



building software with ease

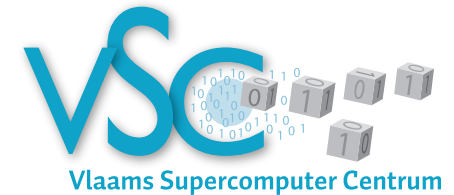
conference call with University of Auckland (New Zealand)
Oct 15th 2013

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 currently consists of 8 FTEs
- ▶ tasks include system administration of HPC infrastructure, user training, user support, ...



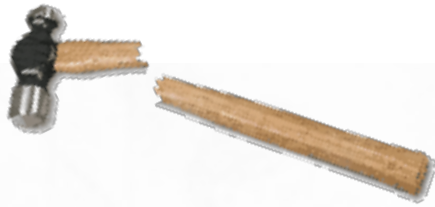
Building scientific software is... fun!

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- ❑ poor and/or outdated **documentation**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

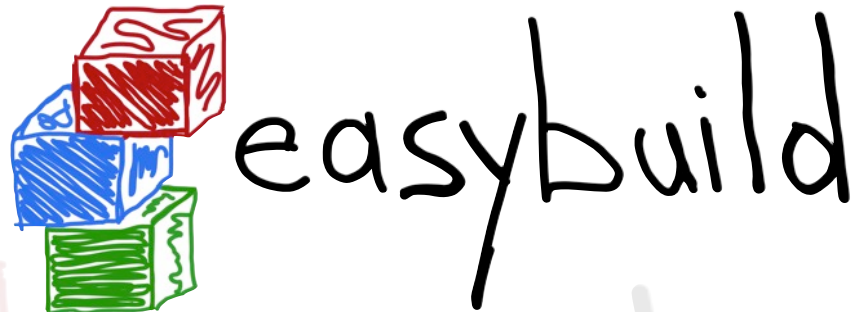
- ❏ building from **source** is preferred in an HPC environment
 - ❏ **performance** is critical, instruction selection is key (e.g. AVX)
- ❏ not a lot of packaged scientific software available (RPMs, ...)
 - ❏ requires **huge effort**, which is duplicated across distros
- ❏ existing build tools are
 - ❏ 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), gmckpack (ALADIN)



Our build tool wish list

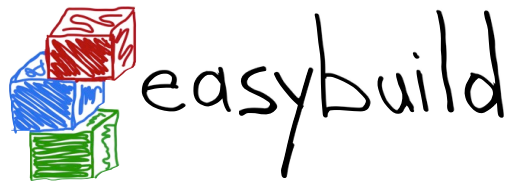
- ▶ **flexible** framework
- ▶ allows for **reproducible** builds
- ▶ supports **co-existence** of versions/builds
- ▶ **automated** builds and **dependency** resolution
- ▶ enables **sharing** of build procedure implementations

Building software with ease



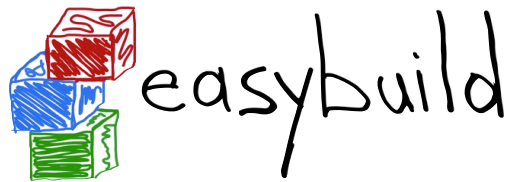
a software build and installation framework

- 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: v1.8.1 (Oct 14th 2013)
- continuously enhanced and extended
- *<http://hpcugent.github.com/easybuild>*



What does EasyBuild need?

- **Linux / OS X**
 - used daily on Scientific Linux 5.x/6.x (Red Hat-based)
 - also tested on Fedora, Debian, Ubuntu, CentOS, SLES, ...
 - a couple of known issues on OS X
 - no Windows support (and none planned for now)
- **Python 2.4** or more recent 2.x
- **environment modules** or **Lmod**
- system C/C++ compiler to bootstrap a GCC toolchain



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)
```

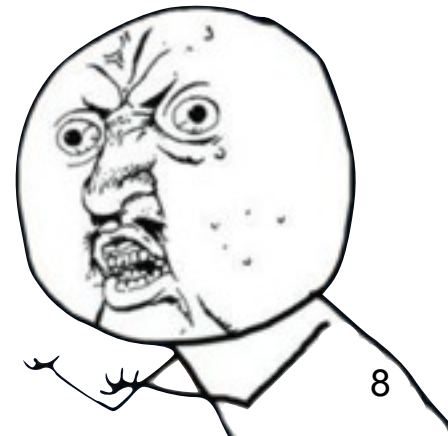
Just use --prefix with easy_install!

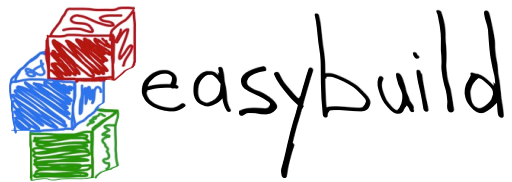
```
$ easy_install --prefix=$HOME easybuild
```

```
$ 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>

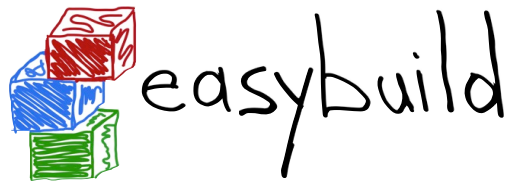
```
$ wget http://hpcugent.github.com/easybuild/bootstrap_eb.py
$ python bootstrap_eb.py $HOME
```

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

```
$ export MODULEPATH=$HOME/modules/all:$MODULEPATH
$ module load EasyBuild/1.8.1
$ eb --version
```

```
This is EasyBuild 1.8.1 (framework: 1.8.1, easyblocks: 1.8.1)
```

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

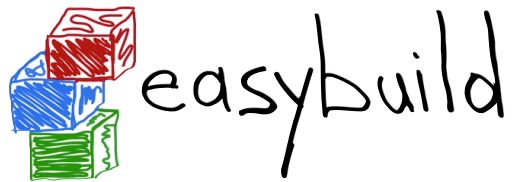
```
$HOME/.local/easybuild/modules/all
```

You can instruct EasyBuild otherwise by configuring it, using:

- a configuration file, e.g., `$HOME/.easybuild/config.py`
- environment variables, e.g., `$EASYBUILD_INSTALLPATH`

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

(note: documentation needs work)

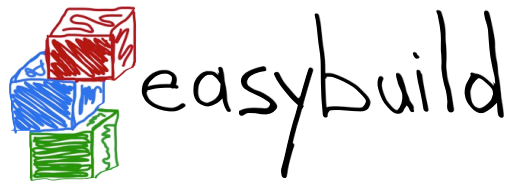


'Quick' demo for the impatient

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




- downloads all required sources (best effort)
- constructs *goalf* toolchain, 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

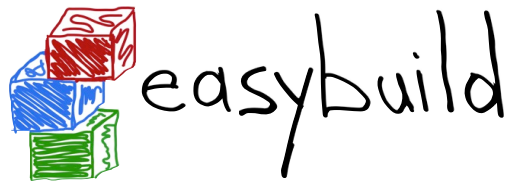
-  Python packages and modules forming the heart of EasyBuild
-  provides (loads of) supporting functionality
-  very modular design w.r.t. toolchains and easyblocks

easyblock

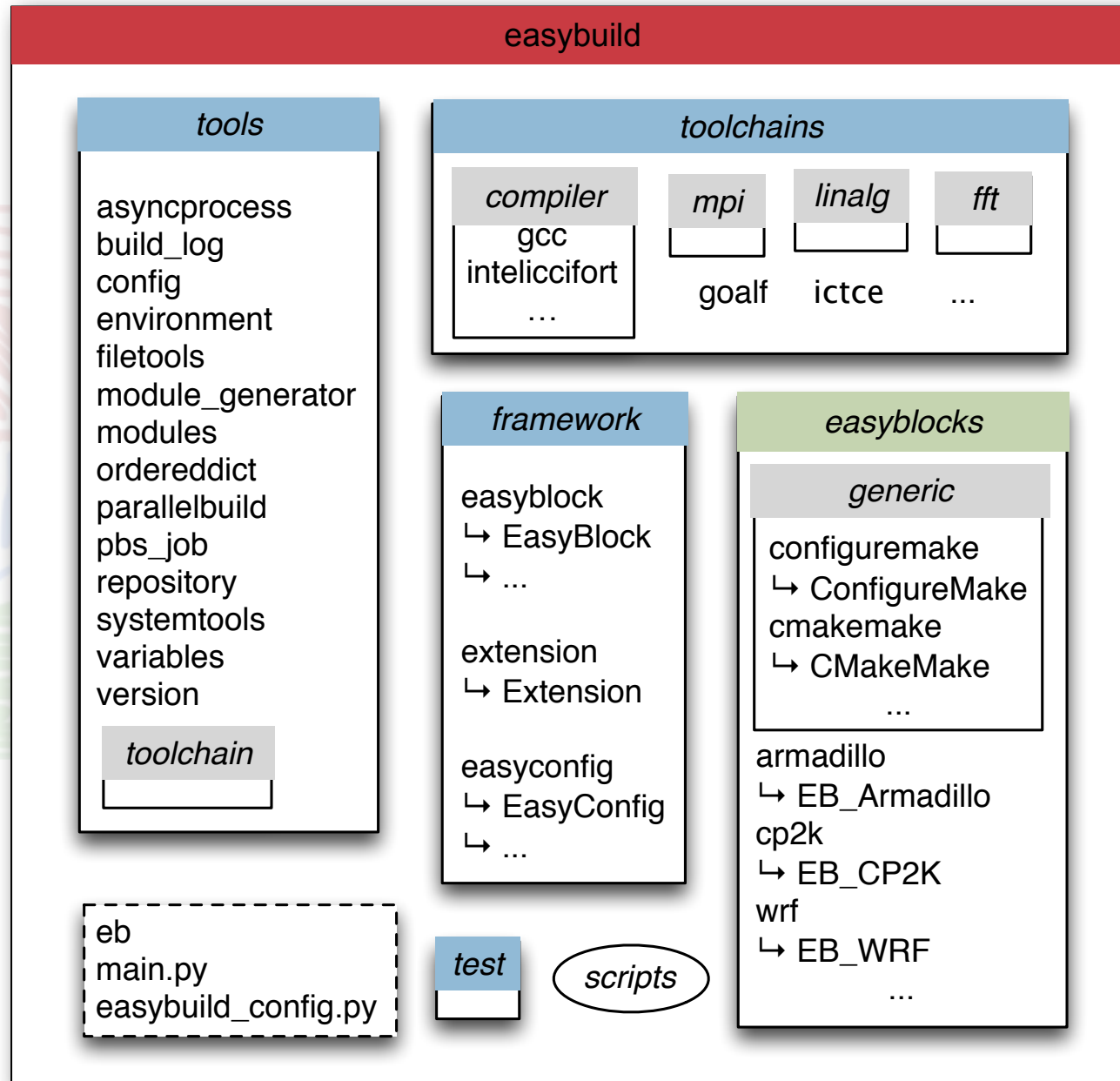
-  Python module providing implementation of a build procedure
-  can be generic or software-specific

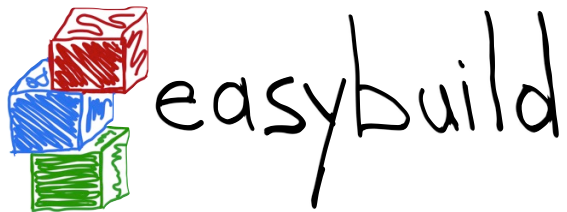
easyconfig file (.eb)

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



High-level design (little bit outdated)

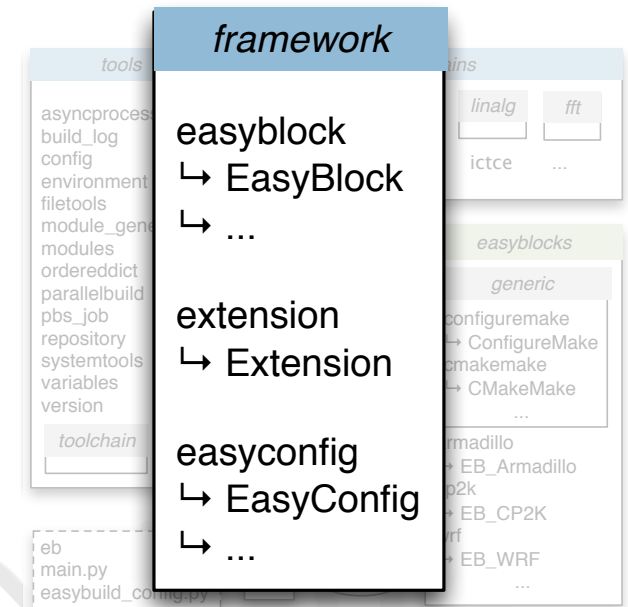


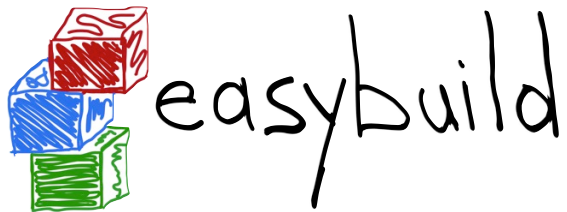


High-level design

framework package

- core of EasyBuild
- 'abstract' class Easyblock
 - should be subclassed
- EasyConfig class (in easyconfig package)
- Extension class
 - e.g., to build and install Python packages, R libraries, ...

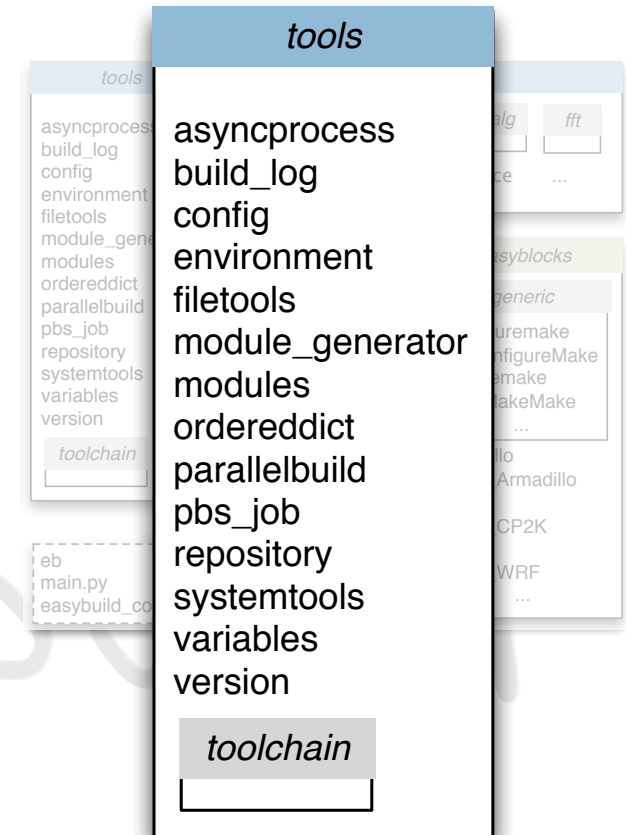




High-level design

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



toolchains package

- support for compiler toolchains

- relies on *tools.toolchain*

- toolchains are defined in here

- organized in subpackages:

- toolchains.compiler*

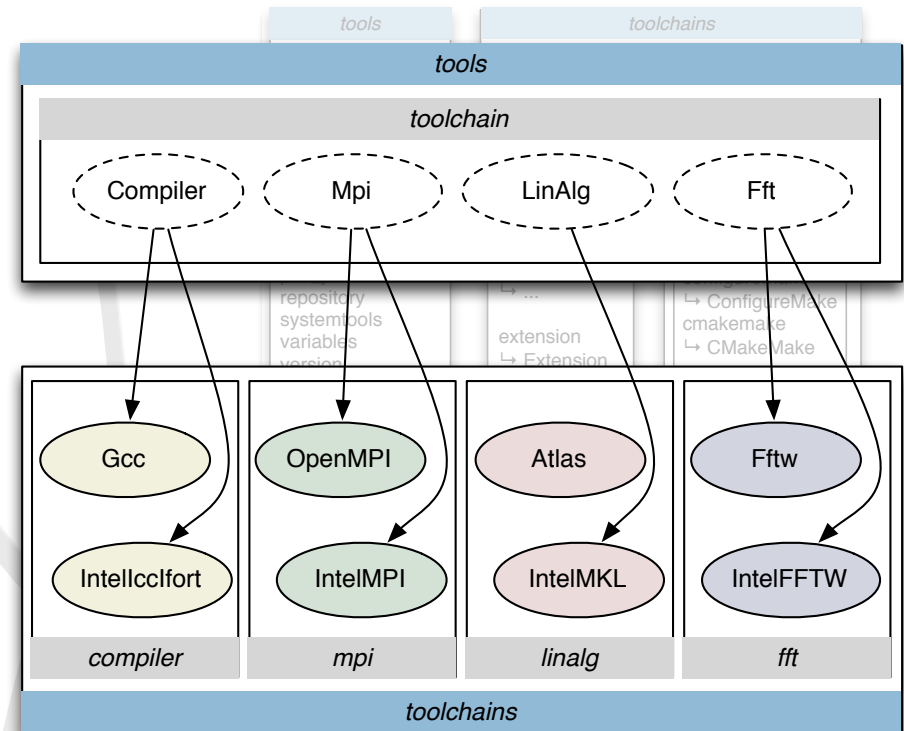
- toolchains.mpi*

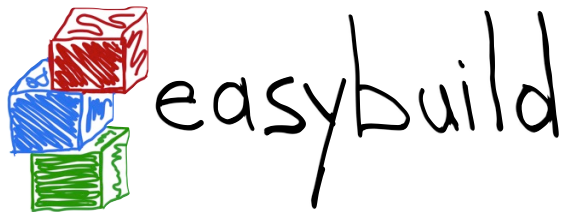
- toolchains.linalg* (BLAS, LAPACK, ...)

- toolchains.fft*

- very modular design for allowing extensibility

- plug in a Python module for compiler/library to extend it





High-level design

test package

- unit testing of EasyBuild

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

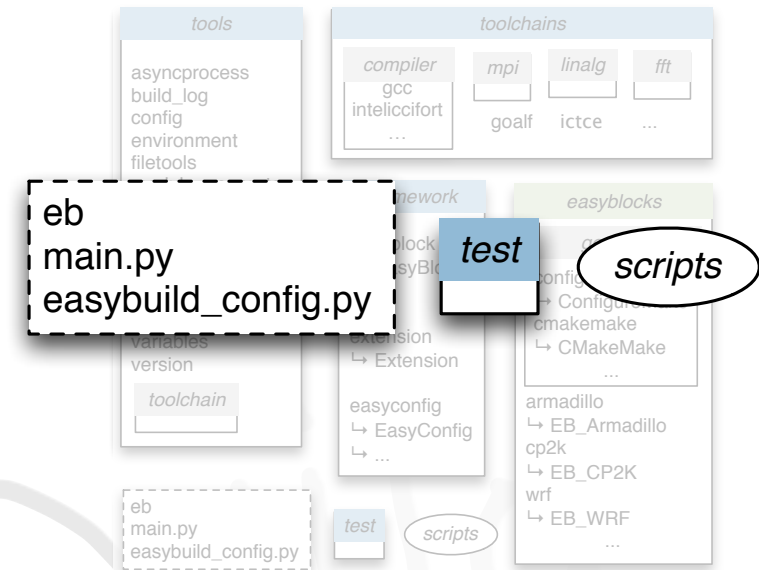
collection of scripts

- mainly for EasyBuild developers

main.py script + *eb* wrapper

default EasyBuild configuration file

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

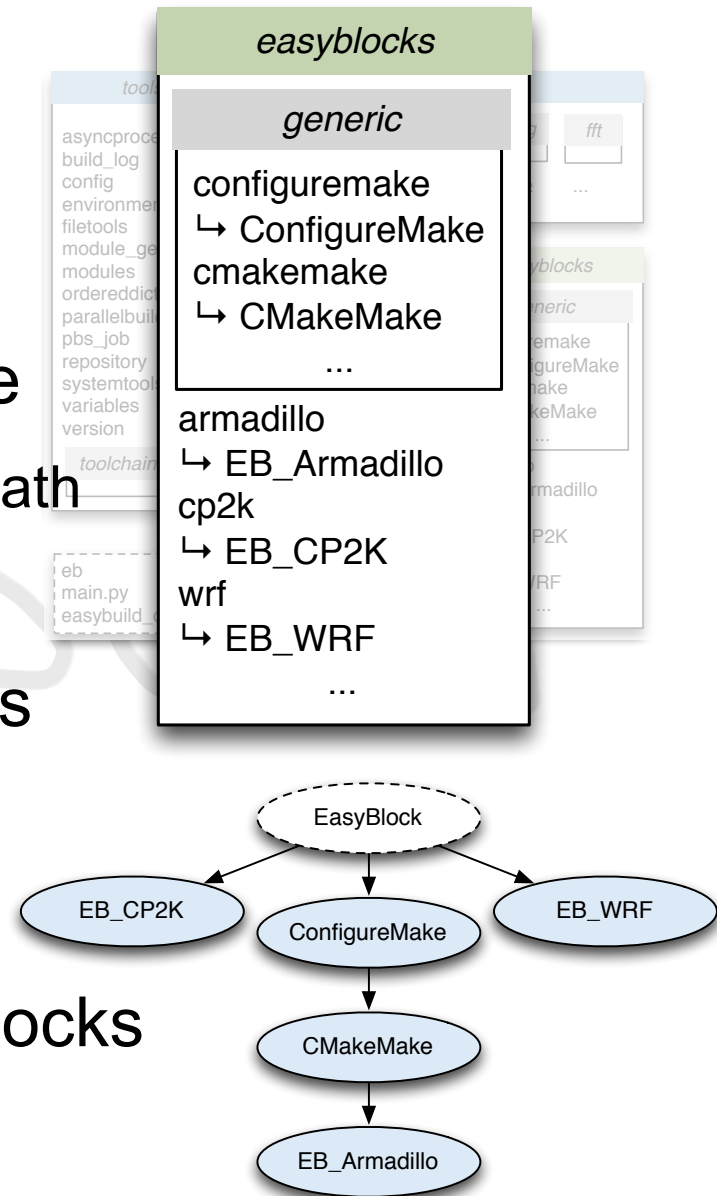


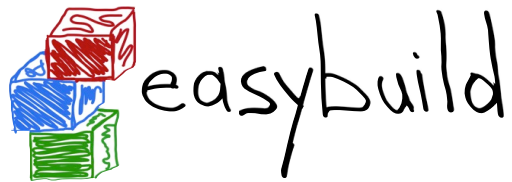


High-level design

easyblocks package

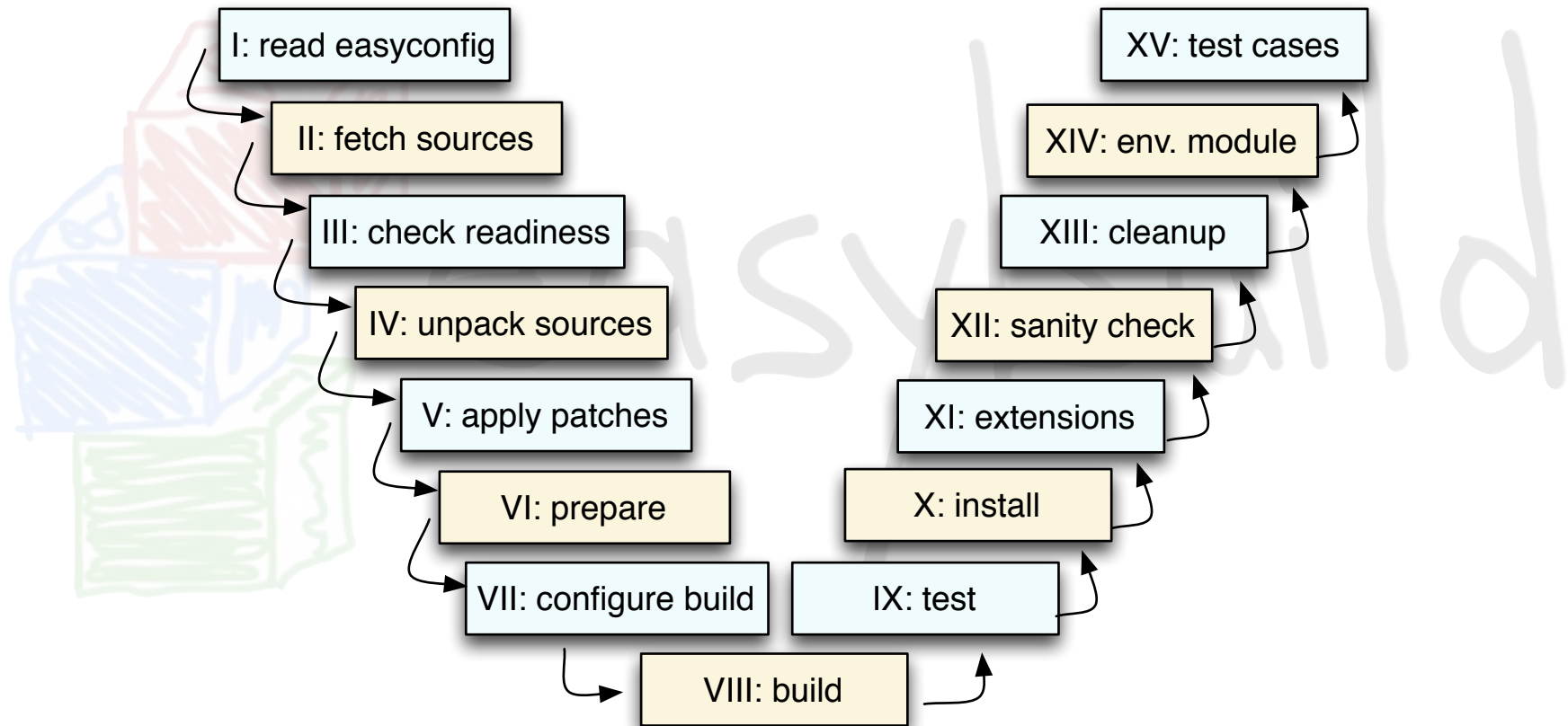
- build procedure implementations
- modular design, dynamically extensible
 - add your easyblock in the Python search path
 - EasyBuild will pick it up
- easyblocks.generic: generic easyblocks**
 - custom support for groups of applications
 - e.g., ConfigureMake, CMakeMake, ...
- easyblocks: application-specific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



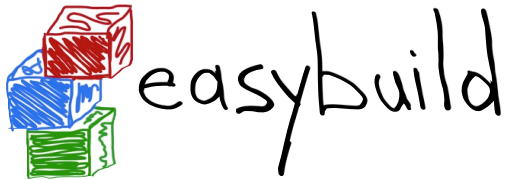


Step-wise install procedure

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**

- 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>

The screenshot shows the Jenkins EasyBuild dashboard. On the left, there is a sidebar with navigation links: Home, Build history, Project Relationship, and Check File Fingerprint. Below these are sections for Build Queue (No builds in the queue) and Build Executor Status (2 idle). The main area displays a table of build history for the EasyBuild Quorum. The table has columns for Status (S), Weather (W), Name, Last Success, Last Failure, and Last Duration. The builds listed are:

S	W	Name	Last Success	Last Failure	Last Duration
●	☀	easybuild-framework_unit-test_hpcugent_devops	18 hr (#14)	23 days (#13)	6.8 sec
●	☀	easybuild-framework_unit-test_hpcugent_master	4 days 16 hr (#5)	N/A	7.3 sec
●	☀	easybuild-5.8-rc0-test_devops	4 days 19 hr (#2)	N/A	0.35 sec
●	☀	easybuild-5.8-rc0-test_master	6 days 14 hr (#2)	N/A	0.4 sec
●	☀	easybuild-5.8-rc0-test_released	4 days 3 hr (#1)	N/A	0.31 sec

At the bottom of the dashboard, there is a legend for build status icons: a blue circle for 'PASS for all', a yellow circle with a sun for 'PASS for failures', and a red circle with a lightning bolt for 'PASS for just latest builds'. The URL at the bottom is <http://hpcugent.github.com/easybuild/>.



unit tests (framework)



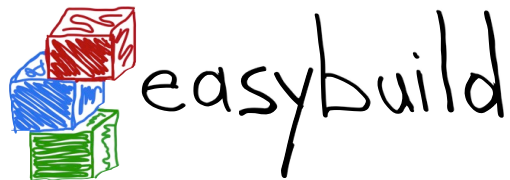
unit tests (easyblocks)



unit tests (easyconfigs)



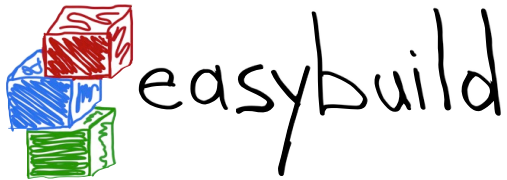
full regression test



List of supported software (v1.8.1)

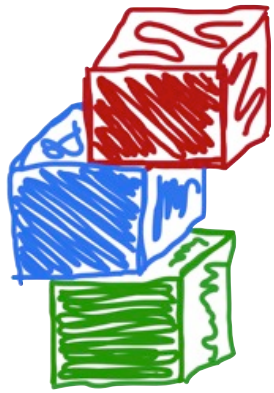
338 different software packages (1,710 example easyconfigs)

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 fasthack 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 Hypricc ifort imkl impi Infernal Inspector Instant lperf ipp itac JasPer Java Jinja2 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 SCOOP SCOTCH setuptools Shapely SHRIMP SLEPc SOAPdenovo Sphinx SQLite Stacks Stow SuiteSparse SWIG sympy Szip Tar tbb Tcl tccl tclsh 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



Current status

- **stable** since EasyBuild v1.0.0 released Nov 13th 2012 (at SC12)
- **EasyBuild v1.8.1** released October 14th 2013
 - planned monthly releases (v1.x.0), bugfix releases as needed
 - v1.9.0 planned for Nov 14th 2013, code freeze on Nov 7th 2013
- some **recent major features**:
 - support for using *Imod* as modules tool
 - support for custom module naming schemes
- more work to do:
 - bring **documentation** wiki up-to-date
 - support for **'fat' easyconfig** files
 - support for **more software** and additional **compiler toolchains**
- small **community**, growing steadily
 - UGent + other Flemish university associations (VSC partners),
 - University of Luxembourg, Cyprus Institute, Gregor Mendel Institute (Austria), ...



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

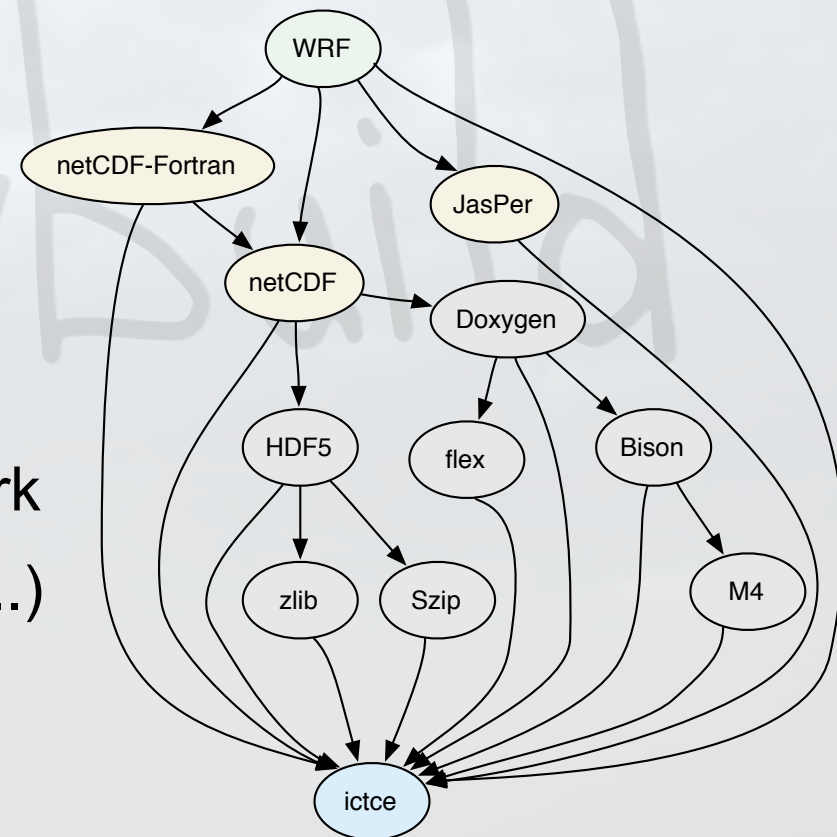
Twitter: [@easy_build](https://twitter.com/easy_build)

IRC: [#easybuild](https://freenode.net) on freenode.net



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
```



easybuild Use case: WRF - easyblock (1/3)

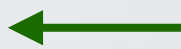
imports, class constructor,
custom easyconfig parameter

```
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)
```

**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



easybuild 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)

```
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 name and version →

→ **software website and description (informative)**

compiler toolchain specification and options →

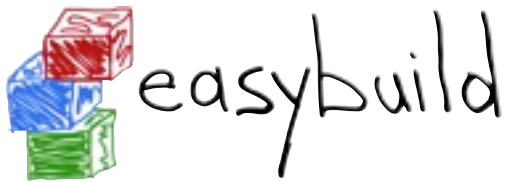
← **list of source files**

← **list of patches for sources**

← **list of dependencies**

custom parameter for WRF →

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



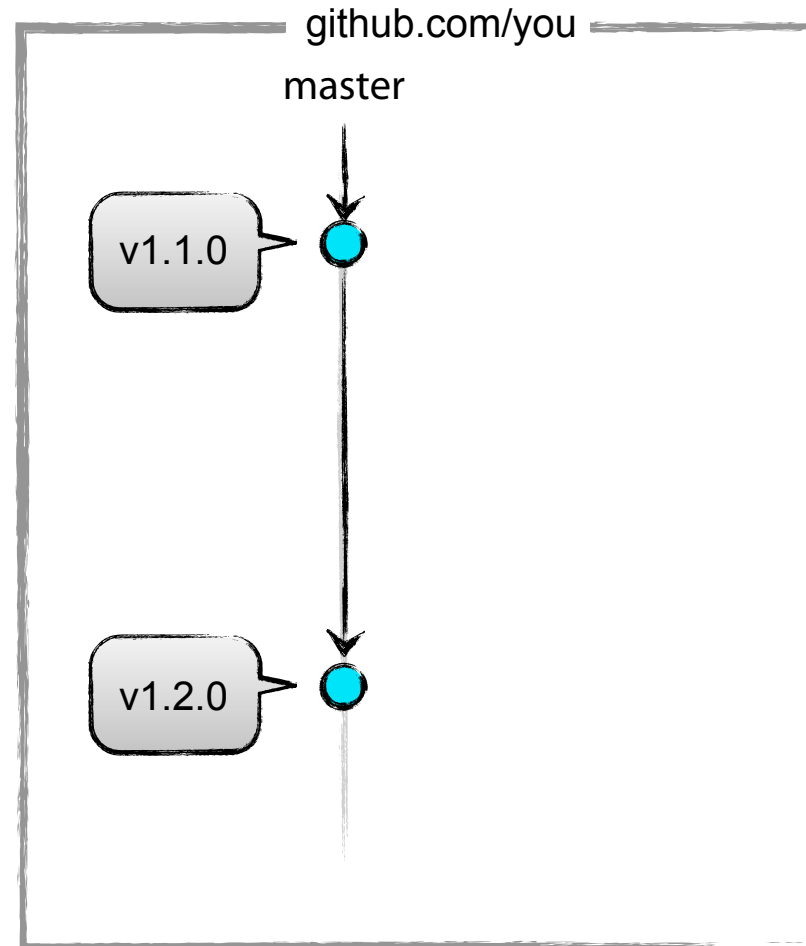
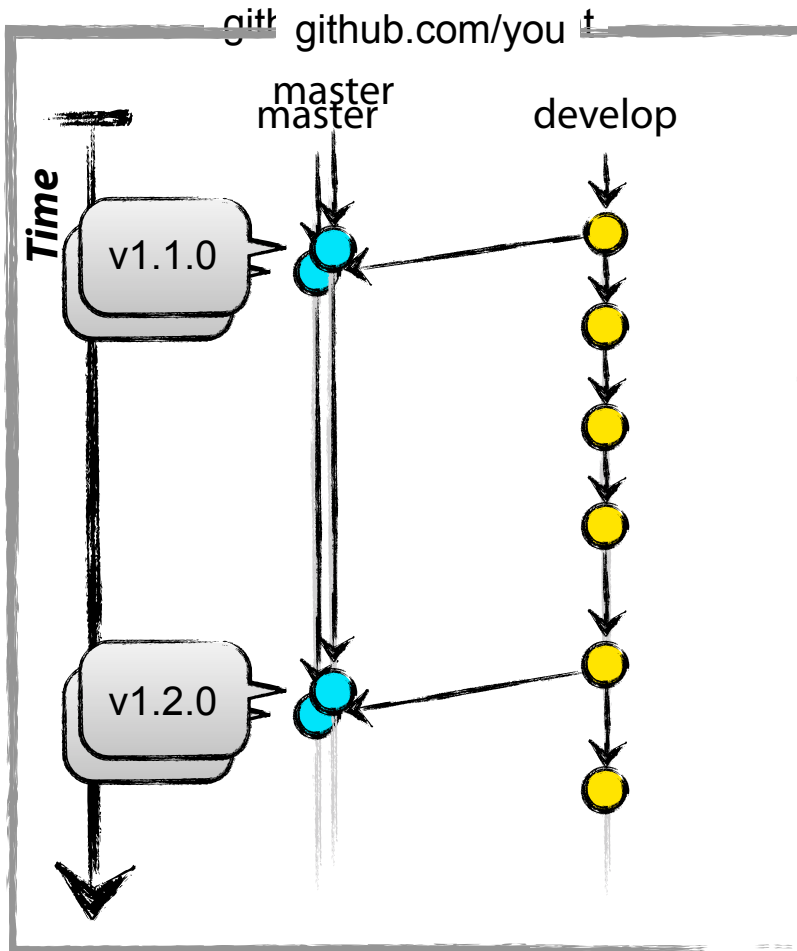
Development workflow with git

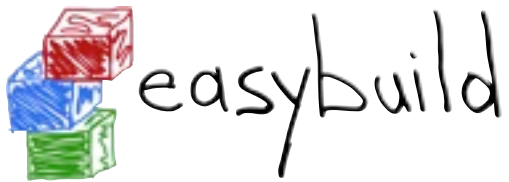
Setting up

fork repository on GitHub,
and clone a working copy



```
git clone git://github.com/you/reponame
```



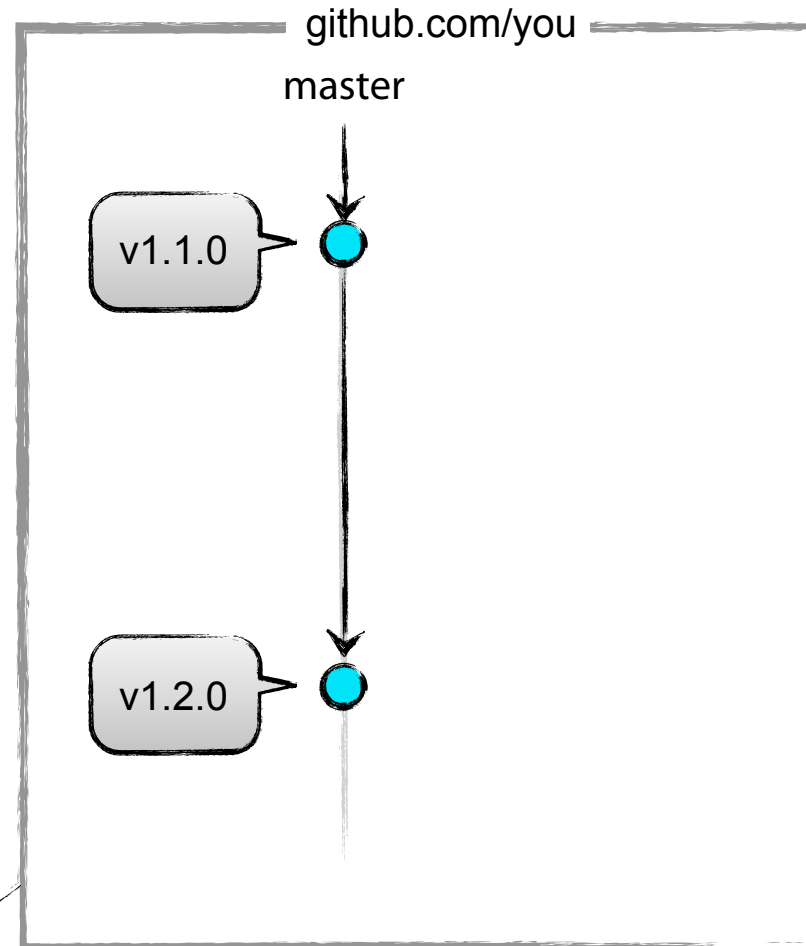
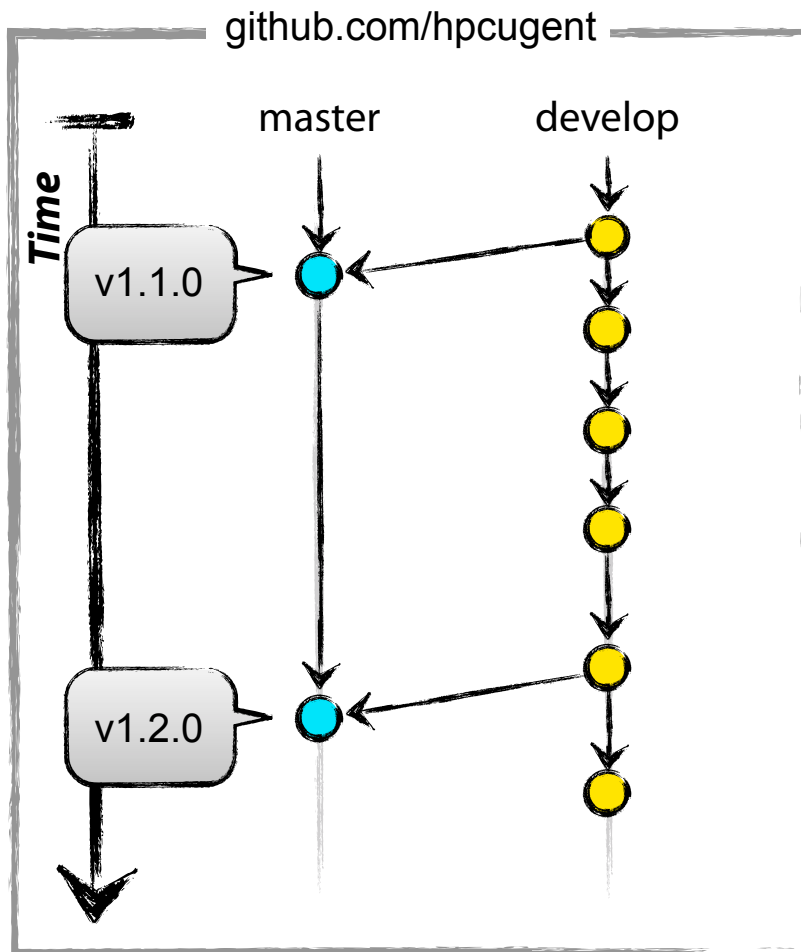


Development workflow with git

Setting up

define upstream remote repository

```
git remote add upstream  
git://github.com/hpcugent/reponame
```

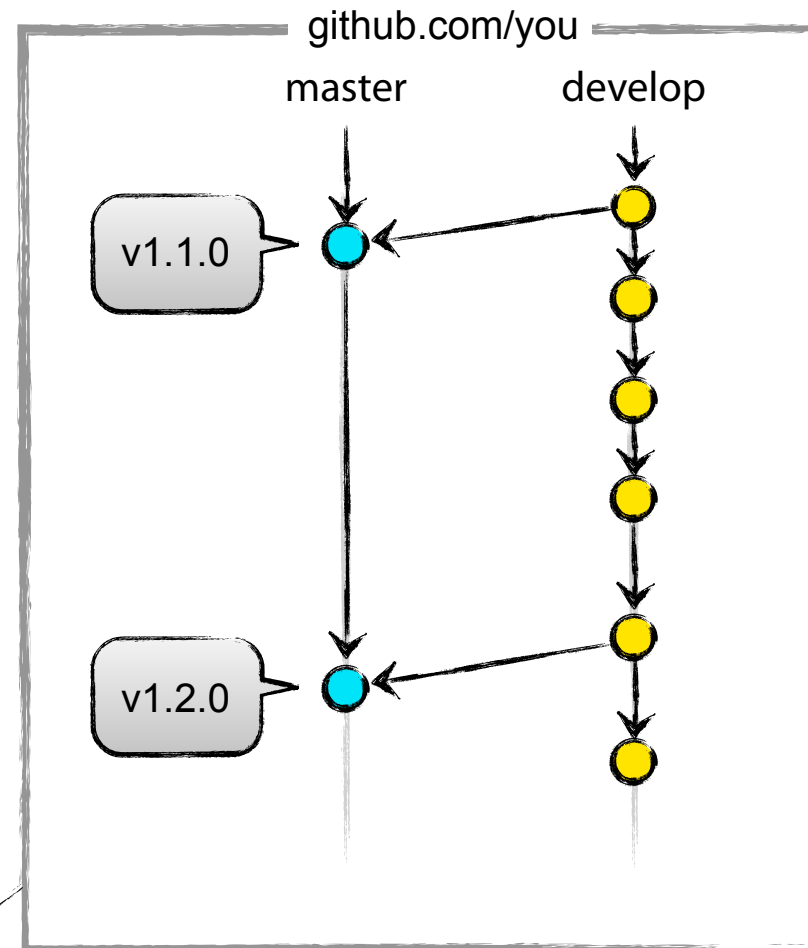
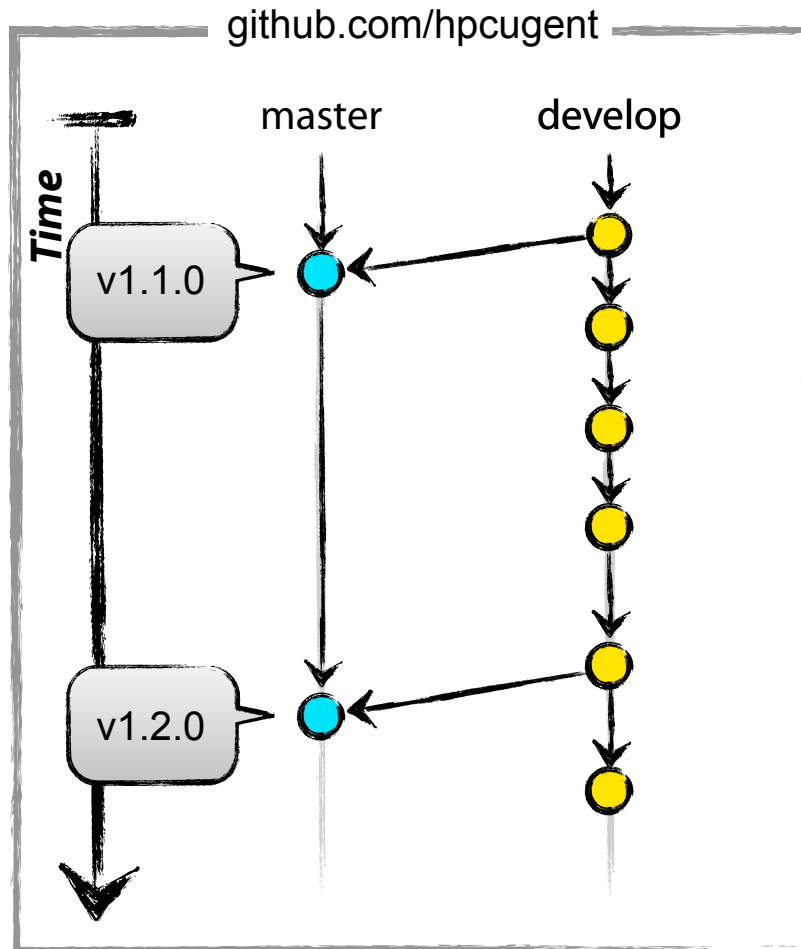


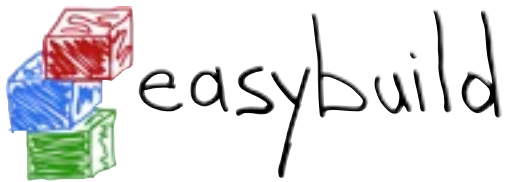
Development workflow with git

Setting up

pull in *develop* branch

```
git checkout -b develop  
git pull upstream develop
```



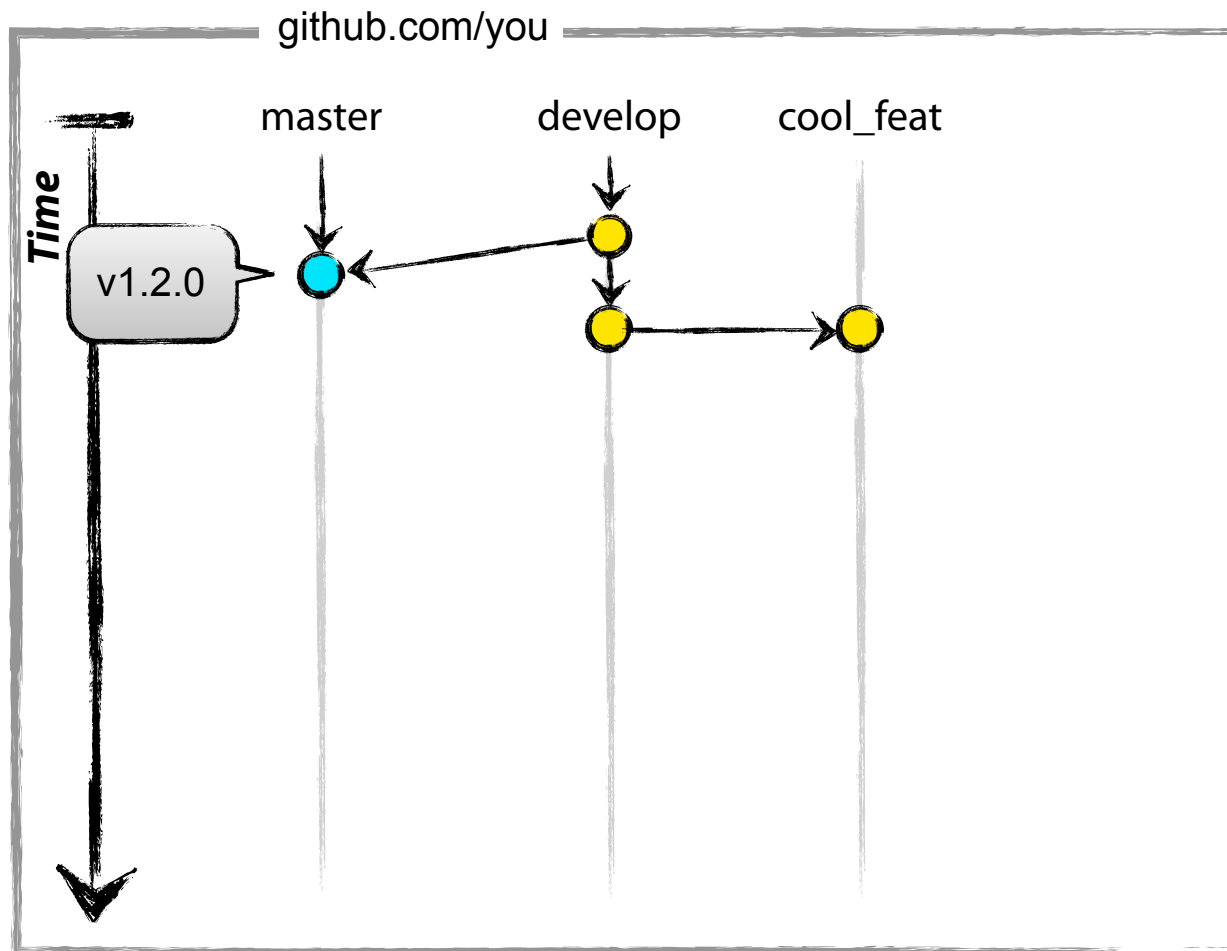


Development workflow with git

Implementing a feature

create a feature branch

```
git checkout develop  
git branch cool_feat  
git checkout cool_feat
```



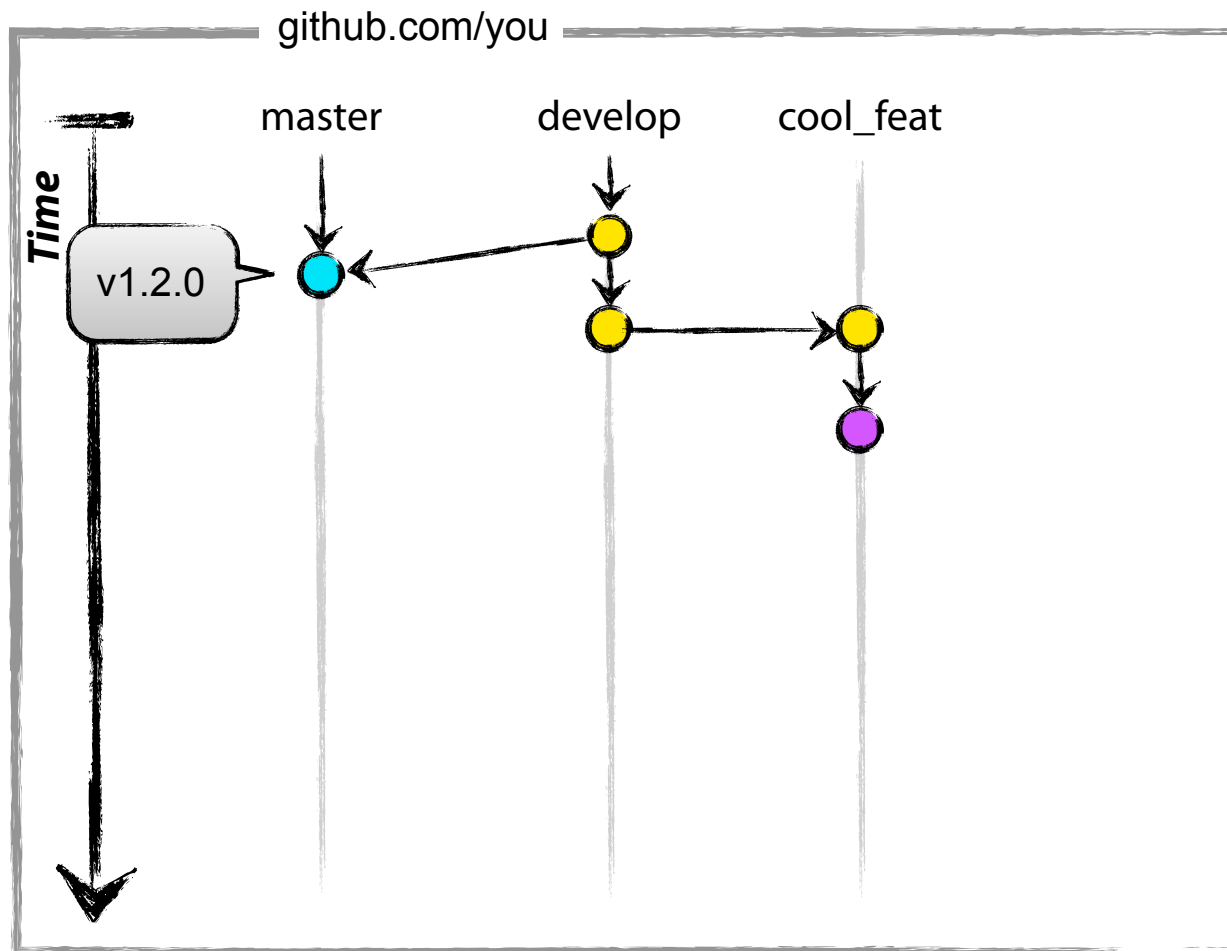


Development workflow with git

Implementing a feature

adjust code, stage and commit

```
vim code.py  
git add code.py  
git commit -m "new stuff"
```



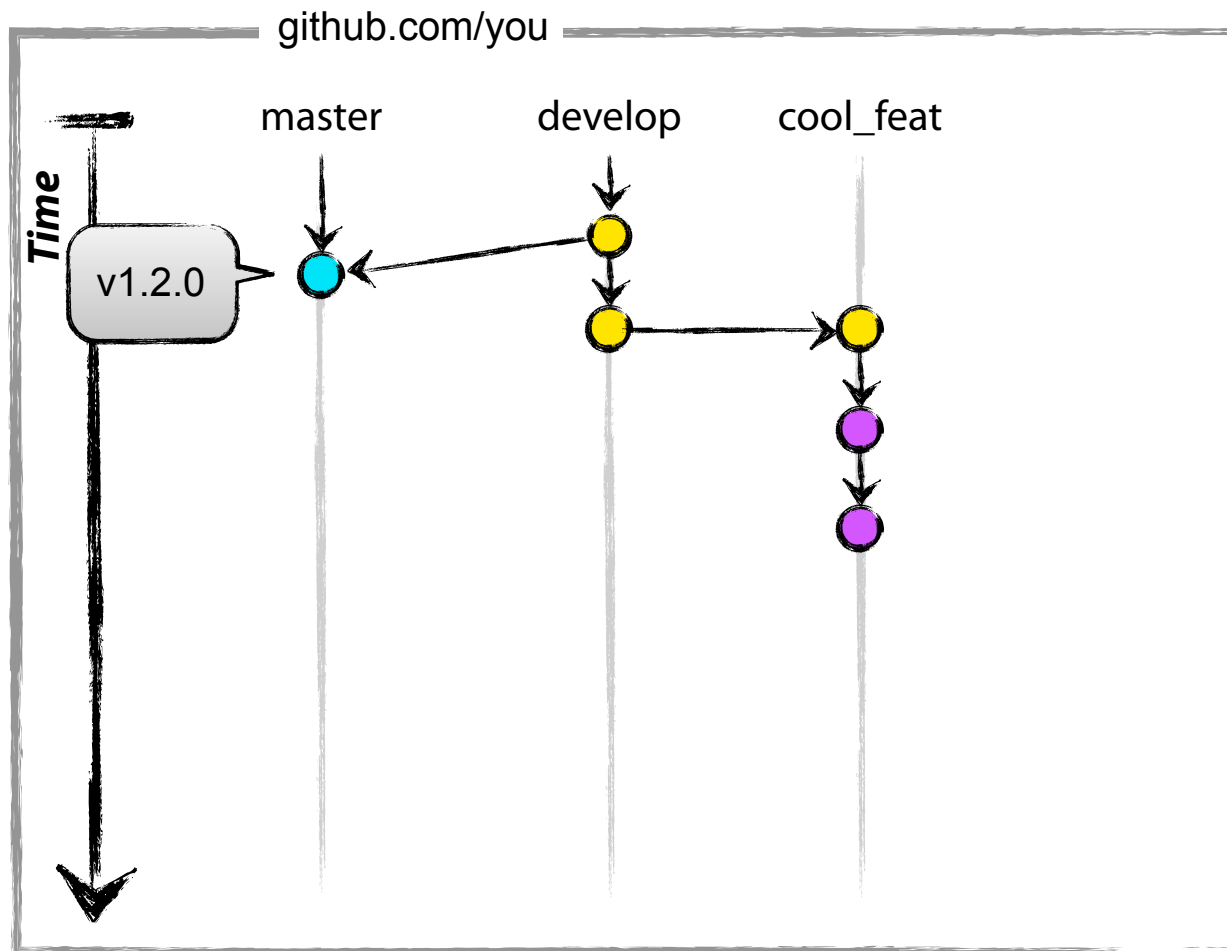


Development workflow with git

Implementing a feature

adjust code, stage and commit,
and fix that bug

```
vim code.py  
git add code.py  
git commit -m "bugfix"
```



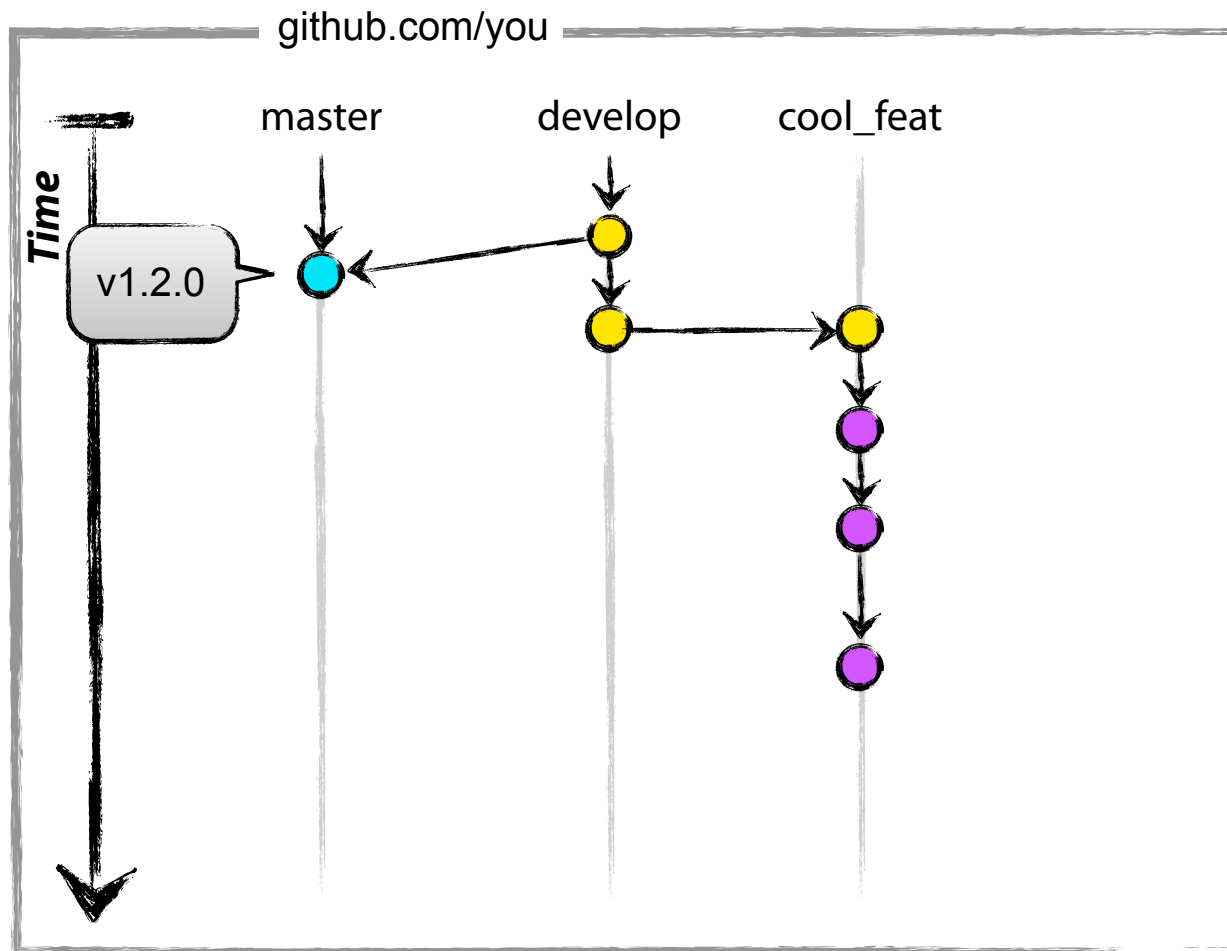


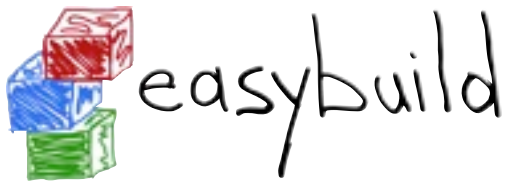
Development workflow with git

Implementing a feature

adjust code, stage and commit,
and fix that bug, and the typo too

```
vim code.py  
git add code.py  
git commit -m "typo (grr)"
```



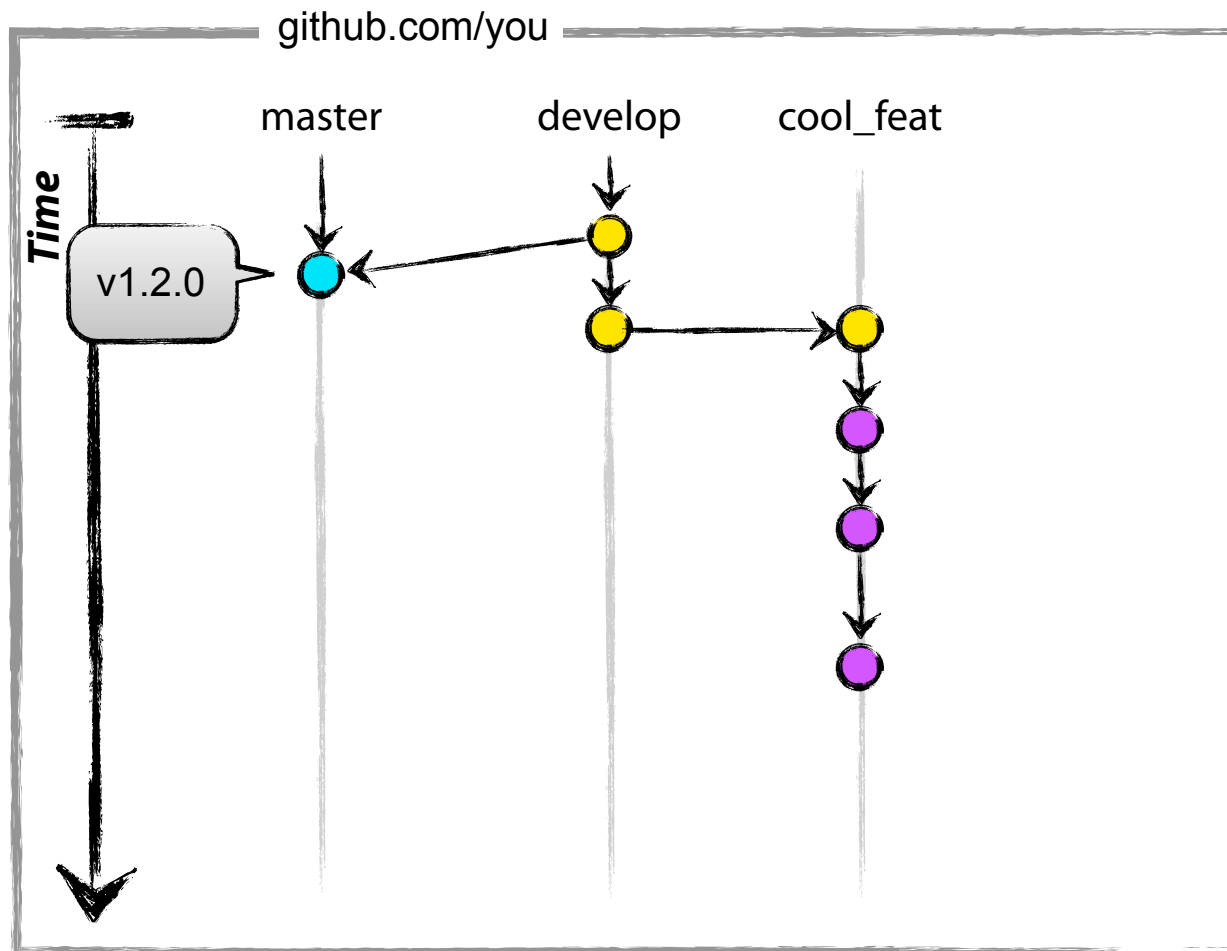


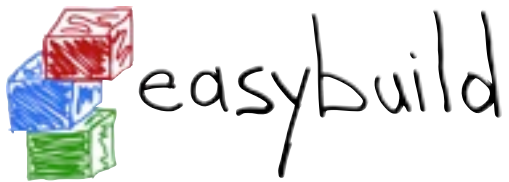
Development workflow with git

Contribute back!

push your feature branch
to your repository

```
git push origin cool_feat
```



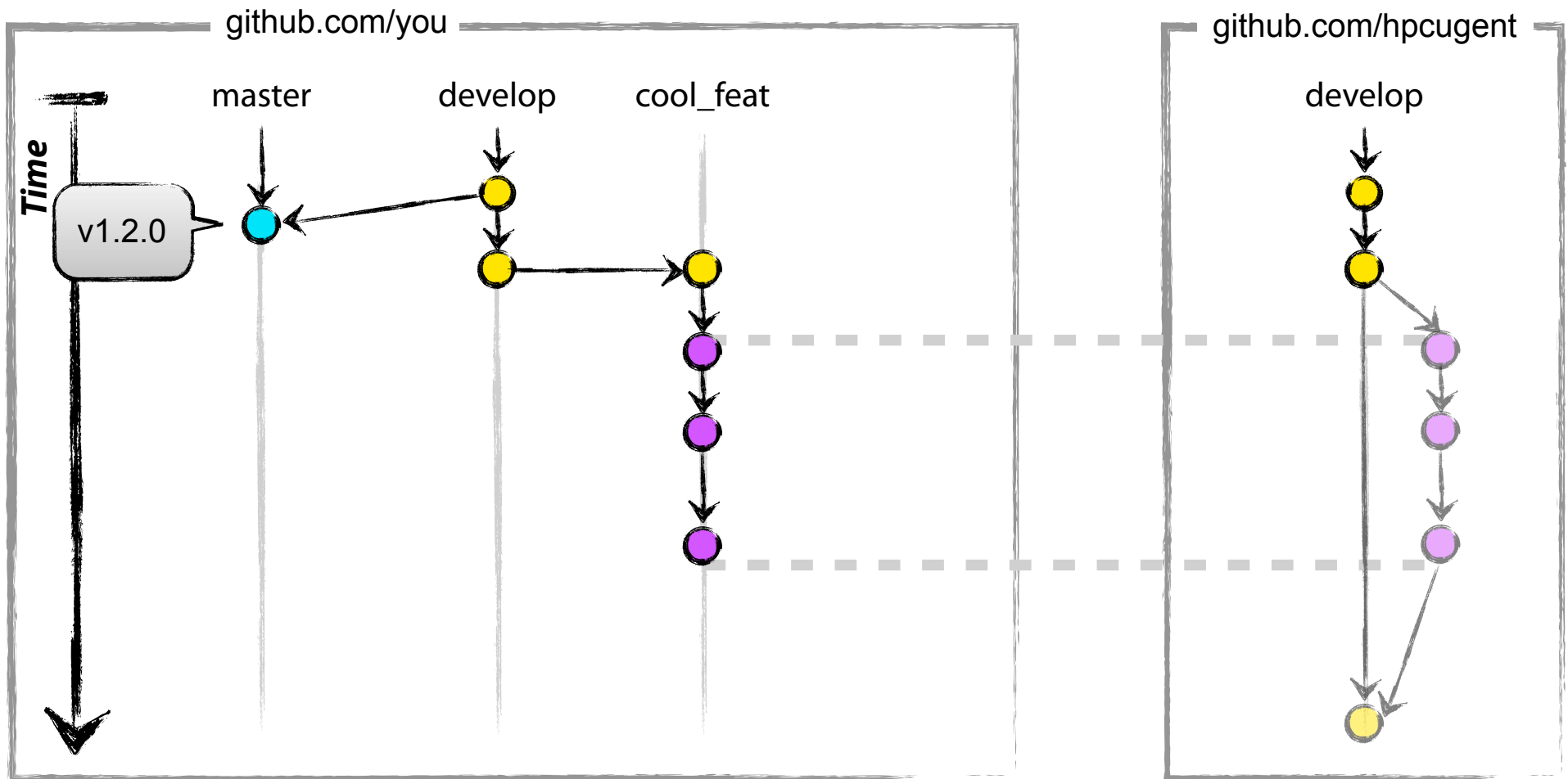


Development workflow with git

Contribute back!

create a pull request via GitHub

Pull Request



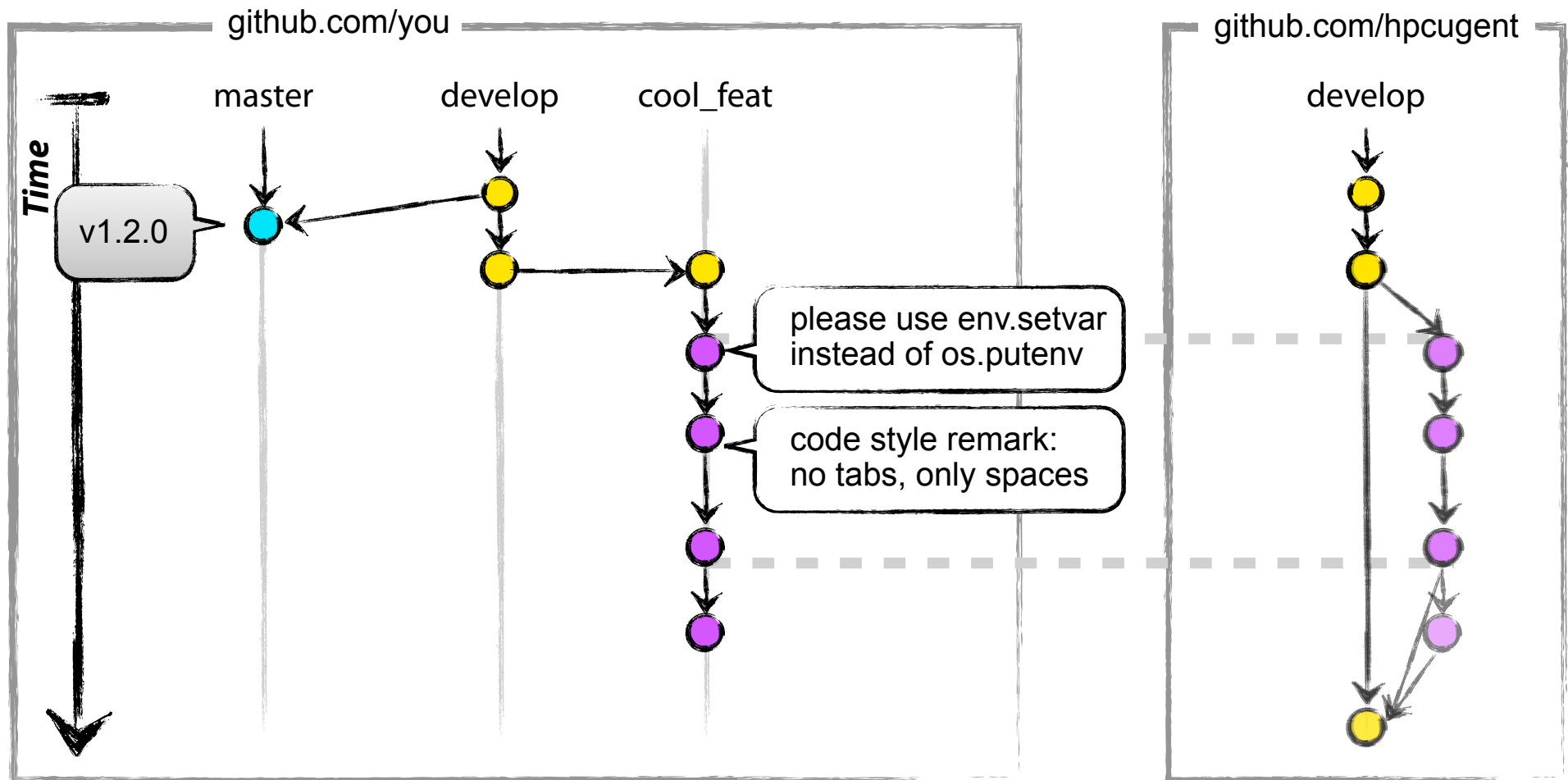


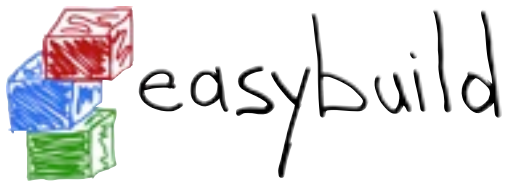
Development workflow with git

Contribute back!

await code review, fix remarks

```
vim code.py  
git add code.py  
git commit -m "fixed remarks"  
git push origin cool_feat
```



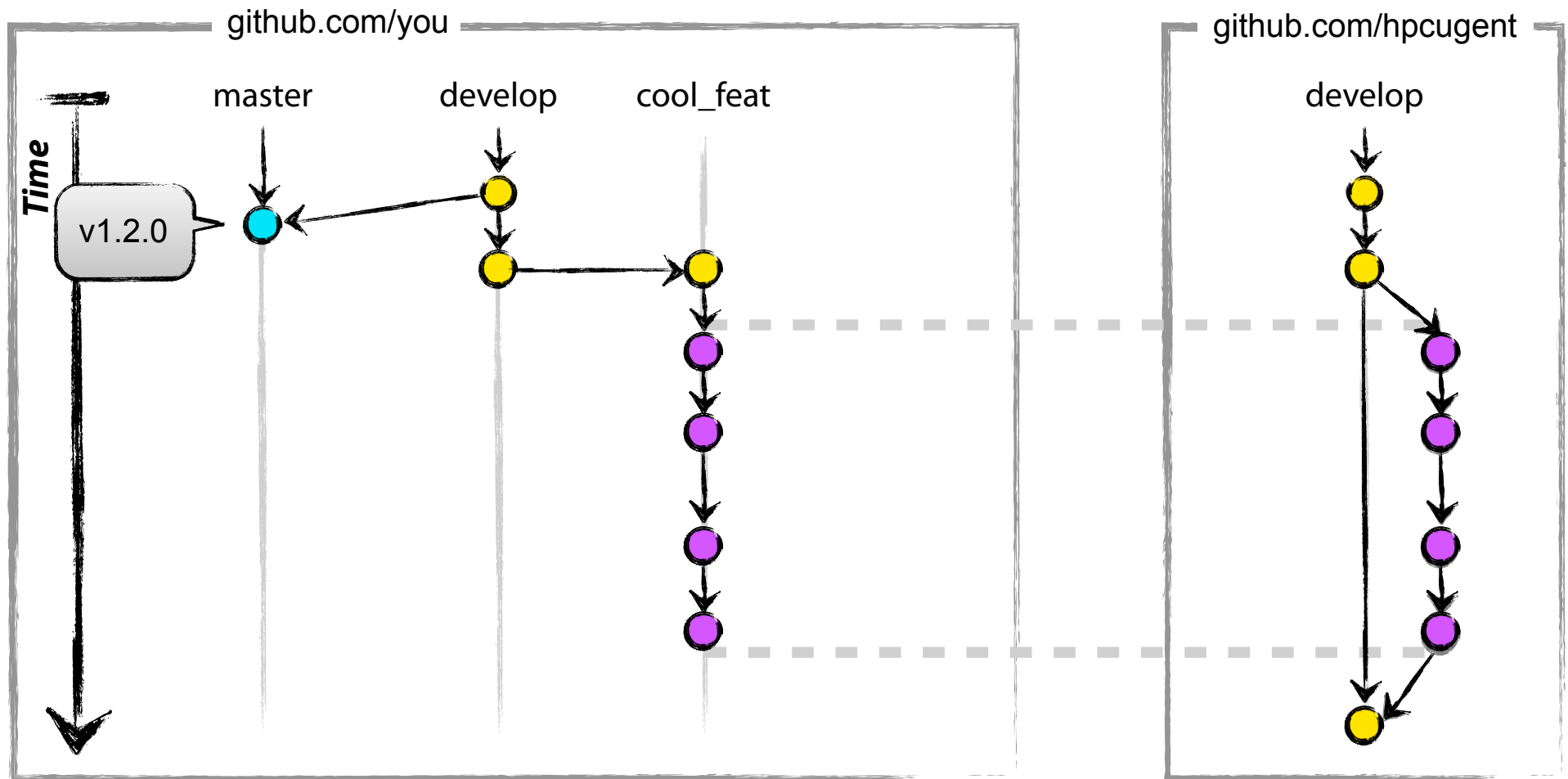


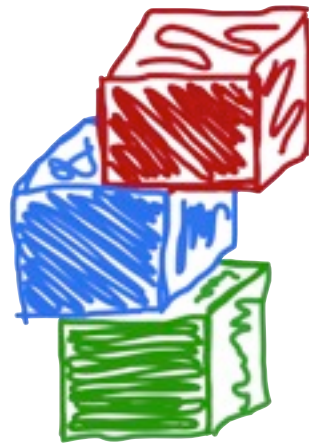
Developing workflow with git

Contribute back!

await the merge, update,
and clean up

```
git checkout develop  
git pull upstream develop  
git branch -d cool_feat
```





easybuild

building software with ease

introduction to EasyBuild
Oct. 15th 2013

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