Lab 1: Git

The git dream team*

August 26, 2024

Introduction

The goal of this lab is to introduce the most common aspects of git as a version control system. The lab is separated into two complementary parts.

- On the one hand, the basics of git usage to manage a **private** project is tackled in the first part. The focus shall be set here on the regular practice regarding the life cycle of files under version control as well as the handling of branches.
- On the other hand, we will consider specific workflows adapted to a **collaborative** usage of git.

To go beyond the aspects considered in this lab, we recommend three online resources:

1. https://git-scm.com/book/en/v2 an exhaustive e-book on GIT.
2. https://www.atlassian.com/git which gathers several tutorials on diverse aspects of GIT.
3. http://gitimmersion.com/, most of the material of this lab is actually adapted from this latter source.

This lab sheet mainly serves as a recap for the various git concepts and commands that we wish you to learn. It will also provide several useful links to additional resources for those who would like to look deeper into some concepts. During the lab, you will have to follow the detailed tutorial available at:

https://foureys.users.greyc.fr/tp-git/gitimmersion

*R. Clouard, S. Fourey, L. Simon

1 Workflow of the poor lonesome cowboy

In this section, you will work on your own. Your goal is to get familiar with the following notions:

1. the creation of a git repository,
2. the life cycles of version-controlled files (see figure 1),
3. the handling of branches,
4. and eventually, remote repositories.

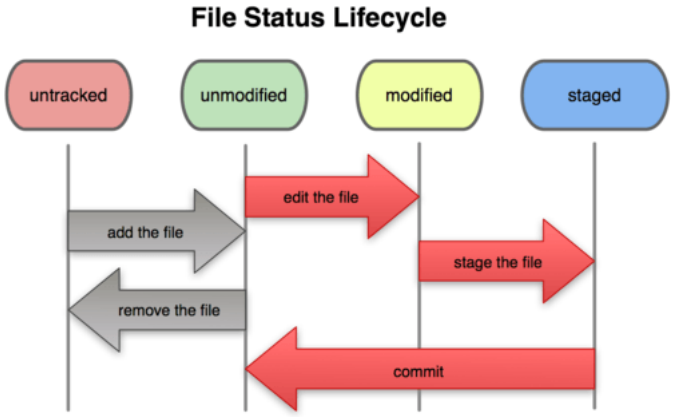

Figure 1: Life cycle of a file under git control.

1.1 Practical work

Concretely, your job here is to load https://foureys.users.greyc.fr/tp-git/gitimmersion and follow steps from `lab_01.html` to `lab_24.html`. These labs are actually extracted and modified from a subset of http://gitimmersion.com.

1.2 Summary of basic usage

The following sequence recaps the most common operations that occur when managing a project with Git.

```
$ git init myproject                          # creating a repository
$ cd myproject
###### basic life cycle #######
$ emacs toto.txt
$ git status                                  # checking which files are modified
```

```
$ git add toto.txt                                  # staging modified files
$ git commit -m "Made an awesome text file"     # committing the staged area
$ emacs toto.txt RCRules.txt
$ git add .
$ git commit -m "Modified toto.txt and created a new file"
###### branching #######
$ git branch
$ git checkout -b newbranch                 # creating and switching to a new branch
$ emacs toto.txt
$ git add .
$ git commit -m "new functionality in newbranch"
$ git checkout master
$ git merge newbranch               # merging changes from another branch in master
###### remotes ######
$ cd ..
$ git clone myproject myclone                       # cloning the repo
$ git clone --bare myproject myproject.git          # cloning as a bare repo
$ cd myclone
$ emacs tata.cpp
$ git commit -a -m "modified tata.cpp"
$ git remote add shared ../myproject.git            # adding a remote
$ git push shared                                   # pushing to the bare remote
$ cd ../myproject
$ git remote add shared ../myproject.git
$ git fetch shared                                  # fetching changes from a remote
$ git merge shared/master master                    # merging a remote branch
```

1.3 Follow-up resources

At home, you may also visit the following resources:

- Concerning GIT basics : http://git-scm.com/book/en/v2/Git-Basics-Getting-a-Git-Repository in particular sections 2.2 and 2.4.
- Concerning branches : http://git-scm.com/book/en/v2/Git-Branching-Branches-in-a-Nutshell in particular sections 3.1, 3.2 and 3.4.
- Besides, an advanced branching model is explained in http://nvie.com/posts/a-successful-git-branching-model/.
- Regarding git internals (skipped in here) you may refer to http://gitimmersion.com/lab_22.html and to section 3.1 of http://git-scm.com/.

2 Distributed workflows

In this section, you will work in pairs. Your goal is to get familiar with a distributed workflow which involves an integration manager (in charge of the official repository) and several developers (see figure 2). You will:

1. create the convenient repositories,
2. follow the steps of a developer who makes and submits a contribution, but also updates its local repository according to the official one,
3. follow the steps of the integration manager who includes a developer's contribution in the blessed repository.

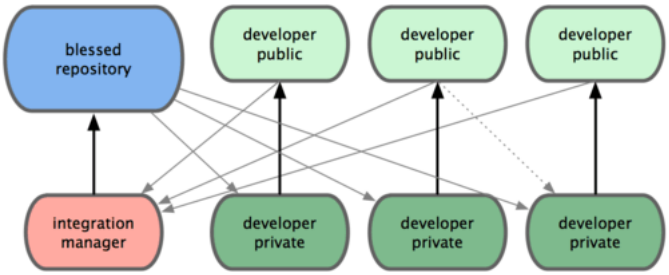

Figure 2: Distributed workflow with an official (blessed) repository.

2.1 Practical work

Your job here is to follow steps from `lab_25.html` to `lab_28.html`. Again, you need to work in pairs and to designate the one who will be in charge of the official repository (Linus in the text). The other team member will be a developer (Dave in the text).

2.2 Summary of basic commands

The following sequence recaps the most common operations that occur while this kind of development workflow is used.

```
###### Integrator's repositories setup (Linus) ###
$ git init --bare ~/public_html/gits/Project.git    # Official bare repository
$ git clone linus@host1:/path/to/Project.git        # Integrator's private rep.
$ mv post-update.sample post-update                 # Enable post-update hook
$ git remote add dave http://host2/path/Project.git # Add Dave's remote
###### Developer's repositories setup (Dave) ######
$ git clone --bare http://official-host/path/Project.git  # Public rep.
```

```
$ git clone dave@host2:/path/to/Project.git                # Private rep.
$ git remote add upstream http://host1/path/Project.git    # Add upstream remote
###### Developer's contribution (Dave) ######
$ git pull upstream master                                 # Get up-to-date master
$ git checkout -b cool-feature                             # Create a feature branch
$ git commit -m 'My nice feature'
$ git push origin cool-feature                             # Publish cool-feature on
                                                             public rep.
###### Integration in official rep. (Linus) ######
$ git checkout -b dave-cool-feature             # Create dedicated local branch
$ git pull dave cool-feature                    # Merge Dave's remote branch
$ git commit                                    # Manual commit in case of conflicts
$ git checkout master                           # \
$ git merge dave-cool-feature                   # / Merge with master branch
```

2.3 Follow-up resources

At home, you may also visit the following resources:

- A short introduction to Git workflows https://www.atlassian.com/git/workflows
- Section of the Pro Git Book dedicated to workflows http://git-scm.com/book/en/v2/Distributed-Git-Distributed-Workflows