Introduction to Monte Python

Benjamin Audren

Institute of Theoretical Physics
École Polytechnique Fédérale de Lausanne

16/05/2014

Outline

1. git, Github, and why it matters
   - Motivation
   - How to use it?
   - Design Strategy
2. How to contribute
3. Tests

Outline

Version Control

Survey

CVS, SVN, git, mercurial?

Who uses it daily (weekly? ever?)

Version Control

Survey

CVS, SVN, git, mercurial?

Who uses it daily (weekly? ever?)

Why bother?

Any (complex enough) code **has** bugs. We are in science, how do we deal with it?

Version Control

Survey

CVS, SVN, git, mercurial?

Who uses it daily (weekly? ever?)

Why bother?

Any (complex enough) code **has** bugs. We are in science, how do we deal with it?

Solution

open-source, **documented**, **version-controlled** software

Motivation

What is Version Control ?

- System that keeps tracks of different **versions** of a code (several file), tracking its entire **history** of creation.
- Accessible from a **central repository**, that stores all the versions, and allow communication between developers.
- Can revert to a previous version to **reproduce** exactly the behaviour at a certain time (bug hunting)

Why does it matter?

Version Control in general

Pros

- Hard-drive failure safe
- Programmer's stupidity safe
- Bug-hunting made easier
- Collaborating
- Robustness

Why does it matter?

Version Control in general

Pros

- Hard-drive failure safe
- Programmer's stupidity safe
- Bug-hunting made easier
- Collaborating
- Robustness

Cons

- getting use to (g)it ...

Softwares

Version Control Softwares

- CVS (Concurrent Versions System)
- Apache Subversion (SVN)
- Git

Softwares

Version Control Softwares

- CVS (Concurrent Versions System)
- Apache Subversion (SVN)
- **Git**

Github

CLASS and Monte Python

Code hosted on Github

- nice code browsing
- diff and downloads of old versions
- easy integration of contributions (forks)
- Wiki complementing the documentation
- Issues ($\simeq$ forum, suggestions)

How to use it?

Without too much effort?

Install git on your machine

- `Debian-based`: sudo apt-get install git
- `Mac OS X`: sudo port install git-core +svn +doc
- `Windows`: `http://msysgit.github.com/`

How to use it?

Without too much effort?

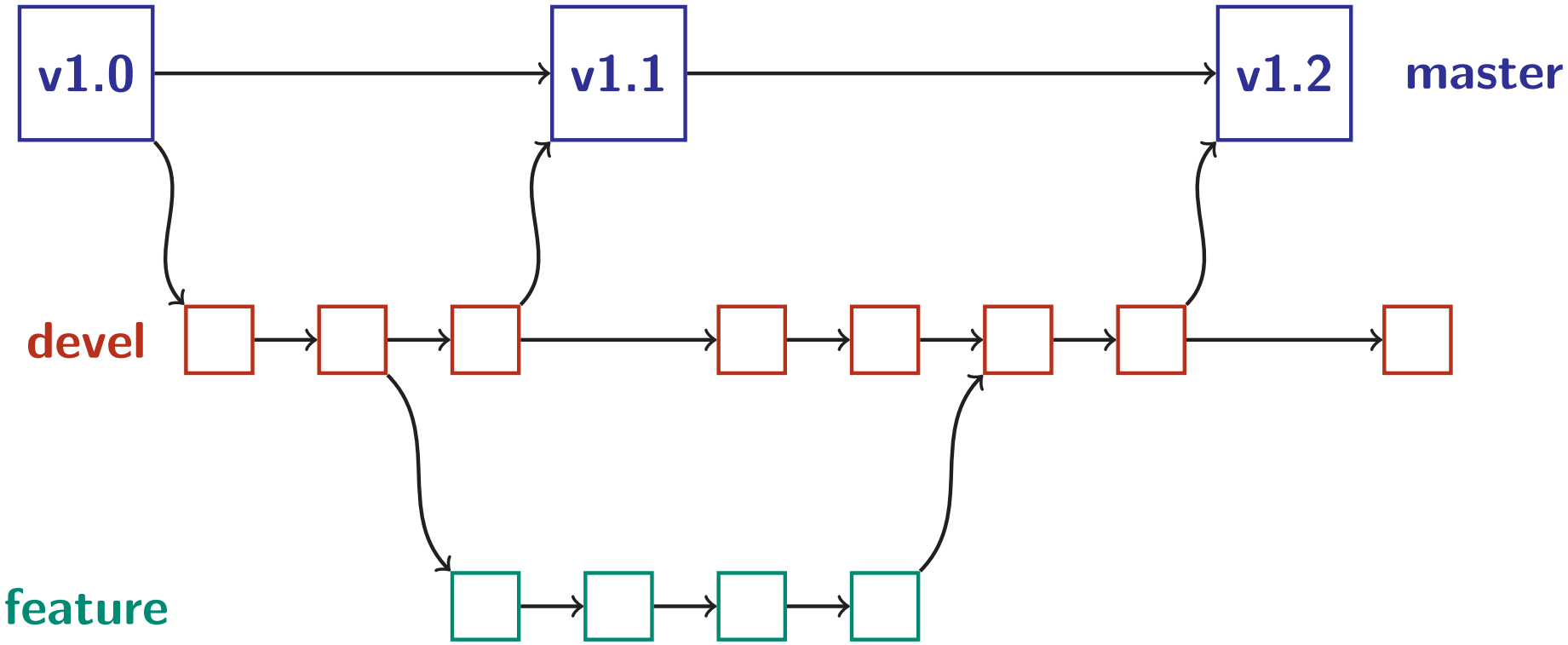
Install git on your machine

- `Debian-based`: sudo apt-get install git
- `Mac OS X`: sudo port install git-core +svn +doc
- `Windows`: http://msysgit.github.com/

Install CLASS and Monte Python via git...

```
cd codes/
git clone https://github.com/lesgourg/class_public.git class
git clone https://github.com/baudren/montepython_public.git montepython
```

Design Strategy for both CLASS and Monte Python



Design Strategy for both CLASS and Monte Python

with words

- **Master** branch (public and private) and **devel** branch(es)

Design Strategy for both CLASS and Monte Python

with words

- **Master** branch (public and private) and **devel** branch(es)
- **master** will - should - contain only stable releases, that you can roll back to in case. Between two versions, there will be a new feature.

Design Strategy for both CLASS and Monte Python

with words

- **Master** branch (public and private) and **devel** branch(es)
- **master** will - should - contain only stable releases, that you can roll back to in case. Between two versions, there will be a new feature.
- **devel** is for the day to day work, potentially ugly commits, that end up doing something.

Design Strategy for both CLASS and Monte Python

with words

- **Master** branch (public and private) and **devel** branch(es)
- **master** will - should - contain only stable releases, that you can roll back to in case. Between two versions, there will be a new feature.
- **devel** is for the day to day work, potentially ugly commits, that end up doing something.
- **feature** branches are for developing big modifications without perturbing day to day work.

Using Github

Practising with the website

- go to `http://github.com/lesgourg/class_public`
- look at the latest commits, the diff,
- download version 2.1.2

Outline

How to contribute

Worflow

- on Github: fork the repository
- You now have your own version of CLASS
- Create a new branch `my_awesome_model`
- Modifying and commiting
- Send us the modification as a Pull-Request
  https://github.com/lesgourg/class_public/wiki/Contributing

How to contribute

Worflow

- on Github: fork the repository
- You now have your own version of CLASS
- Create a new branch `my_awesome_model`
- Modifying and commiting
- Send us the modification as a Pull-Request
  `https://github.com/lesgourg/class_public/wiki/Contributing`

For instance: `https://github.com/baudren/montepython_public/pull/12`

But Github is not free!

True

But you do not need it. Have a machine somewhere in your lab with ssh access, always on? Set up your remote repository! It is that easy.

But Github is not free!

True

But you do not need it. Have a machine somewhere in your lab with ssh access, always on? Set up your remote repository! It is that easy.

How?

- on the remote machine: `git clone github_montepython`
- on your machine:
  `git clone ssh://memyself@remote.mpa.kw.we.de/path/to/montepython montepython`
- that's it

Outline

For the motivated...

For the motivated...
Tests!

Tests are essential

- asking CLASS with some **combination of parameters** might uncover a bug

For the motivated...

Tests are essential

- asking CLASS with some **combination of parameters** might uncover a bug
- Should we store hundreds of parameters and check at **every modification** that CLASS still works?

For the motivated...

Tests are essential

- asking CLASS with some **combination of parameters** might uncover a bug
- Should we store hundreds of parameters and check at **every modification** that CLASS still works?
- No!

For the motivated...

Tests are essential

- asking CLASS with some **combination of parameters** might uncover a bug
- Should we store hundreds of parameters and check at **every modification** that CLASS still works?
- No!
- Automatize away!

How does it work?

CLASS

Principle: using nosetests

In the file `python/test_class.py`
Different scenarios tested, each with different outputs, each with different gauge, each w/ or w/o non linear corrections..

How does it work?

CLASS

Principle: using nosetests

In the file python/test_class.py
Different scenarios tested, each with different outputs, each with different gauge, each w/ or w/o non linear corrections..

```
    @parameterized.expand(
        itertools.product(
            ('LCDM',
             'Mnu',
             'Positive_Omega_k',
             'Negative_Omega_k',
             'Isocurvature_modes', ),
            ({'output': ''}, {'output': 'mPk'}, {'output': 'tCl'},
             {'output': 'tCl pCl lCl'}, {'output': 'mPk tCl lCl', 'P_k_max_h/Mpc':10},
             {'output': 'nCl sCl'}, {'output': 'tCl pCl lCl nCl sCl'}),
            ({'gauge': 'newtonian'}, {'gauge': 'sync'}),
            ({}, {'non linear': 'halofit'})))
    def test_wrapper_implementation(self, name, scenario, gauge, nonlinear):
```

How does it work?

Monte Python

Way to ensure all the default behaviour

check that:

- the code complains with no param file
- the param file is read only once
- the cosmo module has all the needed functions
- the data is well initialized from param file
- that MH, CH, NS and IS are working