

easybuild

building software with ease

Introduction to EasyBuild
EasyBuild hackathon @ Nicosia, Cyprus
Oct 22th 2013

kenneth.hoste@ugent.be
easybuild@lists.ugent.be



HPC-UGent @ Ghent University, Belgium

- central contact for High Performance Computing at university
- established in 2008, part of central IT department (DICT)
- member of Flemish Supercomputer Centre (VSC)
 - collaboration between Flemish university associations





- ▶ our computing infrastructure:
 - ▶ seven Tier 2 systems (capacity computing)
 - ▶ one Tier 1 system
 - #119 in Top500 (June'12), currently at #239
- ▶ HPC-UGent team currently consists of 8 FTEs
 - ▶ system administration of HPC infrastructure
 - ▶ top-down for Tier2 systems: hardware, configuration, user support
 - ▶ Tier1: owned by UGent, setup together with HP, user support
 - ▶ user support and training
 - ▶ EasyBuild grew out of need from this
 - ▶ convincing groups to switch to central infrastructure

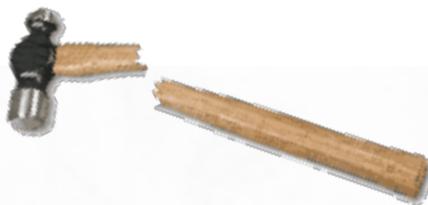
Building scientific software is... fun!

Scientists focus on the *functionality* of their software,
not on portability, build system, ...

Common **issues** with build procedures of scientific software:

- ☒ **incomplete**, e.g. no install step
- ☒ requiring human **interaction**
- ☒ heavily **customised** and **non-standard**
- ☒ uses **hard-coded** settings
- ☒ poor and/or outdated **documentation**

Very time-consuming for user support teams!



Current tools are lacking

- building from **source** is preferred in an HPC environment
 - **performance** is critical, instruction selection is key (e.g. AVX)
- not a lot of packaged scientific software available (RPMs, ...)
 - requires **huge effort**, which is duplicated across distros
- existing build tools are
 - hard to **Maintain** (e.g., bash scripts)
 - stand-alone, **no reuse** of previous efforts
 - **OS-dependent** (HomeBrew, *Ports, ...)
 - **custom** to (groups of) software packages
 - e.g., Dorsal (DOLFIN), gmkpack (ALADIN)



Our build tool wish list

- ▶ **flexible framework**
- ▶ allows for **reproducible** builds
- ▶ supports **co-existence** of versions/builds
- ▶ enables **sharing** of build procedure implementations
- ▶ fully **automates** builds
- ▶ **dependency** resolution

Building software with ease



a software build and installation framework

- written in **Python**
- developed in-house for 2.5 years before public release
- **open-source (GPLv2)** since April 2012
- EasyBuild v1.0: **stable API** (November 2012)
- **monthly releases** (latest: v1.8.2, Oct 18th 2013)
- continuously enhanced and extended
- <http://hpcugent.github.io/easybuild>



EasyBuild dependencies

■ **Linux / OS X**

- used daily on Scientific Linux 5.x/6.x (Red Hat-based)
- also tested on Fedora, Debian, Ubuntu, CentOS, SLES, ...
- some known issues on OS X, focus is on Linux
- no Windows support (and none planned for now)

■ **Python v2.4 or more recent version (2.x, not Python 3 support yet)**

■ **environment modules (or Lmod)**

- system C/C++ compiler to bootstrap a GCC toolchain



Installing EasyBuild :(

EasyBuild suffers from the mess that is Python packaging...

\$ easy_install --user easybuild

error: option --user not recognized (only for recent versions of easy_install / setuptools)

“You should be using pip!”

\$ pip install --user easybuild

pip: No such file or directory (pip not installed)

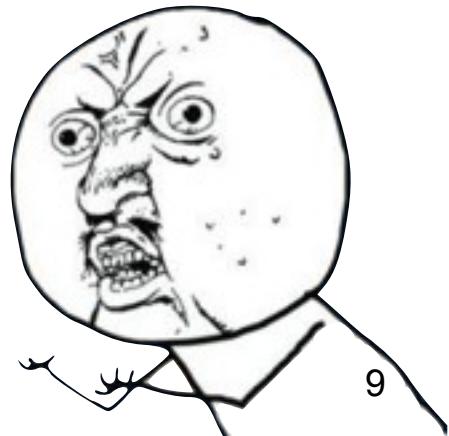
“Just use --prefix with easy_install!”

\$ easy_install --prefix=\$HOME easybuild

\$ export PATH=\$HOME/bin:\$PATH

\$ eb --version

ERROR: Failed to locate EasyBuild's main script
(\$PYTHONPATH is not set correctly)





Bootstrapping EasyBuild

The easiest way to install EasyBuild is by **bootstrapping** it.

<https://github.com/hpcugent/easybuild/wiki/Bootstrapping-EasyBuild>

```
$ wget http://hpcugent.github.com/easybuild/bootstrap_eb.py  
$ python bootstrap_eb.py $HOME
```

This will install EasyBuild using EasyBuild, and produce a module:

```
$ export MODULEPATH=$HOME/modules/all:$MODULEPATH  
$ module load EasyBuild  
$ eb --version
```

This is EasyBuild 1.8.2 (framework: 1.8.2, easyblocks: 1.8.2)

We're also looking into a packaged release (RPM, .deb, ...).



Configuring EasyBuild

By default, EasyBuild will install software to

`$HOME/.local/easybuild/software`

and produce modules files in

`$HOME/.local/easybuild/modules/all`

You can instruct EasyBuild otherwise by **configuring** it, using:

- a **configuration file**, e.g., `$HOME/.easybuild/config.cfg`
- **environment variables**, e.g., `$EASYBUILD_INSTALLPATH`
- **command line**, e.g. `--installpath`

<https://github.com/hpcugent/easybuild/wiki/Configuration>

(note: documentation needs work)



'Quick' demo for the impatient

```
eb HPL-2.0-goalf-1.1.0-no-OFED.eb --robot
```

- downloads all required sources (best effort)
- builds *goalf* toolchain (be patient), and builds HPL with it
 - goalf: GCC, OpenMPI, ATLAS, LAPACK, FFTW, ScaLAPACK, BLACS
- default: source/build/install dir in \$HOME/.local/easybuild

note: we need a better *quick* demo (without ATLAS)



Terminology

■ **framework**

- Python packages and modules forming *the core of EasyBuild*
- provides (loads of) supporting functionality
- very modular and dynamic design w.r.t. easyblocks, toolchains, ...

■ **easyblock**

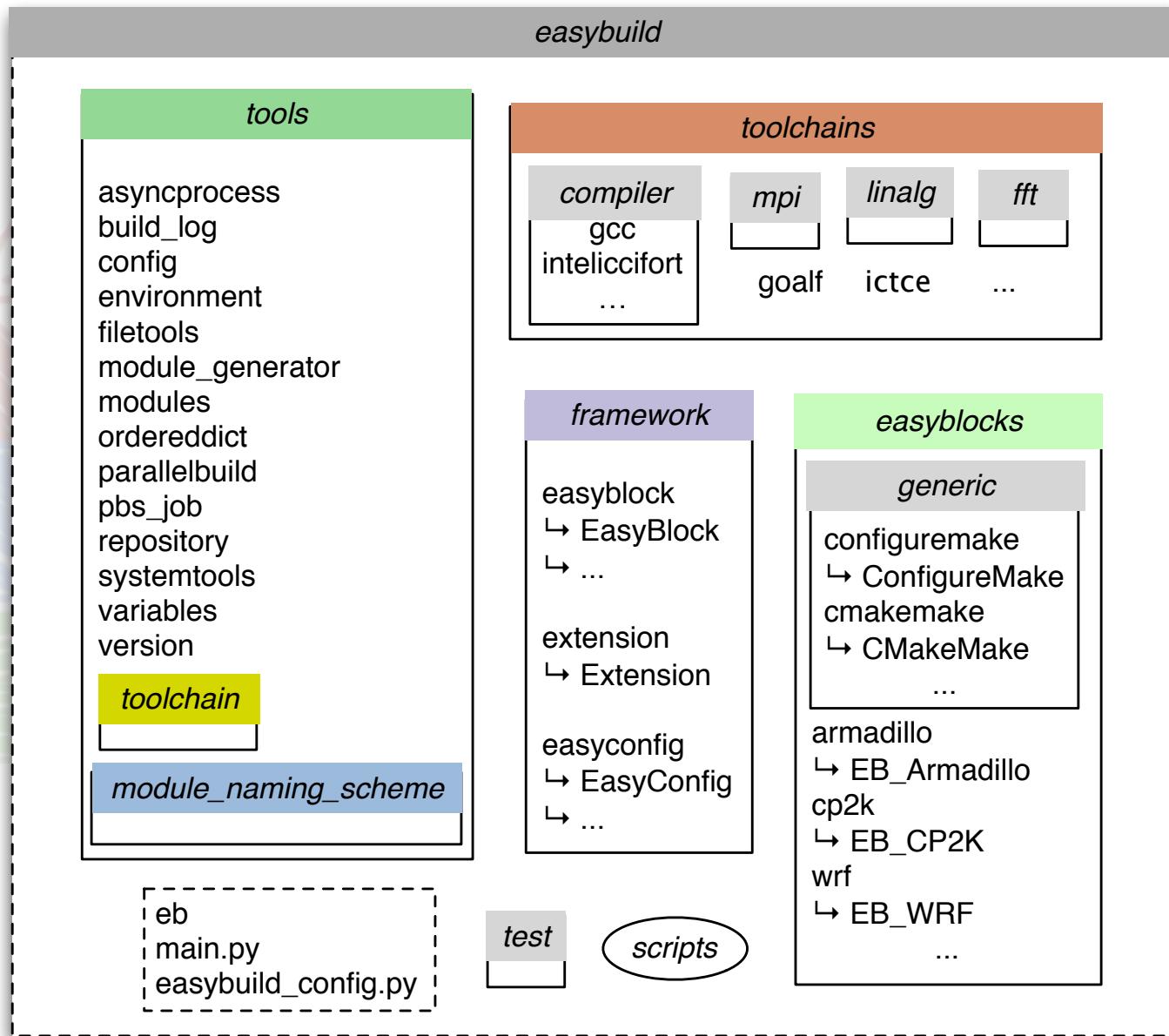
- a Python module providing *implementation of a build procedure*
- can be generic or software-specific

■ **easyconfig file (.eb)**

- *build specification:*
 - software name/version, toolchain, build options, ...
- simple text files, Python syntax



EasyBuild: high-level design

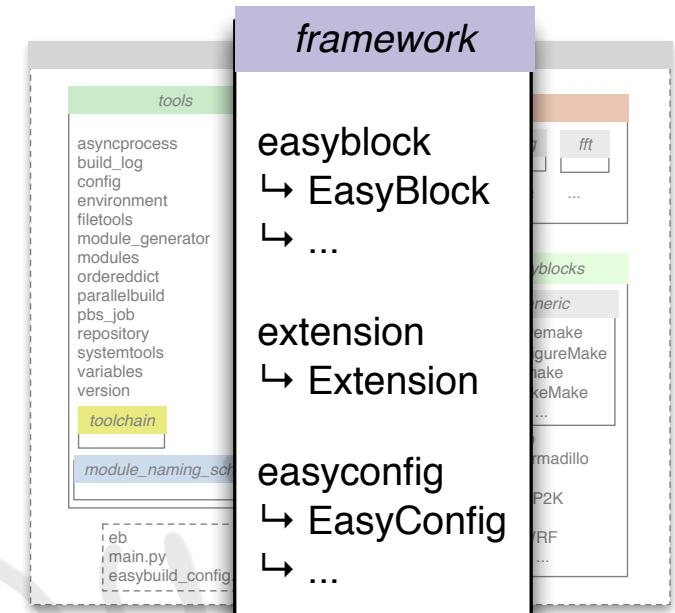




High-level design: framework

framework package

- **core of EasyBuild**
- ‘abstract’ class *Easyblock*
 - should be subclassed
- *EasyConfig* class
 - in *easyconfig* package, next to supporting modules
- *Extension* class
 - e.g., to build and install Python packages, R libraries, ...



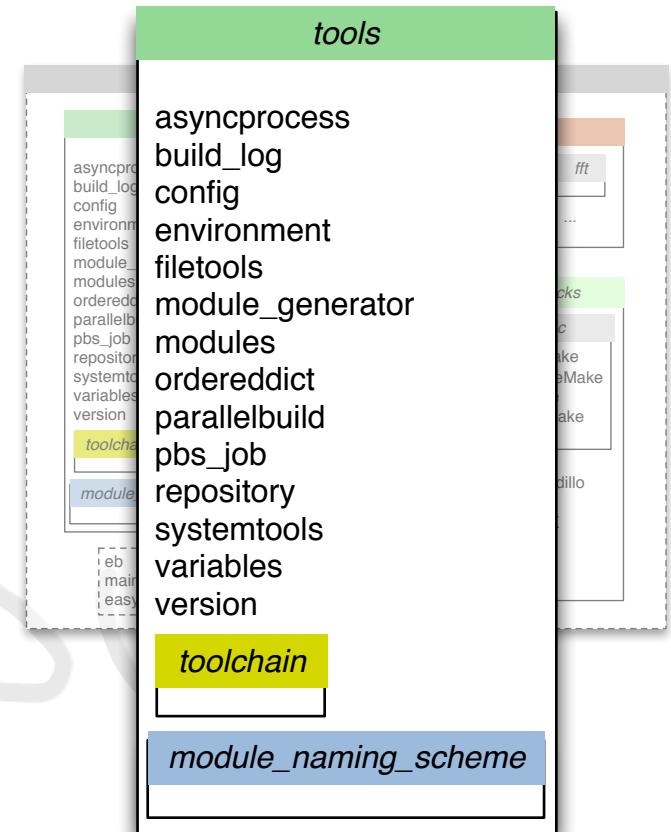


High-level design: framework

tools package

supporting functionality, e.g.:

- ☒ `run_cmd` for shell commands
- ☒ `run_cmd_qa` for interaction
- ☒ `extract_file` for unpacking
- ☒ `apply_patch` for patching
- ☒ `tools.toolchain` package for compiler toolchains
- ☒ `tools.module_naming_scheme` for module naming schemes

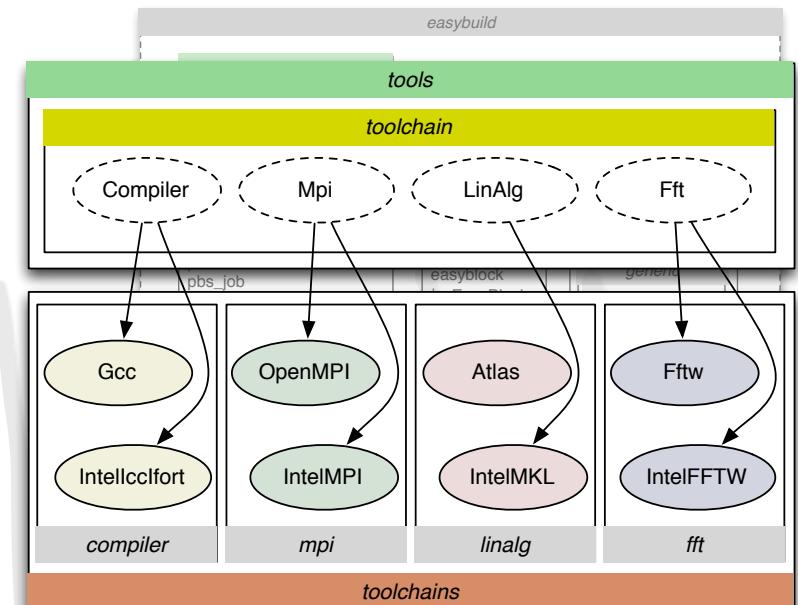




High-level design: framework

toolchains package

- support for **compiler toolchains**
- relies on *tools.toolchain*
- toolchains are defined in here
- organized in subpackages:
 - *toolchains.compiler*
 - *toolchains.mpi*
 - *toolchains.linalg* (BLAS, LAPACK, ...)
 - *toolchains.fft*
- very modular design for allowing extensibility
- plug in a Python module for compiler/library to extend it

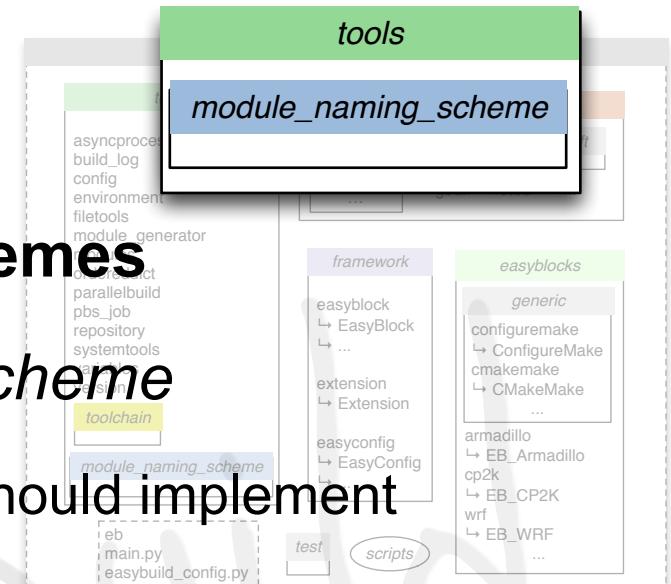




High-level design: framework

module_naming_scheme package

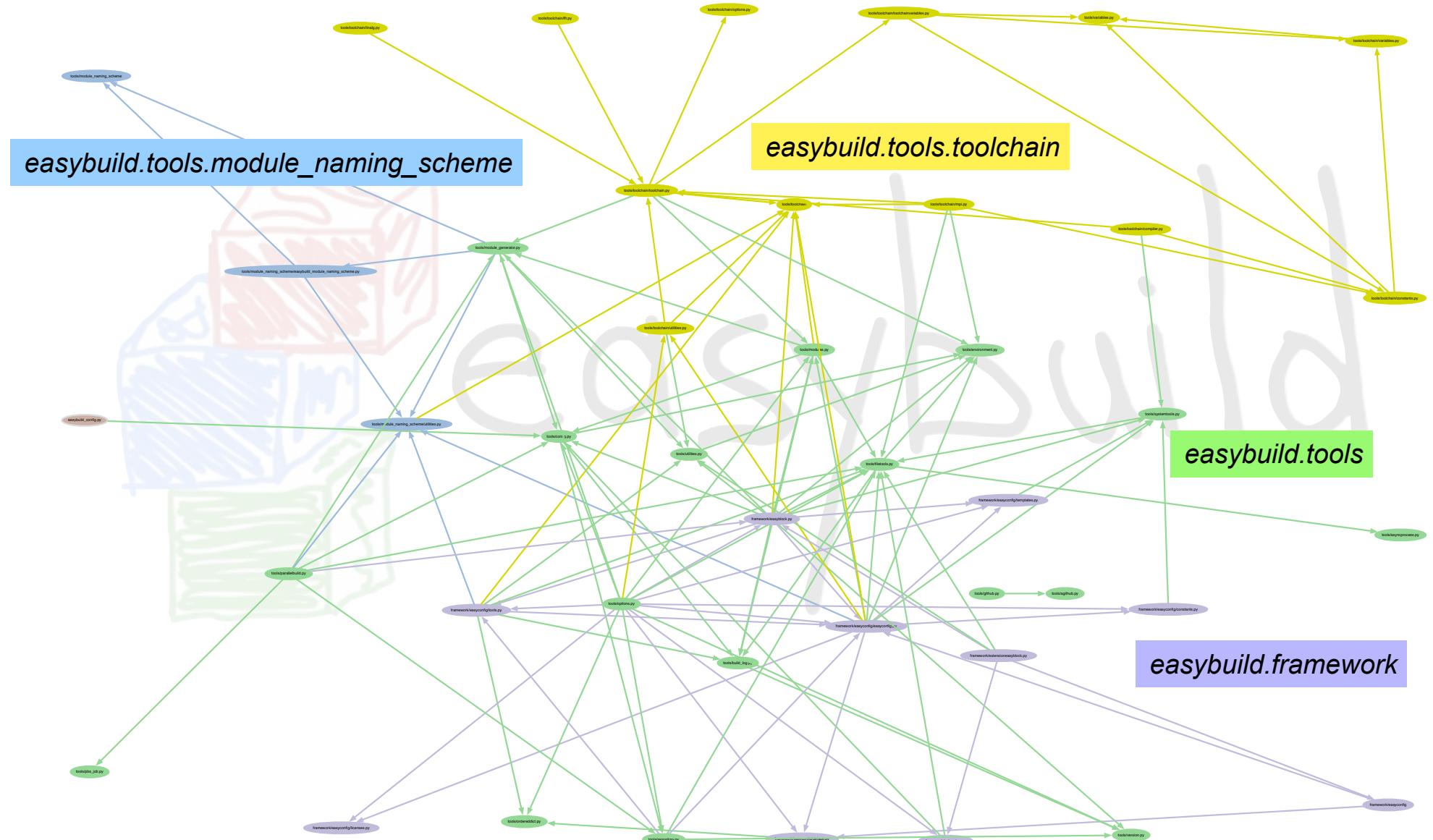
- support for **custom module naming schemes**
- provides ‘abstract’ class *ModuleNamingScheme*
 - empty shell that a module naming scheme should implement
- define your module naming scheme in this namespace
 - EasyBuild picks up any scheme following the specifications
 - see “*Using a custom module naming scheme*” wiki page
- our naming scheme: *EasyBuildModuleNamingScheme*
- available since EasyBuild v1.8.0, with limited capabilities
 - only *name*, *version*, *versionsuffix* and *toolchain* available





easybuild

High-level design: framework





High-level design: framework

test package

- unit testing of EasyBuild

```
python -m test.framework.suite
```

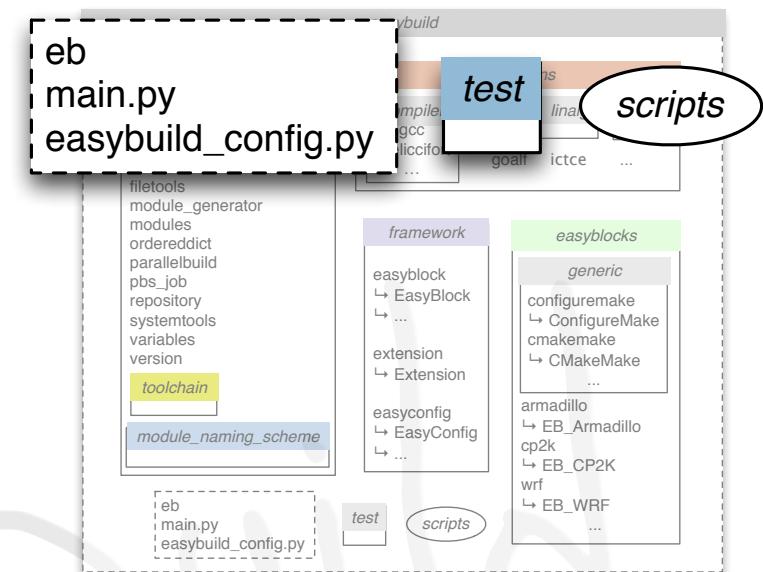
collection of scripts

- mainly for EasyBuild developers

main.py script + eb wrapper

default EasyBuild configuration file

- can be used as a template for your own config file

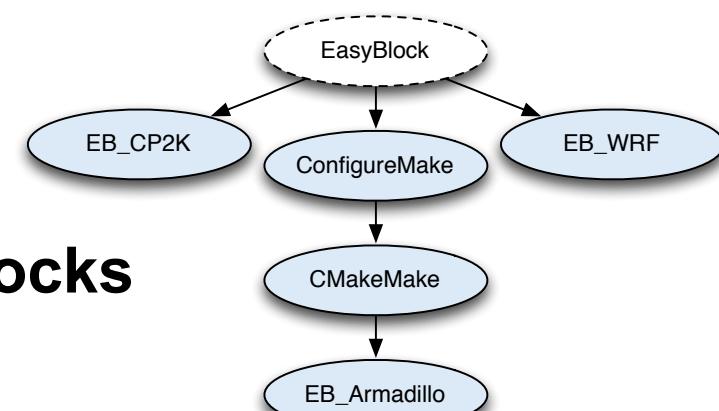
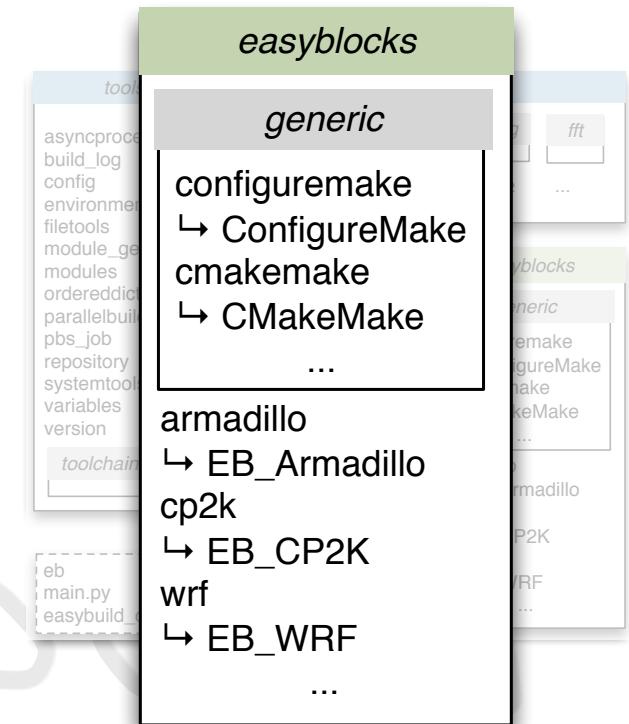




High-level design: easyblocks

easyblocks package

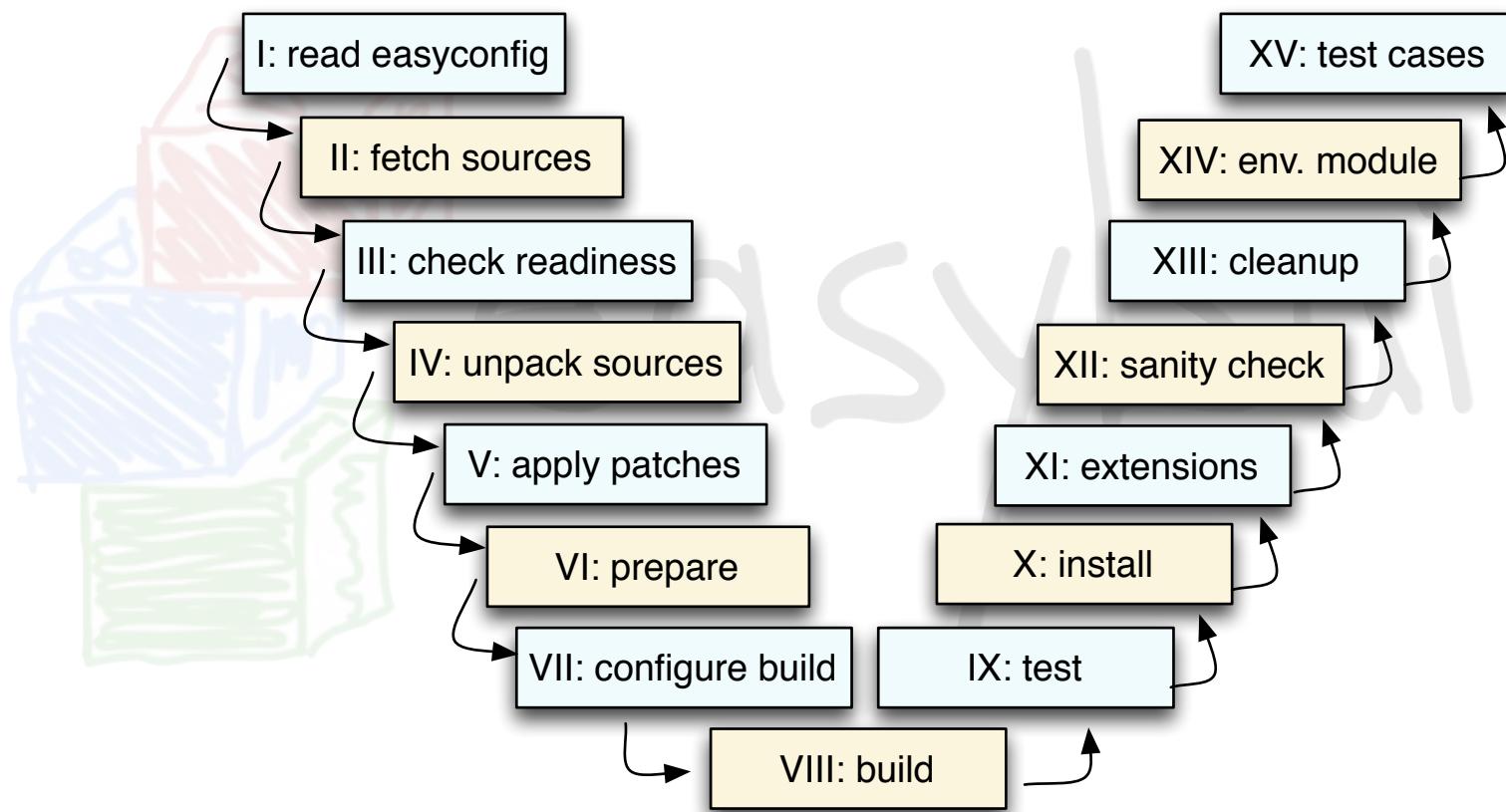
- **build procedure implementations**
- modular design, dynamically extensible
 - add your easyblock in the Python search path
 - EasyBuild will pick it up
- **easyblocks.generic: generic easyblocks**
 - custom support for groups of applications
 - e.g., *ConfigureMake*, *CMakeMake*, ...
- **easyblocks: application-specific easyblocks**
- object-oriented scheme
 - subclass from existing easyblocks or abstract class *EasyBlock*





Step-wise install procedure

build and install procedure as implemented by EasyBuild



most of these steps can be customized if required



Features

■ **logging and archiving**

- entire build process is logged thoroughly, logs stored in install dir
- easyconfig file used for build is archived (file/svn/git repo)

■ **automatic dependency resolution**

- build stack of software with a single command, using --robot

■ **running interactive installers autonomously**

- by passing a Q&A Python dictionary to the `run_cmd_qa` function

■ **building software in parallel**

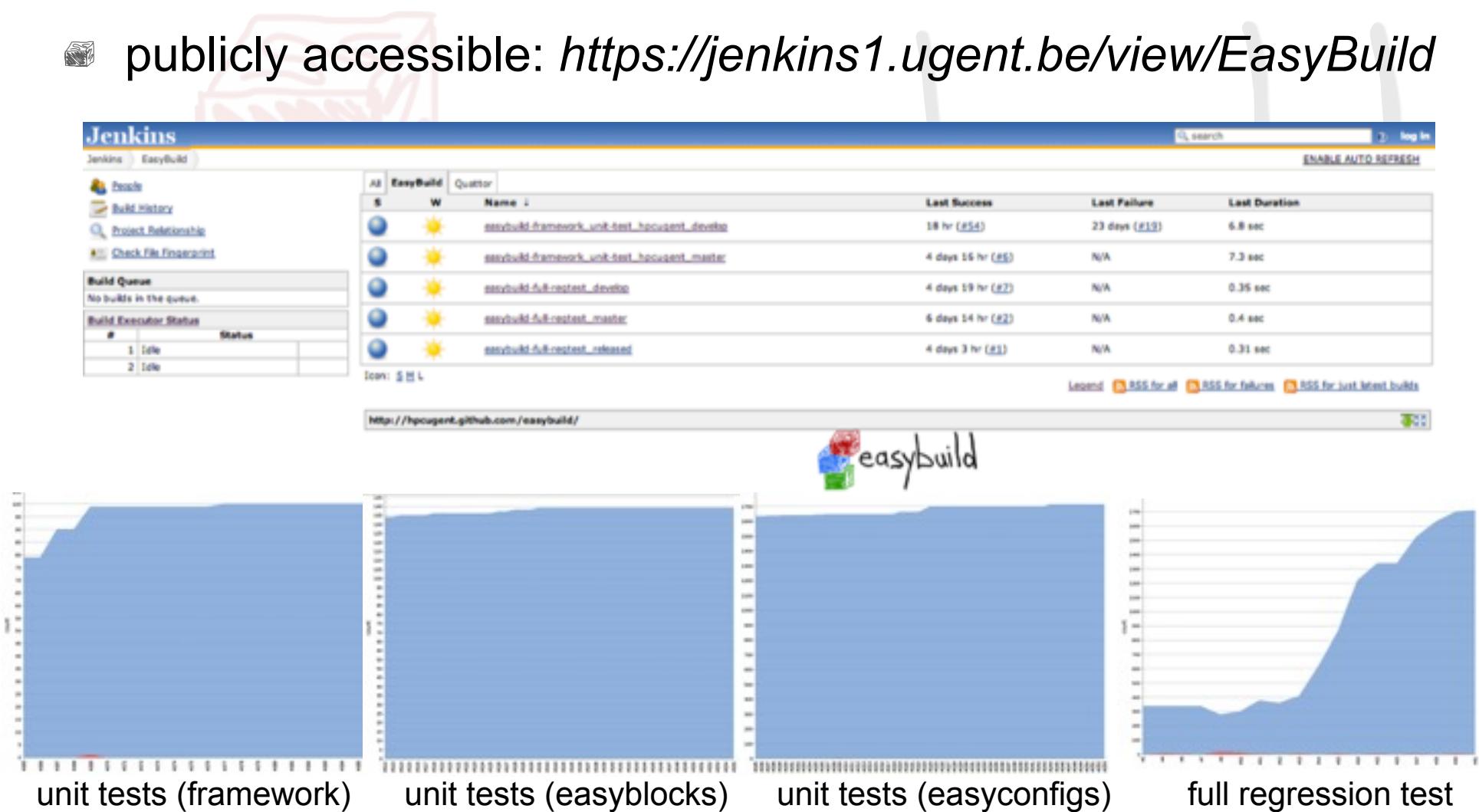
- e.g., on a (PBS) cluster, by using --job

■ **comprehensive testing:** unit tests, regression testing



Comprehensive testing

- **unit tests** are run automagically by Jenkins
- **regression test** results are pulled in on request
- publicly accessible: <https://jenkins1.ugent.be/view/EasyBuild>

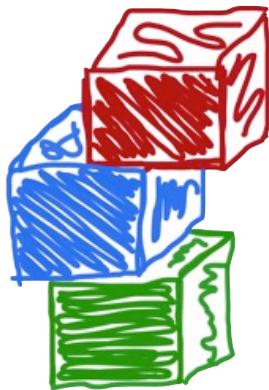




List of supported software (v1.8.2)

338 different software packages (1,710 example easyconfigs)

a2ps ABAQUS ABINIT ABYSS ACML **ALADIN** ALLPATHS-LG AMOS AnalyzeFMRI ant aria2 Armadillo ASE ATLAS Autoconf Automake bam2fastq BamTools Bash bbcp bbFTP bbftpPRO beagle-lib BFAST binutils biodeps BioPerl Biopython BiSearch Bison BLACS BLAST BLAT BOINC Bonnie++ Boost Bowtie Bowtie2 BWA byacc bzip2 cairo CBLAS ccache CCfits CD-HIT CFITSIO cflow CGAL cgdb Chapel Clang ClangGCC CLHEP ClustalW2 CMake Corkscrew **CP2K** CPLEX CRF++ CUDA Cufflinks cURL CVXOPT Cython DB Diffutils DL_POLY_Classic Docutils **DOLFIN** Doxygen **EasyBuild** ECore Eigen ELinks EMBOSS EPD ESMF ESPResSo expat FASTA fastahack FASTX-Toolkit FCM FDTD_Solutions Ferret FFC FFTW FIAT findutils flex FLUENT fmri FoldX fontconfig FRC_align freeglut FreeSurfer freetype FSL g2clib g2lib GATE GATK gawk **GCC** GDAL GDB Geant4 GEOS gettext GHC git GLib GLIMMER GLPK glproto GMP gnuplot gnutls google-sparsehash GPAW gperf Greenlet grib_api GROMACS GSL guile gzip h5py h5utils Harminv HDF HDF5 HH-suite HMMER horton HPCBIOS_Bioinfo HPCBIOS_Debuggers HPCBIOS_LifeSciences HPCBIOS_Math HPCBIOS_Profilers HPL hwloc Hypre icc ifort imkl impi Infernal Inspector Instant Iperf ipp itac JasPer Java Ninja2 JUnit LAPACK lftp libctl libdrm libffi libgtextutils libharu libibmad libibumad libibverbs libidn Libint libint2 libmatheval libpciaccess libpng libpthread-stubs libreadline libsmm LibTIFF libtool libungif libunistring libxc libxcb libxml2 libxslt libyaml likwid lxml LZO M4 make makedepend Maple MariaDB Mathematica MATLAB matplotlib mc MCL MDP Meep MEME Mercurial Mesa Mesquite MetaVelvet METIS molmod Mothur MPFR mpi4py mpiBLAST MPICH MrBayes MTL4 MUMmer MUMPS MUSCLE MVAPICH2 nano NASM NCBI-Toolkit **NCL** ncurses netCDF netCDF-C++ netCDF-Fortran nettle **NEURON** ns numactl numexpr numpy **NWChem** Oases Oger OpenBLAS **OpenFOAM** OpenIFS OpenMPI OpenPGM OpenSSL ORCA orthomcl otcl PAML pandas PAPI parallel ParMETIS ParMGridGen Pasha paycheck PCRE Perl **PETSc** petsc4py phonopy pixman pkg-config PLINK Primer3 problog PSI pyTables Python python-meep PyYAML PyZMQ QLogicMPI Qt **QuantumESPRESSO** R RAxML RCS RNAz ROOT Rosetta SAMtools ScaLAPACK ScientificPython scikit-learn scipy SCons SCOPP SCOTCH setuptools Shapely SHRIMP SLEPc SOAPdenovo Sphinx SQLite Stacks Stow SuiteSparse SWIG sympy Szip Tar tbb Tcl tclcl tcsh Theano TiCCutils TiMBL TinySVM Tk TopHat Tornado TotalView **Trilinos** Trinity UDUNITS UFC UFL util-linux Valgrind Velvet ViennaRNA Viper VSC-tools VTK VTune **WIEN2k** wiki2beamer **WPS WRF** xcb-proto XCrySDen XML XML-LibXML XML-Simple xorg-macros xproto yaff YamCha YAML-Syck Yasm ZeroMQ zlib zsh zsync



easybuild

building software with ease

Do you want to know more?

website: <http://hpcugent.github.com/easybuild>

GitHub: [https://github.com/hpcugent/easybuild\[-framework|-easyblocks|-easyconfigs\]](https://github.com/hpcugent/easybuild[-framework|-easyblocks|-easyconfigs])

PyPi: [http://pypi.python.org/pypi/easybuild\[-framework|-easyblocks|-easyconfigs\]](http://pypi.python.org/pypi/easybuild[-framework|-easyblocks|-easyconfigs])

mailing list: easybuild@lists.ugent.be

Twitter: @easy_build

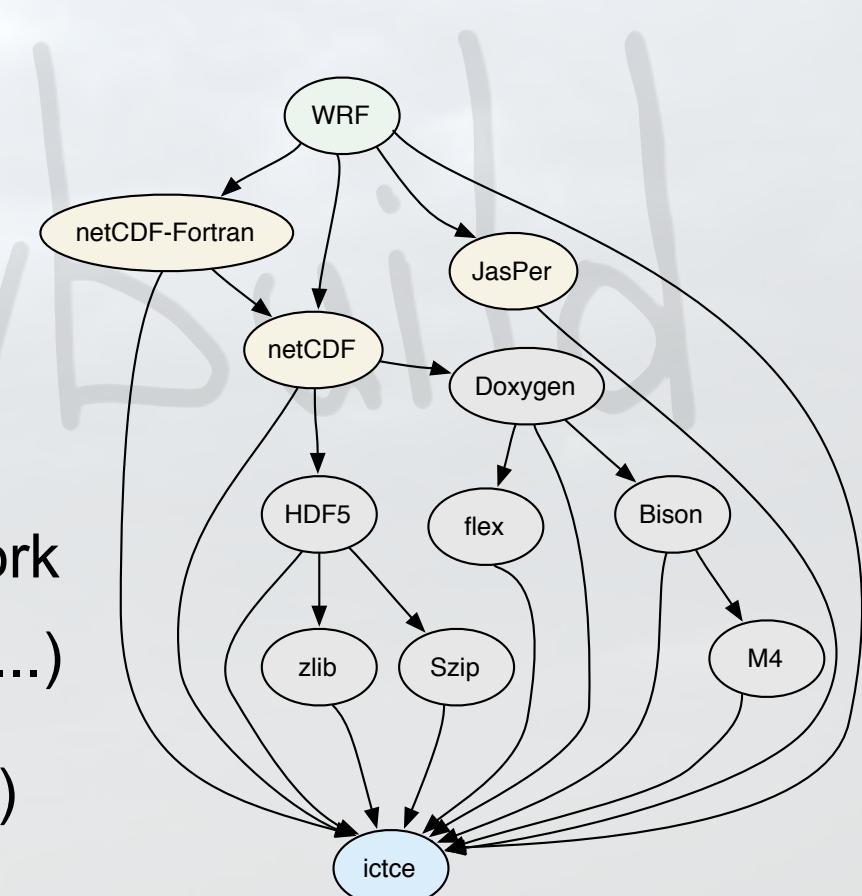
IRC: #easybuild on freenode.net



Use case: building WRF

building and installing WRF (Weather Research and Forecasting Model)

- ▶ <http://www.wrf-model.org>
- ▶ complex(ish) **dependency graph**
- ▶ very **non-standard build procedure**
 - ▶ interactive `configure` script (!)
 - ▶ resulting `configure.wrf` needs work
(hardcoding, tweaking of options, ...)
 - ▶ `compile` script (wraps around `make`)
 - ▶ no actual installation step





Use case: building WRF with eb

building and installing WRF (Weather Research and Forecasting Model)

- ▶ easyblock that comes with EasyBuild implements build procedure
 - ▶ running interactive configure script **autonomously**
 - ▶ **patching** configure.wrf
 - ▶ **building** with compile script
 - ▶ **testing** build with standard included tests/benchmarks
- ▶ easyconfig files for different versions, toolchains, build options, ...
- ▶ building and installing WRF becomes child's play, for example:

```
eb --software=WRF,3.4 --toolchain-name=ictce --robot
```



Use case: easyblock for WRF

part I: imports, class constructor, custom easyconfig parameter

```
1 import fileinput, os, re, sys
2
3 import easybuild.tools.environment as env
4 from easybuild.easyblocks.netcdf import set_ncdf_env_vars
5 from easybuild.framework.easyblock import EasyBlock
6 from easybuild.framework.easyconfig import MANDATORY
7 from easybuild.tools.filetools import patch_perl_script_ autoflush, run_cmd, run_cmd_qa
8 from easybuild.tools.modules import get_software_root
9
10 class EB_WRF(EasyBlock): ← class definition
11
12     def __init__(self, *args, **kwargs):
13         super(EB_WRF, self).__init__(*args, **kwargs)
14         self.build_in_installdir = True ← class constructor,  
specify building in  
installation dir
15
16     @staticmethod
17     def extra_options():
18         extra_vars = [('buildtype', [None, "Type of build (e.g., dmpar, dm+sm).", MANDATORY])]
19         return EasyBlock.extra_options(extra_vars)
20
```

import required functionality



class constructor,
specify building in
installation dir



define custom easyconfig parameters



Use case: easyblock for WRF

part II: configuration (1/2)

```
21 def configure_step(self): ← configuration step function
22     # prepare to configure
23     set_netcdf_env_vars(self.log) ← set environment variables
24     for dependencies
25     jasper = get_software_root('JasPer')
26     if jasper:
27         jasperlibdir = os.path.join(jasper, "lib")
28         env.setvar('JASPERINC', os.path.join(jasper, "include"))
29         env.setvar('JASPERLIB', jasperlibdir)
30
31     env.setvar('WRFIO_NCD_LARGE_FILE_SUPPORT', '1') ← set WRF-specific env var
32     for build options
33
34     patch_perl_script_autoflush(os.path.join("arch", "Config_new.pl")) ← patch configure
35     script to run it
36     autonomously
37     known_build_types = ['serial', 'smpar', 'dmpar', 'dm+sm']
38     self.parallel_build_types = ["dmpar", "smpar", "dm+sm"]
39     bt = self.cfg['buildtype']
40
41     if not bt in known_build_types:
42         self.log.error("Unknown build type: '%s' (supported: %s)" % (bt, known_build_types)) ← check whether specified
43                                         build type makes sense
```



Use case: easyblock for WRF

part II: configuration (2/2)

```
42 # run configure script
43 bt_option = "Linux x86_64 i486 i586 i686, ifort compiler with icc"
44 bt_question = "\s*(?P<nr>[0-9]+).\s*\%s\s*\(%s\)" % (bt_option, bt)
45
46 cmd = "./configure"
47 qa = {"(1=basic, 2=preset moves, 3=vortex following) [default 1]:":
48     "1",
49     "(0=no nesting, 1=basic, 2=preset moves, 3=vortex following) [default 0]:":
50     "0"}
51 std_qa = {r"%s.*\n(.*\n)*Enter selection\s*\[[0-9]+-[0-9]+\]\s*:" %
52           bt_question: "%(nr)s"}
53
54 run_cmd_qa(cmd, qa, no_qa=[], std_qa=std_qa, log_all=True, simple=True)
55
56 # patch configure.wrf
57 cfgfile = 'configure.wrf'
58
59 comps = {
60     'SCC': os.getenv('CC'), 'SFC': os.getenv('F90'),
61     'CCOMP': os.getenv('CC'), 'DM_FC': os.getenv('MPIF90'),
62     'DM_CC': "%s -DMPI2_SUPPORT" % os.getenv('MPIICC'),
63 }
64
65 for line in fileinput.input(cfgfile, inplace=1, backup='orig.comps'):
66     for (k, v) in comps.items():
67         line = re.sub(r"^(%s\s*=)\s*.*$" % k, r"\1 %s" % v, line)
68     sys.stdout.write(line)
```

Annotations from right to left:

- prepare Q&A for configuring
- run configure script autonomously
- patch generated configuration file



easybuild Use case: WRF - easyblock (3/3)

part III: build step & skip install step (since there is none)

```
67 def build_step(self):
68     # build WRF using the compile script
69     par = self.cfg['parallel']
70     cmd = "./compile -j %d wrf" % par
71     run_cmd(cmd, log_all=True, simple=True, log_output=True)
72
73     # build two test cases to produce ideal.exe and real.exe
74     for test in ["em_real", "em_b_wave"]:
75         cmd = "./compile -j %d %s" % (par, test)
76         run_cmd(cmd, log_all=True, simple=True, log_output=True)
77
78 def install_step(self):
79     pass
```

build step function

build WRF (in parallel)

build WRF utilities as well

no actual installation step (build in installation dir)



Use case: installing WRF

specify build details in easyconfig file (.eb)

software name
and version



```
1 name = 'WRF'  
2 version = '3.4'  
3  
4 homepage = 'http://www.wrf-model.org'  
5 description = 'Weather Research and Forecasting'  
6
```

software website
and description
(informative)

compiler toolchain
specification
and options



```
7 toolchain = {'name': 'ictce', 'version': '3.2.2.u3'}  
8 toolchainopts = {'opt': False, 'optarch': False}  
9
```

list of source files

```
10 sources = ['%sV%s.TAR.gz' % (name, version)]  
11 patches = ['WRF_parallel_build_fix.patch',  
12             'WRF-3.4_known_problems.patch',  
13             'WRF_tests_limit-runtimes.patch',  
14             'WRF_netCDF-Fortran_separate_path.patch']  
15
```

list of patches
for sources

```
16 dependencies = [('JasPer', '1.900.1'),  
17                   ('netCDF', '4.2'),  
18                   ('netCDF-Fortran', '4.2')]
```

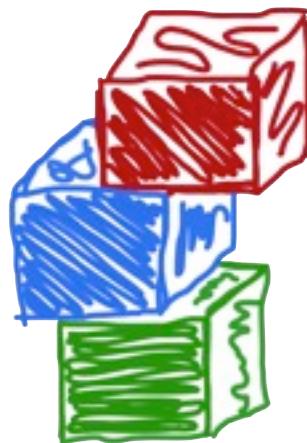
list of dependencies

custom parameter
for WRF



```
20 buildtype = 'dmpar'
```

eb WRF-3.4-ictce-3.2.2.u3-dmpar.eb --robot



easybuild

building software with ease

Introduction to EasyBuild
EasyBuild hackathon @ Nicosia, Cyprus
Oct 22th 2013

kenneth.hoste@ugent.be
easybuild@lists.ugent.be