tutoriel

June 14, 2022

1 Introduction to code versioning with git

schedule: - introduction to files history - introduction to Git, a system of code versioning - the Git cycle - branches

support: jupyter notebook running into the `jupyter/minimal-notebook` docker container - `docker run -it --rm -p 8888:8888 --user root -e NB_USER="tutoriel_git" -e CHOWN_HOME=yes -v "${PWD}:/home/${NB_USER}" jupyter/minimal-notebook` - in a browser, open the last given URL http://127.0.0.1:8888/lab?token=xxx - copy the notebook `tutotiel_git.ipynb` (and its cognate `images` repository) into the `${PWD}/tutotiel_git/` repository (or open a new python notebook)

note: - this notebook is runing with a Python kernel: use `%%sh` for shell (bash) in code cells - the docker container binding need to use `cd ${PWD}/xxx` in each code cell to work into another directory - at the end of the notebook, the `tutoriel_git` repository will look like:
`FAIR_bioinfo_github README.md first_git_example file1.txt file2.txt tutoriel_git.ipynb`

1.1 Really need of a files history?

Most researchers are primarily collaborating with themselves," Tracy Teal explains. "So, we teach it from the perspective of being helpful to a 'future you'."

1.2 Files history, a good practice for reproducible research

"Rule 4: Version Control All Custom Scripts"

1.3 Code control version

Definition: version control, revision control, source control, or source code management: class of systems responsible for managing changes to files

Feature: each revision is associated with a timestamp and the person making the change. Revisions can be compared, restored, and merged

Software: SVN, Git, Mercurial, GNU arch, etc

We choose Git.

1.4 Git vs. GitHub

- will track and version your files - enables you to collaborate with … yourself - open source license GPL (GNU General Public License) - created in 2005 by Linus Torvalds for the development of the Linux kernel

- stores online Git repositories - enables you to collaborate with others (and yourself!) - sources belong to GtHub - first commit in 2007 by Chris Wanstrath

1.5 Git concepts, Git objects

working directory: a user private copy of a whole repository of interest

clone: a local copy of a repository (include all commits and branches), the original repository can be local, or remote (http access)

commit: a git object, the snapshot of your entire repository compressed into a SHA; the command the saves changes by creating the snapshot

HEAD: pointer representing your current working commit. Can be moved (git checkout) to different branches, tags, or commits

branch: a lightweight movable pointer to a commit

merge: combines remote tracking branche into current local branch

Revision graph:

staging area: list of files of the working directory that will be considered for next commit (ie. could be not all the modified files)

1.6 Git setup

Git configuration: check the configuration of your git `user.name` with:

[1]:
```
%%sh
git config --list
```

if not yet done (nothing displayed), tell git our identity:

[2]:
```
%%sh
git config --global user.name 'clairetn'
git config --global user.email 'claire.ctn@gmail.com'
git config --list
```

```
user.name='clairetn'
user.email='claire.ctn@gmail.com'
```

Git repository intitailisation: The initialisation (red arrow) is the creation of a `.git` repository:

Here are 3 ways to initialise a git repository: - `git init`: inside an existing folder (possibly containing files) - `git init myproject`: create folder "myproject" + initializes the .git subfolder inside it - `git clone /gitfolder/path /new/path`: copy the existing git repository to a new one

Initalise a git repository:

[3]:
```
%%sh
git init first_git_example
```

```
Initialized empty Git repository in
/home/jovyan/tutoriel_git/first_git_example/.git/
```

Observe the git folder:

[4]:
```
%%sh
ls -lah first_git_example
```

```
total 12K
drwxr-xr-x 3 1001 1001 4.0K Jun 13 00:33 .
drwxrwxr-x 6 1001 1001 4.0K Jun 13 00:33 ..
drwxr-xr-x 7 1001 1001 4.0K Jun 13 00:33 .git
```

1.7 Git work cycle

A Git work cycle is composed of three steps: - create/delete/change files - place the files to follow to a special space, the staged area with **`add myfiles`**
- keep the actual version of the files included in the staged area with **`commit -m "my reason of change"`**

The **`status`** command explains the git step of each file of the folder:

[5]:
```
%%sh
cd ${PWD}/first_git_example
git status
```

```
On branch master

No commits yet

nothing to commit (create/copy files and use "git add" to track)
```

Now, experiment one git cycle. Create 2 files:

[6]:
```
%%sh
cd ${PWD}/first_git_example
for i in 1 2 ; do
   echo "text of file "${i}"\n" > file${i}.txt ;
done
ls
```

```
file1.txt
file2.txt
```

[7]:
```
%%sh
cd ${PWD}/first_git_example
git status
```

```
On branch master

No commits yet

Untracked files:
  (use "git add <file>…" to include in what will be committed)
        file1.txt
        file2.txt

nothing added to commit but untracked files present (use "git add" to track)
```

Observe: the 2 new files are included in the list of untracked files.

Add `file1.txt` to the list of tracked files, the staged area:

[8]:
```
%%sh
cd ${PWD}/first_git_example
git add file1.txt
git status
```

```
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>…" to unstage)
        new file:   file1.txt

Untracked files:
  (use "git add <file>…" to include in what will be committed)
        file2.txt
```

`file1.txt` pass from untracked to staged (ie. to be committed).

Change again the content of `file1.txt`:

[14]:
```
%%sh
cd ${PWD}/first_git_example
sed 's/text/text change /' file1.txt > tmp ; mv tmp file1.txt
git status
```

```
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>…" to unstage)
        new file:   file1.txt
        new file:   file2.txt
```

```
Changes not staged for commit:
  (use "git add <file>…" to update what will be committed)
  (use "git restore <file>…" to discard changes in working directory)
        modified:   file1.txt
```

observe the 3 states. Note that `file1.txt` appears in "to be commited" and also in "not staged for commit".

[ ]:
```
Stage all files:
```

[10]:
```
%%sh
cd ${PWD}/first_git_example
git add file?.txt
git status
```

```
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>…" to unstage)
        new file:   file1.txt
        new file:   file2.txt
```

And commit:

[15]:
```
%%sh
cd ${PWD}/first_git_example
git commit -m "commit with all files"
git status
```

```
[master (root-commit) 6cd964d] commit with all files
 2 files changed, 4 insertions(+)
 create mode 100644 file1.txt
 create mode 100644 file2.txt
On branch master
Changes not staged for commit:
  (use "git add <file>…" to update what will be committed)
  (use "git restore <file>…" to discard changes in working directory)
        modified:   file1.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

So far, you have initiated a new project whose code is versioned by git. You have created files and all their successives changes were tracked.

To avoid bad changes of code, it is a good practice to test a new code version before use it, and so

separate development code from production code. With the Git **branch** concept, you may manage this separation: develope code from an initial copy of the master code.

1.8 Use branches

We will now create a 2nd project by copying an already existing one (from an online git project site, e.g. github):

[ ]:
```
%%sh
git clone https://github.com/clairetn/FAIR_bioinfo_github.git
ls -lah FAIR_bioinfo_github/
```

Observe the result: - a new folder has been created - its name is directly deduced from the URL - it contains a `.git` repository and a `README.md` file: it is a minimal project!

To developpe a new functionality, add a branch with **`branch`**:

[20]:
```
%%sh
cd ${PWD}/FAIR_bioinfo_github
git branch branch_myfn # create a branch
git branch # list all branches
```

```
fatal: A branch named 'branch_myfn' already exists.
```

```
  branch_myfn
* master
```

The default branch is nammed `master`. The star denotes the working branch.

Move to the new branch with **`checkout`**:

[21]:
```
%%sh
cd ${PWD}/FAIR_bioinfo_github
git checkout branch_myfn
git branch
```

```
Switched to branch 'branch_myfn'
```

```
* branch_myfn
  master
```

Explore the branch (ls, git status):

[22]:
```
%%sh
cd ${PWD}/FAIR_bioinfo_github
ls -lah
git status
```

```
total 16K
drwxrwxr-x 3 1001 1001 4.0K Jun 13 00:57 .
drwxrwxr-x 7 1001 1001 4.0K Jun 13 01:01 ..
drwxrwxr-x 8 1001 1001 4.0K Jun 13 01:01 .git
```

```
-rw-rw-r-- 1 1001 1001  150 Jun 13 00:57 README.md
On branch branch_myfn
nothing to commit, working tree clean
```

The branch `branch_myfn` looks at a strict copie of the origin, the `master` branch.

Realise a git cycle: i) change the `README.md` file by adding your firstname to the authors list, ii) add the file to the staged area, and iii) commit:

[25]:
```
%%sh
cd ${PWD}/FAIR_bioinfo_github
echo "- my firstname "  >> fn.txt ; cat < fn.txt >> README.md ; rm fn.txt # add␣
 ↪fisrtname
more README.md ; echo "----------" # check adding
git status # check status
```

```
::::::::::::::
README.md
::::::::::::::
# FAIR_Bioinfo_github
This file is associated to the FAIR_Bioinfo courses.

## Authors list:
- Claire
- kchennen
- toto
- Paulette Lieby
- Clémence
- my firstname
- my firstname
----------
On branch branch_myfn
Changes not staged for commit:
  (use "git add <file>…" to update what will be committed)
  (use "git restore <file>…" to discard changes in working directory)
        modified:   README.md

no changes added to commit (use "git add" and/or "git commit -a")
```

[27]:
```
%%sh
cd ${PWD}/FAIR_bioinfo_github
git add README.md
git status
echo "-------------"
git commit -m "add firstname"
git status
```

```
On branch branch_myfn
nothing to commit, working tree clean
```

```
------------
On branch branch_myfn
nothing to commit, working tree clean
On branch branch_myfn
nothing to commit, working tree clean
```

Once you have check that the changes are corretcs, back to the `master` branch. Check the version of `README.md` file:

[28]:
```
%%sh
cd ${PWD}/FAIR_bioinfo_github
git checkout master
more README.md
```

```
Switched to branch 'master'

Your branch is up to date with 'origin/master'.
::::::::::::::
README.md
::::::::::::::
# FAIR_Bioinfo_github
This file is associated to the FAIR_Bioinfo courses.

## Authors list:
- Claire
- kchennen
- toto
- Paulette Lieby
- Clémence
```

It is the version before the change in the `branch_myfn` branch.

`merge` and **`delete`** the `branch_myfn`:

[29]:
```
%%sh
cd ${PWD}/FAIR_bioinfo_github
git merge branch_myfn
echo "------------"
more README.md
echo "------------"
git branch -d branch_myfn # -d = delete
git branch
```

```
Updating f93bb12..93de33f
Fast-forward
 README.md | 2 ++
 1 file changed, 2 insertions(+)
------------
::::::::::::::
README.md
```

```
::::::::::::::
# FAIR_Bioinfo_github
This file is associated to the FAIR_Bioinfo courses.

## Authors list:
- Claire
- kchennen
- toto
- Paulette Lieby
- Clémence
- my firstname
- my firstname
------------
Deleted branch branch_myfn (was 93de33f).
* master
```

You make change in a protected area to test it and merge your work when it is ok.

1.9 References

- version control, wikipedia
- git quick guide, tutorial point
- git getting started