UNIVERSITEIT GENT

easybuild

building software with ease

PyBUG meeting @ Ghent
lightning talk - Oct. 1st 2013

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

About HPC UGent:

- central contact for HPC at Ghent University
- part of central IT department (DICT)
- member of Flemish supercomputer centre (VSC)
  - collaboration between Flemish university associations

- seven Tier2 systems, one Tier1 system
  - Top500: #119 (June'12), #163 (Nov'12), #239 (June'13)
- team consists of 7 FTEs
- tasks include system administration of HPC infrastructure, user training, user support, ...

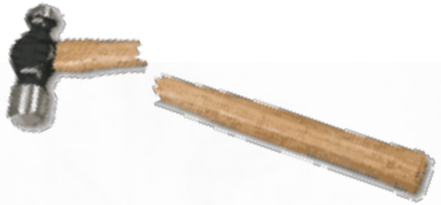
Building scientific software is... fun!

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- poor and/or outdated **documentation**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!

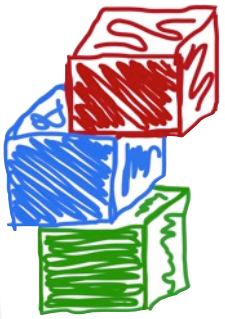


Current tools are lacking

- 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)

Building software with ease



a software build and installation framework

- written in **Python**
- developed in-house (HPC-UGent) for 2.5 years
- **open-source (GPLv2)** since April 2012
- **stable API** since Nov. 2012 (v1.0.0)
- latest release: v1.7.0 (v1.8.0 due this week)
- continuously enhanced and extended
- *http://hpcugent.github.com/easybuild*

Installing EasyBuild

```
$ easy_install --user easybuild
```

error: option --user not recognized (only for recent setuptools)

You should be using pip!

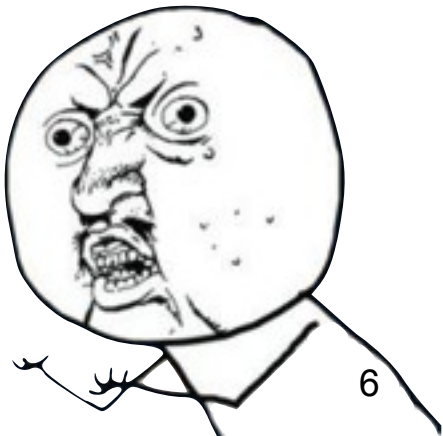
```
$ pip install --user easybuild
```

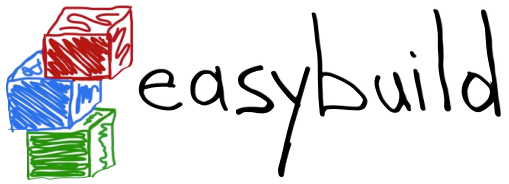
pip: No such file or directory (pip not installed)

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 not set correctly)





Bootstrapping EasyBuild

Easily install EasyBuild by bootstrapping it.

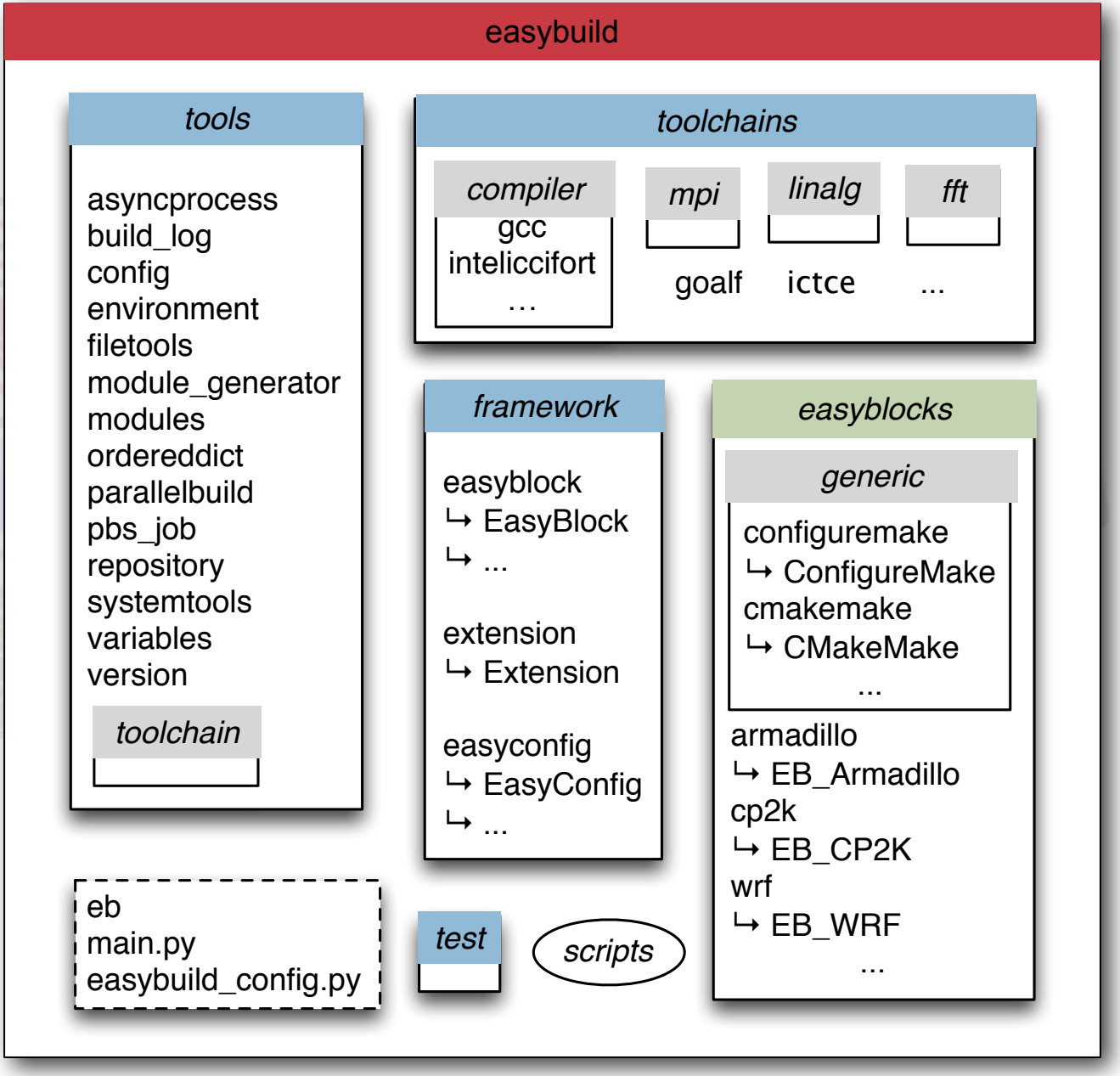
https://github.com/hpcugent/easybuild/wiki/Bootstrapping-EasyBuild

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

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

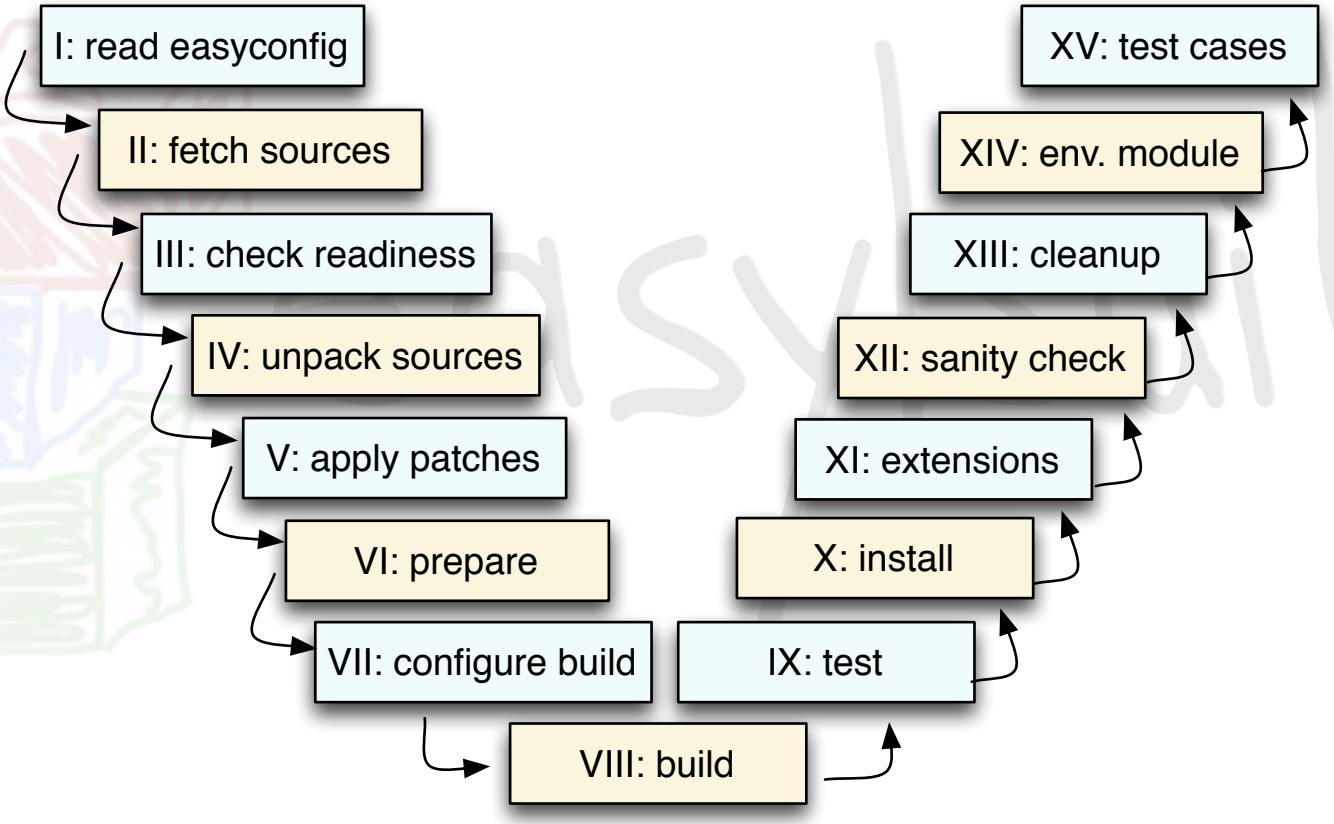
```
$ export MODULEPATH=$HOME/modules/all:$MODULEPATH
$ module load EasyBuild/1.7.0
$ eb --version
This is EasyBuild 1.7.0 (framework: 1.7.0, easyblocks: 1.7.0)
```

High-level design

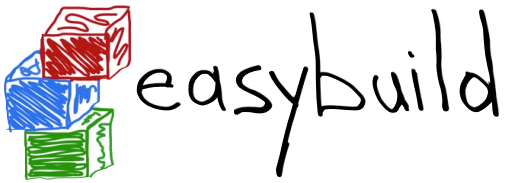


Step-wise install procedure

build and install procedure as implemented by EasyBuild



most of these steps can be customized if required

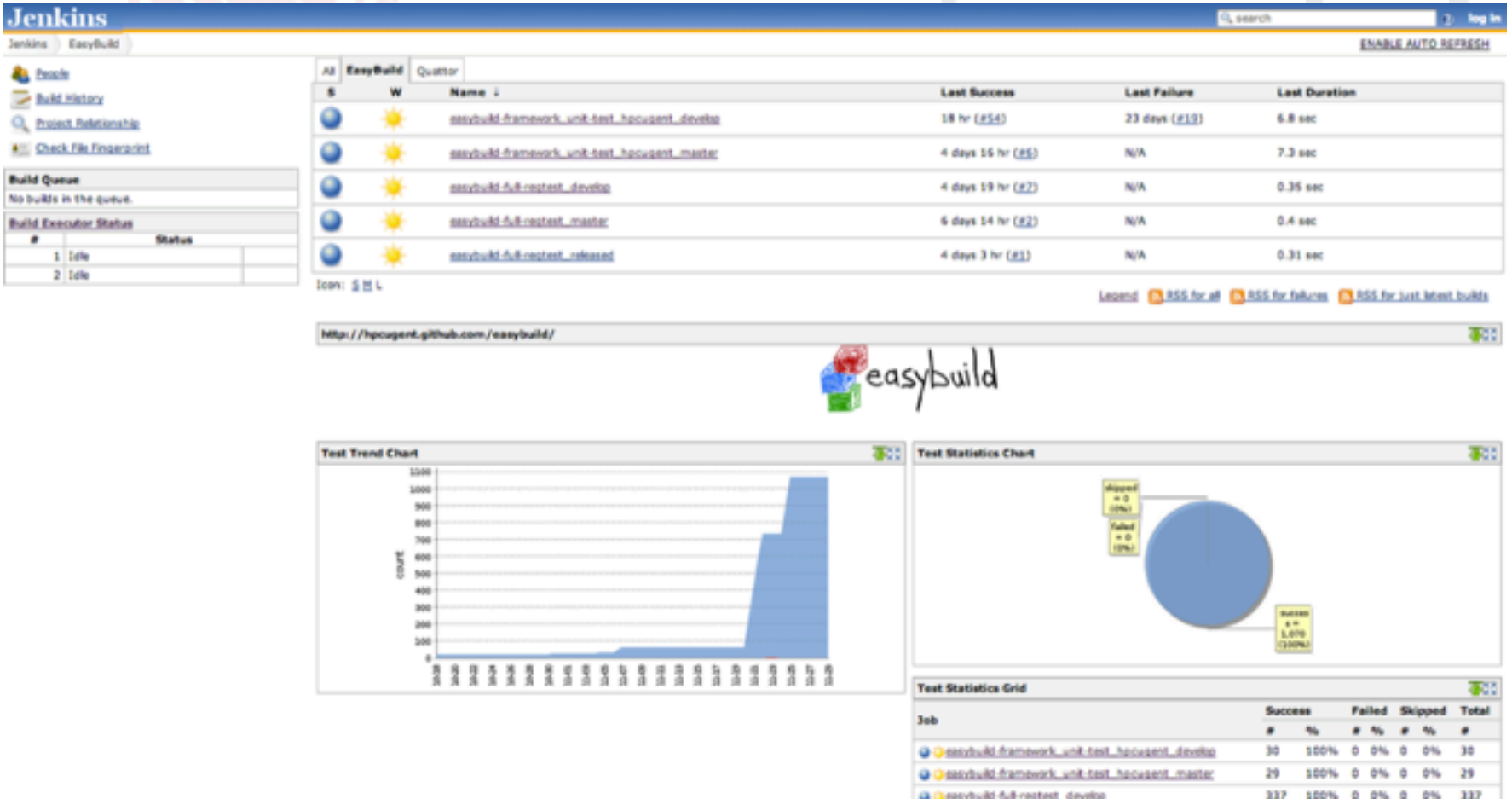


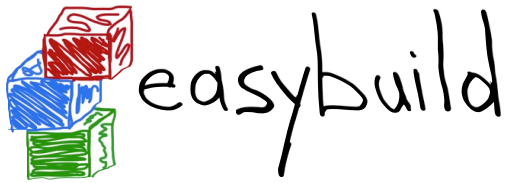
Features

- **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**
  - software stack be built 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
- publicly accessible: *https://jenkins1.ugent.be/view/EasyBuild*

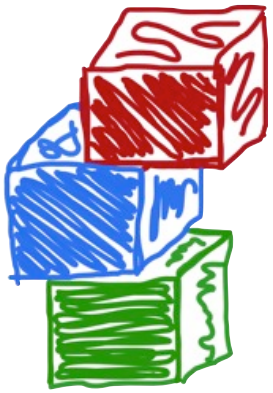




List of supported software (v1.7.0)

329 different software packages (1,631 example easyconfigs)

ABAQUS ABINIT ABySS ACML **ALADIN** ALLPATHS-LG AMOS ASE ATLAS AnalyzeFMRI Armadillo Autoconf Automake a2ps ant aria2 BFAST BLACS BLAST BLAT BWA BamTools Bash BiSearch BioPerl Biopython Bison Bonnie ++ Boost Bowtie Bowtie2 bam2fastq bbFTP bbcp bbftpPRO beagle-lib binutils biodeps byacc bzip2 CBLAS CCfits CD-HIT CFITSIO CGAL CLHEP CMake **CP2K** CPLEX CRF++ CUDA CVXOPT Chapel Clang ClangGCC ClustalW2 Corkscrew Cufflinks Cython cURL cairo ccache cflow cgdb cgmpich cgmpolf cgmvapich2 cgmvolf cgompi cgoolf DL_POLY_Classic **DOLFIN** Diffutils Docutils Doxygen ECore ELinks EMBOSS EPD ESMF ESPResSo **EasyBuild** Eigen expat FASTA FASTX-Toolkit FCM FFC FFTW FIAT FLUENT FRC_align FSL Ferret FreeSurfer findutils flex fmri fontconfig freeglut freetype GATE GATK GCC GDAL GDB GEOS GHC GLIMMER GLPK GLib GMP GPAW GROMACS GSL Geant4 Greenlet g2clib g2lib gawk gettext git glproto gmacml gmvapich2 gmvolf gnuplot gnutls goalf gompi google-sparsehash goolf goolfc gperf grib_api guile gzip HDF HDF5 HH-suite HMMER HPCBIOS_Bioinfo HPCBIOS_Debuggers HPCBIOS_LifeSciences HPCBIOS_Math HPCBIOS_Profilers HPL Harminv Hypre h5py h5utils horton hwloc Infernal Inspector Instant Iperf IronPython icc iccifort ictce ifort iiqmpi imkl impi iomkl ipp iqacml itac JUnit JasPer Java Jinja2 LAPACK LZO LibTIFF Libint lftp libctl libdrm libffi libgtextutils libharu libibmad libibumad libibverbs libidn libint2 libmatheval libpciaccess libpng libpthread-stubs libreadline libsmm libtool libungif libunistring libxc libxcb libxml2 libxslt libyaml likwid lxml M4 MATLAB MCL MDP MEME METIS MPFR MPICH MTL4 MUMmer MUSCLE MVAPICH2 Maple MariaDB Meep Mercurial Mesa MetaVelvet Mono Mothur MrBayes MyMediaLite make makedepend matplotlib mc molmod mpi4py mpiBLAST NASM NCBI-Toolkit NCL **NEURON NWChem** nano ncurses netCDF netCDF-C++ netCDF-Fortran nettle ns numactl numexpr numpy ORCA Oases Oger OpenBLAS **OpenFOAM** OpenIFS OpenMPI OpenPGM OpenSSL orthomcl otcl PAML PAPI PCRE **PETSc** PLINK PSI ParMETIS Pasha Perl Primer3 PyYAML PyZMQ Python pandas parallel paycheck petsc4py phonopy pixman pkg-config problog pyTables python-meep QLogicMPI Qt **QuantumESPRESSO** R RAxML RCS RNAz ROOT Rosetta SAMtools SCOOP SCOTCH SCons SHRiMP SLEPc SOAPdenovo SQLite SWIG ScaLAPACK ScientificPython Shapely Sphinx Stacks Stow SuiteSparse Szip scikit-learn scipy setuptools sympy Tar Tcl Theano TiCCutils TiMBL TinySVM Tk TopHat Tornado TotalView Trilinos Trinity tbb tclcl tcsh UDUNITS UFC UFL util-linux VSC-tools VTK VTune Valgrind Velvet ViennaRNA Viper **WIEN2k WPS WRF** wiki2beamer XCrySDen XML XML-LibXML XML-Simple xcb-proto xorg-macros xproto YAML-Syck YamCha Yasm yaff ZeroMQ zlib zsh zsync



easybuild

building software with ease

HPC UGENT

UNIVERSITEIT GENT

VSC Vlaams Supercomputer Centrum

Do you want to know more?

website: *http://hpcugent.github.com/easybuild*

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

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

mailing list: *easybuild@lists.ugent.be*

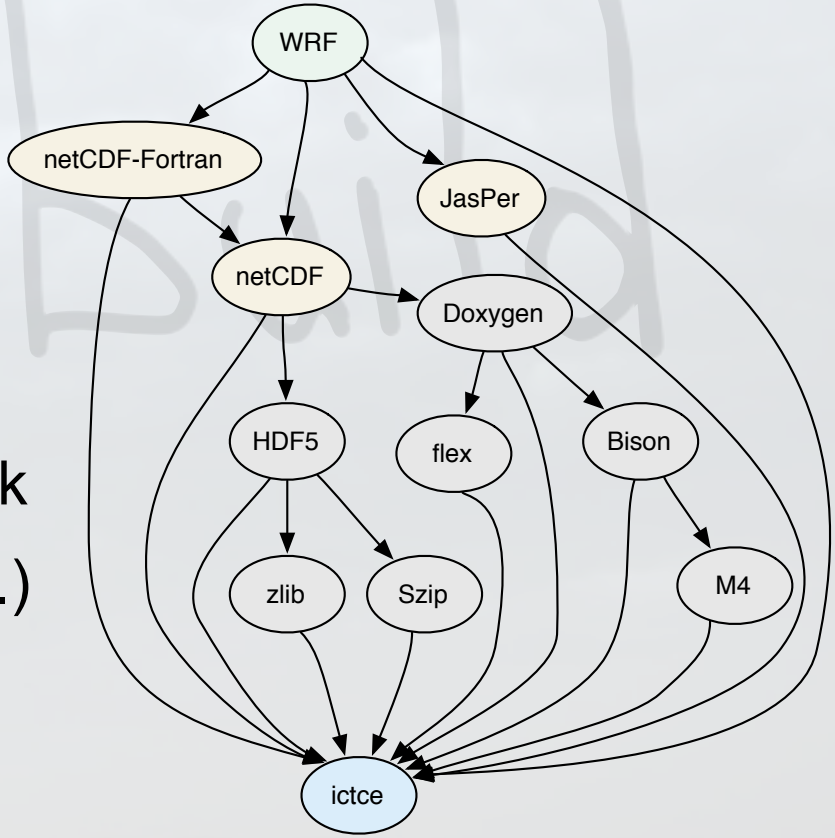
Twitter: *@easy_build*

IRC: *#easybuild on freenode.net*

Example use case (1/2)

*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



Example use case (2/2)

*building and installing **WRF** (Weather Research and Forecasting Model)*

- easyblock that comes with EasyBuild implements build procedure
  - running `configure` script **autonomously**
  - **building** with `compile` and **patching** configure.wrf
  - **testing** build with standard included tests/benchmarks
- various example easyconfig files available
  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: WRF - easyblock (1/3)

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_netcdf_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):
11
12   def __init__(self, *args, **kwargs):
13     super(EB_WRF, self).__init__(*args, **kwargs)
14     self.build_in_installdir = True
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 definition

class constructor, specify building in installation dir

define custom easyconfig parameters

Use case: WRF - easyblock (2/3)

configuration (part 1/2)

```
21  def configure_step(self):
22    # prepare to configure
23    set_netcdf_env_vars(self.log)
24
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')
32
33    patch_perl_script_autoflush(os.path.join("arch", "Config_new.pl"))
34
35    known_build_types = ['serial', 'smpar', 'dmpar', 'dm+sm']
36    self.parallel_build_types = ["dmpar", "smpar", "dm+sm"]
37    bt = self.cfg['buildtype']
38
39    if not bt in known_build_types:
40      self.log.error("Unknown build type: '%s' (supported: %s)" % (bt, known_build_types))
41
```

configuration step function

set environment variables for dependencies

set WRF-specific env var for build options

patch configure script to run it autonomously

check whether specified build type makes sense

Use case: WRF - easyblock (2/3)

configuration (part 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]:": "1",
48            "(0=no nesting, 1=basic, 2=preset moves, 3=vortex following) [default 0]:": "0"}
49      std_qa = {r"%s.*\n(.*\n)*Enter selection\s*\[[0-9]+-[0-9]+\]\s*:" % bt_question: "%(nr)s"}
50
51      run_cmd_qa(cmd, qa, no_qa=[], std_qa=std_qa, log_all=True, simple=True)
52
53      # patch configure.wrf
54      cfgfile = 'configure.wrf'
55
56      comps = {
57              'SCC': os.getenv('CC'), 'SFC': os.getenv('F90'),
58              'CCOMP': os.getenv('CC'), 'DM_FC': os.getenv('MPIF90'),
59              'DM_CC': "%s -DMPI2_SUPPORT" % os.getenv('MPICC'),
60             }
61
62      for line in fileinput.input(cfgfile, inplace=1, backup='.orig.comps'):
63          for (k, v) in comps.items():
64              line = re.sub(r"^(%s\s*=\s*).*$" % k, r"\1 %s" % v, line)
65          sys.stdout.write(line)
66
```

prepare Q&A for configuring

run configure script autonomously

patch generated configuration file

Use case: WRF - easyblock (3/3)

build step & skip install step (since there is none)

build step function

```
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
80
```

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
7  toolchain = {'name': 'ictce','version': '3.2.2.u3'}
8  toolchainopts = {'opt': False, 'optarch': False}
9
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
16 dependencies = [('JasPer', '1.900.1'),
17                 ('netCDF', '4.2'),
18                 ('netCDF-Fortran', '4.2')]
19
20 buildtype = 'dmpar'
```

software website and description (informative)

compiler toolchain specification and options

list of source files

list of patches for sources

list of dependencies

custom parameter for WRF

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

UNIVERSITEIT GENT

easybuild

building software with ease

PyBUG meeting @ Ghent
lightning talk - Oct. 1st 2013

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