionality w.r.t. error reporting (see *Report errors by raising EasyBuildError rather than using log methods*)
- **release 20150316.01** (*Mar 16th 2015*):
  - include list of EasyBuild repositories cloned by `install-EasyBuild-develop.sh` script (see *Installation of latest development version using provided script*)
- **release 20150312.01** (*Mar 12th 2015*):
  - enhance documentation w.r.t. template values in configuration files (see *Templates and constants supported in configuration files*)
  - improve documentation on `--robot` and `--robot-paths` (see *Controlling the robot search path*)
- **release 20150310.01** (*Mar 10th 2015*):
  - document peculiarties w.r.t. dependencies in combination with a `dummy` toolchain (see *Dependencies*)
  - document `clean_gists.py` script (see *clean_gists.py*)
  - mention taking into account of proxy settings for downloading sources (see *Source files and patches*)
- **release 20150306.03** (*Mar 6th 2015*): add release notes for EasyBuild v2.0.0 (see *EasyBuild release notes*)
- **release 20150306.02** (*Mar 6th 2015*):
  - add documentation on GitHub integration features (see *Integration with GitHub*), mainly `--from-pr` (see from_pr)
  - document locations where (specified) easyconfig files are being searched for (see *Specifying builds*)
- **release 20150306.01** (*Mar 6th 2015*):
  - add documentation on removed functionality (see *Removed functionality*)
  - clean up documentation on deprecated functionality (see *Deprecated functionality*)
  - add documentation on provided scripts, in particular `fix-broken-easyconfigs.py` (see *Useful scripts*)
- **release 20150302.01** (*Mar 2nd 2015*): update/cleanup documentation on *Alternative installation methods*
- **release 20150227.02** (*Feb 27th 2015*): add documentation on the EasyBuild unit test suites, see *Unit tests*
- **release 20150227.01** (*Feb 27th 2015*): enhance documentation w.r.t. to (optional dependencies), see *Installing EasyBuild*
- **release 20150220.01** (*Feb 20th 2015*):
  - document new advanced bootstrapping options: skipping stage 0 and providing source tarballs (see *Installing EasyBuild*)
- **release 20150219.01** (*Feb 19th 2015*): first updates for EasyBuild v2.0.0
  - extend section on (default) EasyBuild configuration files to also cover `$XDG_CONFIG_DIRS` (see configuration_file:)
- **release 20150205.01** (*Feb 5th 2015*): include information on deprecated functionality in (generic) easyblocks (see *Deprecated functionality*)

- **release 20150126.01** (*Jan 26th 2015*):
  - fix `pip` installation prefix option (*Alternative installation methods*)
  - clarify need to have modules tool binary available in `$PATH` (*Installing EasyBuild*)
- **release 20150112.01** (*Jan 12th 2015*): mention need to escape `%` when setting log file format via config file (see *Logfile format (–logfile-format)*)
- **release 20150107.01** (*Jan 7th 2015*): document behaviour of *dummy* toolchain (*dummy toolchain*)
- **release 20141219.01** (*Dec 19th 2014*): add release notes for EasyBuild v1.16.1 (see *EasyBuild release notes*)
- **release 20141218.01** (*Dec 18th 2014*): add release notes for EasyBuild v1.16.0 (see *EasyBuild release notes*)
- **release 20141217.01** (*Dec 17th 2014*): document deprecated functionality in EasyBuild v1.x (*Deprecated functionality*)
- **release 20141204.02** (*Dec 4th 2014*): add EasyBuild release notes (see *EasyBuild release notes*)
- **release 20141204.01** (*Dec 4th 2014*): updates for EasyBuild v1.16.0
  - document details w.r.t. (controlling of) robot search path, incl. `--robot-paths` (*Using the EasyBuild command line*)
  - document use of templates and constants in EasyBuild configuration files (*Configuring EasyBuild*)
  - bump EasyBuild version to 1.16.0
  - changed release number scheme for documentation (based on datestamp)
- **release 1.0.3** (*Dec 3rd 2014*): add page on *Code style*
- **release 1.0.2** (*Nov 6th 2014*): typo and grammar fixes, update Lmod installation instructions for Lmod v5.8
- **release 1.0.1** (*Nov 4th 2014*): fix issues with Changelog
- **release 1.0.0** (*Nov 4th 2014*): initial release of revamped EasyBuild documentation @ http://easybuild.readthedocs.org, covering basic topics:
  - introductory topics:
    - *What is EasyBuild?*
    - *Concepts and terminology*
    - *Typical workflow example: building and installing WRF*
  - getting started:
    - *Installing EasyBuild*
    - *Configuring EasyBuild*
  - basic usage topics:
    - *Using the EasyBuild command line*
    - *Writing easyconfig files: the basics*
    - *Understanding EasyBuild logs*

8.2 Configuration Legacy

Legacy configuration is currently **deprecated!**

If you are a new user of EasyBuild you can safely ignore everything below this line, refer instead to Configuring EasyBuild.

8.2.1 Porting from legacy configuration style

In EasyBuild v1.x, a couple of configuration options, other than the standard ones aligned with variables, are available that follow the **legacy configuration style**, including:

- the `-C` and `--config` command line arguments (**use** `--configfiles` **instead**)
- the `$EASYBUILDCONFIG` environment variable (**use** `$EASYBUILD_CONFIGFILES` **instead**)
- the default path `$HOME/.easybuild/config.py` (**new-style default path is** `$XDG_CONFIG_HOME/easybuild/config.cfg`)
- the legacy fallback path `<installpath>/easybuild/easybuild_config.py` (**only default/fallback path is** `$XDG_CONFIG_HOME/easybuild/config.cfg`)

Likewise, the following legacy environment variables allowed to override selected configuration settings:

- `$EASYBUILDBUILDPATH`: build path to be used by EasyBuild (**use** `$EASYBUILD_BUILDPATH` **instead**)
- `$EASYBUILDINSTALLPATH`: install path to be used by EasyBuild (**use** `$EASYBUILD_INSTALLPATH` **instead**)
- `$EASYBUILDSOURCEPATH`: source path to be used by EasyBuild (**use** `$EASYBUILD_SOURCEPATH` **instead**)
- `$EASYBUILDPREFIX`: build/install/source path prefix to be used (**use** `$EASYBUILD_PREFIX` **instead**)

We *strongly* advise to switch to the new way of configuring EasyBuild as soon as possible, since the legacy configuration style will no longer be supported in EasyBuild v2.x.

8.2.2 How EasyBuild used to be configured in the early days

Configuring *EasyBuild* is done by providing a configuration file.

EasyBuild expects the configuration file to contain valid Python code, because it executes its contents (using `exec`). The rationale is that this approach provides a lot of flexibility for configuring EasyBuild.

EasyBuild will use the file that is provided by the path/filename in the following order of preference:

- path/filename specified on the EasyBuild command line (using `--config`),
- path/filename obtained from the environment variable `EASYBUILDCONFIG` (if it is defined)
- `$HOME/.easybuild/config.py` (as of EasyBuild v1.1)
- the (default) configuration file at `<path where EasyBuild was installed>/easybuild/easybuild_config.py`

Configuration variables

The configuration file must define the following five variables: `build_path`, `install_path`, `source_path`, `repository`, and `log_format`. If one of them is not defined, EasyBuild will complain and exit.

Build path (required)

The `build_path` variable specifies the directory in which EasyBuild builds its software packages.

Each software package is (by default) built in a subdirectory of the `build_path` under `<name>/<version>/<toolchain><versionsuffix>`.

Note that the build directories are emptied by EasyBuild when the installation is completed (by default).

Install path (required)

The `install_path` variable specifies the directory in which EasyBuild installs software packages and the corresponding module files.

The packages themselves are installed under `install_path/software` in their own subdirectory aptly named `<name>/<version>-<toolchain><versionsuffix>` (by default), where name is the package name. The corresponding module files are installed under `install_path/modules`.

Setting $MODULEPATH

After the configuration, you need to make sure that `$MODULEPATH` environment variable is extended with the `modules/all` subdirectory of the `install_path`, i.e.:

```
export MODULEPATH=<install_path>/modules/all:$MODULEPATH
```

It is probably a good idea to add this to your (favourite) shell .rc file, e.g., `.bashrc`, and/or the `.profile` login scripts, so you do not need to adjust the `$MODULEPATH` variable every time you start a new session.

Source path (required)

The `source_path` variable specifies the directory in which EasyBuild looks for software source and install files.

Similarly to the configuration file lookup, EasyBuild looks for the installation files as given by the `sources` variable in the .eb easyconfig file, in the following order of preference:

- `<source_path>/<name>`: a subdirectory determined by the name of the software package
- `<source_path>/<letter>/<name>`: in the style of the `easyblocks`/`easyconfigs` directories: in a subdirectory determined by the first letter (in lower case) of the software package and by its full `name`
- `<source_path>`: directly in the source path

Note that these locations are also used when EasyBuild looks for patch files in addition to the various `easybuild/easyconfigs` directories that are listed in the `$PYTHONPATH`.

(replaced by All available easyconfig parameters, –avail-easyconfig-params / -a)

8.3 Available easyconfig parameters for EB_WRF

Overview of easyconfig parameters, including those specific to the easyblock for WRF (indicated with `(*)`):

```
$ eb -a --easyblock EB_WRF

Available easyconfig parameters (* indicates specific for the EB_WRF EasyBlock)
MANDATORY
---------
buildtype(*):          Specify the type of build (serial, smpar (OpenMP), dmpar (MPI), dm+sm (hybr
description:           A short description of the software (default: None)
docurls:               List of urls with documentation of the software (not necessarily on homepage
homepage:              The homepage of the software (default: None)
name:                  Name of software (default: None)
software_license:      Software license (default: None)
software_license_urls:         List of software license locations (default: None)
toolchain:             Name and version of toolchain (default: None)
version:               Version of software (default: None)

EASYBLOCK-SPECIFIC
------------------
rewriteopts(*):                Replace -O3 with CFLAGS/FFLAGS (default: True)
runtest(*):            Build and run WRF tests (default: True)
```

```
TOOLCHAIN
---------
onlytcmod:            Boolean/string to indicate if the toolchain should only load the environment
toolchainopts:                Extra options for compilers (default: None)

BUILD
-----
buildopts:            Extra options passed to make step (default already has -j X) (default: )
checksums:            Checksums for sources and patches (default: [])
configopts:           Extra options passed to configure (default already has --prefix) (default:
easyblock:            EasyBlock to use for building (default: ConfigureMake)
easybuild_version:    EasyBuild-version this spec-file was written for (default: None)
installopts:          Extra options for installation (default: )
maxparallel:          Max degree of parallelism (default: None)
parallel:             Degree of parallelism for e.g. make (default: based on the number of cores,
patches:              List of patches to apply (default: [])
postinstallcmds:      Commands to run after the install step. (default: [])
prebuildopts:         Extra options pre-passed to build command. (default: )
preconfigopts:                Extra options pre-passed to configure. (default: )
preinstallopts:               Extra prefix options for installation. (default: )
runtest(*):           Indicates if a test should be run after make; should specify argument after
sanity_check_commands:        format: [(name, options)] e.g. [('gzip','-h')]. Using a non-tuple is
sanity_check_paths:   List of files and directories to check (format: {'files':<list>, 'dirs':<lis
skip:                 Skip existing software (default: False)
skipsteps:            Skip these steps (default: [])
source_urls:          List of URLs for source files (default: [])
sources:              List of source files (default: [])
stop:                 Keyword to halt the build process after a certain step. (default: None)
tests:                        List of test-scripts to run after install. A test script should retu
unpack_options:               Extra options for unpacking source (default: None)
unwanted_env_vars:    List of environment variables that shouldn't be set during build (default:
versionprefix:                Additional prefix for software version (placed before version and to
versionsuffix:                Additional suffix for software version (placed after toolchain name)

FILE-MANAGEMENT
---------------
buildininstalldir:    Boolean to build (True) or not build (False) in the installation directory
cleanupoldbuild:      Boolean to remove (True) or backup (False) the previous build directory with
cleanupoldinstall:    Boolean to remove (True) or backup (False) the previous install directory wi
dontcreateinstalldir: Boolean to create (False) or not create (True) the install directory (defaul
keeppreviousinstall:  Boolean to keep the previous installation with identical name. Experts only
keepsymlinks:         Boolean to determine whether symlinks are to be kept during copying or if th
start_dir:            Path to start the make in. If the path is absolute, use that path. If not, t

DEPENDENCIES
------------
allow_system_deps:    Allow listed system dependencies (format: (<name>, <version>)) (default: []
builddependencies:    List of build dependencies (default: [])
dependencies:         List of dependencies (default: [])
hiddendependencies:   List of dependencies available as hidden modules (default: [])
osdependencies:               OS dependencies that should be present on the system (default: [])

LICENSE
-------
group:                        Name of the user group for which the software should be available (o
key:                  Key for installing software (default: None)
license_file:         License file for software (default: None)
license_server:               License server for software (default: None)
license_server_port:  Port for license server (default: None)

EXTENSIONS
----------
exts_classmap:                Map of extension name to class for handling build and installation.
```

```
exts_defaultclass:    List of module for and name of the default extension class (default: None)
exts_filter:          Extension filter details: template for cmd and input to cmd (templates for
exts_list:            List with extensions added to the base installation (default: [])

MODULES
-------
include_modpath_extensions:Include $MODULEPATH extensions specified by module naming scheme. (defa
modaliases:           Aliases to be defined in module file (default: {})
modextrapaths:                Extra paths to be prepended in module file (default: {})
modextravars:         Extra environment variables to be added to module file (default: {})
modloadmsg:           Message that should be printed when generated module is loaded (default: {}
modtclfooter:         Footer to include in generated module file (Tcl syntax) (default: )
moduleclass:          Module class to be used for this software (default: base)
moduleforceunload:    Force unload of all modules when loading the extension (default: False)
moduleloadnoconflict: Don't check for conflicts, unload other versions instead  (default: False)

OTHER
-----
buildstats:           A list of dicts with build statistics (default: None)
```

(replaced by eb_help *)*

8.4 List of easyblocks

The list of available easyblocks can easily be obtained as follows:

```
$ eb --list-easyblocks

EasyBlock
|-- Binary
|   |-- EB_ABAQUS
|   |-- EB_Allinea
|   |-- EB_CPLEX
|   |-- EB_CUDA
|   |-- EB_EPD
|   |-- PackedBinary
|   |   |-- EB_Java
|   |   |-- EB_Tornado
|   |-- EB_Mathematica
|   |-- Rpm
|   |   |-- EB_QLogicMPI
|   |-- JAR
|-- EB_ACML
|-- EB_ALADIN
|-- EB_ANSYS
|-- EB_ant
|-- ConfigureMake
|   |-- EB_ARB
|   |-- CMakeMake
|   |   |-- EB_Armadillo
|   |   |-- EB_BamTools
|   |   |-- EB_CGAL
|   |   |-- EB_Clang
|   |   |-- CMakePythonPackage
|   |   |   |-- EB_DOLFIN
|   |   |   |-- EB_UFC
|   |   |-- EB_Geant4
|   |   |-- EB_GROMACS
|   |   |-- EB_netCDF
|   |   |-- EB_OpenBabel
|   |   |-- EB_Trilinos
```

```
|    |-- EB_ATLAS
|    |-- MakeCp
|    |    |-- EB_BamTools
|    |    |-- EB_BLAT
|    |    |-- EB_NAMD
|    |    |-- CmdCp
|    |-- EB_R
|    |-- EB_BLACS
|    |-- EB_Bowtie
|    |-- EB_Bowtie2
|    |-- EB_BWA
|    |-- EB_bzip2
|    |-- EB_CBLAS
|    |-- EB_Chapel
|    |-- EB_Cufflinks
|    |-- EB_DB
|    |-- EB_DL_underscore_POLY_underscore_Classic
|    |-- EB_Python
|    |-- EB_Doxygen
|    |-- EB_ESMF
|    |-- EB_ESPResSo
|    |-- EB_Ferret
|    |-- EB_flex
|    |-- EB_freetype
|    |-- EB_g2clib
|    |-- EB_g2lib
|    |-- EB_GCC
|    |-- EB_GHC
|    |-- EB_Go
|    |-- EB_HDF5
|    |-- EB_HPCG
|    |-- EB_HPL
|    |-- EB_Hypre
|    |-- EB_LAPACK
|    |-- EB_libint2
|    |-- EB_libxml2
|    |-- EB_MetaVelvet
|    |-- EB_METIS
|    |-- EB_Mothur
|    |-- EB_MrBayes
|    |-- EB_Perl
|    |-- EB_MUMmer
|    |-- EB_MUMPS
|    |-- EB_MVAPICH2
|    |-- EB_ncurses
|    |-- EB_netCDF_minus_Fortran
|    |-- EB_NEURON
|    |-- EB_NWChem
|    |-- EB_OpenSSL
|    |-- EB_Pasha
|    |-- EB_PETSc
|    |-- EB_Primer3
|    |-- EB_PSI
|    |-- EB_Qt
|    |-- EB_QuantumESPRESSO
|    |-- EB_ROOT
|    |-- EB_SAMtools
|    |-- EB_ScaLAPACK
|    |-- EB_Scalasca1
|    |-- EB_Score_minus_P
|    |-- EB_SHRiMP
|    |-- EB_SLEPc
|    |-- EB_SOAPdenovo
```

```
|   |-- EB_SuiteSparse
|   |-- EB_SWIG
|   |-- EB_Velvet
|   |-- EB_XCrySDen
|   |-- PerlModule
|-- ExtensionEasyBlock
|   |-- RPackage
|   |   |-- EB_Bioconductor
|   |   |-- EB_Rmpi
|   |   |-- EB_Rserve
|   |   |-- EB_XML
|   |-- PythonPackage
|   |   |-- CMakePythonPackage
|   |   |   |-- EB_DOLFIN
|   |   |   |-- EB_UFC
|   |   |-- EB_EasyBuildMeta
|   |   |-- EB_libxml2
|   |   |-- EB_netcdf4_minus_python
|   |   |-- EB_nose
|   |   |-- FortranPythonPackage
|   |   |   |-- EB_numpy
|   |   |   |-- EB_scipy
|   |   |-- EB_PyQuante
|   |   |-- EB_python_minus_meep
|   |   |-- EB_PyZMQ
|   |   |-- EB_VSC_minus_tools
|   |   |-- VersionIndependentPythonPackage
|   |   |   |-- VersionIndependendPythonPackage
|   |   |   |   |-- VSCPythonPackage
|   |-- PerlModule
|-- EB_BiSearch
|-- EB_Boost
|-- EB_CHARMM
|-- EB_CP2K
|-- EB_Eigen
|-- EB_FDTD_underscore_Solutions
|-- EB_FLUENT
|-- Tarball
|   |-- EB_FoldX
|   |-- EB_FreeSurfer
|   |-- EB_MTL4
|   |-- BinariesTarball
|-- EB_FSL
|-- EB_GenomeAnalysisTK
|-- IntelBase
|   |-- EB_icc
|   |   |-- EB_ifort
|   |-- EB_ifort
|   |-- EB_imkl
|   |-- EB_impi
|   |-- EB_Inspector
|   |-- EB_ipp
|   |-- EB_itac
|   |-- EB_tbb
|   |-- EB_VTune
|-- PackedBinary
|   |-- EB_Java
|   |-- EB_Tornado
|-- EB_libsmm
|-- EB_Maple
|-- EB_MATLAB
|-- EB_Modeller
|-- EB_NCL
```

```
|-- EB_OpenFOAM
|-- EB_OpenIFS
|-- EB_ParMETIS
|-- EB_picard
|-- EB_Rosetta
|-- EB_SCOTCH
|-- EB_TotalView
|-- EB_Trinity
|-- EB_WIEN2k
|-- EB_WPS
|-- EB_WRF
|-- Toolchain

Extension
|-- ExtensionEasyBlock
|   |-- RPackage
|   |   |-- EB_Bioconductor
|   |   |-- EB_Rmpi
|   |   |-- EB_Rserve
|   |   |-- EB_XML
|   |-- PythonPackage
|   |   |-- CMakePythonPackage
|   |   |   |-- EB_DOLFIN
|   |   |   |-- EB_UFC
|   |   |-- EB_EasyBuildMeta
|   |   |-- EB_libxml2
|   |   |-- EB_netcdf4_minus_python
|   |   |-- EB_nose
|   |   |-- FortranPythonPackage
|   |   |   |-- EB_numpy
|   |   |   |-- EB_scipy
|   |   |-- EB_PyQuante
|   |   |-- EB_python_minus_meep
|   |   |-- EB_PyZMQ
|   |   |-- EB_VSC_minus_tools
|   |   |-- VersionIndependentPythonPackage
|   |   |   |-- VersionIndependendPythonPackage
|   |   |   |   |-- VSCPythonPackage
|   |-- PerlModule
```

8.5 List of known toolchains

The list of known toolchains can easily be obtained with:

```
$ eb --list-toolchains
List of known toolchains (toolchainname: module[,module...]):
[...]
```

Toolchain	Components that comprise the toolchain		
Name	Compilers	MPI stack	Included Libraries
ClangGCC	Clang/GCC		
GCC	GCC		
cgmpich	Clang/GCC	MPICH	
cgmpolf	Clang/GCC	MPICH	OpenBLAS/LAPACK, ScaLAPACK(/BLACS), FFTW
cgmvapich2	Clang/GCC	MVAPICH2	
cgmvolf	Clang/GCC	MVAPICH2	OpenBLAS/LAPACK, ScaLAPACK(/BLACS), FFTW
cgompi	Clang/GCC	OpenMPI	
cgoolf	Clang/GCC	OpenMPI	OpenBLAS/LAPACK, ScaLAPACK(/BLACS), FFTW

Continued on next page

Table 8.1 – continued from previous page

Toolchain	Components that comprise the toolchain		
Name	Compilers	MPI stack	Included Libraries
dummy			
foss	GCC	OpenMPI	OpenBLAS/LAPACK, ScaLAPACK(/BLACS), FFTW
gcccuda	GCC, CUDA		
gimkl	GCC	impi	imkl
gmacml	GCC	MVAPICH2	ACML, ScaLAPACK(/BLACS), FFTW
gmpich2	GCC	MPICH2	
gmpolf	GCC	MPICH2	OpenBLAS/LAPACK, ScaLAPACK(/BLACS), FFTW
gmvapich2	GCC	MVAPICH2	
gmvolf	GCC	MVAPICH2	OpenBLAS/LAPACK, ScaLAPACK(/BLACS), FFTW
goalf	GCC	OpenMPI	ATLAS, ScaLAPACK(/BLACS), FFTW
gompi	GCC	OpenMPI	
gompic	GCC, CUDA	OpenMPI	
goolf	GCC	OpenMPI	OpenBLAS/LAPACK, ScaLAPACK(/BLACS), FFTW
goolfc	GCC, CUDA	OpenMPI	OpenBLAS/LAPACK, ScaLAPACK(/BLACS), FFTW
gqacml	GCC	QLogicMPI	ACML, ScaLAPACK(/BLACS), FFTW
iccifort	icc, ifort		
ictce	icc, ifort	impi	imkl
iimpi	icc, ifort	impi	
iiqmpi	icc, ifort	QLogicMPI	
impmkl	icc, ifort	MPICH2	imkl
intel	icc, ifort	impi	imkl
iomkl	icc, ifort	OpenMPI	imkl
iqacml	icc, ifort	QLogicMPI	ACML, ScaLAPACK(/BLACS), FFTW
ismkl	icc, ifort	MPICH2	imkl

Note: The *dummy* toolchain is a special case, see *dummy toolchain*.

8.6 Alternative installation methods

We warmly recommend installing EasyBuild via the bootstrap procedure, see *Bootstrapping EasyBuild*.

This page describes the alternative installation methods:

- *Standard installation of latest release*
- *Installation from downloaded sources*
- *Installation of latest release from GitHub*
- *Installation of latest development version*

Do take into account the list of (required) dependencies (see *Dependencies*).

8.6.1 Standard installation of latest release

Usually, you just want to install the latest (stable) version of each of the EasyBuild packages (framework, easyblocks, easyconfigs).

Python provides a couple of ways to do that. Every version of the EasyBuild packages is released via PyPi.

Installing EasyBuild without admin rights

If you do not have EasyBuild installed yet, or if you just want to install the most recent version of each of the EasyBuild packages, you can use one of the following simple commands:

- using `easy_install` (old tool, but still works):

```
easy_install --prefix $HOME/EasyBuild easybuild
```

> **Note:** If you already have *easybuild* installed, you may need to instruct `easy_install` to install a newer version, using `--upgrade` or `-U`.

- using `pip` (more recent and better installation tool for Python software):

```
pip install --install-option "--prefix=$HOME/EasyBuild" easybuild
```

The `--prefix $HOME/EasyBuild` part in these commands allows you to install EasyBuild without admin rights into `$HOME/EasyBuild`.

> **Note:** For pip v8.0 and newer, `pip install --prefix=$HOM/EasyBuild easybuild` works too.

Adjusting `$PATH` and `$PYTHONPATH` environment variables

After installing EasyBuild with either `easy_install` or `pip` like this, you will need to update the `$PATH` and `$PYTHONPATH` environment variable to make sure the system can find the main EasyBuild command `eb`. On (most) Linux distributions, the command for doing this is:

```
export PATH=$HOME/EasyBuild/bin:$PATH
export PYTHONPATH=$HOME/EasyBuild/lib/python2.7/site-packages:$PYTHONPATH
```

> **Tip:** To determine the path that should be added to the `$PYTHONPATH` environment variable for a given installation prefix, you can use the following command:

```
python -c "import distutils.sysconfig; print distutils.sysconfig.get_python_lib(prefix='$HOME/Easy
```

Install with admin rights

If you do have admin rights on the system where you want to install EasyBuild, you can simply omit the `--prefix $HOME/EasyBuild/` to have EasyBuild installed system-wide. In that case, you do not need to touch the `$PATH` or `$PYTHONPATH` environment variables since the `eb` command will be installed in one of the default paths.

Alternatives to –prefix

As an alternative to `--prefix` when you do not have admin rights, you can specify that EasyBuild should be installed in your `$HOME` directory using the `--user` option.

The full list of commands to install EasyBuild in your `$HOME` directory using `pip` would be:

```
pip install --user easybuild
export PATH=$HOME/.local/bin:$PATH
```

Warning: In our experience, using `--user` creates more problems than it solves. We have run into unexpected behavior with Python software installed in your home directory using `--user`, for example it **always** being preferred over versions installed somewhere else. Hence, we strongly discourage using *–user* to install EasyBuild (or other Python software).

Installing the EasyBuild packages separately

Each of the EasyBuild packages can also be installed separetely:

```
pip install --install-option "--prefix=$HOME/EasyBuild" easybuild-framework
pip install --install-option "--prefix=$HOME/EasyBuild" easybuild-easyblocks
pip install --install-option "--prefix=$HOME/EasyBuild" easybuild-easyconfigs
```

This is the exact same sequence of steps as they will be performed when running `pip install --install-option "--prefix=$HOME/EasyBuild" easybuild`.

8.6.2 Installation from downloaded sources

To install one of the EasyBuild packages from a downloaded source tarball, use the following steps:

```
tar xfvz easybuild-framework-1.0.tar.gz
cd easybuild-framework-1.0
pip install --install-option "--prefix=$HOME/EasyBuild" .
```

Do note that when an EasyBuild package is being installed without having the EasyBuild packages that it depends upon available, both `easy_install` and `pip` will try and pull in the latest available version of those packages from PyPi.

Thus, to have full control over the EasyBuild installation, you need to respect the following installation order: easybuild-framework, easybuild-easyblocks, easybuild-easyconfigs. The easyblocks package depends on the framework package; the easyconfigs package depends on both the framework and easyblocks packages.

If you do not have `pip` or `easy_install` available, you can also fall back to using the `setup.py` script directly:

```
python setup.py --prefix $HOME/EasyBuild install
```

8.6.3 Installation of latest release from GitHub

To install the latest (stable) release of an EasyBuild package directly from GitHub, use the following command:

```
pip install --install-option "--prefix=$HOME/EasyBuild" http://github.com/hpcugent/easybuild-frame
```

Again, the order in which the EasyBuild packages are installed is important to have full control over the installation process, see previous section.

8.6.4 Installation of latest development version

To install the latest development version of an EasyBuild package from GitHub, you can simply adjust the command from the previous section to install from the `develop` branch (or any of the available feature branches in any EasyBuild repository for that matter):

```
pip install --install-option "--prefix=$HOME/EasyBuild" http://github.com/hpcugent/easybuild-frame
```

Note: You should use this only if you are interested in developing for EasyBuild. Although it is well tested, the development version of the EasyBuild repositories may be unstable at a given point in time.

8.6.5 Installation of latest development version using provided script

After you have forked each of the EasyBuild repositories on GitHub (+ vsc-base), you can set up a development version of EasyBuild using the `install-EasyBuild-develop.sh` script.

This script will clone the different EasyBuild repositories from GitHub:

- easybuild: EasyBuild metapackage & documentation sources for http://easybuild.readthedocs.org
- vsc-base: dependency for EasyBuild framework (logging, command line interface, ...)
- easybuild-framework: EasyBuild framework
- easybuild-easyblocks: collection of easyblocks
- easybuild-easyconfigs: collection of easyconfig files
- easybuild-wiki: EasyBuild wiki pages

It can be used as follows:

```
# pick an installation prefix (adjust as you like)
INSTALL_PREFIX=$(mktemp -d $HOME/EasyBuild-XXXXXX)
# download script
curl -O https://raw.githubusercontent.com/hpcugent/easybuild-framework/master/easybuild/scripts/i
# run downloaded script, specifying *your* GitHub username and the installation prefix
bash install-EasyBuild-develop.sh GITHUB_USERNAME $INSTALL_PREFIX
# update $MODULEPATH via 'module use', and load the module
module use $INSTALL_PREFIX/modules
module load EasyBuild-develop
eb --version  ## This should ensure you have a reasonable instance of EasyBuild
```

Note: The above creates a module file which you can load/inspect at will. The interesting aspect about it is that it is pointing to an EasyBuild installation directly on local git repositories, which allows you to customise it easily. Remember to commit/push or otherwise save your changes, if you intend to use them later.

8.7 Installing environment modules without root permissions

This short guide will explain how to install the standard environment modules Tcl/C software package without root permissions on a Linux or Mac OS X system, together with Tcl on which it depends.

8.7.1 Tcl

1. Go to http://www.tcl.tk and download the latest Tcl sources. At the time of writing, the latest available Tcl version was 8.5.15, which can be downloaded from here. The remainder of these commands will assume Tcl v8.5.15 is being installed, you may need to adjust them accordingly. **Note**: Stick to Tcl v8.5.x, don't consider using the more recent v8.6.x or higher, since the environment modules package is not compatible with those Tcl versions.
2. Unpack the Tcl source tarball:

```
tar xfvz tcl8.5.15-src.tar.gz
```

3. Pick a location where you will install Tcl. It should be a directory you have write permissions on. My suggestion would be to use something like `$HOME/.local/Tcl`.
4. Go to the `unix` subdirectory of the unpacked Tcl directory, and run the `configure` script using the `--prefix` option:

```
cd tcl8.5.15/unix
./configure --prefix=$HOME/.local/Tcl
```

If you're building Tcl and environment modules on Mac, you should run `configure` in the `tcl8.5.15/macosx` directory instead.

5. Next, build Tcl using the `make` command. If the system you are building on has multiple cores, running make in parallel will speed up the build. Just use the `-j` option, and pass it a degree of parallelism (just use the number of cores your system has available), e.g.:

```
make -j 4
```

6. The final step consists of installing Tcl to the directory specified in step 4. To do this, simply run:

```
make install
```

All done! Now you are ready to build the environment modules package, which requires Tcl.

8.7.2 Environment Modules

1. Download the latest source tarball for the environment modules tools from http://modules.sourceforge.net/. At the time of writing, the latest available version is 3.2.10 which can be downloaded from here.
2. Unpack the downloaded source tarball:

```
tar xfvz modules-3.2.10.tar.gz
```

3. Configure the build, again use `--prefix` to specify where to install the environment modules tool in the end. If you needed to install Tcl by hand as outlined in the previous section, you'll also need to specify where it was installed using the `--with-tcl` option:

```
cd modules-3.2.10
./configure --prefix=$HOME/.local/environment-modules --with-tcl=$HOME/.local/Tcl/lib
```

4. Build with `make`, consider parallel build if your system is recent enough:

```
make -j 4
```

5. Install:

```
make install
```

Alright, now just one more thing...

8.7.3 Set up your environment

Because you've installed environment modules and Tcl in a non-default location, you need to make sure your environment is setup up correctly to use them.

To make a long story short, these are the commands you need to execute:

```
export PATH=$HOME/.local/environment-modules/Modules/3.2.10/bin:$PATH
export LD_LIBRARY_PATH=$HOME/.local/Tcl/lib:$LD_LIBRARY_PATH
# adjust line below if you're using a shell other than bash, check with 'echo $SHELL'
source $HOME/.local/environment-modules/Modules/3.2.10/init/bash
```

Tip: Add these three lines in your `.bashrc` file, that way they'll be executed every time you log in.

8.8 Installing Lmod without root permissions

This short guide will show how to install Lmod (and Lua, on which it depends) on Linux, without requiring root permissions.

8.8.1 Lua

Build and install Lua using the source tarball available in the Lmod SourceForge repository (http://sourceforge.net/projects/lmod/files/). This version is a lot easier to build, and already includes the required extra Lua modules. At the time of writing this relates to the `lua-5.1.4.8.tar.gz` tarball.

Step 1: Download and unpack lua-5.1.4.8.tar.gz.

Step 2: Configure the Lua build, provide a custom installation prefix (e.g. `$HOME/lua`) and specify to statically link libraries (i.e. `libreadline` and `ncurses`), to avoid problems when modules that provide these libraries are being loaded. Then build and install via `make`:

```
./configure --with-static=yes --prefix=$HOME/lua && make && make install
```

Step 3: Make sure the `lua` binary is available in your `$PATH` (only required when building Lmod, see below):

```
export PATH=$HOME/lua/bin:$PATH
```

Optionally, check whether the `lua` binary indeed doesn't link to any unexpected `readline` or `ncurses` libraries:

```
$ ldd $HOME/lua/bin/lua
        linux-vdso.so.1 (0x00007fffad7ff000)
        libm.so.6 => /lib64/libm.so.6 (0x00007ff9914db000)
        libdl.so.2 => /lib64/libdl.so.2 (0x00007ff9912d7000)
        libc.so.6 => /lib64/libc.so.6 (0x00007ff990f2a000)
        /lib64/ld-linux-x86-64.so.2 (0x00007ff9917d9000)
```

8.8.2 Lmod

Step 1: Download and unpack the latest available Lmod version, Lmod-6.1.tar.bz2 at the time of writing.

```
tar xfvj Lmod-6.1.tar.bz2 && cd Lmod-6.1
```

Step 2: Configure, build and install Lmod build, in a custom prefix:

```
./configure --prefix=$HOME && make install
```

Step 3: Update `$PATH` so `lmod` is available (put this in your `.bashrc`):

```
export PATH=$HOME/lmod/6.1/libexec:$PATH
```

Optionally, give it a spin:

```
$ lmod --version

Modules based on Lua: Version 6.1  2016-02-05 16:31
    by Robert McLay mclay@tacc.utexas.edu
```

Step 4: Define `module` function to use `lmod` (optional for use with EasyBuild):

```
source $HOME/lmod/6.1/init/bash
export LMOD_CMD=$HOME/lmod/6.1/libexec/lmod
```

8.9 Useful links

Tools used to convert from other wiki formats and prepare .rst when in pre-production mode:

- http://johnmacfarlane.net/pandoc/try/?text=%0A&from=markdown_github&to=rst
- http://rst.ninjs.org

8.10 Sphinx and Read the Docs reference documents online

- https://github.com/rtfd/readthedocs.org
- http://docutils.sourceforge.net/docs/ref/rst/restructuredtext.html
- http://matplotlib.org/sampledoc/cheatsheet.html
- http://openalea.gforge.inria.fr/doc/openalea/doc/_build/html/source/sphinx/rest_syntax.html
- http://thomas-cokelaer.info/tutorials/sphinx/rest_syntax.html
- http://docutils.sourceforge.net/docs/user/rst/quickref.html
- http://nipy.org/devel/guidelines/sphinx_helpers.html
- https://pythonhosted.org/an_example_pypi_project/sphinx.html
- http://openmdao.org/docs/documenting/sphinx.html

8.11 Third party Sphinx examples, themes and possible extensions

- http://sphinx.readthedocs.org/en/latest/examples.html
- http://django.readthedocs.org/en/latest/
- http://docs.cython.org/index.html
- http://docs.writethedocs.org/tools/sphinx-themes/
- http://ryan-roemer.github.io/sphinx-bootstrap-theme/examples.html
- http://django-profiletools.readthedocs.org/en/latest/
- http://pythonhosted.org/korean/
- https://pythonhosted.org/cloud_sptheme/cloud_theme.html#features
- http://openmdao.org/docs/documenting/sphinx.html
- http://sphinx-doc.org/genindex.html

8.12 EasyBuild release notes

The latest version of EasyBuild provides support for building and installing **998** different software packages, using 37 different (compiler) toolchains. It contains 174 software-specific easyblocks and 29 generic easyblocks, alongside 6,548 easyconfig files.

8.12.1 v2.8.2 (July 13th 2016)

bugfix release

framework

- various small enhancements, including:
  - add support for rst output for `--list-*` and `--avail-*` (#1339)
  - add support for 'eb `--check-conflicts`' (#1747, #1807, #1833)
  - ensure nice error message when non-existing path is passed to `apply_regex_substitutions` (#1788)

  - add check for module output, empty stdout is a sign of trouble with Lmod (#1793)
  - add multi-threaded FFT to toolchain (#1802)
  - avoid special characters like '[', ']' in path to temporary directory (#1808)
  - add support for `--zip-logs` (#1820)
  - add support for `--extra-modules` (#1821)
  - add type conversion for 'checksums' and 'patches' parameter in .yeb easyconfigs (#1826, #1840)
  - add support for filtering tests by name (#1828)
  - add support for `--avail-toolchain-opts` (#1830, #1839)
  - use absolute path for robot and easyconfig files (#1834)
  - add backup URL for tarballs hosted on SourceForge in `install_eb_dep.sh` script (#1843)
- various bug fixes, including:
  - fix installation of Lua in `install_eb_dep.sh` script (#1789)
  - fix OpenMP flag for Cray compiler wrappers (#1794)
  - only reset `$MODULEPATH` before loading a module if environment was reset (#1795)
  - include vsc-install as dependency in `setup.py` (#1805)
  - cache `$PATH` & `$PYTHONPATH` in test setUp, restore them in tests where '`eb`' is used (#1806)
  - don't reset `$MODULEPATH` in stage 2 of bootstrap script, support forced installation during stage 2 (#1810)
  - fix issue with templates defined by deps being required while still parsing deps (#1812)
  - skip unneeded unuse/use commands on tail of `$MODULEPATH` in `check_module_path` (#1813)
  - fix auto-convert for all `*dependencies` params in `.yeb` easyconfigs, ensure version is a string (#1818)
  - fix `keyring` version in Travis config (#1819)
  - fix dumping of `.yeb` easyconfig files in easyconfigs archive (#1822)
  - fix format of supported easyconfig templates in help output (#1825)
  - stick to `pydot` 1.1.0 for Python 2.6 in Travis config (#1827)

easyblocks

- new easyblocks for 5 software packages that require customized support:
  - Amber (#958), Extrae (#955), Gurobi (#962), Paraver (#956), Tau (#887)
- various enhancements, including:
  - add support for building & installing old GROMACS versions (#569, #960)
  - add support for building Boost with Cray toolchain (#849)
  - libxsmm support for CP2K (#942)
  - pick up specified components for imkl (#943)
  - add support for building GROMACS with double precision (#946, #960)
  - add support for building GROMACS with CUDA support and using dynamic libraries using `Cray` toolchains (#951, #960)
  - also install vsc-install in `EasyBuildMeta` easyblock, if tarball is provided (#957)
  - enhance PSI easyblock to support PSI4 1.0 (#965)
- various bug fixes, including:

- also install scripts with MRtrix 0.3.14 (#941)
- enhance Qt easyblock to support Qt3 (#944)
- create 'release' symlink in MRtrix install dir (#947)
- fix make_installdir in Inspector & VTune easyblocks (#952)
- make Binary and MakeCp easyblocks aware of 'keepsymlinks' (#959)
- correctly define $G4* environment variables in Geant4 easyblock (#961, #970)
- prepend tmp install path to $PYTHONPATH in numpy test step, move to build dir when removing 'numpy' subdir (#963)
- correct full path to ALADIN config file & patch it to use right Fortran compiler flags (#964)
- ensure correct compiler command/flags are used for SAMtools (#966)

easyconfigs

- added example easyconfig files for 54 new software packages:
  - Amber (#3200), Bullet (#3175), CONTRAlign (#690), Cluster-Buster (#3191), damageproto (#3222, #3308), DCA++ (#3219), EIGENSOFT (#3147, #3163), Extrae (#507), fdstools (#3237), ffnet (#3273), GP2C (#3257), Gurobi (#3239), gc (#3202, #3261), gputools (#546), IMa2p (#3300), IOzone (#3253), i-cisTarget (#3191, #3194), icmake (#3243), io_lib (#3255), Kent_tools (#3191), libcmaes (#3256), libpsortb (#3259), libxsmm (#3099), MEGACC (#3263), MM-align (#1428), MOSAIK (#880), MView (#1345), MySQL-python (#3172, #3189), magma (#3219), mrFAST (#862), mrsFAST (#862), mysqlclient (#3172, #3232), NTL (#3183), PARI-GP (#3257), Paraver (#508), psutil (#3171, #3231), PSI4 (#3293), Qwt (#3157), RMBlast (#3142), STAMP (#3191), Seqmagick (#3264), splitRef (#946), TAU (#509), TRF (#3141), TVB (#3053, #3247, #3251), TVB-deps (#3053, #3247, #3251), tvb-data (#3053, #3247, #3251), tvb-framework (#3053, #3247, #3251), tvb-library (#3053, #3247, #3251), VampirTrace (#509), Voro++ (#3174), wheel (#3235), wxPropertyGrid (#508), xonsh (#3159)
- added easyconfigs for update of common toolchains: foss/2016b (#3271), intel/2016b (#3270)
- added new easyconfigs for existing toolchains: CrayGNU/2016.03 & CrayGNU/2016.04 (#3291), foss/2016.06 (#3184), intel/2016.03-GCC-5.4 (#3185)
- added additional easyconfigs for various supported software packages, including:
  - Boost 1.61.0, GCC 5.4.0, GROMACS 3.3.3, HDF5 1.8.17, netCDF 4.4.1, numpy 1.11.0, Perl 5.24.0, PETSc 3.7.2, Python 2.7.12, Python 3.5.2, Qt 3.3.8, R 3.3.1
- various enhancements, including:
  - use check_conflicts function in easyconfigs tests (#2981)
  - also include vsc-install in list of sources for recent EasyBuild easyconfigs, to support offline installation (#3203)
  - enable building of libmysqld.* in MariaDB easyconfigs (#3230)
  - add ALDEx2, phyloseq to bundles for Bioconductor 3.2 (#3211, #3241)
  - add biom, geepack, lubridate, pim to list of R 3.2.3 extensions (#3186, #3211, #3275)
- various bug fixes, including:
  - add patch for Boost 1.60.0 to fix bug resulting in TypeError (#3162)
  - add fftw dependency to CP2K 2.6.0 easyconfigs using CrayGNU (#3176)
  - fix location of libelf.h, only (also) installed as include/libelf.h is there's no /usr/include/libelf.h (#3201)
  - fix software name for Guile & GnuTLS (was 'guile' & 'gnutls') (#3207)
  - added missing space in Geant4 configopts to specify -DGEANT4_INSTALL_DATA (#3209)
  - fix Cython download URL in Python 2.7.11 easyconfigs (#3212)

  - add missing build deps for X stack in easyconfigs using `foss/2016a` or `intel/2016a` (#3222, #3308)
  - fix overruling of exts_list in Perl 5.22.2 easyconfig (#3224)
  - add missing dependency on GMP in R 3.2.3 easyconfigs (#3226)
  - don't hard specify toolchain for binutils build dep in likwid easyconfig, since it matches parent toolchain (#3240)
  - fix `homepage` & `source_urls` in HMMER easyconfigs (#3246)
  - stick to `pydot` 1.1.0 for Python 2.6 in Travis config (#3282)
  - add `python-dev(el)` to OS deps in GC3Pie easyconfigs (#3310)

8.12.2 v2.8.1 (May 30th 2016)

bugfix release

framework

- various bug fixes, including:
  - fix error message on missing module command in bootstrap script (#1772)
  - expand '~' in paths specified to `--include-*` (#1774)
  - break after deleting cache entry to avoid attempt to delete cache entry again (#1776)
  - avoid changing `$MODULEPATH` when prepending with symlink of path already at head of `$MODULEPATH` (#1777)
  - filter out duplicates in `find_flexlm_license` (#1779)
  - stick with GitPython < 2.0 with Py2.6 in Travis configuration (#1781)
  - don't use `LooseVersion` to define `version_major`/`version_minor` (#1783)

easyblocks

- various enhancements, including:
  - update MRtrix easyblock for version 0.3.14 (#932)
  - update Inspector easyblock for recent versions (#934)
  - update VTune easyblock for recent versions (#935)
  - add debug message to IntelBase easyblock w.r.t. switching to 'exist_lic' (#936)

easyconfigs

- added example easyconfig files for 13 new software packages:
  - drFAST (#906), git-lfs (#2478), grabix (#3127), JWM (#3007), libcroco (#3007), librsvg (#3007), MaCH (#3136), mayavi (#3106), OpenMM (#2762), Pysam (#3080), SeqPrep (#3097), vt (#3128), wkhtmltopdf (#3098)
- added new easyconfigs for existing toolchains: `intel/2016.03-GCC-4.9` (#3088)
- added additional easyconfigs for various supported software packages, including:
  - Boost 1.61.0, ESMF 7.0.0, Inspector 2016 update 3, IPython 4.2, netCDF-C++4 4.3.0, netCDF-Fortran 4.4.4, Perl 5.22.2, VTune 2016 update 3
- various bug fixes, including:
  - apply libreadline patch to fix bug triggering segmentation fault (#3086)

8.12.3 v2.8.0 (May 18th 2016)

feature + bugfix release

framework

- significant speedup improvements of EasyBuild itself, thanks to:
  - stop creating `ModulesTool` instances over and over again (#1735)
  - cache result of '`module avail`' calls (#1742)
- add support for using PGI as toolchain compiler (#1342, #1664, #1759, #1761, #1764)
  - incl. new toolchain definitions `pompi` and `pomkl` (#1724)
- add test configuration for Travis (#1733, #1737, #1743, #1767)
- various other enhancements, including:
  - add `get_total_memory()` function in `systemtools` module (#1623)
  - ignore `__init__.py` files in `--include-*` (#1704)
  - use `-fopenmp` rather than `-openmp` for Intel compilers, since `-openmp` is deprecated (#1718)
  - add modules to metadata for Cray modules (#1721)
  - make sure user write permissions are set after failed removal attempt of installation directory (#1722)
  - escape special characters in software name in `find_related_easyconfigs` (#1726)
  - add support for `CrayPGI` compiler toolchain (#1729)
  - ensure read permission to all installed files for everybody (unless other options specify otherwise) (#1731)
  - also consider `$LMOD_CMD` in bootstrap script (#1736)
  - translate PyPI download URL to alternate URL with a hash (#1749)
  - make `get_software_libdir` compatible with `-x` (#1750)
  - set `$LMOD_REDIRECT` to '`no`' when initialising Lmod (#1755)
  - add test for broken modules tool setup affecting '`module use`' (#1758)
- various bug fixes, including:
  - isolate '`options`' tests from easyblocks other than the ones included in the tests (#1699)
  - don't run '`module purge`' in tests, since EasyBuild may be made available through a module (#1702)
  - avoid rehandling `--include-*` options over and over again during `--show-config` (#1705)
  - remove useless `test_cwd` (#1706)
  - fix bootstrap script: make sure setuptools installed in stage0 is still available at end of stage1 (#1727)
  - forcibly create target branch in `--update-pr` (#1728)
  - remove check whether '`easybuild`' is being imported from dir that contains `easybuild/__init__.py` (#1730)
  - (re)install vsc-base during stage1 using `--always-copy` in bootstrap script, if needed (#1732)
  - use `os.path.realpath` in `test_wrong_modulepath` to avoid symlinked path breaking the test (#1740)
  - unset `$PYTHONPATH` in before tested bootstrapped EasyBuild module (#1743)
  - take into account that paths in modulepath may be symlinks in `test_module_caches` (#1745)

  - change to install dir rather than buildpath in sanity check of extension, latter may not exist (#1746, #1748)
  - only load modules using short module names (#1754)
  - (re)load modules for build deps in extensions_step (#1762)
  - fix `modpath_extensions_for method`: take into account modules in Lua syntax (#1766)
  - fix broken link to VSC website in license headers (#1768)

easyblocks

- add test configuration for Travis (#895, #897, #900, #926)
- new easyblocks for 4 software packages that require customized support:
  - binutils (#907), libQGLViewer (#890), SuperLU (#860), wxPython (#883)
- various other enhancements, including:
  - update SuiteSparse easyblock for version >= 4.5 (#863)
  - enhance imkl easyblock to install on top of PGI (#866, #916)
  - enable runtime logging of install cmd in `IntelBase` (#874)
  - enhance Qt easyblock to support installing with `dummy` toolchain (#881)
  - delete libnuma symbolic links in PGI installation directory (#888)
  - enhance PDT easyblock to support installing with `dummy` toolchain (#894)
  - add support for building Clang with OpenMP support (#898)
  - update Score-P easyblock for additional compilers, MPI libraries & dependencies (#889)
  - drop deprecated '`testrb`' from sanity check in Ruby easyblock (#901)
  - enhance WRF easyblock to support versions >= 3.7 (#902)
  - update QuantumESPRESSO easyblock for version 5.3.0 (#904)
  - add support in PythonPackage easyblock to use '`setup.py develop`' (#905)
  - update Qt easyblock for Qt 5.6.0 (#908)
  - extend bzip2 easyblock to also build dynamic libraries (#910)
  - make threading an explicit option rather than relying on MPI library in SCOTCH easyblock (#914)
  - update PGI easyblock to install siterc file so PGI picks up `$LIBRARY_PATH` (#919)
  - enhance sanity check paths for compiler commands in PGI easyblock (#919)
  - also filter out `-ldl` from $LIBBLAS & co for Intel MKL in numpy easyblock (#920)
  - define `$MIC_LD_LIBRARY_PATH` for impi (#925)
- various bug fixes, including:
  - don't hardcode Python version in `test_make_module_pythonpackage` (#876)
  - make PythonPackage easyblock compatible with `--module-only` (#884, #906)
  - remove check whether '`easybuild`' is being imported from dir that contains `easybuild/__init__.py` (#891)
  - fix passing compiler configure option in PDT easyblock (#894)
  - fix bug in Score-P easyblock w.r.t. `--with-libbfd` (#889)
  - fix extension filter for Ruby (#901)
  - fix `ACTIVATION_TYPES` list in IntelBase + minor style change (#913)
  - correctly define `$MIC_LD_LIBRARY_PATH` in imkl 11.3.x and newer (#915)

  - fix broken link to VSC website in license headers (#927)

easyconfigs

- added example easyconfig files for 69 new software packages:
  - ALPS (#2888), annovar (#3010), BayeScEnv (#2765), BayesAss (#2870), BerkeleyGW (#2925), Blitz++ (#2784, #3004), bam-readcount (#2850), Commet (#2938), CrossTalkZ (#2939), cuDNN (#2882), DBus (#2855), DFT-D3 (#2107), DIAL (#3056), dask (#2885), dbus-glib (#2855), FFLAS-FFPACK (#2793), FLAC (#2824), FLANN (#3015, #3029), FLEUR (#3043), GConf (#2855), GROMOS++ (#1297), GST-plugins-base (#2855), GStreamer (#2855), GTOOL (#2805), Givaro (#2793), gdist (#2935), gromosXX (#1297), HISAT2 (#2809), i-PI (#2940), Kraken (#3037, #3041), LAME (#2823), LASTZ (#3002), LinBox (#2793), Loki (#2839), libQGLViewer (#2923, #3008), libXxf86vm (#2855), MDSplus (#2787, #2838, #3027), MRIcron (#2831), Mawk (#2732), minieigen (#2839), mpmath (#3058), NBO (#3047, 3048), NGS (#2803), NGS-Python (#2810), ncbi-vdb (#2808), OptiX (#2795), PCL (#3024), PEAR (#2731), PLplot (#2990), Postgres-XL (#2891), PyGTS (#2969), RSeQC (#2788), Rust (#2920, #2943), rainbow (#2730), SHAPEIT (#2806), SIONlib (#2908), Saxon-HE (#2773), Singularity (#2901), SoX (#2825), Subread (#2790), SuperLU (#2665), travis (#2953), VASP (#2950), Wannier90 (#2906, #3042), wget (#3041), wxPython (#2855), xf86vidmodeproto (#2855), Yade (#2839), Yambo (#2932)
- add test configuration for Travis (#2942, #2944, #2954, #3061)
- added easyconfigs for new PGI-based toolchains
  - `pomkl/2016.03` (#2899, #2900, #3046), `pomkl/2016.04` (#3044), `CrayPGI/2016.04` (#2927)
- added new easyconfigs for existing toolchains:
  - `foss/2016.04` (#3013), `intel/2016.02-GCC-5.3` (#2523), `intel/2016.03-GCC-5.3` (#3009)
- added additional easyconfigs for various supported software packages: version updates, different toolchains, ...
  - incl. CGAL 4.8, Clang 3.8.0, icc/ifort 2016.2.181 & 2016.3.210, imkl 11.3.2.181 & 11.3.3.210, impi 5.1.3.181, LLVM 3.8.0, OpenCV 2.4.12, pandas 0.18.0, Qt 5.6.0, Scalasca 2.3, Score-P 2.0.1, SuiteSparse 4.5.2, WRF 3.8
- various other enhancements, including:
  - enhance ORCA easyconfig for compatibility with SLURM (#1819)
  - enable `-fPIC` in GraphicsMagick easyconfig, required by Octave (#2764)
  - clean up binutils easyconfigs to use binutils easyblock (#3006)
  - add `include/GraphicsMagick` to `$CPATH` in GraphicsMagick easyconfigs (#3034)
  - update SuiteSparse easyconfigs according to updated SuiteSparse easyblock (#3050)
- various bug fixes, including:
  - fix Perl extensions download urls (#2738)
  - add Autoconf as build dep for `GCCcore` (#2772)
  - fix versions of extensions in Bioconductor 3.2 bundles (#2769)
  - fix (build) deps for `intel/2016a` easyconfigs of cairo, libXext, libXrender (#2785, #2874)
  - use ‘`env`‘ wherever preconfig/build/installopts is used to set environmental variables (#2807, #2811, #2812)
  - add zlib as explicit dep in Tk easyconfigs (#2815)
  - consistently specify to use `-fgnu89-inline` flag in M4 1.4.17 easyconfigs (#2774, #2779, #2816)
  - fix homepage and description in Pygments easyconfigs (#2822)

  - include pkg-config as build dependencies for libXau, libXdmcp, libxcb (#2827)
  - consistently use `XORG_*_SOURCE` constants (#2829, #2830, #2848)
  - update source URLs in ScientificPython easyconfig files (#2847)
  - add checksums in SuiteSparse easyconfigs (#2849)
  - fix build deps for GObject-Introspection (#2852)
  - correctly specify Perl location in git easyconfig (#2866)
  - fix bitstring 3.1.3 download URL in Python easyconfigs, source tarball on PyPI disappeared (#2880)
  - fix Perl dependency in worker easyconfigs, it requires non-standard Perl modules (#2884)
  - add XZ as dependency in Python 3.5.1 easyconfigs, required for lzma (#2887)
  - fix download URL for packmol (#2902)
  - drop `usempi` toolchain in numexpr easyconfigs, not needed (#2937)
  - fix use of `resolve_dependencies` in tests according to changes in framework (#2952)
  - add dependency extensions for MarkupSafe and jsonscheme in IPython 3.2.3 easyconfigs (#2967)
  - add patch for matplotlib 1.5.1 to fix Tcl/Tk library paths being used (#2971)
  - add xproto build dependency for makedepend v1.0.5 (#2982)
  - disable parallel build for Doxygen (#2986)
  - fix source URLs for `FreezeThaw` and `Tie::Function` extensions for Perl v5.22.1 (#2988)
  - add `sed` command in worker easyconfig files to fix module_path in conf/worker.conf (#2997, #3000)
  - drop toolchainopts from Eigen easyconfigs, since it is headers-only (#3025)
  - clean up dummy bzip2 easyconfig, define buildopts rather than defining `$CC` and `$CFLAGS` via `os.environ` (#3036)
  - use `%(pyshortver)s` template rather than hardcoding 2.7 in VTK easyconfigs (#3052)
  - correct install location of OpenCV Python bindings (#3054)
  - include XZ as dependency for libunwind (#3055)
  - add patch to fix broken OpenSSL tests due to expired certificates (#3057)
  - fix broken link to VSC website in license headers (#3062)

8.12.4 v2.7.0 (March 20th 2016)

feature + bugfix release

framework

- stabilize Cray support
  - enable '`dynamic`' toolchain option by default for `Cray*` toolchains (#1581)
  - remove FFTW from the Cray toolchains definition (#1585)
  - add external modules metadata for Cray systems (#1638)
  - fix independency of Cray toolchains w.r.t. toolchain build environment (#1641, #1647)
  - remove requirement to use `--experimental` for Cray toolchains (#1663)
- enable Python optimization mode in '`eb`' (#1357)
- improved GitHub integration
  - improve error handling on git commands + better logging for `--new-pr`/`--update-pr` (#1590)

  - use `git` rather than `https` in `--new-pr`/`--update-pr` (#1602)
  - add `-u` as shorthand for `--upload-test-report` (#1605)
  - fix `--from-pr` for PRs that include renamed/deleted files (#1615)
  - add support for `--install-github-token` and `--check-github` (#1616)
  - fix `fetch_easyconfigs_from_pr` w.r.t. duplicate files in PRs (#1628)
- various other enhancements, including:
  - add support for `--search-filename` and `--terse` (#1577)
  - support complete bash completion (#1580)
  - add support for `%(*ver)s` and `%(*shortver)s` templates (#1595, #1604)
    - incl. `%(javaver)s`, `%(javashortver)s`, `%(perlver)s`, `%(perlshortver)s`, `%(pyver)s`, `%(pyshortver)s`, `%(rver)s`, `%(rshortver)s`
  - define `HOME` constant that can be used in easyconfig files (#1607)
  - implement support for generating '`swap`' statements in module files (#1609)
  - add support for `--show-config` (#1611, #1620)
  - simplified support for `--minimal-toolchains` (#1614, #1619, #1622, #1625, #1646)
  - add support for `--dump-env-script` (#1624)
  - enhance ModulesTool.exist to also recognize partial module names (#1630)
  - improve error message for toolchain definition errors (#1631)
  - make default is_short_modname_for check less strict to support versionless external modules as deps (#1632)
  - mention hostname in comment made by `--upload-test-report` (#1635)
  - support providing additional relative path for prefix in external module metadata (#1637)
  - add `ThematicModuleNamingScheme` (#1645)
  - enhance logging format: remove logger name, mention location instead (#1649, #1654)
  - update kernel versions for SLES12 (#1659)
  - raise `EasyBuildError` rather than `ImportError` in `only_if_module_is_available` decorator (#1662)
- various bug fixes, including:
  - fix Lmod spider output in generated modules (#1583)
  - correctly define '`easybuild`' namespaces (#1593, #1666, #1680)
    - this change requires that the `setuptools` Python package is available (at runtime)
    - using custom easyblocks by adding them in the Python search path (`$PYTHONPATH`) may require adjustments, i.e. also using `pkg_resources.declare_namespace` in the `__init__.py` files; *we highly recommend to use* `--include-easyblocks` *instead*, see http://easybuild.readthedocs.org/en/latest/Including_additional_Python_modules.html
    - note: this has the side-effect of not being able anymore to reliably use '`eb`' in the parent directory of the easybuild-framework repository (#1667)
  - fix template for `savannah.gnu.org` source URL (#1601)
  - stop running '`module purge`', only restore environment (#1608)
  - fix license headers: Hercules foundation is now FWO (#1629)
  - avoid that `fancylogger` tries to import `mpi4py` to determine MPI rank (#1648)

  - fix error in tests when '`file`' backend is not available in Python keyring (#1650)
  - update develop install script (#1651)
  - handle allowed system deps during `prepare_step` rather than during parsing of easyconfig (#1652)
  - add function to find FlexLM licenses: `find_flexlm_license` (#1633, #1653)
  - fix availability check for external modules with partial module name (#1634, #1643)
  - fix bootstrap script to ensure `setuptools` is also installed (#1655)
  - fix issue in bootstrap script with `vsc-base` being picked up from the OS (#1656)
  - fix bootstrap script for environment where '`python`' is Python 3.x (#1660)
  - remove `--experimental` for tests related to `--package` (#1665)
  - ensure path to setuptools is included in `$PYTHONPATH` being used to test scripts (#1671)
  - sanitize environment before initializing easyblocks (#1676)
  - remove `reload` statements in `include.py`, since they are not required and break `--include-toolchains` (#1679)

easyblocks

- new easyblocks for 6 software packages that require customized support:
  - ADF (#826), MPICH (#844, #852, #868), mutil (#859), pplacer (#835), psmpi (#852), SNPhylo (#865)
- various other enhancements, including:
  - implement support for '`use_pip`' in PythonPackage easyblock (#719, #831)
  - add support in CUDA easyblock to install wrappers for host compilers (#758)
  - update sanity check for picard version 1.124 and above (#796)
  - use '`module swap`' for all components in `CrayToolchain` (#823)
  - update PSI4 easyblock to cope with changed name of PSI4 data dir (#824)
  - use `find_flexlm_license` function and avoid defining `$CPATH` in PGI easyblock (#837)
  - use `find_flexlm_license` function in `IntelBase` generic easyblock (#839)
  - add unit test to check module file generated by `PythonPackage` easyblock (#841)
  - rework MVAPICH2 easyblock on top of new MPICH easyblock (#844)
  - add CUDA support in CP2K easyblock (#850)
  - also define `$LD` in `buildopts` for GATE (#855)
  - use `find_flexlm_license` function in TotalView easyblock (#839)
  - enhance `MakeCp` easyblock to also support renaming of files while copying them (#859)
  - hunt for usable '`python`' command in `PythonPackage` easyblock when system Python is used (#861)
  - add sanity check in `easybuild/__init__.py` w.r.t. current working dir (#869)
  - change suffix of original file to `.easybuild` when using `fileinput` in impi easyblock (#870)
- various bug fixes, including:
  - make sure Python unicode settings match that of the system Python (#817)
  - remove FFTW related statements in HPL easyblock, since HPL doesn't require FFTW at all (#822)
  - use `pkg_resources.declare_namespace` rather than `pkgutil.extend_path` to declare '`easybuild`' namespaces (#827)
  - fix license headers: Hercules foundation is now FWO (#836)

  - fix check for non-empty lib dirs in `PythonPackage` easyblock (#840)
  - consider all Python lib dirs in sanity check for libxml2 (#842)
  - correctly handle deprecated configure options (`--with-hwloc`/`--enable-mpe`) in MVAPICH2 easyblock (#853)
  - use correct configure option for checkpoint/restart in MVAPICH2 easyblock (#854)
  - ensure list of Python lib dirs always has a '`lib/...`' entry (#858)
  - check whether `rpm`/`rpmrebuild` commands are available using '`which`', rather than checking for OS deps (#864)
  - fix `test_step` in UFC easyblock (#872)

easyconfigs

- added example easyconfig files for 63 new software packages:
  - ATSAS (#616, #2587), astropy (#2724, #2727), attr (#2706), BamUtil (#2654), BBMap (#2322), BH (#2508), CheMPS2 (#2445), CosmoloPy (#2723, #2727), csvkit (#2639), Firefox (#2648), FreeXL (#2422), GL2PS (#2667), Glade (#2631), htop (#2538), IGV (#2019), IGVTools (#2019), ImageMagick (#2438), jModelTest (#2529), KEALib (#2420), libcerf (#2656), libgcrypt (#2201), libglade (#2631), libgpg-error (#2201), libspatialite (#2431), LittleCMS (#2438), MAST (#2542), MLC (#2577), MPJ-Express (#2529), mutil (#2201), neon (#758), NextClip (#2544), npstat (#2686, #2703), Octopus (#2643), QuickFF (#2721), p4vasp (#2328), PCMSolver (#2445), PFFT (#2643), PHYLIP (#2694), pkgconfig (#2475, #2476), Platypus (#2618), pplacer (#1056), PRINSEQ (#2437, #2444, #2585), PyFFmpeg (#2501, #2519), PyGObject (#2443), PyGTK (#2443), PyOpenGL (#2628), pyringe (#2533), qrupdate (#2675), rgeos (#2635), rpmrebuild (#2402), shift (#2201), SNAPE-pooled (#2688), SNPhylo (#2701), sratoolkit (#2715), STAR-Fusion (#2463), statsmodels (#2719), StringTie (#2527), synchronicity (#2508), testpath (#2461), USEARCH (#2537), VarScan (#2464), vsc-install (#2165), Whoosh (#2725), xprop (#2645)
- added new easyconfigs for existing toolchains:
  - `intel/2016.02-GCC-4.9` (#2620), `gmpolf/2016a` & `gmvolf/2016a` (#2589)
- stable Cray-specific easyconfigs
  - delete deprecated Cray toolchains and easyconfig files (#2400)
  - don't hardcode `PrgEnv` version, remove `craype` and `fftw` components in Cray toolchains (#2554)
  - remove `-XC` versionsuffix for stable definitions for `Cray*` toolchains (#2714)
  - support for various software packages with `CrayGNU` and `CrayIntel` toolchains: CP2K, GROMACS, WRF
- added additional easyconfigs for various supported software packages: version updates, different toolchains, ...
  - including BWA 0.7.13, CMake 3.4.3, GATE 7.2, GROMACS 5.1.2, Mesa 11.1.2, netCDF 4.4.0, Perl 5.22.1, Python 3.5.1, R 3.2.3, R-bundle-Bioconductor 3.2, scipy 0.17.0, SuiteSparse 4.5.1
- various other enhancements, including:
  - copy `contrib` dir in Velvet easyconfigs so scripts are also available (#2456)
  - redefine matplotlib 1.5.1 easyconfig as a bundle, also include `cycler` extension (dep for matplotlib) (#2470)
  - add bitstring extension to Python 2.7.11 easyconfigs (#2471)
  - enable building of MetaVelvet in Velvet 1.2.10 easyconfigs (#2473)
  - add custom sanity check for libjpeg-turbo (#2480)
  - add Velvet easyconfigs that include BioPerl dependency, so VelvetOptimizer can use it (#2495, #2729, #2733)

  - add source URL in RAxML 7.2.6 easyconfigs (#2536)
  - update MPICH easyconfigs to use new MPICH easyblock (#2589)
  - free libX11 & co from unneeded Python dependency/versionsuffix (#2549, #2563, #2605, #2664)
  - add '`--enable-utf --enable-unicode-properties`' configure options in PCRE easyconfigs (#2561) * required for latest R versions
  - add HCsnip, metagenomeSeq in Bioconductor 3.1 bundles (#2553, #2578)
  - add additional extensions in R 3.2.x easyconfigs that are required for extra Bioconductor extensions (#2547, #2556)
  - update psmpi easyconfig files to use the new psmpi easyblock (#2619)
  - add easyconfig for Python 2.7.11 on top of X11-enabled Tk (#2614, #2621)
  - add virtualenv as extension in Python 2.7.11 easyconfigs (#2660)
- various bug fixes, including:
  - fix software name for GTK+ (was 'gtk+'), PyCairo (was 'pycairo') and Gdk-Pixbuf (was 'gdk-pixbuf') (#2468)
  - don't hardcode `CC`/`CXX` in OpenMPI easyconfigs (#2472)
  - remove Google Code source URL for mpi4py (#2474)
  - rename ffmpeg to FFmpeg (#2425, #2481)
  - use available easyblock for flex (#2486)
  - fix determining list of easyconfigs in unit test suite, don't assume locations are correct (#2530)
  - fix specifying DB dependency in DB_File easyconfigs (#2539)
  - remove hard-coded `-xSSE4.2` for numpy/scipy with Intel compilers (#2546)
  - fix license headers: Hercules foundation is now FWO (#2550)
  - add `--with-zlib` configure argument in libxml easyconfigs (#2555)
  - don't hardcode `optarch=True` in xextproto/xtrans easyconfigs (#2601)
  - change toolchain version to '' in easyconfigs that use `dummy` toolchain and include dependencies (#2612)
  - GLib doesn't require libxml2 with Python bindings (#2632)
  - add patch file to imkl 10.2.6.038 32-bit easyconfig to fix installer not being able to deal with '`--`' in build path (#2634)
  - add missing 'pkgconfig' dependency for h5py (#2476, #2650)
  - correct software name in FastQC easyconfigs (was 'fastqc'), use '`dummy`' toolchain for all FastQC version (#2657, #2666)
  - add missing libxml2 dependencies in GLib easyconfigs (#2658)
  - fix Xerces-C++ download location (#2668)
  - enable `XML::Bare` extension in all Perl easyconfigs (#2672)
  - update dead link for SuiteSparse (#2679)
  - remove custom `exts_filter` in easyconfigs used `PythonPackage` easyblock (#2683, #2685)
  - add M4 as build dep for binutils & flex (#2681)
  - add missing dependencies in Python 3.5.x easyconfigs: SQLite, Tk, GMP (#2704)
  - fix (OS) deps, add checksums, remove parameter definition with default values in MVAPICH2 easyconfigs (#2707)

- style cleanup in various easyconfigs (#2378, #2387, #2395, #2396, #2488-#2493, #2496-#2500, #2502-#2504, #2602)
  - working towards automated style review of pull requests

8.12.5 v2.6.0 (January 26th 2016)

feature + bugfix release

framework

- add (experimental) support for opening/updating (easyconfigs) pull requests (`--new-pr`, `--update-pr`) (#1528)
- sanitize environment before each installation by undefining `$PYTHON*` (#1569, #1572)
- various other enhancements, including:
  - allow user-local modules with hierarchical naming schemes (`--subdir-user-modules`) (#1472)
  - enhance `--extended-dry-run` output to include paths for requirements in `make_module_req` (#1520)
  - rewrite read_file to use '`with`' (#1534)
  - add support for `eb --last-log` (#1541)
  - support using fixed install dir scheme (`--fixed-installdir-naming-scheme`) (#1546)
  - add edge attributes for build dependencies in `--dep-graph` output (#1548)
  - check whether dependencies marked as external module are hidden (#1552)
  - implement support for `--modules-header` (#1558)
  - add support to specify '`else`' body for conditional statements in modules (#1559)
  - add extra test for `--include-easyblocks` for generic easyblocks (#1562)
  - allow user to define the default compiler optimization level (`--default-opt-level`) (#1565)
  - make `toolchain.get_variable` more robust w.r.t. dummy toolchain (#1566)
- various bug fixes, including:
  - fix missing '`yaml`' module check in tests (#1525)
  - fix 'develop' install script (#1529)
  - correctly quote FPM option values in packagin support (#1530)
  - correctly handle '`.`' in software name w.r.t. `$EB*` environment variables (#1538)
  - exclude logs and test reports from packages (#1544)
  - also pass down `--job-cores` for `pbs_python` job backend (#1547)
  - skip dependencies marked as external modules when packaging (#1550)
  - fix syntax for `set_alias` statement in Lua syntax (#1554)
  - handle the case of all 'offline' nodes correctly for `--job` (#1560)
  - fix `test_modules_tool_stateless` unit test for stateless ModulesTool with Lmod as modules tool (#1570)

easyblocks

- add generic easyblock for Cray toolchains (#766)
- new easyblocks for 2 software packages that require customized support:
  - EggLib (#811), PGI (#658)

- various other enhancements, including:
  - update BamTools easyblock for versions 2.3.x and newer: some shared libraries are now static) (#785)
  - don't hardcode `.so`, use `get_shared_lib_ext` instead (#789, #790, #791, #793, #794, #803, #815)
  - enhance CPLEX easyblock by adding more subdirs to `$PATH`, define `$LD_LIBRARY` and `$CPLEXDIR` (#797)
  - make sanity check for netcdf4-python work with both egg and non-egg installs (#799)
  - update sanity check in PETSc/SLEPc easyblocks for v3.6.x (#800)
  - update Trinity easyblock for 2.x versions (#802)
  - update DOLFIN easyblock for v1.6.0 (#804)
  - check for `libkokkoscore.a` rather than `libkokkos.a` for Trilinos 12.x (#805)
  - add an option to skip the sanitizer tests of Clang (#806)
  - update Molpro easyblock to support binary installs and 2015 version (#807)
  - make `ConfigureMake` more robust w.r.t. custom easyconfig parameters (#810)
- various bug fixes, including:
  - add back support for Eigen 2.x in Eigen easyblock (#798)
  - fix for vsc-base being picked up from OS in EasyBuildMeta easyblock (#813)
  - remove `setuptools.pth` if it includes absolute paths after installing EasyBuild (#813)

easyconfigs

- add easyconfigs for `foss/2016a` and `intel/2016` common toolchains (#2310, #2311, #2339, #2363)
  - incl. easyconfigs for Boost, CMake, Python, Perl using these toolchains
- added example easyconfig files for 21 new software packages:
  - BLASR (#922), BioKanga (#2247), BoltzTraP (#2365), basemap (#2221), CppUnit (#2271), EggLib (#2335), FLASH (#2281), GLM (#2288), hub (#2249), MACS2 (#1983), MotEvo (#843), numba (#2243), PGI (#1833, #2367), PLY (#2305), PaStiX (#2319, #2326), patchelf (#2327), pip (#2284), RSEM (#2316), RcppArmadillo (#2289), SCDE (#2289), slepc4py (#2318)
- added additional easyconfigs for various supported software packages: version updates, different toolchains, ...
  - including BamTools 2.4.0, Boost 1.60.0, Clang 3.7.1, DOLFIN/FFC/FIAT/Instant/UFL 1.6.0, GATE 7.0, GCC 5.3.0, LLVM 3.7.1, pandas 0.17.1, PETSc 3.6.3, SAMtools 1.3, scipy 0.16.1, SLEPc 3.6.2, Trilinos 12.4.2, Trinity 2.1.1, VTK 6.3.0
- various other enhancements, including:
  - added new `Cray*` toolchain versions with pinned dependency versions (#2222)
  - don't hardcode `.so`, use `SHLIB_EXT` constant instead (#2245)
  - add custom sanity check in GEOS easyconfigs (#2285)
- various bug fixes, including:
  - add Autotools (M4) as a build dependency in GMP v6.x easyconfigs (#2096)
  - remove argparse from list of extensions in Python 3.2+ easyconfigs, since it became part of stdlib (#2323)
- various style fixes, including:
  - get rid of tabs (#2302)
  - remove trailing whitespace (#2341)

8.12.6 v2.5.0 (December 17th 2015)

feature + bugfix release

framework

- add support for IBM XL compilers on Power7 and PowerPC (BlueGene) (#1470)
- add support fo generic compilation using `--optarch=GENERIC` (#1471)
  - see also *Controlling compiler optimization flags*
- update experimental support for `.yeb` easyconfigs (#1515)
  - support clean way to specify toolchain + dependencies in `.yeb` easyconfigs
- various other enhancements, including:
  - add support for 'whatis' easyconfig parameter (#1271)
  - add support for SLES 12 and kernel 3.12.x (#1412)
  - add GCCcore toolchain definition (#1451)
  - use 'diff --git' lines to determine patched files in pull request with `--from-pr` (#1460)
  - add proper option parser to bootstrap script (#1468)
  - add `get_gcc_version()` function in systemtools module (#1496)
  - don't load fake module in sanity_check_step during a dry run (#1499)
  - allow string values to be passed in `make_module_req` by hoisting them into a list (#1502)
  - add support for listing build dependencies as hidden dependencies (#1503)
  - also consider `lib32/pkgconfig` and `lib64/pkgconfig` for `$PKG_CONFIG_PATH` (#1505)
  - add support to `make_module_dep` to specify module to unload before loading a dependency module (#1506)
  - add support to `make_module_extra` to specify alternative root/version for `$EBROOT/$EBVERSION` (#1508)
  - packaging support is no longer considered experimental (#1510)
- various bug fixes, including:
  - also consider `lib64` in sanity check performed during EasyBuild bootstrap (#1464)
  - also add description/homepage to packages created with FPM (#1469)
  - fix develop setup script to install EasyBuild-develop module in subdirectory (#1480)
  - don't create a whole set of temporary 'minimal-easyconfigs' subdirs with `--minimal-toolchains` (#1484)
  - only keep polling if exit code is `None` in `run_cmd_qa`, to correctly deal with negative exit codes (#1486)
  - fix bootstrap script for missing sys_platform by using newer distribute 0.6.49 in stage 0 (#1490)
  - make sure that extra custom easyconfig parameters are known for extensions (#1498)
  - add missing import for `EasyBuildError` in `easybuild/toolchains/linalg/libsci.py` (#1512)
  - isolate tests from possible system-wide configuration files (#1513)
  - only use `glob` in `make_module_req` on non-empty strings (#1519) * this fixes the problem where `$CUDA_HOME` and `$CUDA_PATH` are not defined in module files for CUDA

easyblocks

- update easyblocks for Intel tools to support 2016 versions (#691, #745, #756, #777)

  - IntelBase easyblock has been enhanced to support specifying which components to install
- new easyblocks for 3 software packages that require customized support:
  - Intel Advisor (#767), DIRAC (#778), MRtrix (#772)
- various other enhancements, including:
  - update numpy and SuiteSparse easyblock to use scikit-umfpack (#718)
  - add an option to allow removal of the `-Dusethreads` flag in Perl easyblock (#724)
  - update Doxygen easyblock for 1.10.x (CMake) (#734)
  - update sanity check in Qt easyblock for Qt 5.x (#740)
  - add support for multilib build of GCC on PowerPC (#741)
  - add support to OpenFOAM and SCOTCH easyblocks to support 64-bit integers, via 'i8' toolchain option (#744)
  - fix sanity check to support numpy 1.10 (dropped _dotblas.so) (#757, #761, #762)
  - update IPP easyblock for v9.x (#759)
  - cleaner output for PythonPackage under dry run, make numpy easyblock dry-run aware (#760, #671)
  - add support for using netCDF-Fortran as dependency in ALADIN easyblock (#764)
  - add support for tbb 4.4.x in tbb easyblock (#769)
  - add support for specifying altroot/altversion in Bundle easyblock (#773)
  - update OpenFOAM easyblock for OpenFOAM-Extend 3.2 + use apply_regex_substitutions (#770)
- various bug fixes, including:
  - fix module path extension of system compiler in HMNS setup (#742)
  - only restore `$PYTHONPATH` if it was defined in EasyBuildMeta easyblock (#743)
  - make sure `$PYTHONPATH` is defined correctly in module file for Python packages created with `--module-only` (#748)
  - fix WRF easyblock to produce correct module under `--module-only --force` (#746, #752)
  - don't hardcode '`openPBS`' in GATE easyblock, use value for `default_platform` easyconfig parameter (#753)
  - avoid adding lib subdirs to `$*LIBRARY_PATH` if no libraries are there in `PythonPackage` easyblock (#755)
  - fix installing Python bindings for libxml2 to correct installation prefix (#765)

easyconfigs

- add GCCcore easyconfig that can be used as base for all compilers (without getting in the way) (#2214)
  - along with easyconfig for `GCC/4.9.3-2.25`: bundle of GCCcore 4.9.3 and binutils 2.25
  - intended to replace the GNU toolchain
- added example easyconfig files for 39 new software packages:
  - DIRAC (#2212), GeoIP (#2172, #2185), GeoIP-C (#2172, #2185), graph-tool (#1591), gtkglext (#2217), Intel Advisor (#2210), InterProScan (#2225, #2227, #2234), intltool (#2136), kallisto (#2173), LibUUID (#1930), LuaJIT (#2153), libXcursor (#2136), libXrandr (#2136), libXtst (#2143), libdap (#1930), libtasn1 (#2208), libxkbcommon (#2136), MRtrix (#2217, #2218), MultiNest (#2166, #2168), Nipype (#2150), PPfold (#2183, #2187), p11-kit (#2208), pangox-compat (#2217), Qt5 (#2136), randrproto (#2136), rhdf5 (#2175), Stampy (#2180, #2182), scikit-umfpack (#2061), scp (Python pkg) (#2196), sleuth (#2175), traits (#2150), vincent (#2169, #2185), XKeyboardConfig (#2136), xcb-util (#2136), xcb-util-image (#2136), xcb-util-keysyms (#2136), xcb-util-renderutil (#2136), xcb-util-wm (#2136), zlibbioc (#2175)

- **added new easyconfigs for existing toolchains:** `intel/2015.08` (#2194), `intel/2016.00` (#2209), `intel/2016.01` (#2219), `iomkl/2015.03` (#2155)
- added additional easyconfigs for various supported software packages: version updates, different toolchains, ...
  - including CMake 3.4.1, HDF5 1.8.16, netCDF 4.3.3.1, netCDF-Fortran 4.4.2, numpy 1.10.1, Octave 4.0.0, OpenFOAM 3.0.0, OpenFOAM-Extend 3.2, Python 2.7.11
- various other enhancements, including:
  - add tidyr to R 3.2.1 easyconfigs (#2174)
  - enable C++ support in MIGRATE-N (#2178)
  - also installed shared libraries for AMD and UMFPACK in SuiteSparse (#2061)
  - fix software name for ParaView (was: Paraview) (#2132)
  - enable building of shared libraries for binutils (#2133)
  - harden binutils built with dummy toolchain by linking to system libraries via `RPATH` (#2228)
  - enhance easyconfig unit tests to check that each easyconfig file is in the right subdirectory (#2232)
- various bug fixes, including:
  - fix ALADIN patch file to not use relative paths, and adjust list of ALADIN sources accordingly ((#2207), (#2213))
  - rename patch files for OpenFOAM to be in line with other patches (#2226)
  - fix typo in bzip2 source URLs (#2204)
  - force linking of ncurses in libreadline (#2206)
  - enable `-fPIC` in all zlib 1.2.8 easyconfigs (#2220)
  - move Net-LibIDN/SRA-Toolkit/bbftpPRO/o2scl easyconfigs to right location (#2232)
  - restrict parallel build in OpenFOAM-Extend easyconfigs via '`maxparallel`', not '`parallel`' (#2233)

8.12.7 v2.4.0 (November 10th 2015)

feature + bugfix release

framework

- add support for `--extended-dry-run`/`-x` (#1388, #1450, #1453, #1455)
  - detailed documentation is available at *Extended dry run*
- fix checking of sanity check paths w.r.t. discriminating between files and directories (#1436)
  - this impacts several easyconfig files where `sanity_check_paths` was not 100% correct
- make '`eb`' script aware of Python v3.x, fall back to using `python2` if required (#1411)
- add experimental support for parsing .yeb easyconfig files in YAML syntax (#1447, #1448, #1449)
  - see also easyconfig_yeb_format
- add experimental support for resolving dependencies with minimal toolchains (#1306)
  - see also *Using minimal toolchains for dependencies*
- various other enhancements, including:
  - refactor `extract_cmd` function to get rid of if/elif/else spaghetti blob (#1382)
  - add support for `--review-pr` (#1383)

  - add `apply_regex_substitutions` function to perform runtime patching from easyblocks (#1388, #1458)
  - add support for specifying alternate name to be part of generated module name (#1389) * via '`modaltsoftname`' easyconfig parameter
  - support overriding # used cores via `--parallel` (#1393)
  - also define `$FC` and `$FCFLAGS` in build environment (#1394)
  - add support extracting for `.tar.Z` files (#1396)
  - include easybuild/scripts in instalation (#1397)
  - ignore hidden directories in find_base_dir (#1413, #1415)
  - add `only_if_module_is_available` decorator function to guard functionality that uses optional dependencies (#1416)
  - give easyblocks the possibility to choose `maxhits` for `run_cmd_qa` (#1417)
  - use class name (string) rather than License instances as values for software license constants (#1418)
  - support controlling recursive unloading of dependencies via '`recursive_module_unload`' easyconfig parameter (#1425)
  - implement basic support for type checking of easyconfig parameters (#1427)
  - support auto-converting to expected value type for easyconfig parameters (enabled by default) (#1428, #1437)
  - add support for `--rebuild` command line option, alternative for `--force` which doesn't imply `--ignore-osdeps` (#1435)
  - add support for Mercurial easyconfig repository (#979, #1446)
  - add dedicated class for psmpi toolchain MPI component, and use it in gpsmpi and ipsmpi toolchains (#1454)
- various bug fixes, including:
  - fix extracting of comments from an easyconfig file that includes 'tail' comments (#1381)
  - fix dev version to follow PEP-440, as required by recent setuptools versions (#1403)
    - required to avoid that setuptools transforms the version itself
    - see also https://www.python.org/dev/peps/pep-0440/#developmental-releases
  - allow `get_cpu_speed` to return `None` if CPU freq could not be determined (#1421)
  - relax `sanity_check_paths` in EasyBuild bootstrap script to deal with possible zipped .egg (#1422)
  - use empty list as default value for src/patches in Extension class (#1434)
  - skip symlinked files in `adjust_permissions` function (#1439)
  - fix HierarchicalMNS to always use full version number (#1440)

easyblocks

- 3 new generic easyblocks: OCamlPackage (#467), SCons (#689, #700), Waf (#722)
- new easyblocks for 2 software packages that require customized support:
  - OCaml (#467), Samcef (#678)
- various other enhancements, including:
  - add support for installing OpenFOAM with external METIS, CGAL and Paraview (#497)
  - update netCDF easyblock updated for netCDF v4.3.3.1 (#674)
  - update Rosetta easyblock for recent Rosetta versions (#677)

  - make unpacked source dir detection in easyblock for VSC-tools a little bit more flexible (#679)
  - add support for building with Plumed support enabled in CP2K easyblock (#681)
  - update Go easyblock for Go v1.5 (#683)
  - use `apply_regex_substitutions` function in WRF easyblock (#685)
  - update MUMPS easyblock for 5.x (#686)
  - implement runtime patching of `$WM_*` and compiler variables for OpenFOAM (#688)
  - specify sequential compiler to use in compiler command that gets injected in OpenFOAM easyblock (#692)
  - make `PythonPackage` and WRF easyblocks dry-run aware (#696)
    - see also *Guidelines for easyblocks*
  - add support in `PythonPackage` for installing with `easy_install` + installing zipped eggs (#698, #711, #715)
  - update Bowtie easyblock for recent Bowtie versions (#707)
  - update CUDA easyblock for CUDA 7.x(#708)
  - also consider `config/make.sys.in` for want in QuantumESRESSO easyblock (#714)
  - define `$NWCHEM_LONG_PATH` if needed in NWChem easyblock (#720)
  - remove custom post-install step in PDT easyblock (#723)
    - no longer needed now that `adjust_permissions` functions ignores symlinks
  - use `$LIBS` in HPL easyblock (#727, #736)
- various bug fixes, including:
  - also define `$MCRROOT` for MCR in module (#687)
  - add missing ‘`super`‘ call in `configure_step` of easyblock for python-meep (#694)
  - only prepend existing non-empty paths to `$PYTHONPATH` in `PythonPackage` easyblock (#697)
  - fix extra_options definition in `CMakePythonPackage` easyblock (#698)
  - fix dev version to follow PEP-440, as required by recent setuptools versions (#702, #703, #704)
    - required to avoid that setuptools transforms the version itself
    - see also https://www.python.org/dev/peps/pep-0440/#developmental-releases
  - consider both `lib` and `lib64` in sanity check paths for flex (#705)
  - also copy signature file and don’t copy CMake files in Eigen easyblock (#709)
  - fix directory names in `make_module_req_guess` of ANSYS easyblock (#713)
  - fix imports for `set_tmpdir` in easyblock unit tests after function was moved in EasyBuild framework (#726)
  - use `--with-tcltk*` configure options for Python to point to ensure Tcl/Tk deps are picked up (#729)
  - fix order of subdirs for QuantumESPRESSO binaries (#730)
  - correctly handle having both `$FC`/`$FCFLAGS` and `$F90`/`$F90FLAGS` defined when building MVAPICH2 (#732)
  - fix OpenSSL sanity check paths: lib/engines is a directory (#731, #733)
  - fix sanity check paths for netcdf-python (#735)

easyconfigs

- added example easyconfig files for 45 new software packages:

  - animation (#2007), ANSYS CFD (#1969), ANTLR (#1191, #1980), APR (#1970), APR-util (#1970), Aspera Connect (#2005), ChIP-Seq (#2119), deap (#2082), DISCOVARdenovo (#1932), FastQC (#1984), fontsproto (#1618, #2038), GraphicsMagick (#2007), HBase (#1990), ISIS (#1972), libedit (#293), libfontenc (#1618, #2038), libGLU (#1627), libXdamage (#1618, #2038), libXfont (#1618, #2038), LLVM (#1620, #1989, #2031), MIGRATE-N (#1944), MIRA (#1938), mympingpong (#2049), MySQLdb (#2011), NCO (#1191, #1980), NIPY (#2064), Nilearn (#2064), NiBabel (#2064), PBZIP2 (#1038), PIL (#2062), PhyloCSF (#2018), pycairo (#2085), pydicom (#2063), Salmon (#2051), Samcef (#1941), scikit-image (#1974, #2006), Serf (#1970), SSAHA2 (#1039), Subversion (#1970), SWASH (#2059), time (#1954), Trim_Galore (#1984), Trimmomatic (#1987), WEKA (#1986), x264 (#2017)

- added new easyconfigs for existing toolchains: `gimkl/2.11.5` (#2093)
- added additional easyconfigs for various supported software packages: version updates, different toolchains, ...
  - including Clang + LLVM 3.7.0, CMake 3.3.2, CUDA 7.5.18, hanythingondemand v3.0.1, Mesa 11.0.2, mpi4py v2.0.0, ncurses 6.0, OpenFOAM 2.4.0, Paraview 4.4.0, Python 3.5.0, QuantumESPRESSO v5.2.1
- various other enhancements, including:
  - enable '`pic`' toolchain option in libxml2 easyconfigs (#1993)
  - extend list of R libraries included in R v3.2.1 easyconfigs (#2042, #2046, #2067, #2072)
  - add Rsubread in Bioconductor easyconfigs (#1971)
- various bug fixes, including:
  - fix software name for `BEEF` (was '`libbeef`') (#1679)
  - add patch to install `qhull.pc` (pkgconfig) file with Qhull (#1975)
  - don't enable experimental nouveau API in libdrm easyconfigs (#1994)
  - fix dev version to follow PEP-440, as required by recent setuptools versions (#1997)
    - required to avoid that setuptools transforms the version itself
    - see also https://www.python.org/dev/peps/pep-0440/#developmental-releases
  - correct homepage in Cufflinks easyconfigs (#2060)
  - fix imports for `set_tmpdir` in easyblock unit tests after function was moved in EasyBuild framework (#2097)
  - add patch for Tk 8.6.4 to fix problem with `tk.tcl` not being found (#2102)
  - don't use `%(version)s` template in toolchain version, causes problems with HierarchicalMNS (#2104)
  - fix sanity check paths in several easyconfig (#2109, #2120, #2121, #2125)
    - required because of bug fix in `sanity_check_step` implementation
    - CVXOPT, h5py, LIBSVM, libunistring, MDP, monty, PhyloCSF, Pyke, pandas, pycosat, pyhull, pymatgen, python-dateutils, Seaborn, Theano, XML-LibXML, XML-Simple

8.12.8 v2.3.0 (September 2nd 2015)

feature + bugfix release

framework

- requires vsc-base v2.2.4 or more recent (#1343)
  - required for `mk_rst_table` function in `vsc.utils.docs`
- various other enhancements, including:

  - add support for generating documentation for (generic) easyblocks in `.rst` format (#1317)
  - preserve comments in easyconfig file in `EasyConfig.dump()` method (#1327)
  - add `--cleanup-tmpdir` option (#1365)
    * enables to preserve the used temporary directory via `--disable-cleanup-tmpdir`
  - enhance `EasyConfig.dump()` to reformat dumped easyconfig according to style guidelines (#1345)
  - add support for extracting `.iso` files using 7z (p7zip) (#1375)
- various bug fixes, including:
  - correctly deal with special characters in template strings in `EasyConfig.dump()` method (#1323)
  - rework `easybuild.tools.module_generator` module to avoid keeping state w.r.t. fake modules (#1348)
  - fix dumping of hidden deps (#1354)
  - fix use of `--job` with hidden dependencies: include `--hidden` in submitted job script when needed (#1356)
  - fix `ActiveMNS.det_full_module_name()` for external modules (#1360)
  - fix `EasyConfig.all_dependencies` definition, fix tracking of job dependencies (#1359, #1361)
  - fix `ModulesTool.exist()` for hidden Lua module files (#1364)
  - only call `EasyBlock.sanity_check_step` for non-extensions (#1366)
    * this results in significant speedup when installing easyconfigs with lots of extensions, but also results in checking the default sanity check paths if none were defined for extensions installed as a module
  - fix using module naming schemes that were included via `--include-module-naming-schemes` (#1370)

easyblocks

- new easyblocks for 2 software packages that require customized support:
  - MCR (#623), Molpro (#665)
- various other enhancements, including:
  - enhance BWA easyblock to also install man pages (#650)
  - enhance tbb easyblock to consider lib dirs in order and also define `$CPATH`, `$LIBRARY_PATH`, `$TBBROOT` (#653, #654)
  - call `PythonPackage.configure_step` in `ConfigureMakePythonPackage.configure_step` (#668)
  - add '`foldx3b6`' as possible binary name in FoldX easyblock (#671)
  - enhance/cleanup MATLAB easyblock (#672)
  - move preparing of '`intel`' subdir in `$HOME` to `configure_step` in `IntelBase` easyblock (#673)
- various bug fixes, including:
  - add missing super call in `post_install_step` of imkl easyblock (#648, #660)
  - fix regex used to correct `I_MPI_ROOT` in impi `mpivars.sh` scripts (#662)
  - fix regex used to patch `.mk` file in configure step of SuiteSparse easyblock (#666)
  - correctly specify installation prefix via `$GEM_HOME` in `RubyGem` easyblock (#667)

  - add custom sanity check in scipy easyblock (#669)
  - specify to always use the bfd linker for OpenFOAM, to stay away from using `ld.gold` (#670)

easyconfigs

- added example easyconfig files for 19 new software packages:
  - ATK (#1780), Atkmm (#1780), cairomm (#1780), GLibmm (#1780), GlobalArrays (#1868), gdk-pixbuf (#1780), gtk+ (#1780), Gtkmm (#1780), libbeef (#1827), libsigc++ (#1780), libsodium (#1876), MACS (#1869), MCR (#1677), Molpro (#1880), NFFT (#1921), p7zip (#1931), Pangomm (#1780), pygraphviz (#1861), pycosat (#1859)
- added new easyconfigs for existing toolchains: GNU/4.9.3-2.25 (#1836), foss/2015b (#1695), intel/2015b (#1696)
  - add easyconfigs using this toolchain for BLAST+ 2.2.31, Boost 1.58.0, CP2K 2.6.1, OpenFOAM 2.3.1, Perl 5.20.2 + 5.22.0 (bare), Python 2.7.10, R 3.2.1
- added additional easyconfigs for various supported software packages: version updates, different toolchains, ...
  - including Boost 1.59.0, CP2K 2.6.1, GCC 5.2.0
- various other enhancements, including:
  - enhance texinfo easyconfig w.r.t. `texmf`, only use it as a build dependency (#1840)
  - enable building of `ld.gold` in binutils 2.25 (#1885)
- various bug fixes, including:
  - fix enabling MPI support for h5py 2.5.0 (#1825)
  - fix versions of Bioconductor packages + add a couple extra (#1828, #1852, #1895, #1917)
  - put dummy values in place for `builddir`/`installdir` templates in easyconfigs unit tests (#1835)
  - fix easyconfigs unit tests w.r.t. changes made in framework (#1853, #1870, #1874, #1875)
  - add GMP as missing dep in Python 2.7.10 easyconfigs, required for pycrypto extension (#1858)
  - specify installation prefix for SIP (#1888, #1892)
  - add custom sanity check paths in various easyconfigs (#1889, #1894, #1897 - #1909)
    * required because of fix in EasyBuild framework, causing default sanity check paths to be considered for extensions that are installed as a module
    * affected easyconfigs include: AnalyzeFMRI, Biggus, bibtexparser, DB_File, DBD-Pg, DBD-SQLite, DBD-mysql, evmix, fmri, FPM, GraphViz, gsl, GSSAPI, MDP, mpi4py, ncdf, ncdf4, netifaces, NetLibIDN, networkx, ordereddict, Parallel-ForkManager, paycheck, PyQuante, Pyke, PyQt, r2py, rjags, runjags, scikit-learn, SOBAcl, vsc-processcontrol, vsc-mympirun-scoop, XML, XML-Dumper, XML-Parser, XML-Twig, YAML-Syck
  - don't enable '`static`' toolchain option in SuiteSparse 4.4.3 easyconfig (#1911)
  - add –exclude unpack options for OpenFOAM 2.3.1 to avoid cyclic symlink causing problems when unpacking (#1925)

8.12.9 v2.2.0 (July 15th 2015)

feature + bugfix release

framework

- add support for using GC3Pie as a backend for `--job` (#1008)
  - see also *Submitting jobs using –job*

- add support for `--include-*` configuration options to include additional easyblocks, toolchains, etc. (#1301)
  - see *Including additional Python modules (–include-*)*
- add (experimental) support for packaging installed software using FPM (#1224)
  - see *Packaging support*
- various other enhancements, including:
  - use https for PyPI URL templates (#1286)
  - add GNU toolchain definition (#1287)
  - make bootstrap script more robust (#1289, #1325):
    - exclude 'easyblocks' pkg from `sys.path` to avoid that `setup.py` for easybuild-easyblocks picks up wrong version
    - undefine `$EASYBUILD_BOOTSTRAP*` environment variables, since they do not correspond with known config options
  - improve error reporting/robustness in `fix_broken_easyconfigs.py` script (#1290)
  - reset keep toolchain component class 'constants' every time (#1294)
  - make `--strict` also a build option (#1295)
  - fix purging of loaded modules in unit tests' setup method (#1297)
  - promote `MigrateFromEBToHMNS` to a 'production' MNS (#1302)
  - add support for `--read-only-installdir` and `--group-writable-installdir` configuration options (#1304)
  - add support for *not* expanding relative paths in `prepend_paths` (#1310)
  - enhance `EasyConfig.dump()` method to use easyconfig templates where possible (#1314), #1319), #1320), #1321)
- various bug fixes, including:
  - fix issue with cleaning up (no) logfile if `--logtostdout`/`-l` is used (#1298)
  - stop making `ModulesTool` class a singleton since it causes problems when multilple toolchains are in play (#1299)
  - don't modify values of 'paths' list passed as argument to prepend_paths in `ModuleGenerator` (#1300)
  - fix issue with `EasyConfig.dump()` + cleanup (#1308), #1311)
  - reenable (and fix) accidentally disabled test (#1316)

easyblocks

- modified easybuild.easyblocks package declaration to support giving preference to custom easyblocks (#617)
- 2 new generic easyblocks: RubyGem (#565), SystemCompiler (#633)
- new easyblocks for 5 software packages that require customized support:
  - NEMO (#564), pbdMPI (#612), #620), pbdSLAP (#620), PDT (#624), Ruby (#565)
- various other enhancements, including:
  - update CUDA easyblock for v6.x (#476)
  - make METIS easyblock take into account configopts (#494)
  - enable building of EOMIP and EOMEA for NWChem versions 6.5 and up (#508)
  - make out-of-source build with CMake truly out-of-source (#615)

  - add support in Bundle easyblock to run full sanity check (#627)
  - also take platform-specific Python lib dirs into account in PythonPackage easyblock (#628)
  - fix mpivars scripts in Intel MPI installation for versions where the installation is moved (#629)
  - don't restrict `det_pylibdir` function to only EasyBuild-provided Python (#631), #641)
  - support snappy and other optional native libs in Hadoop easyblock (#632), #638), #640), #642)
- various bug fixes, including:
  - fix Xmipp easyblock, use provided `install.sh` script (#630)
  - update Clang easyblock to disable tests that may fail when unlimited stack size is used (#622)
  - fix setting of `$INTEL_LICENSE_FILE` for port@server values (#635)

easyconfigs

- added example easyconfig files for **62** new software packages:
  - ADF (#899), AutoDock_Vina (#808), bibtexparser (#1726), Biggus (#1770), Bismark (#990), blasr (#1662), BSMAP (#1171), Check (#811), Circuitscape (#1222), CONTRAfold (#689), cramtools (#1741), DBD-Pg (#1066), DendroPy (#995), EMAN2 (#1737), ETSF_IO (#727), eudev (#1578), fastqc (#1636), FDS (#814), #1617), #1625), FPM (#1440), frealign (#1619), g2log (#1035), GC3Pie (#1692), #1756), #1768), GenotypeHarmonizer (#1672), gensim (#1762), GraphViz (#1658), hisat (#1674), IDBA-UD (#1045), IMa2 (#828), IMPUTE2 (#824), JUBE (#1396), LAMARC (#760), libXScrnSaver (#1653), MATIO (#1004), MuTect (#1483), ncdf (#617), NEMO (#1640), ngspice (#1116), ordereddict (#1774), OSU Micro-Benchmarks (#1777), Parallel-ForkManager (#847), pBWA (#1009), PeakSeq (#1412), Pillow (#1702), Pindel (#1126), PLUMED (#1596), #1665), PostgreSQL (#1066), PROJ (#1006), PyAMG (#1222), Pyke (#1776), rpy2 (#1775), Sailfish (#1035), SCANMS (#1386), Seaborn (#1763), snpEff (#1680), SOBAcl (#1658), SPIDER (#1624), #1723), STAR (#1043), #1676), system GCC (#1778), tabix (#1059), tecplot360ex (#1100), Vampir (#512), VampirServer (#512), verifyBamID (#1675)
- added easyconfigs for 4 new software bundles:
  - R-bundle-Bioconductor (#1573), #1795), R-bundle-devtools (#1621), #1759), R-bundle-extra (#1387), #1759), R-bundle-pbd (#1659)
- added easyconfigs for new GNU toolchain (#1346), #1669)
- added new easyconfigs for existing toolchains: goolf/1.5.16, intel/2014.06
- added additional easyconfigs for various supported software packages: version updates, different toolchains, ...
  - including BLAST 2.2.31+, Clang 3.6.1, CUDA 6.x, GCC 4.9.3, GROMACS 5.0.5, HDF5 1.8.15 + 1.8.15-patch1, Python 2.7.10, R 3.2.0 + 3.2.1, WRF 3.6.1
- various other enhancements, including:
  - update all ncurses easyconfigs to enable ncursesw and use ConfigureMake easyblock (#1337)
  - update PDT easyconfigs to use PDT easyblock (#1687)
  - add custom `sanity_check_paths` in libxml2 easyconfigs (#1690)
  - enhance unit tests to also cover EasyConfig.dump() method on all easyconfigs (#1761)
  - include snappy as dependency in Hadoop easyconfigs (#1758), #1773)
  - enable SSL support in CMake v3.2.3 easyconfigs (#1784)
  - add additional extensions in R easyconfigs (#1637)
- various bug fixes, including:
  - add patch file required for correct CUDA-aware OpenMPI v1.7.3 build (#631)
  - fix issue with OpenMPI dependency in ECore easyconfigs (#777)

- don't run the Bloom tests for Jellyfish, they can randomly fail (#1016)
- fix source URLs in BioPerl easyconfigs (#1075)
- patch out svnversion command in Python 2.5.6 to fix build on recent systems (#1576)
- consistently use https for PyPI URLs in homepage/source_urls (#1616), #1722)
- include Tcl and Tk as dependencies in R easyconfig (#1623)
- add patch for installing paycheck as Py3 extension (#1629)
- add Perl dependency in git 2.x easyconfigs (#1631)
- fix easyconfig for GMP 6.0.0, don't use 6.0.0a sources (#1635)
- fix source_urls in QuantumESPRESSO 5.0.2 easyconfigs (#1652)
- include Tk as dependency in Python 2.7.9 easyconfigs (#1654)
- include tk-devel is list of OS deps for Python 2.7.9 Cray easyconfigs, make sure '`import Tkinter`' works (#1655)
- drop gpfs versionsuffix and stop using no longer existing `--enable-gpfs` configopt for recent HDF5 versions (#1657)
- include missing libxml2 dep in GLib easyconfigs (#1666)
- fix source URLs in Qt easyconfigs (#1673)
- fix source URLs for argparse Python extension (#1697)
- fix source URLs for deap Python extension (#1699)
- fix easyconfigs unit tests after making ModulesTool a non-singleton class (#1708)
- fix names for Xmipp easyconfigs and patches (#1719)
- add patch for Qt 4.8.6 to fix build issue on RHEL6 with intel/2015a (#1734)
- add M4 as build dep for GCC 5.1.0 (#1735)
- fix Bioconductor extension versions in R 3.1.3 easyconfigs (#1748)
- remove custom exts_filter definition from Python 3.4.3 easyconfig (#1765)
- fix source_urls in netCDF easyconfigs (#1766)
- fix source_urls in netCDF-C++ and netCDF-Fortran easyconfigs (#1767)

8.12.10 v2.1.1 (May 18th 2015)

bugfix release

framework

- fix issue with missing load statements when `--module-only` is used, don't skip ready/prepare steps (#1276)
- enhance `--search`: only consider actual filename (not entire path), use regex syntax (#1281)
- various other bug fixes, including:
  - fix `generate_software_list.py` script w.r.t. dependencies marked as external modules (#1273)
  - only use `$LMOD_CMD` value if `lmod` binary can't be found in $PATH (#1275)
  - fix location of `module_only` build option w.r.t. default value (#1277)
  - fix combined use of `--hide-deps` and `hiddendependencies` (#1280)
  - remove log handlers that were added during tests, to ensure effective cleanup of log files (#1282)

    * this makes the unit test suite run ~3x faster!
  - define `$CRAYPE_LINK_TYPE` if '`dynamic`' toolchain option is enabled for Cray compiler wrappers (#1283)

easyblocks

- fix compatibility of easyblocks with `--module-only` + dedicated unit test module (#610)
- minor enhancements, including:
  - update GROMACS easyblock for version 5 (#513)
- various other bug fixes, including:
  - only check compiler being used if FFTW interfaces are being built in Intel MKL easyblock (#606)

easyconfigs

- added example easyconfig files for **3** new software packages:
  - networkx (#1592), Platanus (#1597), SaguaroGW (#1600)
- added new easyconfigs for existing toolchains: `ictce/7.3.5`, `CrayCCE/5.2.40`, `CrayGNU/5.2.40`, `CrayIntel/5.2.40`
- added easyconfigs using `CrayGNU/5.2.25` and `CrayGNU/5.2.40` toolchains (#1610, #1611)
- added additional easyconfigs for various supported software packages: version updates, different toolchains, ...
  - including Boost 1.58.0, GROMACS 5.0.4, Python 3.4.3
- various bug fixes, including:
  - enable usempi in GROMACS easyconfig using CrayGNU toolchain (as required by GROMACS easyblock) (#1590)
  - use system-provided tcsh when building WRF on Cray systems, to avoid hanging build (#1595)
  - only use '`dynamic`' toolchain option, not '`shared`', in easyconfigs using Cray toolchain (#1609)

8.12.11 v2.1.0 (April 30th 2015)

feature + bugfix release

framework

- requires vsc-base v2.2.0 or more recent
  - added support for LoggedException (vsc-base#160, vsc-base#163, vsc-base#165, vsc-base#167)
  - added support for add_flex action in GeneralOption (vsc-base#162)
  - added support to GeneralOption to act on unknown configuration environment variables (vsc-base#162)
- add support for only (re)generating module files: `--module-only` (#1018)
  - module naming scheme API is enhanced to include det_install_subdir method
  - see *Only (re)generating (additional) module files using –module-only*
- add support for generating module files in Lua syntax (note: requires Lmod as modules tool) (#1060, #1255, #1256, #1270)
  - see `--module-syntax` configuration option and *Module files syntax (–module-syntax)*
- deprecate `log.error` method in favor of raising `EasyBuildError` exception (#1218)
  - see *Report errors by raising EasyBuildError rather than using log methods*

- add support for using external modules as dependencies, and to provide metadata for external modules (#1230, #1265, #1267)
  - see *Using external modules*
- add experimental support for Cray toolchains on top of PrgEnv modules: `CrayGNU`, `CrayIntel`, `CrayCCE` (#1234, #1268)
  - see https://github.com/hpcugent/easybuild/wiki/EasyBuild-on-Cray
- various other enhancements, including:
  - clear list of checksums when using `--try-software-version` (#1169)
  - sort the results of searching for files (e.g., `--search` output) (#1214)
  - enhance test w.r.t. use of templates in cfgfile (#1217)
  - define '`%(DEFAULT_REPOSITORYPATH)s`' template for cfgfiles (see `eb --avail-cfgfile-constants`) (#1220)
  - also reset `$LD_PRELOAD` when running module commands, in case module defined `$LD_PRELOAD` (#1222)
  - move location of '`module use`' statements in generated module file (*after* '`module load`' statements) (#1232)
  - add support for `--show-default-configfiles` (#1240)
    - see *Default configuration files*
  - report error on missing configuration files, rather than ignoring them (#1240)
  - clean up commit message used in easyconfig git repository (#1248)
  - add `--hide-deps` configuration option to specify names of software that must be installed as hidden modules (#1250)
    - see *Installing dependencies as hidden modules using –hide-deps*
  - add support for appending/prepending to `--robot-paths` to avoid overwriting default robot search path (#1252)
    - see *Prepending and/or appending to the default robot search path*
  - enable detection of use of unknown `$EASYBUILD`-prefixed environment variables (#1253)
    - see *Environment variables*
  - add `--installpath-modules` and `--installpath-software` configuration options (#1258)
    - see *Software and modules install path (–installpath, –installpath-software, –installpath-modules)*
  - use dedicated subdirectory in temporary directory for each test to ensure better cleanup (#1260)
  - get rid of `$PROFILEREAD` hack when running commands, not needed anymore (#1264)
- various bug fixes, including:
  - make bootstrap script robust against having `vsc-base` already available in Python search path (#1212, #1215)
  - set default value for unpack_options easyconfig parameter to `''`, so `self.cfg.update` works on it (#1229)
  - also copy rotated log files (#1238)
  - fix parsing of `--download-timeout` value (#1242)
  - make `test_XDG_CONFIG_env_vars` unit test robust against existing user config file in default location (#1259)
  - fix minor robustness issues w.r.t. `$XDG_CONFIG*` and `$PYTHONPATH` in unit tests (#1262)

  - fix issue with handling empty toolchain variables (#1263)

easyblocks

- replace '`log.error`' with '`raise EasyBuildError`' in all easyblocks (#588)
- one new generic easyblock: `ConfigureMakePythonPackage` (#540)
- new easyblocks for 2 software packages that require customized support:
  - TINKER (#578), Xmipp (#581)
- various other enhancements, including:
  - enhance WIEN2k easyblock for recent versions + cleanup (#486)
  - define `$PYTHONNOUSERSITE` in PythonPackage easyblock so user-installed packages are not picked up (#577)
  - add support in CP2K easyblock for building on top of MPICH/MPICH2 (#579)
  - enhance Hadoop easyblock to support parallel builds (#580)
  - drop `-noroot` for recent FLUENT versions, honor `installopts`, enable `-debug` (#582)
  - include `prebuildopts` in build command for Python packages (#585)
  - also install `rosetta_tools` subdirectory for Rosetta (#586)
  - update SLEPc easyblock for v3.5 + style cleanup (#593)
  - minor fix in HPL easyblock w.r.t. checking defined environment variables (#595)
  - tweak CP2K easyblock w.r.t. LAPACK/FFTW support (#596)
  - minor update to GCC easyblock to support GCC v5.x (#598)
  - update sanity check in R easyblock for version 3.2.0 (#602)
- various bug fixes, including:
  - fix Score-P easyblock for compiler-only toolchains, include Qt as optional dependecy (#552)
  - fix definition of `$MKLROOT`, should be set to '`mkl`' subdir of install dir (#576)
  - add `-libmpichf90` to list of MPI libraries in NWChem easyblock (#584)
  - stop using '`$root`' to make easyblocks compatible with module files in Lua syntax (#590)
  - also set `$PYTHONPATH` before installing Python package in temporary directory in `test_step` (#591)
  - unset `buildopts`/`installopts` before installing Python extensions in Python easyblock (#597)
  - allow not including vsc-base sources when installing EasyBuild (gets installed with easybuild-framework) (#600)
  - use `self.initial_environ` rather than `self.orig_environ` in EasyBuildMeta easyblock (#600)
  - make GCC easyblock compatible with `--module-only` by setting default value for `self.platform_lib` in constructor (#603)

easyconfigs

- added example easyconfig files for **27** new software packages:
  - AFNI (#1322, #1521), BCFtools (#1492), getdp (#1518), gmsh (#1518), gtest (#1244), hanythingondemand (#1530), mawk (#1369), Minimac (#815), Minimac3 (#1502), monty (#1548), Octave (#1563), pbs_python (#1530), pigz (#1036), Pygments (#1536), pyhull (#1539), pymatgen (#1549), PyQt (#1322, #1521), Ray (#1494), requests (#1536), seqtk (#1524), SIP (#1322, #1521), S-Lang (#1369), Spark (#1554), spglib (#1549), TINKER (#1465), tmux (#1369), Xmipp (#1489)
- added easyconfigs for new (Cray-specific) toolchains (#1538): `CrayGNU`, `CrayIntel`, `CrayCCE`

  - initially supported software (using CrayGNU toolchains): CP2K, GROMACS, HPL, Python + numpy/scipy, WRF (#1558)
  - see also https://github.com/hpcugent/easybuild/wiki/EasyBuild-on-Cray
- added new easyconfigs for existing toolchains: `goolf/1.5.16`, `intel/2014.06`
- added additional easyconfigs for various supported software packages: version updates, different toolchains, ...
  - including GCC v5.1.0, OpenFOAM v2.3.1, R v3.1.3 and v3.2.0, PETSc/SLEPc v3.5.3, WIEN2k v14.1
- various other enhancements, including:
  - include pbr dependency for lockfile Python extension in Python v2.7.9 easyconfigs + mock/pytz/pandas (#1462, #1540)
  - include SQLite as dependency in Python v2.7.9 easyconfigs (#1468)
  - set `$LD_PRELOAD` for Hoard and jemalloc (#1470)
  - fix homepage in SCOTCH easyconfigs (#1485)
  - adding missing six/ecdsa dependencies for dateutil/paramiko Python packages in Python easyconfigs (#1504, #1505, #1506, #1507, #1508, #1509, #1510)
  - enable `pic` toolchain option in expat easyconfigs (#1562)
  - extend list of source URLs for Bioconductor packages in R easyconfigs to include 'release' source URLs (#1568)
- various bug fixes, including:
  - adding missing zlib dependency in all Tcl easyconfig files (#1344)
  - fix homepage in FLUENT easyconfigs (#1472)
  - use `--with-verbs` rather than deprecated `--with-openib` in OpenMPI configure options (#1511)
  - stop relying on `OS_NAME` constant to specify OS dependencies in OpenMPI easyconfigs (#1512)
  - replace use of '`$root`' with '`%(installdir)s`' to ensure compatibility with module files in Lua syntax (#1532)
  - stop relying on '`$MKLROOT`' in ROOT easyconfigs (#1537)
  - use proper Bundle easyblock for biodeps/PRACE (#1566)
  - make source_urls in Cube and Scalasca easyconfigs compatible with –try-software-version (#1574)
  - add patch for Cube to fix configure script w.r.t. Qt dependency, add –without-java-reader configure option (#1574)

8.12.12 v2.0.0 (March 6th 2015)

feature + bugfix release

framework

- requires vsc-base v2.0.3 or more recent
  - avoid deprecation warnings w.r.t. use of `message` attribute (vsc-base#155)
  - fix typo in log message rendering `--ignoreconfigfiles` unusable (vsc-base#158)
- removed functionality that was deprecated for EasyBuild version 2.0 (#1143)
  - see *Removed functionality*
  - the fix_broken_easyconfigs.py script can be used to update easyconfig files suffering from this (#1151, #1206, #1207)

  - for more information about this script, see *fix_broken_easyconfigs.py*
- stop including a crippled copy of vsc-base, include vsc-base as a proper dependency instead (#1160, #1194)
  - vsc-base is automatically installed as a dependency for easybuild-framework, if a Python installation tool is used
  - see *Required Python packages*
- various other enhancements, including:
  - add support for Linux/POWER systems (#1044)
  - major cleanup in `tools/systemtools.py` + significantly enhanced tests (#1044)
  - add support for '`eb -a rst`', list available easyconfig parameters in ReST format (#1131)
  - add support for specifying one or more easyconfigs in combination with `--from-pr` (#1132)
    - see from_pr_specifying_easyconfigs
  - define `__contains__` in EasyConfig class (#1155)
  - restore support for downloading over a proxy (#1158)
    - i.e., use `urllib2` rather than `urllib`
    - this involved sacrificing the download progress report (which was only visible in the log file)
  - let `mpi_family` return `None` if MPI is not supported by a toolchain (#1164)
  - include support for specifying system-level configuration files for EasyBuild via `$XDG_CONFIG_DIRS` (#1166)
    - see *Default configuration files*
  - make unit tests more robust (#1167, #1196)
    - see *Unit tests*
  - add hierarchical module naming scheme categorizing modules by `moduleclass` (#1176)
  - enhance bootstrap script to allow bootstrapping using supplied tarballs (#1184)
    - see *Advanced bootstrapping options*
  - disable updating of Lmod user cache by default, add configuration option `--update-modules-tool-cache` (#1185)
    - for now, only the Lmod user cache can be updated using `--update-modules-tool-cache`
  - use available `which()` function, rather than running '`which`' via `run_cmd` (#1192)
  - fix `install-EasyBuild-develop.sh` script w.r.t. vsc-base dependency (#1193)
  - also consider robot search path when looking for specified easyconfigs (#1201)
    - see *Specifying builds*
- various bug fixes, including:
  - stop triggering deprecated/no longer support functionality in unit tests (#1126)
  - fix `from_pr` test by including dummy easyblocks for HPL and ScaLAPACK (#1133)
  - escape use of '`%`' in string with command line options with `--job` (#1135)
  - fix handling specified patch level 0 (+ enhance tests for fetch_patches method) (#1139)
  - fix formatting issues in generated easyconfig file obtained via `--try-X` (#1144)
  - use `log.error` in `tools/toolchain/toolchain.py` where applicable (#1145)
  - stop hardcoding `/tmp` in `mpi_cmd_for` function (#1146, #1200)
  - correctly determine variable name for `$EBEXTLIST` when generating module file (#1156)

  - do not ignore exit code of failing postinstall commands (#1157)
  - fix rare case in which used easyconfig and copied easyconfig are the same (#1159)
  - always filter hidden deps from list of dependencies (#1161)
  - fix implementation of `path_matches` function in `tools/filetools.py` (#1163)
  - make sure plain text keyring is used by unit tests (#1165)
  - suppress creation of module symlinks for `HierarchicalMNS` (#1173)
  - sort all lists obtained via `glob.glob`, since they are in arbitrary order (#1187)
  - stop modifying `$MODULEPATH` directly in `setUp`/`tearDown` of toolchain tests (#1191)

easyblocks

- one new generic easyblock for installing a bundle of modules: `Bundle` (#550)
  - and let the `Toolchain` generic easyblock derive from `Bundle`
- new easyblocks for 2 software packages that require customized support:
  - GAMESS-US (#470, #544, #558), Hadoop (#563)
- various other enhancements, including:
  - move support for `staged_install` from CPLEX easyblock to generic `Binary` easyblock (#502)
  - fix sanity check in picard easyblock for v1.119 that doesn't include `sam.jar` (#522)
  - log warning message when unlinking jellyfish symlink fails in Trinity easyblock (#534)
  - revamp `EB_libint2` easyblock into `EB_Libint` that works for both Libint v1x and v2.x (#536)
  - update CP2K easyblock for recent versions (no more '`fes`') (#537)
  - update SuiteSparse easyblock for recent versions (#541)
  - fix easyblock unit tests after dropping support for deprecated behaviour in framework (#543)
  - rework PSI easyblock to support future releases (#545)
  - enable always generating a 'verbose' Makefile in the generic CMakeMake easyblock (#546)
  - remove functionality in (generic) easyblocks that was deprecated for EasyBuild v2.0 (#547)
  - enhance generic RPackage easyblock to support installing extensions in a separate prefix (#551)
  - deprecate GenomeAnalysisTK easyblock, since it's basically equivalent to Tarball (#557)
  - update SAMtools easyblock for v1.2 (#562)
  - take `preconfigopts` easyconfig parameter into account in ROOT easyblock (#566)
  - update easyblock for installing EasyBuild
    * to support bootstrapping with provided source tarballs (#559)
    * to also cover vsc-base dependency, and verify `easy-install.pth` (#567)
  - update disabling sanitizer tests for Clang 3.6 (#570)
- various bug fixes, including: * fix handling of LTO in GCC easyblock (#535) * relocate FDTD RPM to fix installation on SL6 (#538)

easyconfigs

- added example easyconfig files for **29** new software packages:
  - bsoft (#1353), Coot (#1400), Cuby (#1258), DSRC (#1242), Exonerate (#568), fastqz (#1242), FSA (#568), fqzcomp (#1242), GAMESS-US (#1153, #1406), Grep (#1308), Hadoop (#1418), Hoard (#1415), IMB (#1284), ISL (#1389), jemalloc (#1416), libdwarf (#1283), libelf (#1283), MPC (#1310), multitail (#1327), Pmw (#1403), Quip (#1242), rCUDA (#720), SCALCE (#1242), SMALT (#568), STREAM (#1332), worker (#1307), Xerces-C++ (#1370), XQilla (#1370), ZPAQ (#1242)

- added easyconfigs for new (common) toolchains
  - `foss/2015a` (#1239), `gompi/1.5.16` (#1380), `gmvolf/1.7.20` (#1397), `goolf/1.7.20` (#1294), intel/2015a (#1238), `intel/2015.02` (#1393), `iomkl/2015.01` (#1325), `iomkl/2015.02` (#1401)
- added new software bundle: `Autotools` (#1385)
- various other enhancements, including:
  - don't define `$LDSHARED` in zlib easyconfigs (#1350)
    - this fixes the long-standing "`no version information available`" issue with zlib
    - see also framework#108
  - add unit test to check that all `extra_options` keys are defined in `EasyConfig` instance (#1378)
  - add source MD5 checksums in all GCC easyconfigs (#1391)
  - speeding up the unit tests by avoiding rereading of same easyconfig file (#1432)
  - fix conflict detection in unit tests by considering build deps separately from runtime deps (#1447)
  - fix toolchain for Bison build dep in `MVAPICH2-1.9-iccifort-2011.13.367.eb` easyconfig (#1448)
  - use `Bundle` generic easyblock for HPCBIOS bundles and fix `moduleclass` (#1451)
- various bug fixes, including:
  - revert version of Libint dependency to 1.1.4 in CP2K v2.5.1 easyconfig (#1144)
  - added Java dependencies to EMBOSS easyconfigs (#1167)
  - don't list '`lto`' as a language in GCC easyconfigs (#1286)
    - related to the fixes in the GCC easyblock, see easyblocks#535
  - rename libint2 easyconfigs as Libint v2 easyconfigs (#1287)
  - fix mpi4py `source_urls` in Python easyconfigs (#1306)
  - consistently use CLooG 0.18.0 for GCC 4.8 series (#1392)
  - rename GenomeAnalysisTk easyconfigs to GATK (#1399)
  - include `openssl-devel` SLES11 OS dependency in cURL/MySQL/Python easyconfigs (#1422)
  - add missing Perl dependency in parallel easyconfigs (#1430)
  - correct name in BLAST+ easyconfigs (#1443)
  - fix name for sparsehash easyconfigs (#1452)

8.12.13 v1.16.2 (March 6th 2015)

bugfix release

framework

(no changes compared to v1.16.1, simple version bump to stay in sync with easybuild-easyblocks)

easyblocks

- make `EB_EasyBuildMeta` easyblock aware of vsc-base to make upgrading to EasyBuild v2.0.0 possible (#573)

easyconfigs

(no changes compared to v1.16.1, simple version bump to stay in sync with easybuild-easyblocks)

8.12.14 v1.16.1 (December 19th 2014)

bugfix release

framework

- fix functionality that is broken with `--deprecated=2.0` or with `$EASYBUILD_DEPRECATED=2.0`
  - don't include easyconfig parameters for `ConfigureMake` in `eb -a`, since fallback is deprecated (#1123)
  - correctly check software_license value type (#1124)
  - fix `generate_software_list.py` script w.r.t. deprecated fallback to `ConfigureMake` (#1127)
- other bug fixes
  - fix logging issues in tests, sync with vsc-base v2.0.0 (#1120)

easyblocks

- fix EasyBuild versions for which `$EASYBUILD_DEPRECATED=1.0` is set in EasyBuild sanity check (#531)

easyconfigs

- set default easyblock to ConfigureMake in TEMPLATE.eb (#1277)

8.12.15 v1.16.0 (December 18th 2014)

feature + bugfix release

framework

- deprecate automagic fallback to `ConfigureMake` easyblock (#1113)
  - easyconfigs should specify `easyblock = 'ConfigureMake'` instead of relying on the fallback mechanism
  - **note: automagic fallback to `ConfigureMake` easyblock will be removed in EasyBuild v2.0**
  - see also Automagic fallback to ConfigureMake
- stop triggering deprecated functionality, to enable use of `--deprecated=2.0` (#1107, #1115, #1119)
  - see Deprecated functionality for more information
- various other enhancements, including:
  - add script to clean up gists created via `--upload-test-report` (#958)
  - also use `-xHost` when using Intel compilers on AMD systems (as opposed to `-msse3`) (#960)
  - add Python version check in `eb` command (#1046)
  - take `versionprefix` into account in `HierarchicalMNS` module naming scheme (#1058)
  - clean up and refactor `main.py`, move functionality to other modules (#1059, #1064, #1075, #1087)
  - add check in `download_file` function for HTTP return code + show download progress report (#1066, #1090)
  - include info log message with name and location of used easyblock (#1069)
  - add toolchains definitions for `gpsmpi`, `gpsolf`, `impich`, `intel-para`, `ipsmpi` toolchains (#1072, #1073)
    - support for Parastation MPI based toolchains
  - enforce that `hiddendependencies` is a subset of `dependencies` (#1078)

    - this is done to avoid that site-specific policies w.r.t. hidden modules slip into contributed easyconfigs
  - enable use of `--show_hidden` for `avail` subcommand with recent Lmod versions (#1081)
  - add `--robot-paths` configure option (#1080, #1093, #1095, #1114)
  - support use of `%(DEFAULT_ROBOT_PATHS)s` template in EasyBuild configuration files (#1100)
    - see also Controlling the robot search path
  - use `-xHost` rather than `-xHOST`, to match Intel documentation (#1084)
  - update and cleanup `README` file (#1085)
  - deprecate `self.moduleGenerator` in favor of `self.module_generator` in `EasyBlock` (#1088)
  - also support `MPICH` MPI family in `mpi_cmd_for` function (#1098)
  - update documentation references to point to http://easybuild.readthedocs.org (#1102)
  - check for OS dependencies with *both* `rpm` and `dpkg` (if available) (#1111)
- various bug fixes, including:
  - fix picking required software version specified by `--software-version` and clean up `tweak.py` (#1062, #1063)
  - escape `$` characters in module load message specified via `modloadmsg` easyconfig parameter) (#1068)
  - take available hidden modules into account in dependency resolution (#1065)
  - fix hard crash when using patch files with an empty list of sources (#1070)
  - fix Intel MKL BLACS library being used for MPICH/MPICH2-based toolchains (#1072)
  - fix regular expression in `fetch_parameter_from_easyconfig_file` function (#1096)
  - don't hardcode queue names when submitting a job (#1106)
  - fix affiliation/mail address for Fotis in headers (#1105)
  - filter out `/dev/null` entries in patch file in `det_patched_files` function (#1108)
  - fix `gmpolf` toolchain definition, to have `gmpich` as MPI components (instead of `gmpich2`) (#1101)
    - 'MPICH' refers to MPICH v3.x, while MPICH2 refers to MPICH(2) v2.x (MPICH v1.x is ancient/obsolete)
    - **note**: this requires to reinstall the `gmpolf` module, using the updated easyconfig from easybuild-easyconfigs#1217

easyblocks

- new easyblocks for **2** software packages that requires customized support:
  - Chimera (#524), GATE (#518)
- fix use of deprecated functionality, enhance unit tests to check for it (#523)
- various other enhancements, including:
  - update PETSc easyblock for recent versions (v3.5) (#446)
  - only include major/minor version numbers for FLUENT subdir (#480)
  - factor out 'move after install' code from impi easyblock to `IntelBase`, use it for itac (#487)
  - support custom easyconfig parameters in `EB_impi` easyblock to correct behavior of MPI wrapper commands (#493)
  - consider both `lib` and `lib64` in sanity check for GROMACS (#501)

  - take `preinstallopts` and `installopts` into account in `Binary` easyblock (#503)
  - add sanity check command `abaqus information=all` for ABAQUS (#504)
  - update and clean up `README`, refer to http://easybuild.readthedocs.org documentation (#505, #516)
  - rename deprecated `self.moduleGenerator` to `self.module_generator` (#506)
    - see also easybuild-framework#1088
  - check whether specified `type` value is a known value (`psmp` or `popt`) in CP2K easyblock (#509)
  - add support to `SAMtools` easyblock for installing recent versions (v1.x) (#512)
  - fix version check + sanity check in Geant4 easyblock + style fixes (#517)
  - added `$root/modlib` to `$PYTHONPATH` guesses in Modeller easyblock (#525)
  - mark `license` custom easyconfig parameter as deprecated in `IntelBase` (#527)
- various bug fixes, including:
  - fix Libxc version check in CP2K easyblock (#478)
  - correctly specify `mkl_libs` when building *numpy* with GCC and imkl (#485)
  - extend noqa for OpenFOAM-Extend in build_step (#488, #520)
  - correctly set `$LD_LIBRARY_PATH`, `$LIBRARY_PATH` and `$PKG_CONFIG_PATH` for `R` (#495)
  - fix default value for `files_to_copy` in `MakeCp` easyblock (#500)
  - treat `MPICH` MPI family as MPICH v3.x instead of MPICH v1.x (#519)
    - see also easybuild-framework#1112
  - fix affiliation/mail address for Fotis in headers (#521)
  - clean up in `extra_options` methods, avoid casting return value to `dict` or returning via parent (#528)

easyconfigs

- added example easyconfig files for **39** new software packages:
  - ANTs (#1232), BEOPS (#1264), Chhimera (#1255), ctffind (#1249), DBD-SQLite (#1064), DBD-mysql (#1063), DIALIGN-TX (#668), ffmpeg (#1088), GObject-Introspection (#1079), GTS (#1079), Graphviz (#1079), GraphViz2 (#1079), grace (#1131), HarfBuzz (#1079), HTSlib (#1161), GSS-API (#1048), Kerberos_V5 (#1048), libevent (#1063), libXdmcp (#1129), libXft (#1017), libXinerama (#1017), libXrender (#1017), Maven (#1094), MySQL (#1063), Net-LibIDN (#1060), OpenCV (#1088), OpenMD (#1105), Qhull (#1105), Pango (#1079), psmpi (#1245, #1246), RELION (#1017), renderproto (#1017), rjags (#1125), runjags (#1125), SPRNG (#1138, #1141), xineramaproto (#1017), XML-Dumper (#1061), XML-Parser (#1061), XML-Twig (#1061)
- added easyconfigs for new toolchains
  - `intel/2014.10` & `intel/2014.11` (#1219), `intel-para/2014.12` (#1246), `gpsolf/2014.12` (#1245), `iompi/6.6.4` (#1215)
- include `easyblock = 'ConfigureMake'` in relevant easyconfigs to deal with deprecation of automagic fallback to `ConfigureMake` (#1248)
  - see also easybuild-framework#1113 and Automagic fallback to ConfigureMake
- clean up use of deprecated functionality in existing easyconfigs (#1252, #1259)
  - stop using deprecated `makeopts`, `premakeopts` and `shared_lib_ext`
  - check for use of deprecated functionality in easyconfigs unit tests
  - see also http://easybuild.readthedocs.org/en/latest/Deprecated-functionality.html#easyconfig-parameters
- various other enhancements, including:

  - also build `fftw3_threads` libraries, and enhance sanity checks (#1013)
  - add unit test to verify specified `sanity_check_paths` (#1119)
  - update and clean up `README`, refer to http://easybuild.readthedocs.org documentation (#1184, #1224)
- various bug fixes, including:
  - fix unit tests w.r.t. changes in framework (#1146)
  - remove unnecessary build dependencies for OpenMPI (#1168)
  - remove duplicate line in OpenMPI easyconfigs (#1207)
  - fix affiliation/mail address for Fotis in headers (#1237)
  - fix permissions of easyconfig files for consistency (#1210)
  - disable symbol lookup feature in cairo to fix build on SL6 (#1241)
  - fix easyconfig `gmpolf` toolchain w.r.t. MPICH software name (#1217)
    - see also easybuild-framework#1112
  - fix `source_urls` for WRF and WPS (#1225)
  - fix and clean up GATE easyconfigs (#1228)
  - fix broken CLHEP builds by including `-gcc` in `$CXXFLAGS` (#1254)
  - add patch to fix broken test in Go (#1257)
  - fix name of GMAP easyconfigs, should be GMAP-GSNAP (#1268)
  - fix easyconfig filenames, enhance unit test to check easyconfig filenames (#1271)

8.12.16 v1.15.2 (October 7th 2014)

bugfix release

framework

- fix `$MODULEPATH` extensions for Clang/CUDA, to make `goolfc`/`cgoolf` compatible with `HierarchicalMNS` (#1050)
- include `versionsuffix` in module subdirectory with `HierarchicalMNS` (#1050, #1055)
- fix unit tests which were broken with bash patched for ShellShock bug (#1051)
- add definition of gimpi toolchain, required to make gimkl toolchain compatible with `HierarchicalMNS` (#1052)
- don't override `COMPILER_MODULE_NAME` obtained from `ClangGCC` in Clang-based toolchains (#1053)
- fix wrong code in `path_to_top_of_module_tree` function (#1054)
  - because of this, load statements for compilers were potentially included in higher-level modules under `HierarchicalMNS`

easyblocks

- only disable sanitizer tests for recent Clang versions (#481, #482)
- pass down installopts to CUDA install command (#483)

easyconfigs

- disable parallel build for slalib (#968)
- fix compatibility of goolfc with HierarchicalMNS by using GCC toolchain for installing CUDA (#1106, #1115)
- fix zlib OS dependency spec for Debian in Boost easyconfigs (#1109)

- fix compatibility of gimkl with HierarchicalMNS by using gimpi subtoolchain (#1110)
- make both GCC and Clang first-class members in Clang-based toolchains to fix compatibility with HierarchicalMNS (#1113)

8.12.17 v1.15.1 (September 23rd 2014)

bugfix release

framework

- take into account that multiple modules may be extending `$MODULEPATH` with the same path, when determining path to top of module tree (see #1047)
  - this bug caused a load statement for either `icc` or `ifort` to be included in higher-level modules installed with an Intel-based compiler toolchain, under the `HierarchicalMNS` module naming scheme
- make `HierarchicalMNS` module naming scheme compatible with `cgoolf` and `goolfc` toolchain (#1049)
- add definition of `iompi` (sub)toolchain to make `iomkl` toolchain compatible with `HierarchicalMNS` (#1049)

easyblocks

(no changes compared to v1.15.0, simple version bump to stay in sync with easybuild-framework)

easyconfigs

- minor bug fixes, including:
  - use SHLIB_EXT in GMP/MPFR easyconfigs (#1090)
  - fix TopHat homepage and source_urls since page moved (#1092)
  - make iomkl toolchain compatible with HierarchicalMNS (#1099)

8.12.18 v1.15.0 (September 12th 2014)

feature + bugfix release

framework

- various other enhancements, including:
  - fetch extension sources in fetch_step to enhance `--stop=fetch` (#978)
  - add `iimpi` toolchain definition (#993)
  - prepend robot path with download location of files when `--from-pr` is used (#995)
  - add support for excluding module path extensions from generated modules (#1003)
    - see `include_modpath_extensions` easyconfig parameter
  - add support for installing hidden modules and using them as dependencies (#1009, #1021, #1023)
    - see `--hidden` and `hiddendependencies` easyconfig parameter
  - stop relying on `conflict` statement in module files to determine software name of toolchain components (#1017, #1037)
    - instead, the `is_short_modname_for` method defined by the module naming scheme implementation is queried
  - improve error message generated for a missing easyconfig file (#1019)
  - include path where tweaked easyconfigs are placed in robot path (#1032)

  - indicate forced builds in `--dry-run` output (#1034)
  - fix interaction between `--force` and `--try-toolchain --robot` (#1035)
  - add `--software` option, disable recursion for `--try-software(-X)` (#1036)
- various bug fixes, including: * fix `HierarchicalMNS` crashing when MPI library is installed with a dummy toolchain (#986) * fix list of FFTW wrapper libraries for Intel MKL (#987) * fix stability of unit tests (#988, #1027, #1033) * make sure `$SCALAPACK_INC_DIR` (and `$SCALAPACK_LIB_DIR`) are defined when using `imkl` (#990) * fix error message on missing FFTW wrapper libs (#992) * fix duplicate toolchain elements in `--list-toolchains` output (#993) * filter out load statements that extend the `$MODULEPATH` to make the module being installed available (#1016) * fix conflict specification included in module files (#1017) * avoid `--from-pr` crashing hard unless `--robot` is used (#1022) * properly quote GCC version string in archived easyconfig (#1028) * fix issue with `--repositorypath` not honoring `--prefix` (#1031) * sync with latest vsc-base version to fix log order (#1039) * increase # commits per page for `--from-pr` (#1040)

easyblocks

- added support for **2** new software packages that requires customized support:
  - Modeller (#392), NAMD (#397)
- various enhancements, including:
  - fix locale used for running Perl unit tests (#425)
  - fix Rmpi easyblock to correctly configure for Intel MPI (#435)
  - add support in generic Rpackage easyblock for handling patches (#435)
  - enhance OpenFOAM easyblock: fix building MPI build of Pstream and (pt)scotchDecomp + overall cleanup (#436)
  - enhance NWChem easyblock for version 6.3, clean up running of test cases (#441)
  - enhance noqa for interactive configuration of Qt build procedure (#447)
  - add custom sanity check for R in easyblock (#449)
  - make perlmodule take into account `(pre){config,build,install}opts` (#450)
  - add support for specifying an activation key after installing Mathematica (#452)
  - consider both `lib` and `lib64` directories in netCDF sanity check (#453)
  - fix sanity check for ANSYS for v15.0.x (#458)
  - fix Trilinos easyblock for recent version (#462)
  - enhance impi easyblock to handle install subdir for impi v5.0.1 and future version (#465, #472)
  - include $CFLAGS in linker flags for HPL (#466)
  - update sanity test checks for GROMACS 5.x (#471)
- various bug fixes:
  - fix building of FFTW wrappers for Intel MKL v11.1.x + cleanup of imkl easyblock (#445)

easyconfigs

- added example easyconfig files for **13** new software packages:
  - Circos (#780), DB_File (#913), Emacs (#970), evmix (#1077), GD (#780), gsl (#1077), IOR (#949), JAGS (#1076), libgd (#780), MethPipe (#1070), Modeller (#825), NAMD (#835), netCDF-C++4 (#1015)
- added easyconfigs for new toolchains (#986, #1051):
  - gimkl/1.5.9, ictce/7.1.2

- added additional easyconfigs for various supported software packages: version updates, different toolchains, ...
  - including Python 2.7.8/3.4.1, Perl 5.20.0, R 3.1.1, NWChem 6.3, OpenFOAM-Extend 3.1, GCC 4.9.1, Clang 3.4.2, ...
- various enhancements, including:
  - make existing ictce/intel toolchains compatible with HierarchicalMNS (#1069)
    - this involves installing impi with an iccifort toolchain, and imkl with an iimpi toolchain
- various bug fixes, including:
  - download link for Perl modules changed to use cpan.metapan.org
  - fix missing MPI-based OpenFOAM libraries (`Pstream`, `(pt)scotchDecomp`), make sure provided SCOTCH is used (#957)

8.12.19 v1.14.0 (July 9th 2014)

feature + bugfix release

framework

- important changes
  - required Lmod version bumped to v5.6.3 (#944)
    - required due to enhancements and bug fixes in Lmod, e.g. making `--terse avail` significantly faster, and correctly handling a `prepend-path` statement that includes multiple directories at once
  - required Tcl/C environment modules version set to 3.2.10 (
    - hard requirement due to fixed `modulecmd` segmentation fault bug, that only tends manifests itself when making a large amount of changes in the environment (e.g. `module load <toolchain>`)
  - renamed `EasyBuildModuleNamingScheme` to `EasyBuildMNS`
- enhanced custom module naming schemes functionality to support hierarchical module naming schemes (#953, #971, #975)
  - extended API for custom module naming schemes to allow tweaking different aspects of module naming
    - see `easybuild/tools/module_naming_scheme/mns.py` for abstract `ModuleNamingScheme` class
    - an example hierarchical module naming scheme is included, see `HierarchicalMNS`
  - split up full module names into a module subdirectory part, which becomes part of `$MODULEPATH`), and a 'short' module name (is exposed to end-users)
    - example: `GCC/4.7.2` in `Core` subdir, `OpenMPI/1.6.5` in `Compiler/GCC/4.7.2` subdir
  - make `ModuleNamingScheme` class a singleton, move it into `easybuild.tools.module_naming_scheme.mns` module
  - implement `ActiveMNS` wrapper class for quering active module naming scheme
  - implement toolchain inspection functions that can be used in a custom module naming scheme
    - `det_toolchain_compilers`, `det_toolchain_mpi` in `easybuild.tools.module_naming_scheme.toolchain`
  - significant code cleanup & enhanced unit tests
- enhance & clean up `tools/modules.py` (#944, #953, #963, #964, #969)

  - make `ModulesTool` a singleton to avoid repeating module commands over & over again needlessly
  - use `module use`, `module unuse` rather than fiddling with `$MODULEPATH` directly
  - improve debug logging (include full stdout/stderr output of module commands)
  - remove deprecated functionality (`add_module`, `remove_module`, indirect module load)
- various other enhancements, including:
  - added toolchain definitions for 'common' toolchains: `intel` and `foss` (#956)
  - implement caching for easyconfig files, parsed easyconfigs and toolchains (#953)
  - enable `--ignore-osdeps` implicitely when `-D`, `--dry-run` or `--dep-graph` are used (#953)
  - flesh out `use_group` and `det_parallelism` function, include them in `easybuild.tools.systemtools` (#953)
  - make symlinking of module files part of module naming scheme API (#973)
    * list of symlinks paths can be controlled using `det_module_symlink_paths()` method
  - added support for new configuration options:
    * tweaking compiler flags triggered by `optarch` toolchain options using `--optarch` (#949)
    * filtering out dependencies from easyconfig files using `--filter-deps` (#957)
    * filtering environment included in test reports with `--test-report-env-filter` (#959) e.g. `--test-report-env-filter='^SSH|USER|HOSTNAME|UID|.*COOKIE.*'`
    * made suffix used for module files install path configurable, using `--suffix-modules-path` (#973)
  - added support for additional easyconfig parameters: * define aliases in module files (`modaliases`) (#952) * add print message on module load (`modloadmsg`) and Tcl footer (`modtclfooter`) in module files (#954, #974)
- various bug fixes, including:
  - don't try to tweak generated easyconfigs when using `--try-X` (#942)
  - currently create symlinks to module files modules/all under a custom module naming scheme (#953)
  - restore traceback error reporting on hard crashes (#965)

easyblocks

- added **one** new *generic* easyblock: CmdCp (#395)
- added support for **2** new software packages that requires customized support:
  - ANSYS (#398), HPCG (#408)
- various enhancements, including:
  - updated ABAQUS easyblock so that it works for version 13.2 (#406)
  - enhance BLAT easyblock by using `super`'s `build_step` and `prebuildopts`/`buildopts` (#423)
  - enhance Mothur easyblock to guess correct start dir for recent versions (#424)
  - replace use of deprecated (pre)makeopts with (`pre`)''buildopts'' in all easyblocks (#427)
  - fix poor mans version of toolchain compiler detection in imkl easyblock (#429)
- various bug fixes:
  - only check for `idb` for older versions of icc (#426)
  - fix issues with installing RPMS when `rpmrebuild` is required (#433)
  - correctly disable parallel build for ATLAS (#434)

  - fix sanity check for Intel MPI v5.x (only provides bin64) (#432)
  - add `$MIC_LD_LIBRARY_PATH` for MKL v11.x (#437)

easyconfigs

- added example easyconfig files for **17** new software packages:
  - ANSYS (#836), Beast (#912), ELPH (#910, #911), FastTree (#900, #947), GEM-library (#858), HPCG (#853), mdtest (#925), ncview (#648), PRANK (#917), RDP-Classifier (#903), SDPA (#955), SIBELia (#886), SOAPaligner (#857), SPAdes (#884), stemming (#891), WHAM (#872), YAXT (#656)
- added easyconfigs for new toolchains (#935, #944, #948):
  - foss/2014b, ictce/6.3.5, intel/2014b
- added additional easyconfigs for various supported software packages: version updates, different toolchains, ...
- various enhancements, including:
  - replace use of deprecated `(pre)makeopts` with `(pre)buildopts` in all easyblocks (#954)
  - disable running embossupdate on installation of EMBOSS (#963)
- various bug fixes, including: * really enable OpenMP support in FastTree easyconfigs (#947) * fix easyconfigs unit tests after changes in framework (#958)

8.12.20 v1.13.0 (May 29th 2014)

feature + bugfix release

framework

- make `--try-X` command line options work recursively (i.e. collaborate with `--robot`) (#922)
  - EasyBuild will first build a full dependency graph of the specified easyconfigs, and then apply the `--try` specifications
    - the elements of the dependency graph for the used toolchain and its dependencies are left untouched
  - this makes `eb foo-1.0-goolf-1.4.10.eb --try-toolchain=ictce,5.5.0 --robot` also work when `foo` has dependencies
  - caveat: the specified easyconfig files must all use the same toolchain (version)
- add support for testing easyconfig pull requests from EasyBuild command line (#920, #924, #925, #932, #933, #938)
  - add `--from-pr` command line option for downloading easyconfig files from pull requests
  - add `--upload-test-report` command line option for uploading a detailed test report to GitHub as a gist
    - this requires specifying a GitHub username for which a GitHub token is available, using `--github-user`
    - with `--dump-test-report`, the test report can simply be dumped to file rather than being uploaded to GitHub
    - see also https://github.com/hpcugent/easybuild/wiki/Review-process-for-contributions#testing-result
- the `makeopts` and `premakeopts` easyconfig parameter are deprecated, and replaced by `buildopts` and `prebuildopts` (#918)
  - both `makeopts` and `premakeopts` will still be honored in future EasyBuild v1.x versions, but should no longer be used

- various other enhancements, including:
  - add `--disable-cleanup-builddir` command line option, to keep the build dir after a (successful) build (#853)
    - the build dir is still cleaned up by default for successful builds, i.e. `--cleanup-builddir` is the default
  - also consider lib32 in paths checked for `$LD_LIBRARY_PATH` and `$LIBRARY_PATH` (#912)
  - reorganize support for file/git/svn repositories into `repository` package, making it extensible (#913)
  - add support for `postinstallcmds` easyconfig parameter, to specify commands that need to be run after the install step (#918)
  - make `VERSION=` part in version of C environment modules tool optional, which is required for older versions (#930)
- various bug fixes, including:
  - fix small issues in bootstrap script: correctly determine EasyBuild version and make sure modules path exists (#915)
  - fix github unit tests (#916)
  - disable useless debug logging for unit tests (#919)
  - fix unit test for `--skip` (#929)
  - make sure `start_dir` can be set based on location of unpacked first source file (#931)
  - the `vsc` package shipped with easybuild-framework is synced with vsc-base v1.9.1 (#935)
    - fancylogger (used for logging in EasyBuild) is now robust against strings containing UTF8 characters
    - the `deprecated` logging function now does a non-strict version check (rather than an erroneous strict check)
    - the `easybuild.tools.agithub` module is removed, `vsc.utils.rest` now provides the required functionality
  - fix support for unpacking gzipped source files, don't unpack `.gz` files in-place in the source directory (#936)

easyblocks

- added support for **1** new software package that requires customized support:
  - Go (#417)
- various enhancements, including:
  - enhance OpenFOAM easyblock so OpenFOAM-Extend fork can be built too + style fixes (#253, #416)
  - enhance CPLEX easyblock by adding support for staged installation (#372)
  - include support for `configure_cmd_prefix` easyconfig parameter in ConfigureMake generic easyblock (#393)
  - enhance GCC easyblock for building v4.9.0 and versions prior to v4.5 (#396, #400)
  - enhance easyblocks for Intel ipp and itac to support recent versions (#399)
  - enhance Clang easyblock: install static analyzer (#402), be more robust against changing source dir names (#413)
  - enhance FoldX easyblock, update list of potential FoldX binaries to support recent versions (#407)
  - add runtime patching in Boost easyblock to fix build issue with old Boost versions and recent glibc (> 2.15) (#412)

  - enhance `MakeCp` generic easyblock: use location of 1st unpacked source as fallback base dir for `files_to_copy` (#415)
- various bug fixes:
  - fix installing Mathematica when X forwarding is enabled (make sure `$DISPLAY` is unset) (#391)
  - fix permissions of installed files in SAMtools easyblock, ensure read/execute for group/other (#409)
  - fix implementation of `det_pylibdir` function in PythonPackage generic easyblock (#419, #420)
    - determine Python lib dir via `distutils` Python, which works cross-OS (as opposed to hard-coding `lib`)

easyconfigs

- added example easyconfig files for **32** new software packages:
  - APBS (#742), BayesTraits (#770), bc (#888), BitSeq (#791), CEM (#789), CVS (#888), eXpress (#786), file (#888), GEMSTAT (#861), GMAP (#594), Go (#887), iscp (#602), IsoInfer (#773), Jellyfish (#868), less (#888), libcircle (#883), mcpp (#602), MMSEQ (#795), MUSTANG (#800), OpenFOAM-Extend (#437), popt (#759), pscom (#759), psmpi2 (#759), QuadProg++ (#773), rSeq (#771), RSEQtools (#870), Ruby (#873), segemehl (#792), SOAPec (#879), SOAPdenovo2 (#874), SRA-Toolkit (#793), texinfo (#888)
- added easyconfig for new toolchain goolfc/1.4.10
- added additional easyconfigs for various supported software packages: version updates, different toolchains, ...
  - e.g., older versions of Boost (1.47.0), GCC (4.1-4.4), & recent versions of Clang, GCC, Lmod, etc.
- various enhancements, including:
  - add OpenSSL dependency for cURL, to enable HTTPS support (#881)
  - also install esl-X binaries for HMMER (#889)
- various bug fixes, including:
  - properly pass down compiler flags for ParMGridGen (#437)
  - specify proper make options for PLINK, fixing the build on SL6 (#594, #772)
  - fix netloc version (0.5 rather than 0.5beta) (#707)
  - remove Windows-style line ending in netCDF patch file (#796)
  - bump version of OpenSSL dep for BOINC (#882)

8.12.21 v1.12.1 (April 25th 2014)

bugfix release

framework

- return to original directory after executing a command in a subdir (#908)
- fix bootstrap with Lmod, fix issue with module function check and Lmod (#911)

easyblocks

(no changes compared to v1.12.0, simple version bump to stay in sync with easybuild-framework)

easyconfigs

(no changes compared to v1.12.0, simple version bump to stay in sync with easybuild-framework)

8.12.22 v1.12.0 (April 4th 2014)

feature + bugfix release

framework

- various enhancements, including:
  - completed support for custom module naming schemes (#879, #904)
    - a fully parsed easyconfig file is now passed to the `det_full_module_name` function
    - this does require that an easyconfig file matching the dependency specification is available
  - added more features to better support using a shared install target with multiple users (#902, #903, #904)
  - further development on support for new easyconfig format (v2.0) (#844, #848)
    - not considered stable yet, so still requires using `--experimental`
  - enhanced bootstrap script to also support Lmod and `modulecmd.tcl` module tools (#869)
  - added support to `run_cmd_qa` function to supply a list of answers (#887)
  - detect mismatch between definition of `module` function and selected modules tool (#871)
    - allowing mismatch now requires `--allow-modules-tool-mismatch`; an empty `module` function is simply ignored
  - provide lib64 fallback option for directories in default sanity check paths (#896)
  - add support for adding JAR files to `$CLASSPATH` (#898)
  - enhanced and cleaned up unit tests (#877, #880, #884, #899, #901)
  - code cleanup and refactoring
    - get rid of global variable for configuration settings in `config.py`, use singleton instead (#874, #888, #890, #892)
    - track build options via singleton in `config.py` rather than passing them around all over (#886, #889)
    - avoid parsing easyconfig files multiple times by passing a parsed easyconfig to the easyblock (#891)
    - deprecate list of tuples return type of `extra_options` static method (#893, #894)
    - move OS dependency check to `systemtools.py` module (#895)
- bug fixes, including:
  - fix linking with `-lcudart` if CUDA is part of the toolchain, should also include `-lrt` (#882)

easyblocks

- various enhancements, including:
  - also run `Perl Build build` for Perl modules (usually not required, but sometimes it is) (#380)
  - added glob support in generic makecp block (#367, #384)
  - enhance sanity check in GCC easyblock such that it also passes/works on OpenSuSE (#365)
  - add multilib support in GCC easyblock (#379)
- various bug fixes: * Clang: disable sanitizer tests when vmem limit is set (#366) * make sure all libraries are installed for recent Intel MKL versions (#375) * fix appending Intel MPI include directories to `$CPATH` (#371) * statically link readline/ncurses libraries in Python and NWChem easyblocks (#370, #383, #385) * fix easyblock unit tests after changes in framework (#376, #377, #378)

easyconfigs

- added example easyconfig files for **6** new software packages:
  - CLooG (#653), ELPA (#738), LIBSVM (#788), netaddr (#753), netifcas (#753), slalib-c (#750)
- added easyconfigs for new toolchains:
  - ClangGCC/1.3.0 (#653), goolf/1.4.10-no-OFED (#749), goolf/1.5.14(-no-OFED) (#764, #767), ictce/6.2.5 (#726), iomkl (#747)
- added additional easyconfigs for various supported software packages: version updates, different toolchains, ...
- various enhancements, including:
  - tweak BOINC easyconfig to make use of `glob` support available for `files_to_copy` (#781)
  - enable `-fPIC` for libreadline, so it can be linked into shared libs (e.g. `libpython2.x.so`) (#798)
- various bug fixes, including: * fix Qt source_urls (#756) * enable `-fPIC` in ncurses 5.9 ictce/5.2.0 easyconfig, just like in the others (#801) * fix unit tests after changes to framework (#763, #766, #769)

8.12.23 v1.11.1 (February 28th 2014)

bugfix release

framework

- various bug fixes, including:
  - fix hard crash when `$LMOD_CMD` specified full path to lmod binary, but `spider` binary is not in `$PATH` (#861, #873)
  - fix bug in initialisation of repositories, causing problems when a repository subdirectory is specified (#852)
  - avoid long wait when dependency resolution fails if `--robot` is not specified (#875)

easyblocks

(no changes compared to v1.11.0, simple version bump to stay in sync with easybuild-framework)

easyconfigs

(no changes compared to v1.11.0, simple version bump to stay in sync with easybuild-framework)

8.12.24 v1.11.0 (February 16th 2014)

feature + bugfix release

framework

- various enhancements, including:
  - add checksum support for extensions (#807)
  - make checksum functionality more memory efficient by reading in blocks (#836)
  - rewrite of dependency solving for speed and better reporting of missing dependencies (#806, #818)
  - refactoring of `main.py` (#815, #828)
    - function/method signatures to pass down build options
    - move functions from main.py into easybuild.framework.X or easybuild.tools
  - provide better build statistics (#824)
  - add –experimental, `--deprecated` and `--oldstyleconfig` command line options (#838)
    - with `--experimental`, new but incomplete (or partially broken) features are enabled

    - with `--deprecated`, removed of deprecated functionality can be tested (anything deprecated will fail hard)
    - with `--disable-oldstyleconfig`, support for the old style configuration is disabled
  - define `$LIBRARY_PATH` in generated module files (#832)
  - more constants for source URLs (e.g. for downloads from bitbucket) (#831)
  - prefer `$XDG_CONFIG_HOME` and `~/.config/easybuild` over `~/.easybuild` for configuration files (#820)
  - add support for specifying footers to be appended to generated module files (#808)
    - see `--modules-footer` command line option
  - track version of modules tool + cleanup of `modules.py` (#839)
  - move actual `run_cmd` and `run_cmd_qa` implementations from `tools.filetools` into `tools.run` (#842, #843)
  - add support for generating modules that support recursive unloading (#830)
    - see `--recursive-module-unload` command line option
  - add flexibility support for specifying OS dependencies (#846)
    - alternatives can be specified, e.g. (`openssl-devel`, `libssl-dev`)
  - initial (incomplete) support for easyconfig files in new format (v2.0) (#810, #826, #827, #841)
    - requires `--experimental` to be able to experiment with format v2 easyconfig files
- various bug fixes, including:
  - fix problems with use of new-style configuration file (#821)
  - fix removal of old build directories, unless `cleanupoldbuild` easyconfig parameter is set (#809)
  - fix support for different types of repository path specifications (#814)
  - fix unit tests sensitive to `$MODULEPATH` and available easyblocks (#845)

easyblocks

- added **one** new *generic* easyblock: `VSCPythonPackage` (#348)
- added support for **1** new software package that requires customized support:
  - netcdf4-python (#347)
- various enhancements, including:
  - add support for installing recent tbb versions (#341)
  - use `makeopts` in the build step of the generic `PythonPackage` easyblock (#352)
  - define the `$CMAKE_INCLUDE_PATH` and `$CMAKE_LIBRARY_PATH` in the generic `CMakeMake` easyblock (#351, #360)
  - update Clang easyblock to support v3.4 (#346)
  - make sure Python is built with SSL support, adjust Python easyblock to pick up OpenSSL dep (#359)
    - note: providing OpenSSL via an OS package is still recommended, such that security updates are picked up
  - add support for recent netCDF versions (#347)
  - update SuiteSparse easyblock for new versions, and clean it up a bit (#350)
- various bug fixes:
  - fix name of `VersionIndependentPythonPackage` easyblock, deprecate `VersionIndependendPythonPackage` easyblock (#348)

  - fix detection of machine architecture in FSL easyblock (#353)
  - fix bug in NWChem easyblock w.r.t. creating local dummy `.nwchem` file (#360)
  - make sure `$LIBRARY_PATH` is set for Intel compilers and Intel MPI, fix 64-bit specific paths (#360)

easyconfigs

- added example easyconfig files for **30** new software packages:
  - argtable (#669), Clustal-Omega (#669), Coreutils (#582), GMT (#560), gperftools (#596), grep (#582), h4toh5 (#597), libunwind (#596), Lmod (#600, #692), Lua (#600, #692), MAFFT (#654), Molekel (#597), NEdit (#597), netcdf4-python (#660), nodejs (#678), OCaml (#704), patch (#582), PhyML (#664), PRACE Common Production Environment (#599), protobuf (#680), python-dateutil (#438), sed (#582), sickle (#651), Tesla-Deployment-Kit (#489), TREE-PUZZLE (#662), VCFtools (#626), Vim (#665), vsc-mympirun-scoop (#661), vsc-processcontrol (#661), XZ (#582)
- added additional easyconfigs for various supported software packages: version updates, different toolchains, ...
  - OpenSSL with ictce toolchain (#703)
- various enhancements, including:
  - using more constants and templates (#613, #615)
  - specify OS dependency for SSL support, with OpenSSL dependency as fallback (#703)
- various bug fixes, including:
  - fix unit tests after (internal) framework API changes (#667, #672)
  - fix homepage in vsc-mympirun easyconfig file (#681)
  - align OpenMPI easyconfigs (#650)
  - add additional source URL in Qt easyconfigs (#676)
  - specify correct $PATH specification and define `$CHPL_HOME` for Chapel (#683)

8.12.25 v1.10.0 (December 24th 2013)

feature + bugfix release

framework

- various enhancements, including:
  - set unique default temporary directory, add `--tmpdir` command line option (#695)
  - add support for computing and verifying source/patch file checksums (#774, #777, #779, #801, #802)
    - cfr. `checksums` easyconfig parameter
  - add support for *eb –configfhelp*, which prints out an example configuration file (#775)
  - add initial support for `eb` tab completion in bash shells (#775, #797, #798)
    - see also https://github.com/hpcugent/easybuild/wiki/Setting-up-tab-completion-for-bash
    - note: may be quite slow for now
  - enhancements for using Lmod as modules tool (#780, #795, #796):
    - ignore Lmod spider cache by setting `$LMOD_IGNORE_CACHE` (requires Lmod 5.2)
    - bump required Lmod version to v5.2
    - get rid of slow workaround for detecting module directories (only required for older Lmod versions)
    - fix version parsing for Lmod release candidates (rc)

    * improve integration with *lmod spider* by adding `Description: ``` prefix to ```module-whatis` field of module
  - add `--dry-short-short`/`-D` and `--search-short`/`-S` command line options (#781)
  - add toolchain definition for 'gompic', intended for using with CUDA-aware OpenMPI (#783)
  - add support for specifying multiple robot paths (#786)
  - add various source URL constants, add support for `%(nameletter)s` and `%(nameletterlower)s` templates (#793)
  - add `buildininstalldir` easyconfig parameter (#794)
  - add `--ignore-osdeps` command line option (#799, #802)
- various bug fixes, including:
  - enable `-mt_mpi` compiler flag if both `usempi` and `openmp` toolchain options are enabled (#771)
  - only use `libmkl_solver*` libraries for Intel MKL versions prior to 10.3 (#776)
  - fix toolchain support for recent Intel tools (#785)
  - code style fixes in `main.py` to adhere more to PEP8 (#789)
  - make sure `easyblock` easyconfig parameter is listed in `eb -a` (#791)
  - fix error that may pop up when using `skipsteps=source` (#792)

easyblocks

- added **one** new *generic* easyblock: `VersionIndependendPythonPackage` (#329, #334)
- added support for **2** new software packages that require customized support:
  - Charmm (#318), GROMACS (#335, #339)
- various enhancements, including:
  - fix support for recent SAMtools version (>= 0.1.19) (#320)
  - fix support for recent Intel tools (icc, ifort, imkl, impi) (#325, #327, #338)
  - enhance generic easyblock for installing RPMs (#332)
  - pick up `preinstallopts` in generic `PythonPackage` easyblock (#334)
  - enhance CP2K easyblock (libxc support, new versions) + style cleanup (#336)
- various bug fixes:
  - use unwanted env var functionality to unset `$MKLROOT` rather than failing with an error (#273)
  - always include `-w` flag for preprocessor to supress warnings that may break QuantumESPRESSO configure (#317)
  - link with multithreaded LAPACK libs for ESMF (#319)
  - extend `noqanda` list of patterns in CUDA easyblock (#328, #337)
  - add `import _ctypes` to Python sanity check commands to capture a common build issue (#329)
  - bug fixes in generic `IntelBase` easyblock (#331, #333, #335)
    * remove broken symlink `$HOME/intel` if present
    * fix use of gettempdir to obtain a common (user-specific) tmp dir to symlink `$HOME/intel` to
  - fix build of recent scipy versions with GCC-based toolchain (#334)
  - fix use of gettempdir to obtain common (user-specific) tmp dir for `$HOME/.nwchemrc` symlink (#340, #342)
  - extend `noqanda` list in Qt easyblock (#342)

easyconfigs

- added example easyconfig files for **18** new software packages:
  - BEDTools (#579, #632, #635), CAP3 (#548), CHARMM (#584), cutadapt (#620), ErlangOTP (#556, #630), Ghostscript (#547, #632), HTSeq (#554, #632), Jansson (#545), libjpeg-turbo (#574), Molden (#566), netloc (#545), o2scl (#633), packmol (#566), PP (#405), qtop (#500), TAMkin (#566), vsc-base (#621), vsc-mympirun (#621)
- added easyconfigs for new toolchains (#545, #609, #629):
  - gcccuda/2.6.10, gompic/2.6.10, goolfc/2.6.10, ictce/6.0.5, ictce/6.1.5
- added additional easyconfigs for various supported software packages: version updates, different toolchains, ...
  - new versions of icc, ifort, imkl, impi (#609, #629)
  - large collection of ictce/5.3.0 easyconfigs (#627)
- various enhancements, including:
  - extended list of Python packages as extensions to Python (#625)
  - add MPI-enabled version of GROMACS + easyconfigs using ictce (#606, #636)
  - clean up templating of `source_urls` (#637)
- various bug fixes, including:
  - provide alternative download URL for Mesa (#532)
  - add Python versionsuffix in OpenBabel filenames (#566)
  - apply no-gets patch in all M4 v1.4.16 easyconfigs (#623)
  - fix patching of Python w.r.t. `libffi`/`_ctypes` issues (#625, #642)
  - bug fixes in GROMACS easyconfigs (#606)
    - change versionsuffix for non-MPI GROMACS easyconfigs to `-mt`
    - stop using 'CMakeMake' easyblock for GROMACS now that there's a dedicated GROMACS easyblock, which correctly specifies building against external BLAS/LAPACK libraries
  - fix Qt dependency for CGAL (#642)
  - fix libctl, libmatheval, Meep, PSI build issues caused by full paths in `guile-config`/`python-config` shebang (#642)
  - make sure HDF easyconfigs specify dedicated `include/hdf` include dir (#642)
    - this is required to avoid build issues with NCL, because HDF ships it's own `netcdf.h`
    - this also triggered removal of patch files for NCL that rewrote `include/hdf` to `include`
  - fix WPS v3.5.1 patch file after upstream source tarball was changed, supply checksum for verification (#642)

8.12.26 v1.9.0 (November 17th 2013)

feature + bugfix release

framework

- add support for Tcl environment modules (`modulecmd.tcl`) (#728, #729, #739)
  - special care was taken to make sure also the DEISA variant of `modulecmd.tcl` can be used
- code refactoring to prepare for supporting two formats for easyconfig files (#693, #750)
  - this prepares the codebase for supporting easyconfig format v2.0

  - some initial work on adding support for the new easyconfig format is included, but it's by no means complete yet
  - the current easyconfig format (now dubbed v1.0) is still the default and only supported format, for now
  - for more details, see https://github.com/hpcugent/easybuild/wiki/Easyconfig-format-two
- various other enhancements, including:
  - include a full version of vsc-base (see the `vsc` subdirectory) (#740)
    - this is a first step towards switching to using vsc-base as a proper dependency
  - implement get_avail_core_count function in systemtools module that takes cpusets and co into account (#700)
    - the `get_core_count` function is now deprecated
  - add `impmkl` toolchain definition (#736)
  - make regtest more robust: put holds on jobs without dependencies, release holds once all jobs are submitted (#751)
  - add support for specifying multiple alternatives for sanity check paths (#753)
  - add `get_software_libdir` function to modules.py (along with unit tests) (#758)
  - add support for more file extensions and constants w.r.t. sources (#738, #760, #761)
  - add MPICH2 support in `mpi_cmd_for` function (#761)
- various bug fixes, including:
  - fix checking of OS dependencies on Debian/Ubuntu that have `rpm` command available (#732)
  - make unit tests more robust w.r.t. non-writeable `/tmp` and loaded modules prior to starting unit tests (#752, #756)
  - also call `EasyBlock`'s sanity check in `ExtensionEasyblock` if paths/commands are specified in easyconfig (#757)
  - set compiler family for dummy compiler, add definition of toolchain constant for dummy (#759)
- other
  - add build status badges for master/develop branches to `README` (#742)
  - add scripts for installing EasyBuild develop version or setting up git development environment (#730, #755)

easyblocks

- added support for **8** new software packages that require customized support:
  - Allinea DDT/MAP (#279), ARB (#291), GenomeAnalysisTK (#278), OpenBabel (#305, #309), picard (#278), PyQuante (#297), Scalasca v1.x (#304), Score-P (#304)
    - the Score-P easyblock is also used for Cube 4.x, LWM2, OTF2, and Scalasca v2.x
- various enhancements, including:
  - add support building ScaLAPACK on top of MPICH2, required for gmpolf toolchain (#274)
  - add support to ConfigureMake easyblock to customize configure `--prefix` option (#287)
  - add support for specifying install command in Binary easyblock (#288)
  - enhance CMakeMake easyblock to specify srcdir via easyconfig parameter, and to perform out-of-source builds (#303)
- various bug fixes:
  - use correct configure flag for Szip in HDF5 easyblocks, should be `--with-szlib` (#286, #301)
  - add support for serial HDF5 builds (#290, #301)

  - enhance robustness of Qt easyblock w.r.t. interactive configure (#295, #302)
  - enhance support for picking up license specification via environment variables (#298, #307)
  - extend list of values for `$CPATH` in libint2 easyblock (#300)
  - fix `extra_options` `super` call in Clang easyblock (#306)
  - add support in Boost easyblock to specify toolset in easyconfig file (#308)
- other:
  - add build status badges for master/develop branches to README (#289)

easyconfigs

- added example easyconfig files for **58** new software packages:
  - Allinea (#468), ARB + dependencies (#396, #493, #495), arpack-ng (#451, #481), CDO (#484, #521), Cube (#505), ed (#503), FLTK (#503), GenomeAnalysisTK (#467), GIMPS (#527), GTI (#511), IPython (#485, #519), LWM2 (#510), MPICH2 (#460), MUST (#511), ncdf (#496, #522), OPARI2 (#505), OpenBabel (#504, #524), OTF (#505), OTF2 (#505), PandaSEQ (#475), ParaView (#498, #514), ParFlow (#483, #520), PCC (#486, #528), PDT (#505), picard (#467), PnMPI (#511), PyQuante (#499, #523), pysqlite (#519), Scalasca (#505), Score-P (#505), SDCC (#486, #528), Silo (#483, #520), Stride (#503), SURF (#503), TCC (#486, #528)
  - ARB dependencies (23): fixesproto, imake, inputproto, kbproto, libICE, libSM, libX11, libXau, libXaw, libXext, libXfixes, libXi, libXmu, libXp, libXpm, libXt, lynx, motif, printproto, Sablotron, xbitmaps, xextproto, xtrans
- added easyconfigs for new gmpich2/1.4.8, gmpolf/1.4.8 and goolf/1.6.10 toolchains (#460, #525, #530)
- added additional easyconfigs for various software packages (list too long to include here)
  - version updates, different toolchains, ...
- various bug fixes, including:
  - enable building of shared libraries for MPICH (#482)
  - fix HDF configure option for Szip, should be `--with-szlib` (#533)
    - for HDF5, this issue is fixed in the HDF5 easyblock
- other
  - add build status badges for master/develop branches to README (#490)

8.12.27 v1.8.2 (October 18th 2013)

bugfix release

framework

- fix regular expression used for obtaining list of modules from `module avail` (#724)
  - modules marked as default were being hidden from EasyBuild, causing problems when they are used as dependency

easyblocks

- fix installing of EasyBuild with a loaded EasyBuild module (#280)
  - this is important to make upgrading to the latest EasyBuild version possible with a bootstrapped EasyBuild

easyconfigs

- port thread pool patch to PSI 4.0b4 and include it to avoid hanging tests (#471)

8.12.28 v1.8.1 (October 14th 2013)

bugfix release

- various bug fixes, including:
  - fix bugs in regtest procedure (#713)
    - force 2nd and 3rd attempt of build in case 1st attempt failed
  - fix copying of install log to install directory (#716)
  - only create first source path if multiple paths are specified (#718)
  - detect failed PBS job submission by checking obtained job ID for `None` value (#713, #717, #719, #720)

easyblocks

- various bug fixes:
  - fix problems in PSI easyblock causing build to fail (#270)
  - fix issues with running NWChem test cases, fail early in case broken symlink is present (#275)

easyconfigs

- added additional easyconfigs for various software packages (#457):
  - Boost, bzip2, libreadline, ncurses, PSI, Python, zlib
- various bug fixes, including:
  - fix faulty easyconfig file names for HPCBIOS_Math, MUSCLE, XML-LibXML and YAML-Syck (#459, #462)
  - stop (re)specifying default maximum ratio for failed tests in NWChem easyconfig (#457)

8.12.29 v1.8.0 (October 4th 2013)

feature + bugfix release

framework

- add support for using alternative module naming schemes (#679, #696, #705, #706, #707)
  - see https://github.com/hpcugent/easybuild/wiki/Using-a-custom-module-naming-scheme for documentation
  - module naming scheme classes that derive from the 'abstract' `ModuleNamingScheme` class can be provided to EasyBuild
    - the Python module providing the class must be available in the `easybuild.tools.module_naming_scheme` namespace
    - a function named `det_full_module_name` must be implemented, that determines the module name in the form of an string based on the supplied dictionary(-like) argument
  - the active module naming scheme is determined by EasyBuild configuration option `--module-naming-scheme`
  - for now, only the `name`/`version`/`versionsuffix`/`toolchain` easyconfig parameters are guaranteed to be provided
    - consistently providing all easyconfig parameters (i.e., also for dependencies) requires more work (see #687)
  - implementing this involved a number of intrusive changes:
    - the API of the modules.py module needed to be changed, breaking backward compatibility

        - the function for which the signatures were modified are considered to be internal to the framework, so this should have very minor impact w.r.t. easyblocks not included with EasyBuild
        - affected functions include: `available`, `exists`, `show`, `modulefile_path`, `dependencies_for`
    * the format for specifying dependencies was extended, to allow for specifying a custom toolchain
        - this allows to fix inaccurate dependency specifications, for example: `('OpenMPI', '1.6.4-GCC-4.7.2')" to "('OpenMPI', '1.6.4', '', ('GCC', '4.7.2'))`
        - see also easyconfigs#431
    * the recommended version for Lmod was bumped to v5.1.5
        - using earlier 5.x version still works, but may be very slow when 'available' is used, due to bugs and a missing feature in Lmod versions prior to v5.1.5 on which we rely

- various other enhancements, including:
  - only (try to) change group id if it is different from what is wanted (#685)
  - added toy build unit test (#688)
  - support for specifying a list of source paths in EasyBuild configuration (#690, #702)
  - add function to determine CPU clock speed in `systemtools.py` (#694, #699)
- various bug fixes, including:
  - avoid importing toolchain modules over and over again to make toolchain constants available in toolchain module (*#679 <https://github.com/hpcugent/easybuild-framework/pull/679>*)
  - only change the group id if current gid is different from what we want in `adjust_permissions` function (*#685 <https://github.com/hpcugent/easybuild-framework/pull/685>*)
  - restore original environment after running 'module purge' (*#698 <https://github.com/hpcugent/easybuild-framework/pull/698>*)
    * important sidenote: this results in resetting the entire environment, and has impact on the sanity check step;
    * any environment variables that are set before the EasyBlock.sanity_check_step method fires, are no longer there when the sanity check commands are run (cfr. easyblocks#268)

easyblocks

- added **one** new *generic* easyblock: `BinariesTarball` (#255)
- added support for **5** new software packages that require customized support:
  - DB (#226), FDTD Solutions (#239), FoldX (#256), Mathematica (#240), MUMPS (#262)
- various enhancements, including:
  - support optionally running configure in generic `MakeCp` easyblock (#252)
  - enhanced Clang easyblock to support building Clang 3.3 (#248)
  - add support for `$INTEL_LICENSE_FILE` specifying multiple paths (#251)
  - enhanced ATLAS easyblock to support building ATLAS 3.10.1 (#258)
- various bug fixes:
  - add zlib lib dir in link path dirs for WPS (#249)
  - stop using deprecated `add_module` function in imkl easyblock (#250)
  - fixed PSI easyblock w.r.t. support for building plugins (#254, #269)
  - move OS check for Clang to `check_readiness_step`, to allow a build job to be submitted from SL5 (#263, #264)

  - enable verbose build for DOLFIN, to allow for proper debugging if the build fails (#265)
  - make sure `$LDFLAGS` and `$INSTANT_*_DIR` env vars are set for DOLFIN sanity check commands (#268)
    * this is required after resetting the environment after running module purge (see framework release notes)
  - don't try to load module in LAPACK test-only build (#264, #266)

easyconfigs

- added example easyconfig files for **9** new software packages:
  - BOINC (#436), DB (#343, #449), fastahack (#374), FDTD Solutions (#387), FoldX (#440, #442), Mathematica (#394), Mesquite (#447), MUMPS (#447), ParMGridGen (#447)
- added additional easyconfigs for goalf, gompi, ClangGCC, cgmvapich2, cgmvolf toolchains (#350, #441)
- added additional easyconfigs for various software packages:
  - ATLAS, Bison, bzip2, Clang, CMake, cURL, EasyBuild, expat, FFTW, GDB, gettext, git, HPL, LAPACK, libreadline, M4, METIS, MVAPICH2, Mercurial, ncurses, OpenBLAS, OpenMPI, ParMETIS, Python, ScaLAPACK, SCOTCH, Valgrind, zlib
- various 'bug' fixes, including:
  - fix source URL for lockfile in Python easyconfigs (#428)
  - correct dependency specifications w.r.t. versionsuffix and toolchain (#431)
    * this is required to support building the affected easyconfigs with a custom module naming scheme
  - correct PSI patch file to avoid errors w.r.t. memcpy not being in scope (#446)
  - fix gettext build with adding `--without-emacs` configure options, add gettext as dependency for a2ps (#448)
  - exclude EMACS support in a2ps because of build failures (#452)

8.12.30 v1.7.0 (September 2nd 2013)

feature + bugfix release

framework

- various enhancements, including:
  - also search for patch files in directory where easyconfig file is located (#667)
  - reduce false positives in reporting of possible errors messages (#669)
  - make module update `$ACLOCAL` if a share/aclocal subdir is found (#670)
  - add `unwanted_env_vars` easyconfig parameter to list environment variables to unset during install procedure (#673)
  - add support for updating list easyconfig values (next to string values) (#676)
  - add `--dry-run` command line option which prints installation overview for specified easyconfig files (#677)
- various bug fixes, including: * ensure that all extensions are listed in `$EBEXTSLISTX` set by module, also when using `--skip` (#671) * report reason for failed sanity check for extensions (#672, #678) * fix `--list-toolchains` command line option (#675)

easyblocks

- added support for **3** new software packages that require customized support:
  - Libint2 (#236), Qt (#210), Rosetta (#218)

- various enhancements, including:
  - allow building OpenFOAM without 3rd party tools (#230)
  - also add sitelib path to `$PERL5LIB`, refactor code to add generic `get_site_suffix` function (#232, #233)
  - support building imkl FFT wrappers using MVAPICH2 MPI library (#234)
  - remove libnpp from CUDA sanity check to support installing CUDA v5.5 (#238)
  - pick up `$INTEL_LICENSE_FILE` for Intel tools, if it is set (#243) * note: gets preference over `license_file` easyconfig parameter
- various bug fixes:
  - call WRF build script with '`tcsh <script>` to ensure that tcsh available in `$PATH` is used (#231)
  - make sure some environment variables that may disrupt the GCC install procedure are unset (#237) * e.g., `$CPATH`, `$C_INCLUDE_PATH`, `$CPLUS_INCLUDE_PATH`, `$OBJC_INCLUDE_PATH`, `$LIBRARY_PATH`
  - code cleanup in GEANT4 easyblock: use `self.version` (instead of `self.get_installversion()`) (#242)
  - enhance list of `noqanda` patterns for CUDA, to get less failing installations (#244)

easyconfigs

- added example easyconfig files for **15** new software packages:
  - Glib (#294, #400), GLPK (#400), horton (#413), libint2 (#413), molmod (#413), Rosetta (#336), SCons (#336), Stacks (#367, #377), sympy (#413), Qt (#294), XML-LibXML (#397), XML-Simple (#397), yaff (#413), YAML-Syck (#380), zsh (#376)
- added additional easyconfigs for various software packages:
  - BLAST, BamTools, BioPerl, Bison, Boost, bzip2, CMake, Cython, CUDA, FFTW, FIAT, GCC, GMP, gettext, git, h5py, HDF5, libffi, libreadline, libxc, matplotlib, METIS, ncurses, Oases, Python, RAxML, ScientificPython, Szip, tcsh, imkl, MVAPICH2, TotalView, VTune, WRF, zlib
- added toolchain easyconfig files for HPCBIOS policies (#402, #407)
  - HPCBIOS_BioInfo, HPCBIOS_Debuggers, HPCBIOS_LifeSciences, HPCBIOS_Math, HPCBIOS_Profilers
- various enhancements, including:
  - added more XML Perl modules to non-bare Perl easyconfigs (#375)
- various 'bug' fixes, including:
  - fix website/description in scipy easyconfigs (#399)
  - specify OpenMPI libibverbs-dev(el) OS dependency in an OS-dependent way (#403)
  - add patch file for M4 to fix building on systems with recent glibc (>=2.16) (#406)
  - align moduleclass in R easyconfigs (#411)
  - fixed filename of Biopython/CD-HIT easyconfig files (#407)
  - disable parallel building of otcl (#419)

8.12.31 v1.6.0 (July 11th 2013)

feature + bugfix release

framework

- added support for using Lmod as module tool (#645)

- various other enhancements, including:
  - allow prepending to/appending to/overwriting list easyconfig parameters using `--try-amend-X` (#658, #664)
- various bug fixes, including:
  - add salt to temporary log file name (#656, #665)
  - fix determining CPU architecture on Rasberry Pi (ARM) systems (#655, #662)
  - fix support for determining base path of tarballs containing a single file (#660)

easyblocks

- added support for **2** new software packages that require customized support:
  - BamTools (#224), BLAT (#214)
- various enhancements, including:
  - update impi easyblock to allow installing impi v4.1.1, which features a slight change in build procedure (#217)
  - enhance `PackedBinary` easyblock to copy both files and directories (#212)
  - added get `sitearch_suffix` to Perl search path and use it in `PerlModule` easyblock (#209)
- various bug fixes:
  - make sure EasyBuild configuration is initialized when running unit tests (#220)
  - make Boost easyblock pick up configopts easyconfig parameter (#221)
  - add `-DMPICH_IGNORE_CXX_SEEK` compiler flag for Mothur when MPI support is enabled (#222)
  - fix Boost sanity check, only check for `libboost_python.so` if Python module is loaded (#223)
  - enhance Trinity support w.r.t. jellyfish (#225, #227)
  - fix checking for `beagle-lib` dep (deprecate checking for BEAGLE) for MrBayes (#228)

easyconfigs

- added example easyconfig files for **26** new software packages:
  - ALLPATHS-LG (#359), AutoMake (#347), BamTools (#319, #338), BLAT (#340), Biopython (#356), cairo (#361), CCfits (#327), CD-HIT (#344), CFITSIO (#327), Diffutils (#347), FASTA (#358, #361), findutils (#347), fontconfig (#361), gawk (#347), gettext (#361), GLIMMER (#357, #361), libidn (#361), LibTIFF (#347), libungif (#347), make (#355), MUSCLE (#339), Oases (#354), pixman (#361), PLINK (#352), RCS (#347), SQLite (#347)
- added additional easyconfigs for various software packages:
  - ant, Bash, Bison, bzip2, cURL, expat, GCC, EasyBuild, freetype, FFTW, GDB, git, HMMER, JUnit, libreadline, libpng, libtool, libxml2, libxslt, M4, makedepend, Mothur, MVAPICH2, Mercurial, ncurses, OpenBLAS, Python, ScaLAPACK, Tcl, tcsh, TopHat, Trinity, Valgrind, Velvet, VTune, zlib (see #169, #297, #298, #301, #309, #323, #331, #332, #341, #347, #349, #351, #355, #361)
- various enhancements, including:
  - added easyconfigs for `ictce/5.4.0`, `ictce/5.5.0` and `gmvolf/1.7.12` toolchain modules (#297, #332, #349)
  - added a template sanity_check_paths as 'MUST' in TEMPLATE.eb (#329)
  - introduced biodeps 'toolchain' to ease keeping common dependencies for bio* software in sync (#309)
  - added collection of easyconfigs for ictce/5.3.0 (#309, #323)
    - bam2fastq, bbFTP, BLAST, Boost, DL_POLY Classic, EMBOSS, FFTW, libharu, libxml2, libxslt, libyaml, lxml, Mercurial, Mothur, mpi4py, ncurses, ns, orthomcl, otcl, PAML, Perl, PyYAML, pandas, problog, scikit-learn, TiCCutils, TiMBL, TinySVM, TopHat, tclcl, YamCha

  - added missing dependencies for various software packages (#323, #328, #348, #361)
  - style fixes in various easyconfigs (#309, #323, #345, #349, #355, #361)
- various 'bug' fixes, including:
  - added `pic` toolchain option for Perl goolf easyconfig (#299)
  - fixed source URLs for R (use correct template `%(version_major)s`) (#302)
  - synced libreadline easyconfigs w.r.t. ncurses dependency (#303)
  - make sure EasyBuild configuration is initialized when running unit tests (#334)
  - specify `lowopt` (-O1) optimization level for OpenIFS, to avoid floating-point related issues (#328)
  - fix naming of 'beagle-lib' (used to be 'BEAGLE'), to avoid name clashes with other software package(s) (#346)

8.12.32 v1.5.0 (June 1st 2013)

feature + bugfix release

framework

- various enhancements, including:
  - define `SHLIB_EXT` constant for shared library extension (`.so`, `.dylib`), deprecate `shared_lib_ext` global var (#630)
  - enhance support for sanity checking extensions (#632, #649)
  - add support for `modextrapaths` easyconfig parameter (#634, #637)
  - allow `source_urls` to be templated for extensions (#639, #646, #647)
  - set `OMPI_*` environment variables for OpenMPI (#640)
  - make BLACS optional as toolchain element, depending on ScaLAPACK version (#638)
- various bug fixes, including:
  - fixed `--list-toolchains`, avoid listing toolchains multiple times (#628)
  - fix templating dictionary after parsing easyconfig file (#633)
  - fix support for ACML as compiler toolchain element (#632)
  - make unit tests clean up after themselves more thoroughly (#641, #642, #643)

easyblocks

- added **one** new *generic* easyblock: `MakeCp` (#208)
- added support for **5** new software packages that require customized support:
  - CBLAS (#192), FreeSurfer (#194), Mothur (#206), OpenIFS (#200), PSI (#191)
- various enhancements, including:
  - add support for building ScaLAPACK 2.x on top of QLogic MPI (#195)
  - support newer BWA versions (#199)
  - explicitly list license server type in ABAQUS install options, required for correct installation of v6.12 (#198)
  - update SCOTCH and OpenFOAM easyblock for recent versions (#201)
- various bug fixes:
  - fix numpy easyblock to get an optimal build (w.r.t. `numpy.dot` performance) (#192)
  - correct build procedure for MUMmer to yield a complete installation (#196, #197)

  - make unit tests clean up after themselves more thoroughly (#203, #204)
  - fix getting Perl version for extensions (#205)

easyconfigs

- added example easyconfig files for **23** new software packages:
  - bam2fastq (#287), CBLAS (#263), EMBOSS (#265, #290), FCM (#272), FRC_align (#273), freeglut (#271), FreeSurfer (#271), FSL (#271), GATK (#287), libharu (#290), libxslt (#235), MariaDB (#292), Mothur (#265) mpi4py (#276), OpenIFS (#272), orthomcl (#265), PAML (#287), pandas (#262), phonopy (#235), problog (#277), PSI (#258), PyYAML (#235), RAxML (#277)
- added additional example easyconfig files for:
  - ABINIT (#235), ACML (#267), BLAST (#275), Boost (#273), BWA (#270), bzip2 (#263), Chapel (#240), CMake (#290), FFTW2 (#247, #267), flex (#267), freetype (#235), grib_api (#272), gzip (#265), Java (#279), libpng (#240, #279), libreadline (#267), libxml2 (#235), libxml (#235), matplotlib (#235), MCL (#265), MUMmer (#265), ncurses (#267), numpy (#267), OpenFOAM (#267), Perl (#265), Python (#276, #263), R (#240, #279), SCOTCH (#267), ScaLAPACK (#267), TopHat (#289), Valgrind (#255), zlib (#267)
- various enhancements, including:
  - enhance unit test suite, include testing for module conflicts (#256) and presence of patch files (#264)
  - use provided constants and templates in easyconfig files where appropriate (#248, #266, #281)
- various 'bug' fixes, including:
  - get rid of hardcoded license_file paths for VTune, Inspector (#253)
  - assign proper moduleclass where they were missing (#278)
  - fix naming for LZO (#280)
  - make unit tests clean up after themselves more thoroughly (#283, #284, #285, #286)
  - fix TopHat dependencies (#289)
  - fix source URLs for XML (#279)
  - fix versions for all listed R extensions (#279)
  - fix CUDA easyconfig file for use on Debian Squeeze (#291)
  - correct easyconfig filename and module name mismatches (bbcp, DL_POLY Classic, ...) (#295)

8.12.33 v1.4.0 (May 2nd 2013)

feature + bugfix release

framework

- the unit tests for easybuild-framework were moved to test/framework, to make things consistent with easybuild-easyblocks and easybuild-easyconfigs (#611, #613, #624)
  - running the framework unit tests should now be using `python -m test.framework.suite`
- various other enhancements, including:
  - extend unit test suite (#593, #599, #603, #618, #620, #622, #624, ...)
  - extended list of constants and templates that can be used in easyconfig files (#566)
  - add support for additional compiler toolchains
    - CUDA-enabled toolchain: `goolfc` (#603, #624)
    - Clang(+GCC)-based toolchains: `cgoolf`, `cgmpolf`, `cgmvolf` (#593, #598, #600)
    - gmvolf (GCC+MVAPICH2+...) (#585)

  - properly decode easyblock to module name using `decode_*` functions (#618)
- various bug fixes, including:
  - fixed default value for `--stop` (#601)
  - remove useless `sleep()` calls in `run_cmd`, `run_cmd_qa` (#599)
  - determine module path based on class name, not software name (#606)
  - remove unwanted characters in build dirs (#591, #607)
  - ignore some error codes spit out by modulecmd that are actually warnings (#609)
  - fix `agithub.py` w.r.t. changes in GitHub API (user-agent string is now obligatory for non-authenticated connections) (#617)
  - fix typo breaking the `adjust_cmd` decorator on SuSE (#615)
  - fix prepending paths with absolute paths in module file (#621)
  - clean up open file handles properly (#620, #624)
  - fix `--search` help and implementation (#622)

easyblocks

- added a unit test suite for easyblocks (#175, #177, #178)
  - every easyblock is parsed and instantiated as a sanity check
- added **one** new *generic* easyblock: `PerlModule` (#183)
- added support for **8** new software packages that require customized support:
  - ABAQUS (#179), Bowtie (#174, #185, #186), Clang (#151), DL_POLY Classic (#118), ESMF (#171), Perl (#183), Intel VTune and Intel Inspector (#180)
- the `CMakeMake.configure_step` parameter `builddir` was renamed to `srcdir`, because the name `builddir` is incorrect (#151)
  - `builddir` will remain supported for legacy purposes up until v2.0
- various enhancements, including:
  - reverted back to hardcoding Python library path, since it's hardcoded by setuptools too (#184)
  - added MPICH support in ScaLAPACK easyblock (#172)
  - enhanced NCL easyblock: add support UDUNITS and ESMF dependencies (#171)
  - enhanced MATLAB easyblock: avoid hardcoding Java options, make sure `$DISPLAY` is unset, extend list of sanity check paths (#181)
  - enhanced TotalView easyblock: add support for license file (#146)

easyconfigs

- added a unit test suite for easyconfigs (#228, #230)
- added example easyconfig files for **20** new software packages: * ABAQUS (#231), BioPerl (#242), Bowtie (#227), Clang (#177), CRF++ (#131), DL_POLY Classic (#132), ESMF, GROMACS (#165), HH-suite (#219), Inspector (#232), likwid (#131), Perl (#242), scikit (#133), TiCCutils (#131), TiMBL (#131), TinySVM (#131), UDUNITS (#167), VTune (#232), YamCha (#131)
- add example easyconfigs for new compiler toolchains (use `eb --list-toolchains` for a full list of supported toolchains):
  - the newly added toolchains only differ in compilers/MPI library, and all feature OpenBLAS, LAPACK, ScaLAPACK and FFTW
  - *goolfc*: GCC, CUDA (co-compiler), OpenMPI (#191)
    - a goolfc easyconfig for GROMACS is inluded as proof-of-concept (#165)

  - *cgmpolf*: Clang (C/C++ compilers), GCC (Fortran compilers), MPICH (#213)
  - *cgmvolf*: Clang, GCC, MVAPICH2 (#218)
  - *cgoolf*: Clang, GCC, OpenMPI (#213)
  - *gmvolf*: GCC, MVAPICH (#202, #222)
- ported already available easyconfigs to new compiler toolchains:
  - *ictce-5.3.0*: 193 easyconfigs (#229)
  - *cgmpolf*: 11 easyconfigs (#213)
  - *cgmvolf*: 11 easyconfigs (#218)
  - *cgoolf*: 12 easyconfigs (#213)
  - *gmvolf*: 10 easyconfigs (#215)
- added additional example easyconfig files for:
  - CMake (#163), git (#210), Java (#206), #221, Mercurial (#201, #215), ncurses (#225), TotalView (#160)
- various enhancements, including:
  - added ESMF and UDUNITS dependencies to NCL easyconfigs (#211)
  - changed value of source_urls in R easyconfigs, to be generic enough for version 3.0 and possibly beyond (#251)
- various 'bug' fixes, including:
  - add `--enable-mpirun-prefix-by-default` configure option for all OpenMPI easyconfigs (#205)

8.12.34 v1.3.0 (April 1st 2013)

feature + bugfix release

framework

- added script to bootstrap EasyBuild with EasyBuild, see https://github.com/hpcugent/easybuild/wiki/Bootstrapping-EasyBuild (#531)
- reorganize framework/easyconfig.py module into framework/easyconfig package with modules (#574, #580)
- support EasyBuild configuration via command line, environment variables and configuration files (#529, #552, #556, #558, #559)
- various other enhancements, including:
  - extended set of unit tests for eb command line options and EasyBuild configuration (#517, #556, #559, #571)
  - made `--search` also useful when easybuild-easyconfigs package is not installed (#524)
  - extended set of default module classes (#525)
  - add support for license easyconfig parameter (which will be mandatory in v2.x) (#526, #569)
  - added `setup.cfg` for `setup.py` to allow creating RPMs (#528)
  - added support for obtaining system information, see `get_os_*` functions in `easybuild/tools/systemtools.py` (#543, #546, #547)
  - added support for iterated builds that require cycling over multiple sets of configure/build/install options, e.g. FFTW (#549)
  - added support for OpenBLAS as toolchain lib for linear algebra (#537, #565)

  - added support for tweaking prefix and separator for library toolchain variables (`LIB*`) (#572, #576)
  - skip already built modules in regression testing mode, to ease regression testing (#582)
- various bug fixes, including: * added `zip_safe` flag to `setup.py`, to silence multitude of warnings (#521) * only define `LIBFFT` for Intel MKL if FFTW interface libraries are available (#518, #567, #579) * set POSIX group early in build process, make EasyBuild aware of consistent chmod/chown failures (#527) * properly order the name/version keys for the toolchain easyconfig parameter when using `--try-toolchain` (#563) * take the `skipsteps` easyconfig parameter into account in regression testing mode as well (#583)

easyblocks

- added support for **2** new software packages that require customized support:
  - CUDA (#145), Ferret (#160, #163)
- remove `license` easyconfig parameter from `IntelBase`, since it clashes with generic `license` parameter (#153, #158)
  - `license_file` should be used instead (see framework#569)
  - using `license` instead of `license_file` will be supported until v2.x for legacy purposes
- various enhancements, including:
  - stop hardcoding Python site-packages library dir, obtain it via `distutils.sysconfig` instead (#141, #156, #159, #161)
  - stop hardcoding list of libraries for BLAS libs, ask toolchain modules or use `$LIBBLAS` instead (#150, #155)
  - enhanced CP2K easyblock, following Intel guidelines for `ictce` builds (#138)
  - added `setup.cfg` for `setup.py` to allow creating RPMs (#140)
  - clean up specifying BLAS/LAPACK libs for building numpy, check whether requires patch is being used for IMKL builds (#155, #161)
- various bug fixes, including:
  - added `zip_safe` flag to `setup.py`, to silence multitude of warnings (#135)
  - install EasyBuild packages in reversed order to avoid funky setuptools issues (#139)
  - fixed a typo in the ScaLAPACK easyblock, and set CCFLAGS and FCFLAGS for v2.x (#149, #162)
  - fix sanity check for python-meep (#159)
  - exclude Python tests from DOLFIN sanity check, since they hang sometimes (#161)
  - add support in ALADIN easyblock for answering question on location of `libgrib_api.a` (#165)

easyconfigs

- added example easyconfig files for **13** new software packages:
  - Bash, CUDA, ccache, Ferret, gzip, libxc, ns, numactl, OpenBLAS, otcl, Tar, tclcl, tcsh
  - several of these easyconfig files were contributed by attendees of the EasyBuild hackathon in Cyprus!
- added example easyconfigs for goolf toolchain (#158)
- added example easyconfigs for builds with goolf toolchain, i.e. for all goalf easyconfigs (#189)
  - for several software packages, a patch file was needed to get them to build with GCC 4.7:
    - AMOS, BEAGLE, Cufflinks, DOLFIN, GATE, ns, Pasha, Trilinos, Trinity
  - for PETSc, a patch file was required to make it build with a toolchain that doesn't include BLACS
- added additional example easyconfig files for:
  - gompi, hwloc, LAPACK, MVAPICH2, OpenMPI, ScaLAPACK

- various enhancements, including:
  - define a proper module class in *all* easyconfigs, cfr. default module classes defined in framework (#150, #159, #161, #162, #179, #181)
  - extend FFTW easyconfig to 'fat' builds that include single, double, long double and quad precision libraries in the same module
    - quad precision is disabled for Intel compiler based builds (it requires GCC v4.6+)
  - the imkl easyconfigs used for the ictce 3.2.2.u3 toolchain now also enable FFTW interfaces
- various 'bug' fixes, including:
  - fix filename for Mercurial and ROOT easyconfig files
  - fix homepage/description for Hypre
  - fix enabling OpenMP support in OpenMPI: use `--enable-openmp`, not `--with-openmp`
  - use correct configure flag for enabling OpenMPI threading support in v1.6 (#186)
    - `--enable-mpi-thread-multiple` instead of `--enable-mpi-threads`
  - explicitely add `--without-openib --without-udapl` configure options in OpenMPI easyconfig using versionsuffix `-no-OFED` (#168)
    - this avoids that OpenMPI auto-detects that it can enable Infiniband (OpenIB) support, which doesn't fit the -no-OFED versionsuffix
    - Note: this makes goalf-1.1.0-no-OFED effectively not suitable to produce software builds that are IB-capable!
  - remove explicit `--with-udapl` from OpenMPI easyconfigs, does more harm than good (#178)
  - remove libibvers, libibmad, libibumad as explicit dependencies for OpenMPI/MVAPICH2 (#173, #182)
    - leave it up to the OS to provide these, since the required version is too much tied to the version of IB drivers
  - use `license_file` in Intel tools easyconfigs, as opposed to the new generic `license` parameter with a different meaning (#180)
  - modify patch for impi to avoid installation problems due to hardcoded path in /tmp (#185)
    - now uses `$USER-$RANDOM` subdir to avoid clashes between different users on the same system
  - the patch file for numpy was extended to also supporting ATLAS and other BLAS libraries spread across multiple directories
    - the extension for ATLAS is required because we now no longer rely on the automatic numpy mechanism to find the ATLAS libs
  - fixed dependencies:
    - libibumad as dependency for libibmad
    - ncurses as dependency for libreadline
    - ncurses and zlib as dependency for SAMtools (+ enhanced patch)
    - remove explicit FFTW dependency for Meep, ... since toolchain already provided FFTW

8.12.35 v1.2.0 (February 28th 2013)

feature + bugfix release

framework

- new backend module for option parsing: generaloption

- support for using constants and string templates in easyconfig files
  - currently disabled for `exts_filter` and `exts_list` easyconfig parameters, for backward compatibility
- various other enhancements, including:
  - support for iqacml and iiqmpi toolchains (Intel compilers + QLogic MPI based)
  - clearer errors messages when sanity check failed
  - unit tests for (about half of) the `eb` command line options
  - support for specifying build/install steps to skip in easyconfig file (`skipsteps`)
  - support for allowing certain dependencies to be resolved by the system instead of modules (`allow_system_deps`)
  - cache ppn value required by regtest, clean up temporary files let behind by `--regtest`/`--job`
  - make sure MPD is used as process manager for Intel MPI (required for impi v4.1 and later)
  - rename template names `name` and `version` used in exts_filter to `ext_name`, `ext_version`
    - `name` and `version` are still supported for legacy reasons
  - cleaned up module docstrings w.r.t. list of authors
- various bug fixes, including:
  - print correct (lowercase) toolchain names with `--list-toolchains`
  - correct easyconfig parameter name `license_server_port`
  - fix string quoting in develop modules
  - ensure `modulecmd` is run with original `$LD_LIBRARY_PATH` value
    - to avoid breaking `modulecmd`, see https://bugzilla.redhat.com/show_bug.cgi?id=719785
  - remove use of hardcoded files/dirs in unit tests
  - fix various inconsistencies w.r.t. paths considered with `--robot`
  - various cleanup and fixes w.r.t. logging
    - use correct logger instance in main script
    - stop passing logger instances around
    - make module logging variables private
  - get rid of `ModuleGenerator` deconstructor, clean up via `EasyBlock.clean_up_fake_module`
  - fix disabling of `optarch` toolchain option (and extend unit tests to check on this)

easyblocks

- added **one** new *generic* easyblock: `Rpm`
- added support for **6** new software packages that require customized support:
  - EasyBuild, EPD (Enthought Python Distribution), freetype, MATLAB, QLogic MPI (RPM), TotalView
  - support for installing EasyBuild with EasyBuild enables bootstrapping an EasyBuild installation!
- various enhancements, including:
  - corrections in WRF/WPS to also enable building with iqacml toolchain
    - use `mpi_cmd_for` instead of hardcoding test commands, using correct Fortran compilers (F90)
  - fix NCL easyblock to also support v6.1.x

        * use correct Fortran compiler (F90), set correct lib/include paths for dependencies (netCDF-Fortran, GDAL)
    - cleanup sweep of license headers and authors list in easyblock module docstrings
    - use new `ext_name` template name in `exts_filter` in Python and R easyblocks
- various bug fixes, including:
    - general code cleanup
        * don't set `sanityCheckOK` in `Toolchain` easyblock
        * get rid of using `os.putenv`
    - NEURON easyblock: don't hardcode number of processes used in test cases
    - make sure `easybuild.easyblocks.generic` namespace is extendable

easyconfigs

- added example easyconfig files for **41** new software packages:
    - a2ps, AnalyzeFMRI, aria, bbcp, bbFTP, bbftpPRO, binutils, Bonnie++, ccache, cflow, cgdb, Corkscrew, EasyBuild, Elinks, EPD, FLUENT, fmri, GDB, GDAL, gnuplot, gnutls, gperf, Iperf, lftp, libyaml, lzo, MATLAB, mc, nano, NASM, nettle, numexpr, parallel, pyTables, QLogic MPI, Stow, TotalView, Valgrind, VTK, Yasm, zsync
- added example easyconfigs for iqacml and iiqmpi toolchains
- added additional example easyconfig files for:
    - ABINIT, ABySS, ACML, BFAST, Bison, BLACS, Cython, cURL, Doxygen, FFTW, flex, g2clib, g2lib, GHC, h5py, HDF, HDF5, HMMER, JasPer, icc, ictce, ifort, imkl, impi, libpng, libreadline, M4, matplotlib, MCL, MEME, mpiBLAST, NCL, ncurses, netCDF, netCDF-Fortran, NWChem, R, ScaLAPACK, Tcl, Tk, WPS, WRF, zlib
- various enhancements, including:
    - fix version of xtable R library in list of extensions for R, to avoid installation failures
- various 'bug' fixes, including:
    - fix toolchain and file names for ABINIT easyconfigs
    - fix sanity check paths for cURL
    - don't disable `optarch` for WRF, it's not needed (only disable heavy optimizations is required)
    - fix homepage/description for ALADIN

8.12.36 v1.1.0 (January 27th 2013)

feature + bugfix release

framework

- improvements w.r.t. support for software extensions (tested on Python and R, see easyblocks package)
    - cleaned up support for building/installing extensions
    - define `ExtensionEasyblock` class that implements support for installing extensions as stand-alone module as well
    - return to build dir before building/installing each extension
    - define `EBEXTSLIST<NAME>` environment variable in module if `exts_list` was defined
    - make sure sanity check for extensions results in an error if it fails
- various enhancements, including:

  - log both framework and easyblocks versions
  - add support for `gimkl`, `gmacml`, `iccifort`, `iomkl` and `ismkl` toolchains
  - define `*_SEQ compiler` variables for sequential compilers
  - add `--list-toolchains` command line option for listing supported toolchains
  - add support for customizing software and modules install path suffixes
  - support both setuptools and distutils installation methods for finding installed easyconfigs
  - also consider robot path in list of paths searched for patch files
  - allow skipping of default extension sanity check (by setting `modulename` to False in options)
- various bug fixes, including:
  - typos in toolchain Python modules w.r.t. `imkl` support, handling of `i8`/`optarch`/`unroll` options
  - purge before loading 'fake' module, unload 'fake' module before removing it, use original `$MODULEPATH`
  - restore environment after unloading fake module, set variables that were incorrectly unset, i.e., that were defined before as well
  - unset `$TMPDIR` for builds submitted as jobs (required by `IntelBase` easyblock)
  - correctly track easyconfig parse error
  - always run all jobs in regtest, also if dependency jobs failed
  - cosmetic adjustments to default EasyBuild configuration file to avoid confusion between e.g. `build_dir` and `build_path` (only latter matters)
  - fix SuSe hack, only prefix command with sourcing of `/etc/profile.d/modules.sh` if it is there
  - leave build directory before it is removed during cleanup step
  - load generated module file before running test cases

easyblocks

- added **3** new *generic* easyblocks: `CMakePythonPackage`, `JAR`, `RPackage`
- added support for **23** new software packages that require customized support:
  - ACML, ALADIN, ant, Bioconductor (R packages), Chapel, Cufflinks, ESPResSo, FLUENT, Geant4, GHC, Java, NEURON, NWChem, PyZMQ, QuantumESPRESSO, R, Rmpi, ROOT, Rserve, SCOOP, Trinity, VSC-tools, XML
- various enhancements, including:
  - clean up of `python.py` easyblock:
    - merge `EB_DefaultPythonPackage` and `PythonPackage` easyblocks into generic *easyblock* `PythonPackage`, which derives from `ExtensionEasyblock`
    - move `EB_FortranPythonPackage` into dedicated *generic* `FortranPythonPackage` easyblock module
    - split off support for building/installing `nose`, `numpy`, `scipy` into dedicated `EB_*` easyblock modules, which allows them to be built as stand-alone modules as well
  - clean up testing of Python packages (`PythonPackage` easyblock)
  - make sure there is no `site.cfg` in home dir when building Python packages, because e.g. `scipy` will pick it up
  - added support for building Intel MKL wrappers with OpenMPI toolchain
  - cleaning up of fake module that was loaded for running tests

  - move calls to functions that rely on environment up in the chain of steps (mostly for cleanup reasons)
  - use better module name for UFC sanity check, minor change to sanity check paths for UFC
- various bug fixes, including:
  - only call `make ptcheck` for ATLAS when multi-threading support is enabled
  - use a symbolic link for $HOME/intel instead of a randomly suffixed subdirectory in home and patching of Intel install scripts
    * latter does not work anymore with recent versions of Intel tools (2013.x)

easyconfigs

- added example easyconfig files for **48** new software packages:
  - ABINIT, ABySS, ACML, ALADIN, ant, BFAST, BLAST, Chapel, CLHEP, Cufflinks, ESPResSo, GATE, GHC, Geant4, Greenlet, google-sparsehash, grib_api, HMMER, Java, JUnit, libibmad, libibumad, libibverbs, MCL, MDP, MEME, mpiBLAST, NCBI Toolkit, NEURON, NWChem, numpy, MDP, Oger, OpenPGM, paycheck, PyZMQ, QuantumESPRESSO, R, ROOT, SCOOP, scipy, Tophat, Trinity, util-linux, VSC-tools, wiki2beamer, XML, ZeroMQ
- added example easyconfigs for gmacml, gmvapich2, iccifort, ictce, iomkl toolchains
- added additional example easyconfig files for:
  - ATLAS, BLACS, Boost, Bowtie2, bzip2, CP2K, Doxygen, FFTW, GCC, HDF5, hwloc, icc, ifort, imkl, impi, JasPer, Libint, libreadline, libsmm, libxml, ncurses, netCDF, M4, Meep, MVAPICH2, OpenMPI, Python, ScaLAPACK, Szip, zlib
- various enhancements, including:
  - major style cleanup of all example easyconfig file (PEP008 compliance)
  - added setuptools to list of Python extensions
  - get rid of `parallel` versionsuffix for HDF5, as its meaningless (MPI-enabled build is always parallel)
- various 'bug' fixes, i.e. added missing dependencies or replaced OS dependencies with proper dependencies

8.12.37 v1.0.2 (December 8th 2012)

bugfix release

framework

- properly catch failing sanity check commands
- fix bug in toolchain support which cause linking environment variables set by toolchain to include too many libraries
  - elements in toolchain variables were being passed by reference instead of by value
- fix selecting a compiler toolchain for a specified software package (`--software-name`) if only a template is a viable option
- fix passing command line parameters with `--job`
- fix list of valid stops (`-s`/`--stop`)
- fix minor issues in help messages (`-h`/`--help`)

easyblocks

- fix typos in WIEN2k easyblock (missing commas after list elements)

easyconfigs

- fixed source URL for ligtextutils (toolchain refactoring error)

8.12.38 v1.0.1 (November 24th 2012)

bugfix release

framework

- fix support for installing with `distutils` (broken import in `setup.py`)
- fix support for ACML as a toolchain element (`toolchains/linalg/acml.py`)
- add name to aggregated regtest XML so that is parsed correctly by Jenkins
- reorder code in `main.py` so that regtest also works with incomplete easyconfig paths
- add bash script for running regression test and sending a trigger to Jenkins to pull in the XML with results
- get rid of assumption that loaded modules should have name like `foo/bar`, make it more flexible
- retry failed builds in regtest twice to ignore fluke errors
- report leaf nodes in dependency graph when regtest is submitted
  - this is required for setting job dependencies in the regtest script for the Jenkins trigger job
- implement and use rmtree2 as more (NFS) robust replacement for `shutil.rmtree`
- bump max hit count for `run_cmd_qa` from 20 to 50, to make false positives of unasnwered questions less likely

easyblocks

- fix support for installing with `distutils` (broken import in `setup.py`)
- only build GMP/CLooG/PPL libraries during GCC build in parallel, don't install in parallel
  - `make -j N install` doesn't work consistently for GMP
- fix GCC build on OS X
  - location of libraries is slightly different (`lib` vs `lib64` dir)
- add support to `ConfigureMake` easyblock for pre-passing tar options to configure
  - see `tar_config_opts` easyconfig parameter
  - workaround for issue with pax hanging `configure` with an interactive prompt
- enhance Q&A for WRF and WIEN2k by adding entries to `qa` dict and `no_qa` list
- use `rmtree2` from `tools/filetools.py` as more (NFS) robust replacement for `shutil.rmtree`

easyconfigs

- remove patch file for OpenMPI to resolve issue with pax hanging configure
  - `tar_config_opts` should be enabled as needed
- disable parallel build for PAPI, seems to be causing problems

8.12.39 v1.0 (November 13th 2012)

- split up EasyBuild across three repositories: framework, easyblocks and easyconfigs
- packaged EasyBuild, different parts can now be installed easily using `easy_install`

framework

- various changes to both internal and external API:
  - renamed main script to main.py (from `build.py`)
  - file and directory organisation
  - module, class, function and function argument renaming and reorganisation

  - split up `Application` into `EasyBlock` and `ConfigureMake` (see easybuild-easyblocks for the latter)
  - created `EasyConfig` class for handling easyconfig files
  - renaming of EasyBuild configuration parameters (non-camelCase)
  - renaming of various easyconfig parameters (non-camelCase)
  - rename `SOFTROOT` and `SOFTVERSION` environment variables set in generated module files to `EBROOT` and `EBVERSION`
  - use 'extension' as generic terminology for Python packages, R libraries, Perl modules, ...
- added support for building software packages in parallel via PBS jobs
- added unit testing framework and initial set of unit tests for basic functionality
  - and run them in Jenkins continuous integration service, see https://jenkins1.ugent.be/view/EasyBuild/
- implement single-command regression test (e.g. to test building all supported software)
  - eb `--regtest` –robot
- switch to new style Python classes
- replaced `toolkit` module with `toolchain` package (total rewrite), providing modular support for toolchains
- adjust default EasyBuild configuration to only use `$HOME/.local/easybuild` by default
- added support for running EasyBuild without supplying an easyconfig file
  - make EasyBuild search for a matching easyconfig file
  - support automatic rewriting of an existing partially-matching easyconfig file (use this with care!)
  - support for automatically generating an easyconfig file according to given specifications (best effort!)
- add support for looking for easyconfig file in Python search path if it can't be found on specified (relative) path (that way, easyconfig files available in the easybuild-easyconfigs package can be used easily)
- various other enhancements and bug fixes, e.g.:
  - extended sanity check capabilities
  - cleaned up logging
  - creating of devel module which allows to mimic environment that was used by EasyBuild to build the software
  - support for creating dependency graphs for a set of easyconfig files
  - grouped options in help output and categorised available easyconfig parameters
  - more consistent code style

easyblocks

- implement class name encoding scheme, see wiki http://github.com/hpcugent/easybuild/wiki/Encode-class-names
  - (non-generic) easyblock class names are now prefixed with `EB_` and non-alphanumeric characters are escaped
- split off generic easyblocks into separate package `easyblocks.generic`
- added custom support for **39** software packages:
  - Armadillo, BiSearch, Boost, Bowtie2, BWA, bzip2, CGAL, CPLEX, DOLFIN, Eigen, flex, FSL, Hypre, libxml2, MetaVelvet, METIS, MTL4, MUMmer, ncurses, OpenFOAM, OpenSSL, ParMETIS, Pasha, PETSc, Primer3, python-meep, SAMtools, SCOTCH, SHRiMP, SLEPc, SOAPdenovo, SuiteSparse, SWIG, Tornado, Trilinos, UFC, Velvet, WIEN2k, XCrySDen

- various enhancements and bug fixes to existing easyblocks

easyconfigs

- added example easyconfig files for **106** new software packages:
  - AMOS, Armadillo, ASE, Autoconf, BiSearch, Boost, Bowtie2, BWA, byacc, bzip2, CGAL, ClustalW2, CMake, CPLEX, cURL, CVXOPT, Cython, Docutils, DOLFIN, ECore, Eigen, expat, FASTX-Toolkit, FFC, FIAT, freetype, FSL, GEOS, git, glproto, GMP, gmvapich2, gompi, GPAW, GSL, guile, h5py, h5utils, Harminv, hwloc, Hypre, Infernal, Instant, Jinja2, libctl, libdrm, libffi, libgtextutils, libmatheval, libpciaccess, libpthread-stubs, libreadline, libtool, libunistring, libxcb, libxml2, makedepend, matplotlib, Meep, Mercurial, Mesa, MetaVelvet, METIS, MPFR, MTL4, MUMmer, ncurses, OpenFOAM, OpenSSL, ORCA, PAPI, ParMETIS, Pasha, PCRE, PETSc, petsc4py, pkg-config, Primer3, python-meep, RNAz, SAMtools, ScientificPython, SCOTCH, setuptools, Shapely, SHRiMP, SLEPc, SOAPdenovo, Sphinx, SuiteSparse, SWIG, Tcl, Theano, Tk, Tornado, Trilinos, UFC, UFL, Velvet, ViennaRNA, Viper, WIEN2k, xcb-proto, XCrySDen, xorg-macros, xproto
- added additional example easyconfig files (versions, builds) for several software packages
  - Bison, BLACS, Doxygen, flex, GCC, HDF5, icc, ifort, libpng, M4, MVAPICH2, OpenMPI, Szip, tbb, zlib
- replaced GCC/OpenMPI based easyconfig files with equivalents using the *gompi* toolchain * ATLAS, BLACS, FFTW, LAPACK, ScaLAPACK
- enhanced Python example easyconfig files (more dependencies required for features, e.g. libreadline, bzip2, zlib, ...)
- corrected file name of easyconfig files to adher to standard as followed by EasyBuild robot dependency resolver
- style cleanup in existing easyconfig files

8.12.40 v0.8 (June 29th 2012)

- added support for building/installing 17 additional software packages:
  - BEAGLE, Doxygen, g2clib, g2lib, HDF, HDF5, JasPer, libpng, Maple, MrBayes, NCL, netCDF, netCDF-Fortran, Szip, WPS, WRF, zlib
- the build procedure for WRF and WPS includes running the tests available for these packages
- various bug fixes and enhancements:
  - made support for interactive installers (`run_cmd_qa`) more robust
  - fixed Python git package check
  - implemented toolkit functions for determine compiler family, MPI type, MPI run command, ...

8.12.41 v0.7 (June 18th 2012)

- fixed BLACS build
  - diagnostic tools to determine `INTERFACE` and `TRANSCOMM` values are now used
- added support for building Bison, CP2K, flex
  - with regression test enabled for CP2K as part of build process
  - note: BLACS built with EasyBuild prior to v0.7 needs to be rebuilt for CP2K to work correctly
- added `--enable-mpi-threads` to OpenMPI example easyconfigs
  - required for correct CP2K psmp build
- adjusted libsmm example easyconfig for lower build time

  - to make the full regression test finish in a reasonable amount of time
- added script to make porting of easyblocks from old to new EasyBuild codebase easier
  - takes care of refactoring, checks for PyLint warnings/errors, ...
- fixed several small bugs
- prefixed EasyBuild messages with ==
- full regression test passed (58 easyconfigs tested)

8.12.42 v0.6 (May 11th 2012)

- added support for Intel compiler toolkit (ictce)
  - which included the Intel compilers, Intel Math Kernel Library (imkl) and Intel MPI library (impi)
- added support for building Python with nose/numpy/scipy packages
- added simple regression test
- this version is able to build all supplied example easyconfigs

8.12.43 v0.5 (April 6th 2012)

- first public release of EasyBuild
  - after a thorough cleanup of the EasyBuild framework of the in-house version
- supports building HPL with goalf compiler toolkit
  - the goalf toolkit consists of the GCC compilers, and the OpenMPI, ATLAS, LAPACK, FFTW and ScaLAPACK libraries
- also support build and installation of MVAPICH2