

easybuild



## Introduction to EasyBuild

November 13th 2017 - SURFsara, Amsterdam

[http://users.ugent.be/~kehoste/EasyBuild\\_20171113\\_SURFsara.pdf](http://users.ugent.be/~kehoste/EasyBuild_20171113_SURFsara.pdf)

[kenneth.hoste@ugent.be](mailto:kenneth.hoste@ugent.be)

<http://easybuilders.github.io/easybuild>

<http://easybuild.readthedocs.io>



GHENT  
UNIVERSITY

<http://www.ugent.be/hpc>

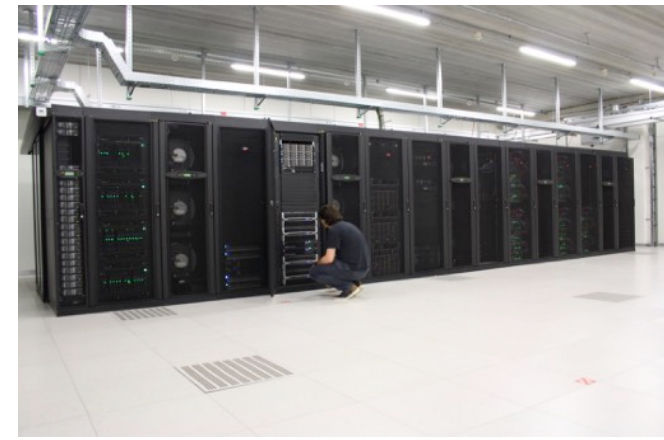
<https://www.vscentrum.be>



# HPC-UGent



- part of central IT department of Ghent University (Belgium)
- centralised scientific computing services, training & support
- for researchers of UGent, industry & knowledge institutes
- member of Flemish Supercomputer Centre (VSC)  
<https://www.vscentrum.be/>
- core values:  
empowerment - centralisation - automation - collaboration





# whoami

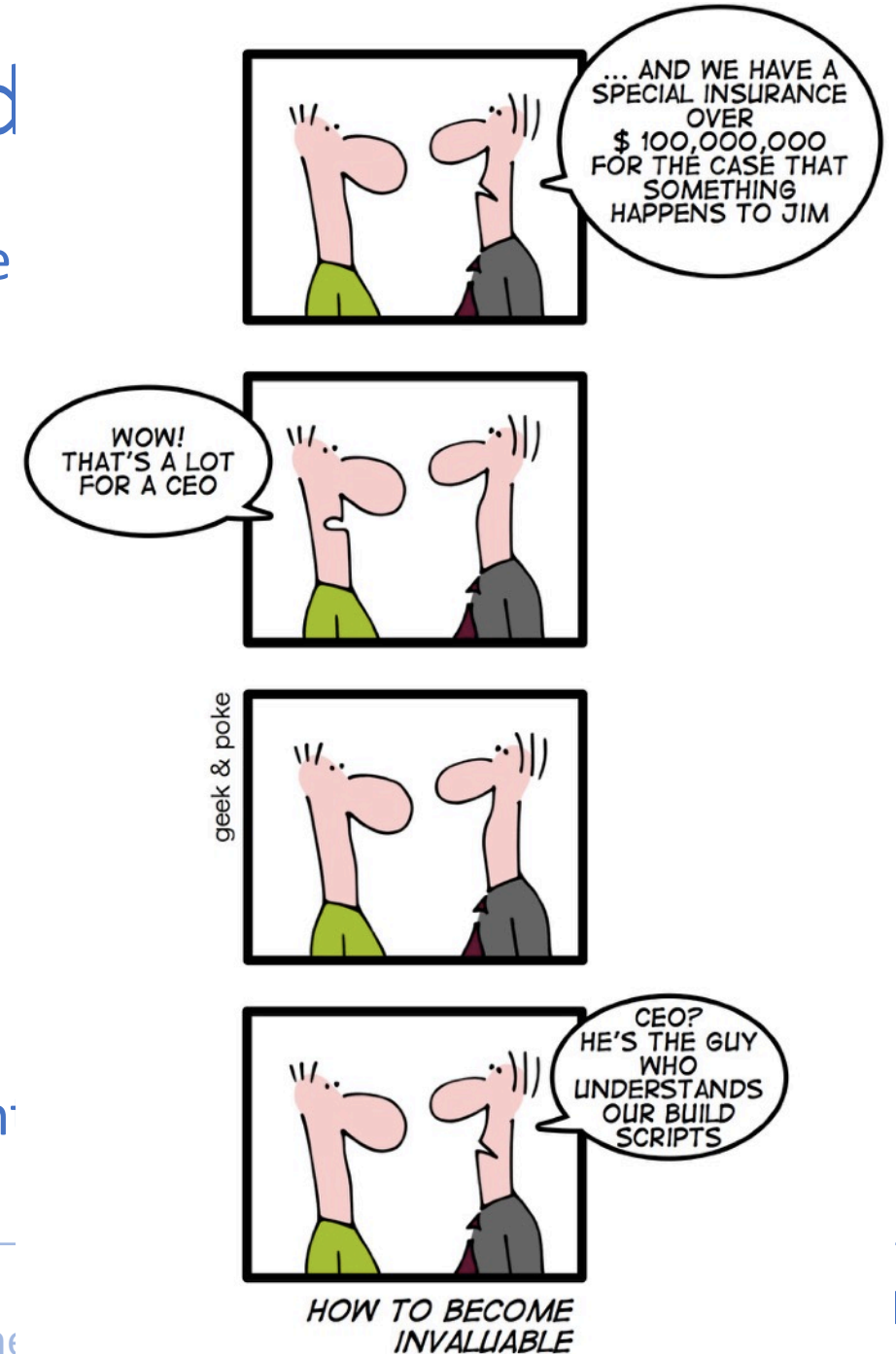
- Masters & PhD in Computer Science from Ghent University (Belgium)
- joined HPC-UGent team in October 2010
- main tasks: user support & training, software installations
- inherited maintenance of EasyBuild in 2011
- slowly also became lead developer & release manager
- big fan of loud music & FOSS (Free & Open Source Software)

kenneth.hoste@ugent.be - *boegel* (GitHub, IRC, Slack) - @kehoste (Twitter)

# Getting Scientific Software Installed

Installation of scientific software is a tremendous problem for HPC sites all around the world.

- ideally built from source (performance is key)
- tedious, time-consuming, frustrating, sometimes simply not worth the (manual) effort...
- huge burden on HPC user support teams
  - over 25% of support tickets at HPC-UGent, but consumes *way* more time...
- very little collaboration among HPC sites (until recent years)



# Common issues with scientific software

Researchers focus on the *science* behind the software they implement, and care little about software engineering, tools, build procedure, portability, ...

Scientists are not software developers or sysadmins (nor should they be).

*“If we would know what we are doing, it would not be called ‘research’.”*

This results in:

- use of non-standard build tools (or broken ones)
- incomplete build procedure, e.g., no configure or install step
- interactive installation scripts
- hardcoded parameters (compilers, libraries, paths, . . . )
- poor/outdated/missing/incorrect documentation



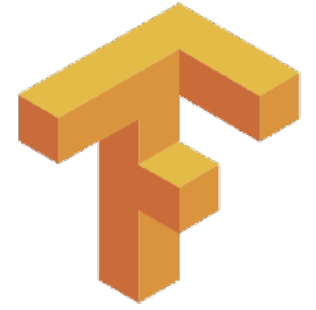
# Prime example 1: WRF



Weather Research and Forecasting Model (<http://www.wrf-model.org>)

- dozen dependencies: netCDF (C, Fortran), HDF5, tcsh, JasPer, ...
- known issues in last release are (only) documented on website
- no patch file provided, infrequent bugfix releases
- interactive 'configure' script, options change between versions
- resulting configuration file 'configure.wrf' needs work:  
fix hardcoded settings (compilers, libraries, . . . ), tweaking of options
- custom 'compile' script (wraps around 'make')  
building in parallel is broken without fixing the Makefile
- no actual installation step

# Prime example 2: TensorFlow



popular open-source software for Deep Learning (<https://www.tensorflow.org/>)

- self-installs most dependencies, but not all... (Python + pkgs, CUDA, cuDNN)
- 'configure' script is a custom interactive script (no autoconf)
  - silent configuration possible, if you know which `$TF_*` environment variables to set (!)
- uses Bazel (<http://bazel.io/>) as build tool (there is a contributed CMake alternative)
  - **resets environment**, may result in unsetting important env vars (e.g., `$INTEL_LICENSE_FILE`)
  - quite different from other build tools; e.g. to build TensorFlow:  

```
bazel build --config=opt //tensorflow/tools/pip_package:build_pip_package
```
  - `--config=opt`, `-c opt` and `-copt=...` all mean different things... `0_o`
- installation via 'pip install' of locally built Python wheel file (.whl)

# What about existing software installation tools?

- package managers: yum (RPMs), apt-get (.deb), ...
- Homebrew (Mac OS X), <http://brew.sh/> ; Linuxbrew, <http://brew.sh/linuxbrew/>
- Portage (Gentoo), <http://wiki.gentoo.org/wiki/Project:Portage>
- pkgsrc (NetBSD & (a lot) more), <http://pkgsrc.org/>

**Existing tools are not well suited to scientific software and HPC systems.**

Common problems:

- usually poor support for old/multiple versions and/or to have builds side-by-side
- not flexible enough to deal with idiosyncrasies of scientific software
- little support for scientific software, non-GCC compilers, MPI, ...

**Modern exceptions (depending on use case): Nix/Guix(?) & Spack (more on Spack later).**



- website: <http://easybuilders.github.io/easybuild/>
- documentation: <http://easybuild.readthedocs.io>
- **framework to build & install scientific software on HPC clusters**
- implemented in Python 2, lead development by HPC-UGent
- Free & Open Source Software (GPLv2)
- supports different compilers & MPI libraries, > **1,300 different software packages**, ...
- **active & helpful worldwide community**

# 5 years of (stable) EasyBuild



- in-house development at HPC-UGent since 2009
  - originally created by Stijn De Weirdt (tech lead at HPC-UGent)
- first public release in April 2012 (EasyBuild v0.5)
- **first stable release on November 13th 2012, during SC'12 (EasyBuild v1.0)**
- initial intention was to get feedback, but gradually a community emerged around it...
- frequent stable releases since then (latest: EasyBuild v3.4.1, Oct 17th 2017)
- development is highly community-driven: bug reports, feature requests, contributions



# Supported software



[http://easybuild.readthedocs.io/en/latest/version-specific/Supported\\_software.html](http://easybuild.readthedocs.io/en/latest/version-specific/Supported_software.html)

- EasyBuild v3.4.1 supports installing **over 1,300 software packages**
  - including CP2K, NAMD, NWChem, OpenFOAM, QuantumESPRESSO, WRF, WPS, ...
  - also a lot of bioinformatics software is supported out of the box...
  - in addition: ~1,000 Python packages, R libraries, Perl modules, X11 libraries, ...
- diverse toolchain support:
  - compilers: GCC, Intel, Clang, PGI, IBM XL, Cray
  - MPI libraries: OpenMPI, Intel MPI, MPICH, MPICH2, MVAPICH2, ...
  - BLAS/LAPACK libraries: Intel MKL, OpenBLAS, ATLAS, ACML, Cray LibSci, ...

# Some EasyBuild terminology



[http://easybuild.readthedocs.io/en/latest/Concepts\\_and\\_Terminology.html](http://easybuild.readthedocs.io/en/latest/Concepts_and_Terminology.html)

- **EasyBuild framework**
  - core of EasyBuild: Python modules & packages
  - provides supporting functionality for building and installing software
- **easyblock**
  - a Python module that serves as a build script, 'plugin' for the EasyBuild framework
  - implements a (generic) software build/install procedure
- **easyconfig file** (\*.eb): build specification; software name/version, compiler toolchain, etc.
- **(compiler) toolchain**: set of compilers with accompanying libraries (MPI, BLAS/LAPACK, . . . )
- **extensions**: additional libraries/packages/modules for a particular applications (e.g., Python, R)

# Feature highlights (1)



- fully **autonomously** building and installing (scientific) software
  - automatic dependency resolution
  - automatic generation of module files (Tcl or Lua syntax)
- thorough **logging** of executed build/install procedure
- **archiving** of easyconfigs and patches
- highly **configurable**, via config files/environment/command line
- **dynamically extendable** with additional easyblocks, toolchains, etc.

# Feature highlights (2)



- support for **custom module naming schemes** (incl. hierarchical)
- **transparency** via support for 'dry run' installation & trace output
- **comprehensively tested**: lots of unit tests, regression testing, ...
- actively developed, frequent stable releases
- **collaboration** between various HPC sites large & small
- worldwide **community**

# What EasyBuild is (not)



EasyBuild is *not*:

- YABT (Yet Another Build Tool); it doesn't replace tools like CMake or make; it wraps around them
- a replacement for your favourite package manager (`yum`, `apt-get`, ...); it leverages some tools & libraries provided by the OS (`glibc`, `OpenSSL`, `libibverbs`, ...)
- a magic solution to all your (software installation) problems...  
you may still run into compiler errors (if nobody else has already taken care of it)

# What EasyBuild is (not)



EasyBuild is:

- a **uniform interface** that wraps around software installation procedures
- a huge **time-saver**, by automating tedious/boring/repetitive tasks
- a way to provide a **consistent software stack** to your users
- an **expert system** for software installation on HPC systems
- a **platform for collaboration** with HPC sites worldwide
- a way to **empower users to self-manage their software stack** on HPC systems

# Example output of using 'eb' command

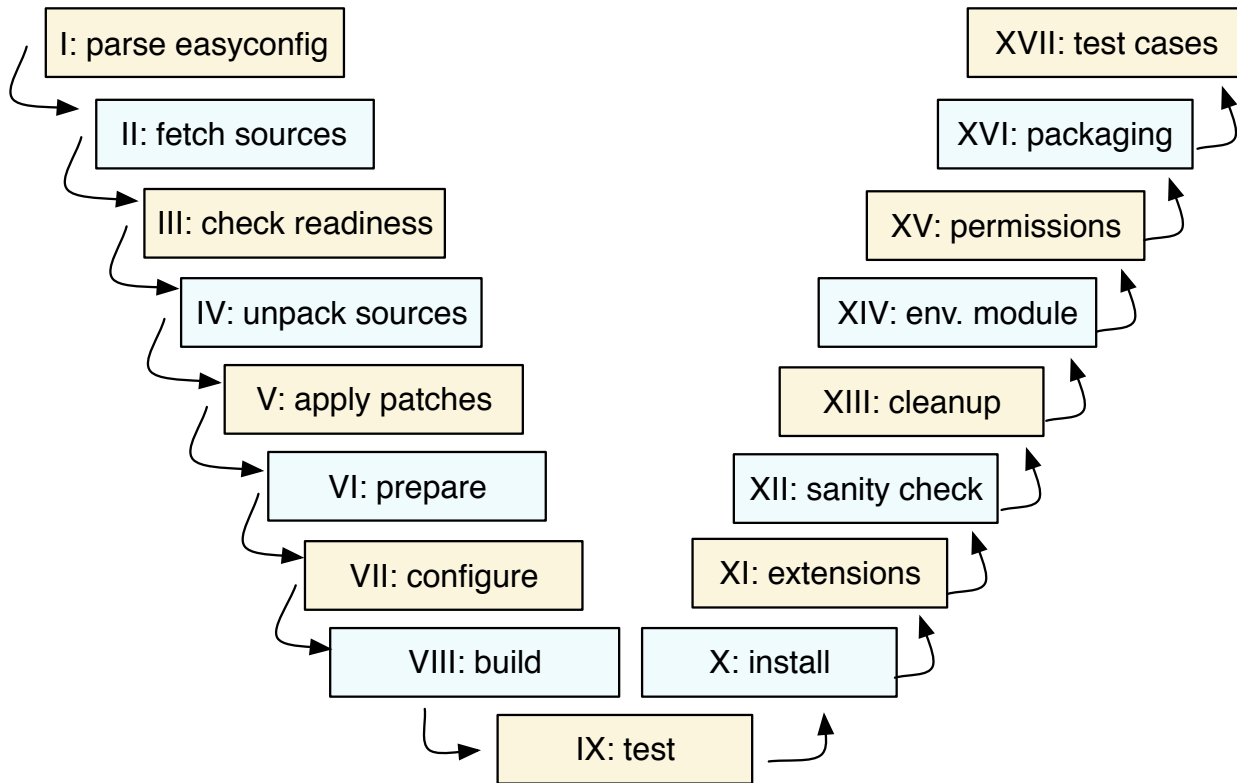


```
$ eb GCC-6.4.0.eb
== temporary log file in case of crash /tmp/eb-GyvPHx/easybuild-U1TkEI.log
== processing EasyBuild easyconfig GCC-6.4.0.eb
== building and installing GCC/6.4.0...
== fetching files...
== creating build dir, resetting environment...
== unpacking...
== patching...
== preparing...
== configuring...
== building...
== testing...
== installing...
== taking care of extensions...
== postprocessing...
== sanity checking...
== cleaning up...
== creating module...
== permissions...
== packaging...
== COMPLETED: Installation ended successfully
== Results of the build can be found in the log file /opt/easybuild/software/GCC/6...
== Build succeeded for 1 out of 1
== Temporary log file(s) /tmp/eb-GyvPHx/easybuild-U1TkEI.log* have been removed.
== Temporary directory /tmp/eb-GyvPHx has been removed.
```

# Step-wise installation procedure



EasyBuild performs a step-wise installation procedure for each software



- download sources (best effort)
- set up build directory & environment
  - unpack sources (& apply patches)
  - load modules for toolchain & deps
  - define toolchain-related env vars (\$CC, \$CFLAGS, ...)
- configure, build, (test), install, (extensions)
- perform a simple sanity check on installation
- generate module file

*each step can be customised via easyconfig parameters or an easyblock*

# Easyconfig files as build specifications



[http://easybuild.readthedocs.io/en/latest/Writing\\_easyconfig\\_files.html](http://easybuild.readthedocs.io/en/latest/Writing_easyconfig_files.html)

- **simple text files including a set of easyconfig parameters** (in Python syntax)
- some are mandatory: software name/version, toolchain, metadata (homepage, descr.)
- other commonly used parameters:
  - easyblock to use
  - list of sources & patches
  - (build) dependencies
  - options for configure/build/install commands
  - files and directories that should be present (sanity check)

# Example easyconfig file



```
name = 'WRF'
version = '3.8.0'

buildtype = 'dmpar' # custom parameter for WRF
versionsuffix = '-' + buildtype # part of module name

homepage = 'http://www.wrf-model.org'
description = "Weather Research and Forecasting (WRF) Model"

toolchain = {'name': 'intel', 'version': '2016b'}

source_urls = ['http://www.mmm.ucar.edu/wrf/src/']
sources = ['%(name)sV%(version_major_minor)s.TAR.gz']
patches = ['WRF-%(version)s_known_problems.patch']

builddependencies = [('tcsh', '6.20.00')]
dependencies = [
    ('JasPer', '2.0.10'),
    ('netCDF', '4.4.1'),
    ('netCDF-Fortran', '4.4.4'),
]
```

# Example easyconfig file



software name and version

```
name = 'WRF'  
version = '3.8.0'  
  
buildtype = 'dmpar' # custom parameter for WRF  
versionsuffix = '-' + buildtype # part of module name
```

software metadata

```
homepage = 'http://www.wrf-model.org'  
description = "Weather Research and Forecasting (WRF) Model"  
  
toolchain = {'name': 'intel', 'version': '2016b'}  
  
source_urls = ['http://www.mmm.ucar.edu/wrf/src/']  
sources = ['%(name)sV%(version_major_minor)s.TAR.gz']  
patches = ['WRF-%(version)s_known_problems.patch']  
  
builddependencies = [('tcsh', '6.20.00')]  
dependencies = [  
    ('JasPer', '2.0.10'),  
    ('netCDF', '4.4.1'),  
    ('netCDF-Fortran', '4.4.4'),  
]
```

# Example easyconfig file



software name and version

```
name = 'WRF'  
version = '3.8.0'  
  
buildtype = 'dmpar' # custom parameter for WRF  
versionsuffix = '-' + buildtype # part of module name
```

software metadata

```
homepage = 'http://www.wrf-model.org'  
description = "Weather Research and Forecasting (WRF) Model"
```

toolchain name & version

```
toolchain = {'name': 'intel', 'version': '2016b'}
```

sources & patches

```
source_urls = ['http://www.mmm.ucar.edu/wrf/src/']  
sources = ['%(name)sV%(version_major_minor)s.TAR.gz']  
patches = ['WRF-%(version)s_known_problems.patch']
```

```
builddependencies = [('tcsh', '6.20.00')]  
dependencies = [  
    ('JasPer', '2.0.10'),  
    ('netCDF', '4.4.1'),  
    ('netCDF-Fortran', '4.4.4'),  
]
```

# Example easyconfig file



software name and version

```
name = 'WRF'  
version = '3.8.0'  
  
buildtype = 'dmpar' # custom parameter for WRF  
versionsuffix = '-' + buildtype # part of module name
```

software metadata

```
homepage = 'http://www.wrf-model.org'  
description = "Weather Research and Forecasting (WRF) Model"
```

toolchain name & version

```
toolchain = {'name': 'intel', 'version': '2016b'}
```

sources & patches

```
source_urls = ['http://www.mmm.ucar.edu/wrf/src/']  
sources = ['%(name)sV%(version_major_minor)s.TAR.gz']  
patches = ['WRF-%(version)s_known_problems.patch']
```

list of (build) dependencies

**note: all versions are *fixed!***

```
builddependencies = [('tcsh', '6.20.00')]  
dependencies = [  
    ('JasPer', '2.0.10'),  
    ('netCDF', '4.4.1'),  
    ('netCDF-Fortran', '4.4.4'),  
]
```

# Example easyconfig file



software name and version	←	<code>name = 'WRF'</code> <code>version = '3.8.0'</code>
build variant (specific to WRF) ( <code>'dmpar'</code> : distributed, MPI)	←	<code>buildtype = 'dmpar' # custom parameter for WRF</code> <code>versionsuffix = '-' + buildtype # part of module name</code>
software metadata	←	<code>homepage = 'http://www.wrf-model.org'</code> <code>description = "Weather Research and Forecasting (WRF) Model"</code>
toolchain name & version	←	<code>toolchain = {'name': 'intel', 'version': '2016b'}</code>
sources & patches	←	<code>source_urls = ['http://www.mmm.ucar.edu/wrf/src/']</code> <code>sources = ['%(name)sV%(version_major_minor)s.TAR.gz']</code> <code>patches = ['WRF-%(version)s_known_problems.patch']</code>
list of (build) dependencies <b>note: all versions are <i>fixed!</i></b>	←	<code>builddependencies = [('tcsh', '6.20.00')]</code> <code>dependencies = [</code> <code>('JasPer', '2.0.10'),</code> <code>('netCDF', '4.4.1'),</code> <code>('netCDF-Fortran', '4.4.4'),</code> <code>]</code>

# Example easyconfig file

no easyblock specified, which implies using a software-specific easyblock (EB\_WRF)



software name and version	<code>name = 'WRF'</code> <code>version = '3.8.0'</code>
build variant (specific to WRF) ( <code>'dmpar'</code> : distributed, MPI)	<code>buildtype = 'dmpar' # custom parameter for WRF</code> <code>versionsuffix = '-' + buildtype # part of module name</code>
software metadata	<code>homepage = 'http://www.wrf-model.org'</code> <code>description = "Weather Research and Forecasting (WRF) Model"</code>
toolchain name & version	<code>toolchain = {'name': 'intel', 'version': '2016b'}</code>
sources & patches	<code>source_urls = ['http://www.mmm.ucar.edu/wrf/src/']</code> <code>sources = ['%(name)sV%(version_major_minor)s.TAR.gz']</code> <code>patches = ['WRF-%(version)s_known_problems.patch']</code>
list of (build) dependencies <b>note: all versions are <i>fixed!</i></b>	<code>builddependencies = [('tcsh', '6.20.00')]</code> <code>dependencies = [</code> <code>('JasPer', '2.0.10'),</code> <code>('netCDF', '4.4.1'),</code> <code>('netCDF-Fortran', '4.4.4'),</code> <code>]</code>

# 5 highlighted features of EasyBuild



- highly configurable and extendable
- integration with Lmod
- "extended dry run" mode
- integration with GitHub
- the EasyBuild community



# Configuring and extending EasyBuild



- configuring can be done via config files, environment and/or command line options
  - *all* configuration options are supported on all 3 levels
  - CLI overrides environment which overrides config files (which override default settings)
  - <http://easybuild.readthedocs.io/en/latest/Configuration.html>
- easy to extend EasyBuild:
  - (also) use own easyconfigs repositories via `--robot-paths`
  - add additional easyblocks, toolchains, module naming schemes via `--include-*`
  - [http://easybuild.readthedocs.io/en/latest/Including\\_additional\\_Python\\_modules.html](http://easybuild.readthedocs.io/en/latest/Including_additional_Python_modules.html)

# Inspecting current configuration



<http://easybuild.readthedocs.io/en/latest/Configuration.html>

Use 'eb --show-config' to get an overview of the current configuration.

```
$ EASYBUILD_PREFIX=/tmp eb --buildpath /dev/shm --show-config
#
# Current EasyBuild configuration
# (C: command line argument, D: default value, E: environment variable, F: configuration file)
#
buildpath      (C) = /dev/shm
installpath    (E) = /tmp
packagepath    (E) = /tmp/packages
prefix         (E) = /tmp
repositorypath (E) = /tmp/ebfiles_repo
robot-paths    (D) = /home/example/easybuild-easyconfigs/easybuild/easyconfigs
sourcepath     (E) = /tmp/sources
```

# Integration with Lmod



- support for using Lmod as modules tool was added in July 2013
- kind of out of necessity...
  - installing lots of modules was (too) easy using EasyBuild
  - traditional Tcl-based modules tool was too slow (no module cache, hierarchy support, ...)
- later also:
  - support for using a hierarchical module naming scheme (July 2014)
  - support for module files in Lua syntax (April 2015)
- Lmod (& Lua syntax) became the default in EasyBuild v3.0.0 (November 2016)

# Integration with Lmod



- synergy between Lmod & EasyBuild has grown over time
  - communities engaging with each other
  - increased adoption for both tools thanks to integration
  - (significant) enhancements to both tools inspired by or thanks to other community
    - improved performance, Lmod's `update_lmod_system_cache_files` script, ...
  - joint papers on "Modern Scientific Software Management"
  - lead developers became good friends :)

# Flat module naming scheme



*legend*

(not available)

(available)

(loaded)

- all modules are always available for loading
- long(er) module names, 'module avail' may be overwhelming for users
- too easy to load incompatible modules together

GCC/5.3.0

GCC/6.1.0

OpenMPI/1.10.2-GCC-5.3.0

OpenMPI/2.1.0-GCC-5.3.0

OpenMPI/1.10.3-GCC-6.1.0

OpenMPI/2.1.0-GCC-6.1.0

FFTW/3.3.4-gompi-2016.04

FFTW/3.3.6-gompi-2016.04

FFTW/3.3.4-gompi-2016.07

FFTW/3.3.6-gompi-2016.07

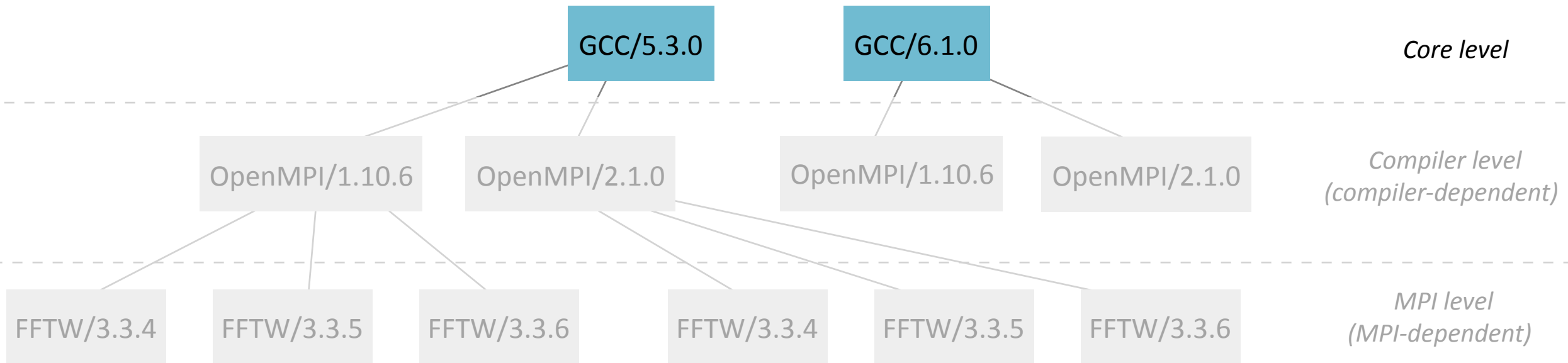
# Hierarchical module naming scheme (1)



legend



- modules are organised in a tree-like fashion
- initially, only 'core' modules are available for loading (typically: Core - Compiler - MPI)
- other modules are only visible via 'module spider'



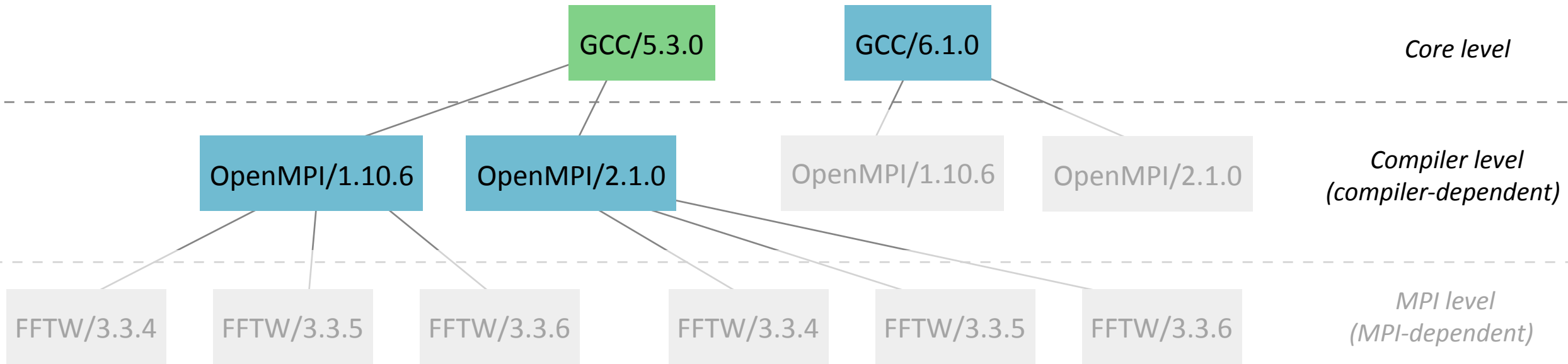
# Hierarchical module naming scheme (2)



legend



- Core modules may extend \$MODULEPATH with an additional location
- loading a Core module may make more modules available
- in this example, loading a GCC module makes OpenMPI modules available



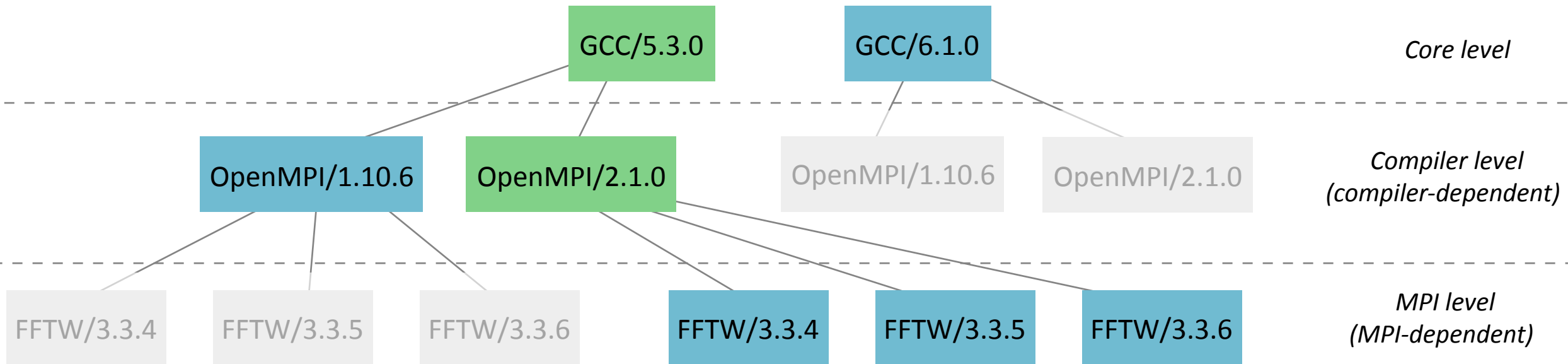
# Hierarchical module naming scheme (3)



legend



- even more modules may be made available by loading other modules
- for example, loading an OpenMPI modules reveals MPI-dependent modules
- EasyBuild can organise modules in hierarchy for you!



# Transparency of performed install procedure



[http://easybuild.readthedocs.io/en/latest/Extended\\_dry\\_run.html](http://easybuild.readthedocs.io/en/latest/Extended_dry_run.html)

- 'eb --extended-dry-run' (or 'eb -x') reveals planned installation procedure
- **runs in a matter of seconds**
- shows commands that will be executed, build environment, generated module file, ...
- any errors that occur in used easyblock are ignored but clearly reported
- not 100% accurate since easyblock may require certain files to be present, etc.
- very useful when debugging easyblocks, instant feedback as a first pass
- implementation motivated by requests from the community
- helps to avoid impression that EasyBuild is a magic black box for installing software

# Example output of `--extended-dry-run` (1)



```
$ eb WRF-3.8.0-intel-2016b-dmpar.eb -x
```

```
== temporary log file in case of crash /tmp/eb-Dh1wOp/easybuild-0bu9u9.log
```

```
== processing EasyBuild easyconfig /home/example/eb/easybuild-easyconfigs/easybuild/easyconfigs/w/WRF/WRF-3.8.0-intel-2016b-dmpar.eb
```

```
...
```

```
*** DRY RUN using 'EB_WRF' easyblock (easybuild.easyblocks.wrf @ /home/example/eb/easybuild-easyblocks/easybuild/easyblocks/w/wrf.py) ***
```

```
== building and installing WRF/3.8.0-intel-2016b-dmpar...  
fetching files... [DRY RUN]
```

```
[fetch_step method]
```

```
Available download URLs for sources/patches:
```

- \* [http://www2.mmm.ucar.edu/wrf/src//\\$source](http://www2.mmm.ucar.edu/wrf/src//$source)
- \* [http://www.mmm.ucar.edu/wrf/src//\\$source](http://www.mmm.ucar.edu/wrf/src//$source)

```
List of sources:
```

- \* WRFV3.8.0.TAR.gz will be downloaded to /home/example/eb/sources/w/WRF/WRFV3.8.0.TAR.gz

# Example output of `--extended-dry-run` (2)



```
$ eb WRF-3.8.0-intel-2016b-dmpar.eb -x
...

building... [DRY RUN]

[build_step method]
  running command "tcsch ./compile -j 4 wrf"
  (in /home/example/eb/software/WRF/3.8.0-intel-2016b-dmpar/WRF-3.8.0)
  running command "tcsch ./compile -j 4 em_real"
  (in /home/example/eb/software/WRF/3.8.0-intel-2016b-dmpar/WRF-3.8.0)
  running command "tcsch ./compile -j 4 em_b_wave"
  (in /home/example/eb/software/WRF/3.8.0-intel-2016b-dmpar/WRF-3.8.0)
...

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

# Example output of `--extended-dry-run` (3)



```
$ eb WRF-3.8.0-intel-2016b-dmpar.eb -x
```

```
...
```

```
[make_module_step method]
```

```
Generating module file /home/example/eb/modules/all/WRF/3.8.0-intel-2016b-dmpar,  
with contents:
```

```
#!/Module  
proc ModulesHelp { } {  
    puts stderr { The Weather Research and Forecasting (WRF) Model }  
}  
module-whatis {Description: WRF - Homepage: http://www.wrf-model.org}  
  
set root /home/example/eb/software/WRF/3.8.0-intel-2016b-dmpar  
  
conflict WRF  
  
if { ![ is-loaded intel/2016b ] } {  
    module load intel/2016b  
}  
if { ![ is-loaded Jasper/1.900.1-intel-2016b ] } {  
    module load Jasper/1.900.1-intel-2016b  
}
```

# Integration with GitHub



[http://easybuild.readthedocs.io/en/latest/Integration\\_with\\_GitHub.html](http://easybuild.readthedocs.io/en/latest/Integration_with_GitHub.html)

- contributing to EasyBuild required sufficient experience with Git
  - + following EasyBuild policy for pull requests: target `develop` branch, PR titles, ...
- this was a limiting factor, not all EasyBuilders are familiar enough with Git
- process can also be quite time-consuming when contributing frequently
- weekend project:
  - leverage `GitPython` and GitHub API to automate contributions
  - enhance 'eb' command line interface with `--new-pr`, `--update-pr`

# Integration with GitHub



[http://easybuild.readthedocs.io/en/latest/Integration\\_with\\_GitHub.html](http://easybuild.readthedocs.io/en/latest/Integration_with_GitHub.html)

- contribution process (manual):
  - `git checkout develop`
  - `git checkout -b my_branch`
  - `git add <easyconfig files>`
  - `git commit -m "my awesome stuff"`
  - `git push origin my_branch`
  - visit `github.com/my_account`
  - issue pull request for `my_branch` to `easybuilders:develop`

# Integration with GitHub



[http://easybuild.readthedocs.io/en/latest/Integration\\_with\\_GitHub.html](http://easybuild.readthedocs.io/en/latest/Integration_with_GitHub.html)

- contribution process (fully automated):

```
eb --new-pr <easyconfig files>
```

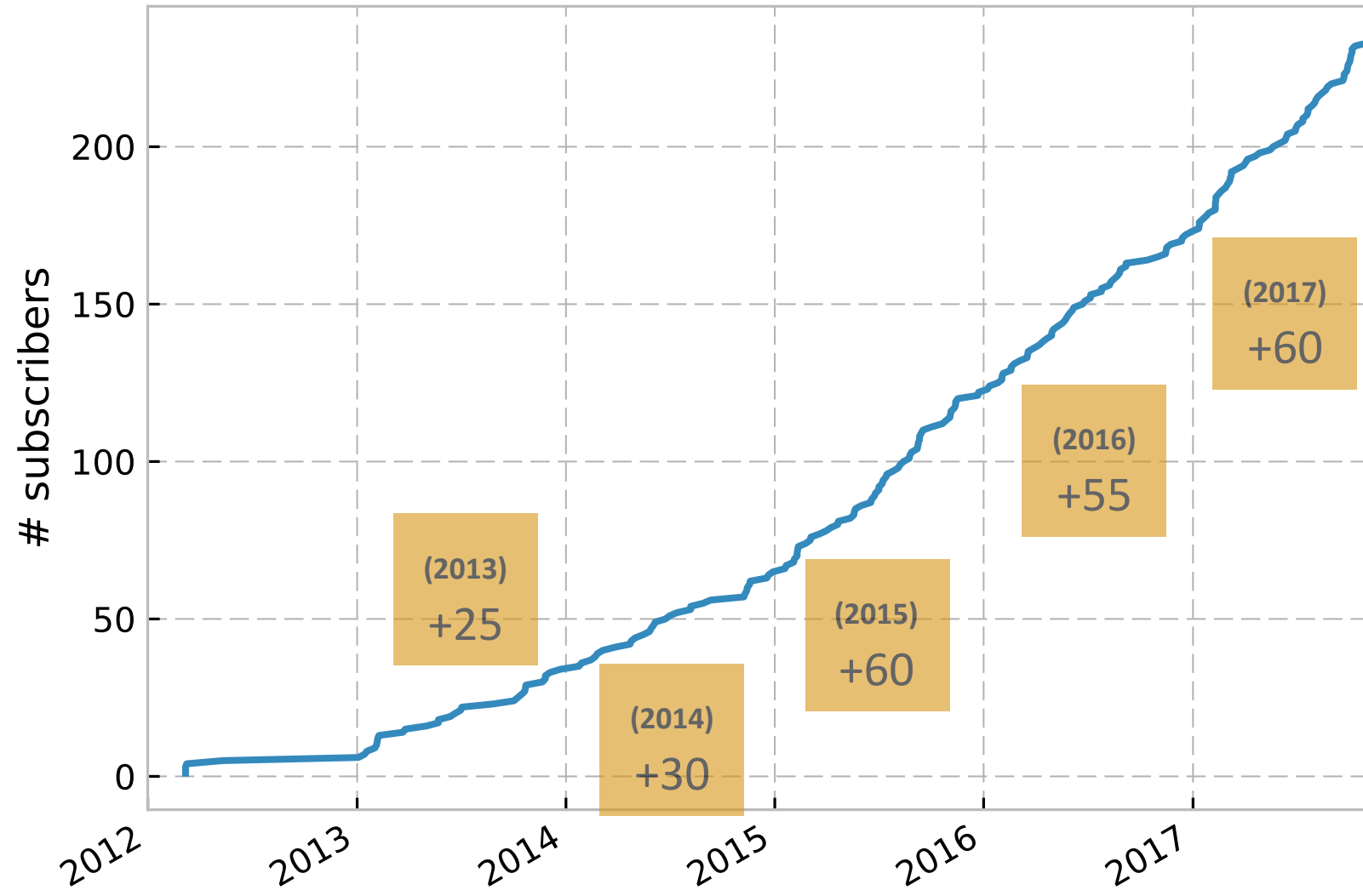
- 4-5 `git` commands + handful of mouse clicks reduced to a single `eb` command!
- to update an existing pull request: `eb --update-pr <PR#> ...`
- significantly lowers the hurdle for contributors: no Git or GitHub experience required
- huge time-saver even for experienced contributors (+ reduced local branches clutter)
- optional: submit test report with  
`eb --from-pr <PR#> --force --robot --upload-test-report`

# The EasyBuild community

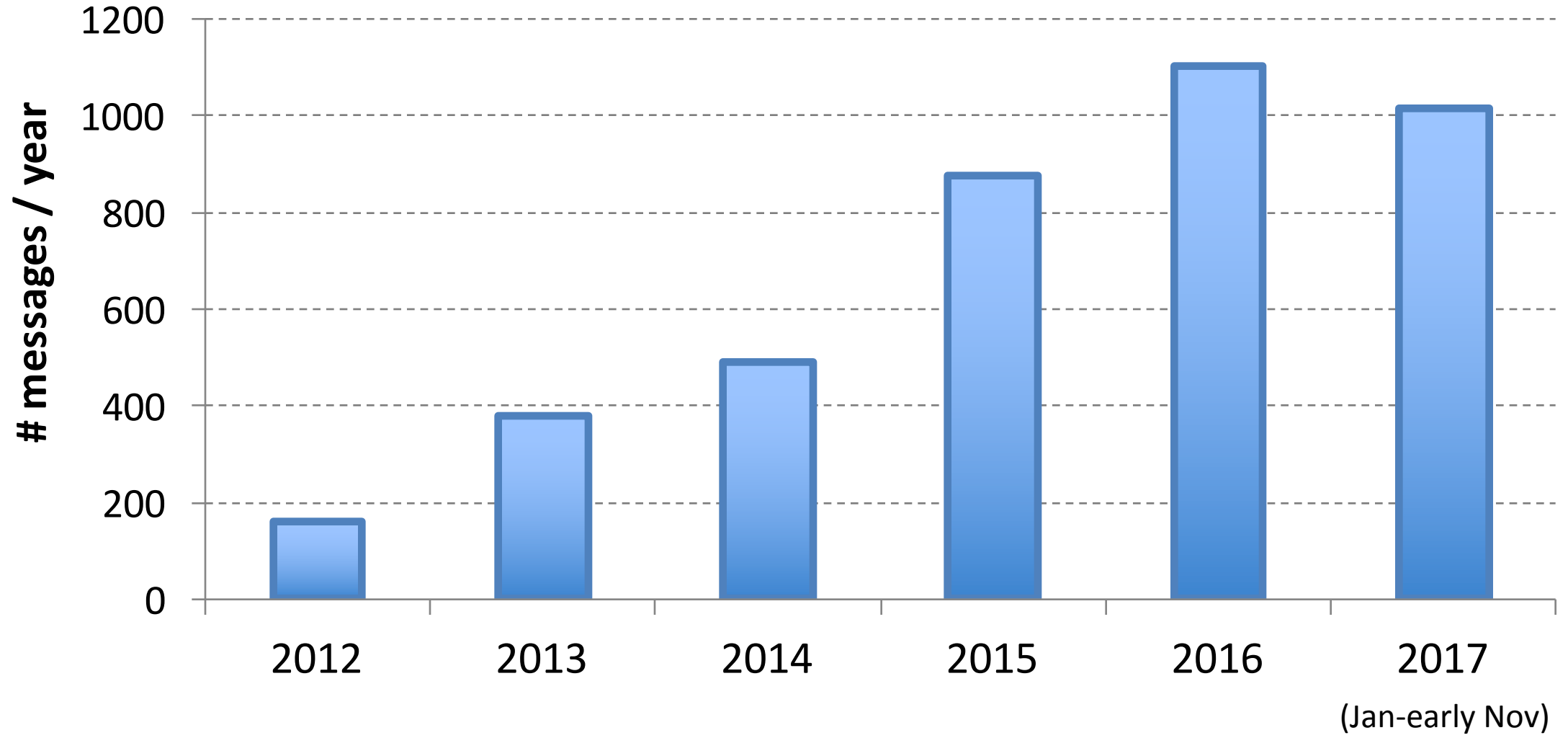


- EasyBuild community has been growing rapidly the last couple of years
- **hundreds of HPC sites and companies worldwide, incl. JSC, CSCS, SURFsara, Pfizer, Bayer, ...**
- very welcoming & supportive to newcomers
- significant overlap between EasyBuild & Lmod communities
- active mailing list: <https://lists.ugent.be/wws/info/easybuild>
- active IRC (`#easybuild` on FreeNode) & Slack channel (`easybuild.slack.com`)
- users are also contributing: bug reports, feature requests, code contributions, ...

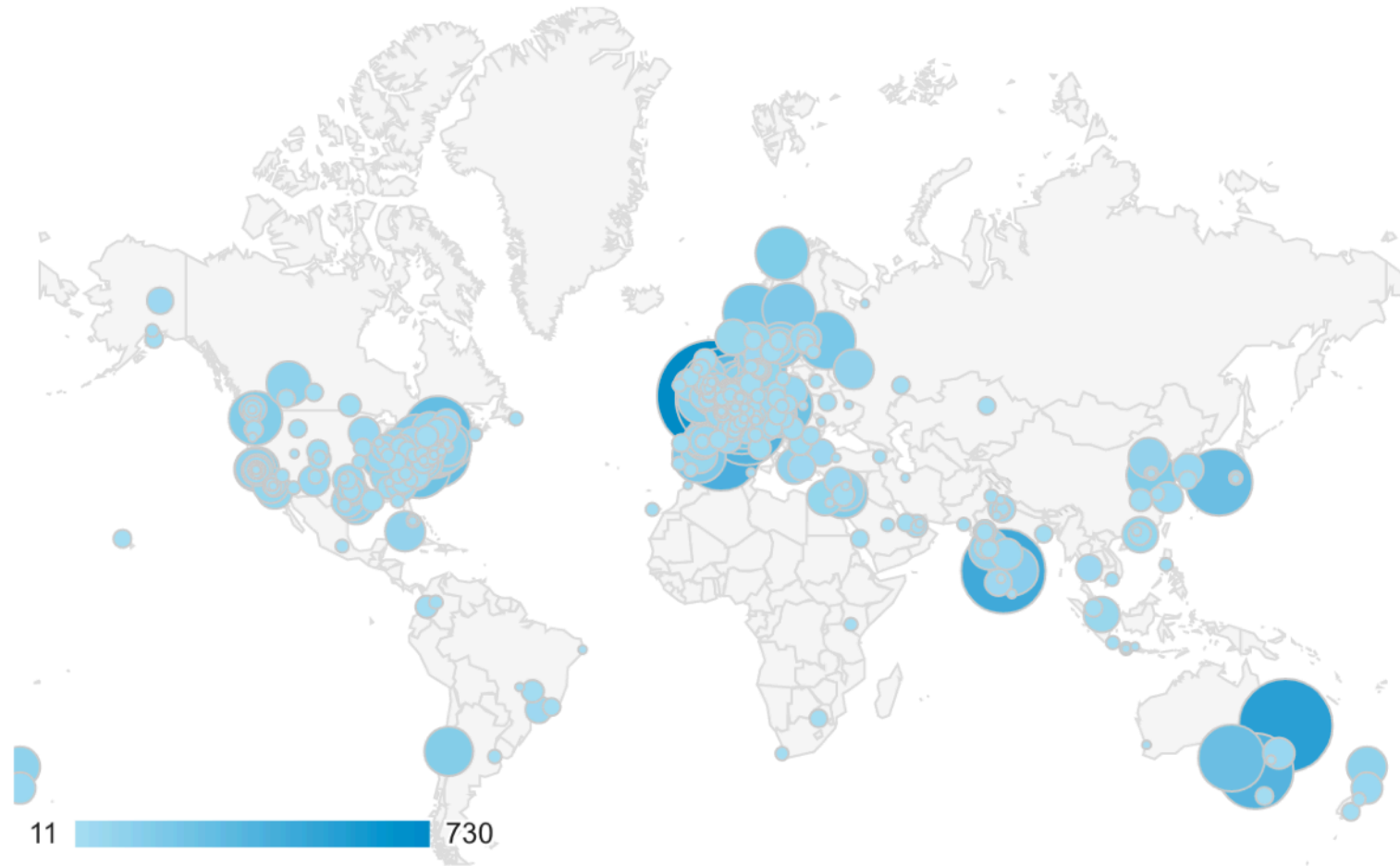
# EasyBuild mailing list (subscribers)



# EasyBuild mailing list (traffic)

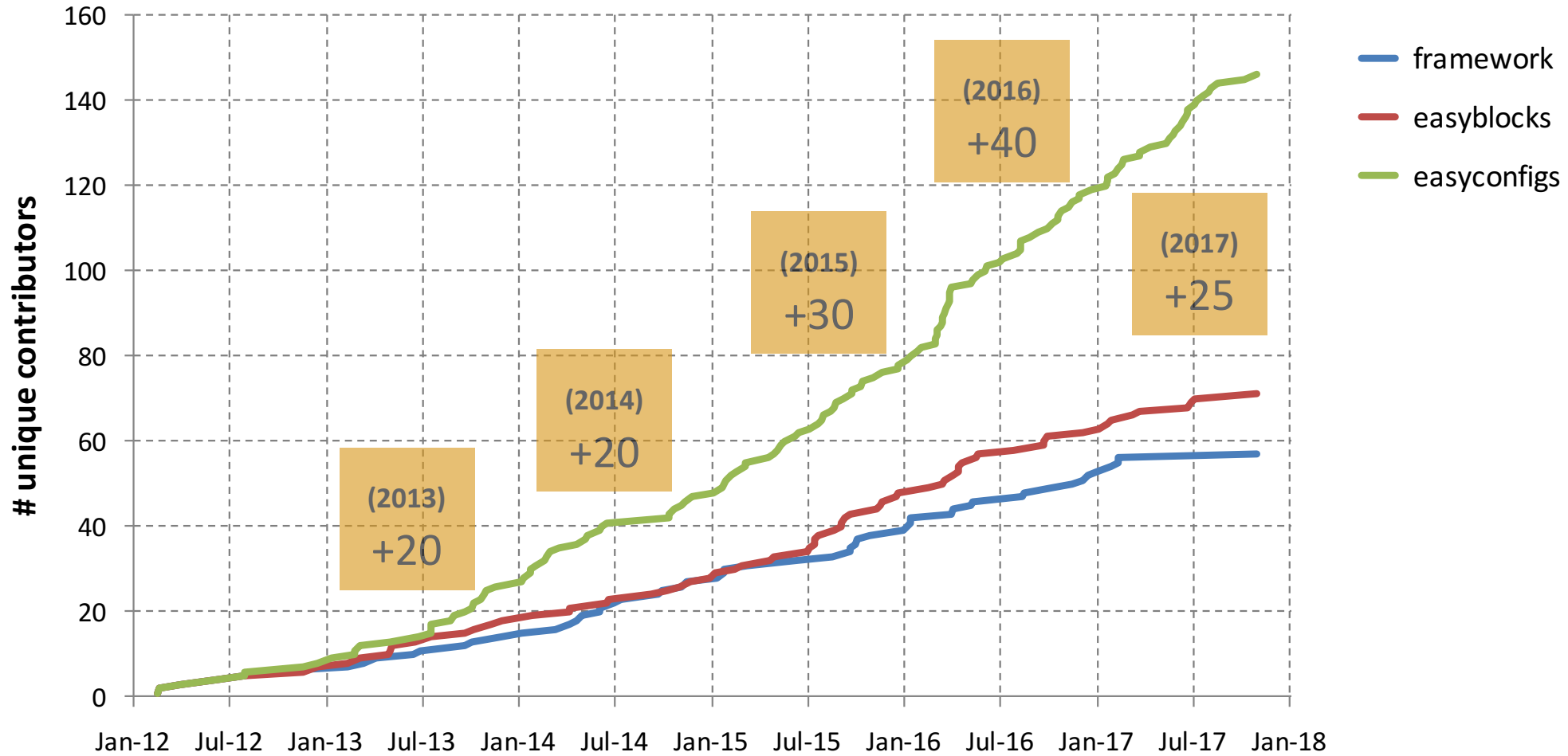


# The sun never sets on EasyBuild...



cities from where EasyBuild documentation was visited at least 10 times during the last year  
source: Google Analytics for [easybuild.readthedocs.io](https://easybuild.readthedocs.io)

# Contributors, contributors, contributors!



**Team of 8 maintainers with admin privileges in EasyBuild repositories since July 2017.**

# Community (common) toolchains



<http://easybuild.readthedocs.io/en/latest/Common-toolchains.html>

- `intel` and `foss`<sup>1</sup> toolchains are most commonly used in EasyBuild community
- helps to focus efforts of HPC sites using one or both of these toolchains
- updated twice a year, clear versioning scheme: `<year>{a,b}` (2016b, 2017a, ...)
- latest version:
  - `foss/2017b`  
binutils 2.28, GCC 6.4.0, OpenMPI 2.1.1, OpenBLAS 0.2.20\* (+ (Sca)LAPACK), FFTW 3.3.6
  - `intel/2017b`  
binutils 2.28 + GCC 6.4.0 as base  
Intel compilers 2017.4.196, Intel MPI 2017.3.196, Intel MKL 2017.3.196

(1) FOSS: Free and Open Source Software

# Other features that were not covered...



<http://easybuild.readthedocs.io>

- letting users manage their software stack on top of centrally provided modules
- installing hidden modules, hiding certain dependencies & toolchains
- support for using RPATH linking (*experimental*)
- partial installations: only (re)generate module file, install additional extensions
- tracing progress via 'eb --trace'
- submitting installations as jobs to an HPC cluster via --job
- creating packages (RPMs, ...) for software installations done with EasyBuild
- using EasyBuild on Cray systems, integration with Cray Programming Environment

# Future work



- **stable support for RPATH (95% done)**
- SHA256 checksums for sources in all easyconfigs included in release (WIP)
- ~~automated style checking of (easyconfig) pull requests~~
- querying of latest available version of a particular software package (via anitya?)
- slightly more flexibility w.r.t. dependency versions (e.g., allow "CMake 3.\*")
- getting rid of dependency on `setuptools` (too much of a PITA...)
- support for using EasyBuild on top of Python 3
- easyconfig files in YAML syntax (.yeb): work in progress...

# Papers on EasyBuild (& Lmod)



## **Modern Scientific Software Management Using EasyBuild and Lmod**

Markus Geimer (JSC), Kenneth Hoste (HPC-UGent), Robert McLay (TACC)

[http://easybuilders.github.io/easybuild/files/hust14\\_paper.pdf](http://easybuilders.github.io/easybuild/files/hust14_paper.pdf)

## **Making Scientific Software Installation Reproducible On Cray Systems Using EasyBuild**

Petar Forai (IMP), Guilherme Peretti-Pezzi (CSCS), Kenneth Hoste (HPC-UGent)

[https://cug.org/proceedings/cug2016\\_proceedings/includes/files/pap145.pdf](https://cug.org/proceedings/cug2016_proceedings/includes/files/pap145.pdf)

## **Scientific Software Management in Real Life: Deployment of EasyBuild on a Large Scale System**

Damian Alvarez, Alan O’Cais, Markus Geimer (JSC), Kenneth Hoste (HPC-UGent)

<http://easybuilders.github.io/easybuild/files/eb-jsc-hust16.pdf>



- requires Python 2(.6), GPLv2
- targeted to HPC user support teams
- stable releases since Nov'12
- supports >1.2k tools/apps/libs + ~1k extensions
- fixed dependency/toolchain versions
- *requires* a modules tool (Lmod/Tmod)
- framework + easyblocks (.py) + easyconfigs (.eb)
- can do both `$LD_LIBRARY_PATH` & `RPATH`
- supports Linux HPC systems (incl. Cray)
- free stickers!

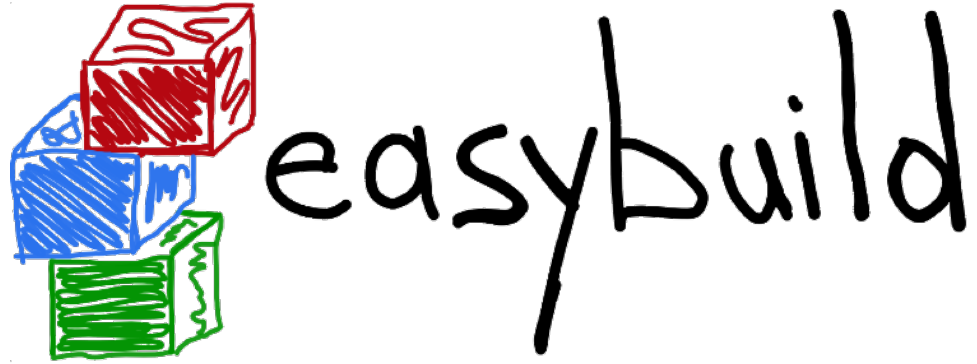
VS



<http://spack.rtd.io/>

- Python 2 & Python 3 compatible, LGPL
- (mainly) for scientific software developers
- alpha software (no stable release yet)
- supports >1.5k software packages (incl. exts)
- very flexible dependency management
- compatible with Lmod/Tmod, dotkit
- core library + packages (.py)
- only does `RPATH`
- supports Linux, macOS, Cray, ...
- free stickers since SC17

Both are backed by an active & supportive community!



## Questions?

[kenneth.hoste@ugent.be](mailto:kenneth.hoste@ugent.be)

<http://easybuilders.github.io/easybuild>

[easybuild@lists.ugent.be](mailto:easybuild@lists.ugent.be)

<http://easybuild.readthedocs.io>

EasyBuild Slack channel (join via <https://easybuild-slack.herokuapp.com/>)



**GHENT  
UNIVERSITY** <http://www.ugent.be/hpc>



<https://www.vscentrum.be>

Vlaams Supercomputer Centrum

(disorganised set of)  
**EXTRA SLIDES**

# EasyBuild requirements



<http://easybuild.readthedocs.io/en/latest/Installation.html#requirements>

- focus is on Linux x86\_64 HPC systems
  - RedHat-based (CentOS, SL, ...), also on Debian and SuSE derivatives
  - also (kind of) works on macOS, but not a target platform
  - Windows support only via new Windows Subsystem for Linux (WSL)
  - ongoing efforts to enhance support on AARCH64 (ARM) and PowerLinux
- Python 2.6 or more recent Python 2, incl. setuptools (no Python 3 support yet)
- system C/C++ compiler (GCC), used to kickstart installation of compiler toolchain
- an environment modules tool, Lmod is highly recommended!



# Installing EasyBuild



<http://easybuild.readthedocs.io/en/latest/Installation.html>

- installing Python packages can be messy, cfr. plethora of Python installation tools
- **bootstrap script** for EasyBuild was created out of frustration

```
# download bootstrap script from
# https://raw.githubusercontent.com/easybuilders/easybuild-framework/develop/easybuild/scripts/bootstrap_eb.py

$ python bootstrap eb.py $PREFIX
$ module use $PREFIX/modules/all
$ module load EasyBuild
```

- standard installation tools like `easy_install`, `pip`, etc. work too
- packaging efforts under way (see OpenHPC; WIP in Fedora)

# Updating EasyBuild



<http://easybuild.readthedocs.io/en/latest/Installation.html#updating-an-existing-easybuild-installation>

- new releases are fully backwards-compatible (except for new major versions)
- to update EasyBuild:
  - re-bootstrap to obtain a module for latest EasyBuild release
  - or use 'eb --install-latest-eb-release' to install module for latest release
  - install a specific EasyBuild version with EasyBuild using an easyconfig file

<https://github.com/easybuilders/easybuild-easyconfigs/tree/develop/easybuild/easyconfigs/e/EasyBuild>

# Configuring EasyBuild (1)



<http://easybuild.readthedocs.io/en/latest/Configuration.html>

You should configure EasyBuild to your preferences, via:

- configuration file(s): key-value lines, text files (e.g., `prefix=/tmp`)
- environment variables (e.g., `$EASYBUILD_PREFIX` set to `/tmp`)
- command line parameters (e.g., `--prefix=/tmp`)

**Each configuration option can be set on each of these configuration levels.**

Priority among different options: cmdline, env vars, config file. For example:

- `--prefix` overrides `$EASYBUILD_PREFIX`
- `$EASYBUILD_PREFIX` overrides `prefix` in configuration file

# Configuring EasyBuild (2)



<http://easybuild.readthedocs.io/en/latest/Configuration.html>

By default, EasyBuild will (ab)use `$HOME/.local/easybuild`.

**Use 'eb --show-config' to get an overview of the current configuration.**

```
$ EASYBUILD_PREFIX=/tmp eb --buildpath /dev/shm --show-config
#
# Current EasyBuild configuration
# (C: command line argument, D: default value, E: environment variable, F: configuration file)
#
buildpath      (C) = /dev/shm
installpath    (E) = /tmp
packagepath    (E) = /tmp/packages
prefix         (E) = /tmp
repositorypath (E) = /tmp/ebfiles_repo
robot-paths    (D) = /Users/kehoste/work/easybuild-easyconfigs/easybuild/easyconfigs
sourcepath     (E) = /tmp/sources
```

# Contributing to EasyBuild



<http://easybuild.readthedocs.io/en/latest/Contributing.html>

EasyBuild has improved significantly thanks to the community.

**You too can contribute back, by:**

- sending feedback
- reporting bugs
- joining the discussion (mailing lists, EasyBuild conf calls)
- sharing suggestions and ideas for changes & additional features
- contributing easyconfigs, enhancing easyblocks, adding support for new software...
- extending & enhancing documentation

# Adding support for additional software



- for each software installation, an **easyconfig file** is required
  - defines easyconfig parameters that specify to EasyBuild what to install, and how
  - existing easyconfig files can serve as examples
  - for version or toolchain updates, a tweaked easyconfig can be *generated* via `eb --try-*`
  - see [http://easybuild.readthedocs.io/en/latest/Writing\\_easyconfig\\_files.html](http://easybuild.readthedocs.io/en/latest/Writing_easyconfig_files.html)
- for 'standard' installation procedures, a **generic easyblock** can be used
  - generic installation can be controlled where needed via easyconfig parameters
- for custom installation procedures, a **software-specific easyblock** is must be implemented
  - see <http://easybuild.readthedocs.io/en/latest/Implementing-easyblocks.html>

# Common generic easyblocks



[http://easybuild.readthedocs.io/en/latest/version-specific/generic\\_easyblocks.html](http://easybuild.readthedocs.io/en/latest/version-specific/generic_easyblocks.html)

- **ConfigureMake**  
standard `./configure` - `make` - `make install` installation procedure
- **CMakeMake**  
same as ConfigureMake, but using *CMake* for configuring
- **PythonPackage**  
installing Python packages (`python setup.py install`, `pip install`, ...)
- **MakeCp**  
no (standard) configuration step, build with `make`, install by copying binaries/libraries
- **Tarball**: just unpack sources and copy everything to installation directory
- **Binary**: run binary installer (specified via `install_cmd` easyconfig parameter)

# Easyconfig files vs easyblocks



<http://easybuild.readthedocs.io/en/latest/Implementing-easyblocks.html>

- thin line between using custom easyblock and 'fat' easyconfig with generic easyblock
- custom easyblocks are "do once and forget", **central solution to build peculiarities**
- reasons to consider implementing a software-specific easyblock include:
  - 'critical' values for easyconfig parameters required to make installation succeed
  - toolchain-specific aspects of the build and installation procedure (e.g., configure options)
  - interactive commands that need to be run
  - custom (configure) options for dependencies
  - having to create or adjust specific (configuration) files
  - 'hackish' usage of a generic easyblock

# Log files



<http://easybuild.readthedocs.io/en/latest/Logfiles.html>

- **EasyBuild thoroughly logs executed installation procedure**
  - active EasyBuild configuration
  - easyconfig file that was used
  - modules that were loaded + resulting changes to environment
  - defined environment variables
  - output of executed commands
  - informative log messages produced by easyblock
- log is file is copied to software installation directory for future reference
- can be used to debug build problems or see how installation was performed exactly

# Log files: example



```
== 2016-04-24 13:34:31,906 main.EB_HPL INFO This is EasyBuild 3.1.2 (framework:
3.1.2, easyblocks: 3.1.2) on host example.
...
== 2016-04-24 13:34:35,503 main.EB_HPL INFO configuring...
== 2016-04-24 13:34:48,817 main.EB_HPL INFO Starting configure step
...
== 2016-04-24 13:34:48,823 main.EB_HPL INFO Running method configure_step part of
step configure
...
== 2016-04-24 13:34:48,823 main.run DEBUG run_cmd: running cmd /bin/bash
make_generic (in /tmp/user/easybuild_build/HPL/2.0/golf-1.4.10/hpl-2.0/setup)
== 2016-04-24 13:34:48,823 main.run DEBUG run_cmd: Command output will be logged
to /tmp/easybuild-W85p4r/easybuild-run_cmd-XoJwMY.log
== 2016-04-24 13:34:48,849 main.run INFO cmd "/bin/bash make_generic" exited with
exitcode 0 and output:
...
```

# In numbers



included in EasyBuild v3.4.1:

- about 50k LoC of Python (incl. framework, easyblocks, and logging/option parsing support)
- about 15k LoC more for unit/integration test suites
- 1,307 supported software applications (excl. extensions, software versions)
- 26 different toolchain definitions (excl. toolchain versions)
- 188 software-specific easyblocks, 30 generic easyblocks
- 7,750 easyconfig files
  - different software versions, variants, using different toolchains, ...

# Basic usage



[http://easybuild.readthedocs.io/en/latest/Using\\_the\\_EasyBuild\\_command\\_line.html](http://easybuild.readthedocs.io/en/latest/Using_the_EasyBuild_command_line.html)

[http://easybuild.readthedocs.io/en/latest/Typical\\_workflow\\_example\\_with\\_WRF.html](http://easybuild.readthedocs.io/en/latest/Typical_workflow_example_with_WRF.html)

- specify software name/version and toolchain to 'eb' command
- commonly via easyconfig filename(s):

```
eb GCC-4.9.2.eb Clang-3.6.0-GCC-4.9.2.eb
```

- check whether required toolchain & dependencies are available using `--dry-run/-D`:

```
eb Python-2.7.14-intel-2017b.eb -D
```

- enable dependency resolution via `--robot/-r`:

```
eb WRF-3.8.0-intel-2016b-dmpar.eb -dr
```

# Using a custom module naming scheme



- a couple of different module naming schemes are included in EasyBuild
  - see `--avail-module-naming-schemes`
  - specify active module naming scheme via `--module-naming-scheme`
  - default: `EasyBuildMNS (<name>/<version>-<toolchain>-<versionsuffix>)`
- **you can implement your own module naming scheme relatively easily**
  - specify how to compose module name using provided metadata
  - via Python module that defines custom derivative class of `ModuleNamingScheme`
  - make EasyBuild aware of it via `--include-module-naming-schemes`
- decouple naming of install dirs vs modules via `--fixed-installdir-naming-scheme`