Introduction to Visual Computing

Assignment# 1.2

git: The basics

February 20, 2018

Description

This class aims at giving you a good overview of the basics of `git`. Steps will be provided both for the command-line (Linux, MacOSX users) and the graphical Git-GUI application (MacOSX, Windows users). Even if you use the graphical user interface, we recommend you to carefully read the equivalent command line instruction to understand what happens "behind the scene".

Objectives

To be able to create a code repository, to understand and be able to create commits, to be able to share code.

Please note that you should use `git` to collaborate within your group during the project and to share code with us at the project's milestones

Specific Challenges

To learn how to think the ***code versioning*** way!

Preliminary steps

If you are not familiar with the concept of "version control" or would like to refresh your mind, watch this short introductory video on YouTube: `https://goo.gl/4JMsh7`

If you have not yet installed `git` on your system, here you can find a brief guide:

`git` installation depends on your operating system. If running Linux, everything is simple, just install `git` with your distribution's favorite package manager (on Debian/Ubuntu, `sudo apt-get install git`).

On Windows/MacOSX, we recommend you to use the git-scm.com official app, that takes care of properly configuring `git` for your system, and also provide an easy to use user interface (but obviously, an "easy to use" interface also means that it hides things from your eyes, and makes the underlying mechanisms harder to understand. Anyway...)

- For Windows : `https://git-scm.com/download/win`
- For Mac : `https://git-scm.com/download/mac`

Note

Once installed, the Windows Git app also provides a link to the Windows shell, conveniently configured to work with Git. We encourage you to make use of it and use the command-line based instructions below.

Besides, you should create a c4science account: it provides an easy way to share and review code within your group. You can use `https://c4science.ch/` as shared server (a remote in `git` parlance). You can login with your GASPAR credential.

Part I

A first git repository

Step 1 – Initial configuration

If this is the first time you are using `git`, you need to tell it what is your name and what is your email address, so that all your code contribution are effectively attributed to you.

From the command-line, type:

```
$ git config --global user.name "Firstname Lastname"
$ git config --global user.email "<email>"
```

If using the GUI, click on ***Create new repository***.

Step 2 – Create a new local repository

Simply create a new directory (like `/home/<username>/cs211/first-repo`) and initialize it by typing `git init` from within the directory. The name of this directory becomes the name of your repository.

That's it: a `git` repository is simply a regular directory, with one special item: a hidden `.git/` directory that stores all the objects `git` manipulates (mainly binary blobs representing files or parts of files).

Step 3 – A first commit

One of the first steps after creating a new repository is to add a `README` file that describes briefly the content of the repository: create such a file and describe in 2-3 lines the Processing sketch you made earlier.

Taking it further (optional)

It's nowadays common practice to write `README`s using the markdown syntax (extension `.md`): markdown is a markup language that lets you write simple text documents that are structured and can be nicely rendered by the computer.
Learn more about markdown on Wikipedia: `http://en.wikipedia.org/wiki/Markdown`

Then, commit this change: since the file `README` is not yet known to `git`, first add it: `git add README`, and then create a new commit with `git commit`.

`git` will ask you for a commit message (a commit message is made of a mandatory one-line ***summary*** – usually maximum 72 characters long – and a longer, optional, ***description*** that explains in greater details what this commit is about).

The commit summary must be concise yet must describe accurately the content of the change. For now, use the simple commit message "Added a README".

> Note
>
> You can use `git commit -m"commit message"` to directly create a commit with a commit message.

If using the GUI, make sure that the ***Unstaged Changes*** list is updated by pressing ***Rescan***, then select the file and press ***Stage Changed***. The file will appear in the ***Staged Changes(Will Commit)***. Then write a commit message and press ***Commit***.

By typing `git log` (or just looking at the Git GUI in Repository Menu ->Visualize master's History), you can see the history of changes in your repo. On Linux, `gitk` is another convenient way to display in a graphical way the history of the repo.

Step 4 – Code versioning

Copy the Processing sketch you have worked on during the first hour to your repo, and add it to the `git` repository (`git add <file name>`). Commit this change.

Now, change the background colour used in your sketch.

Using `git status` or the GUI, review the change and commit it (`git commit <your file>`, choosing an appropriate commit message.

Step 5 – Tagging

In Git, it is possible to mark a specific commit with a tag, ex. v1.0. Creating a tag on the last commit it's easy: just type `git tag <tag name>`. If you want to add a message to your tag, type `git tag -a <tag name> -m "<tag message>"`. To see tags in the repository, use the command `git tag`.

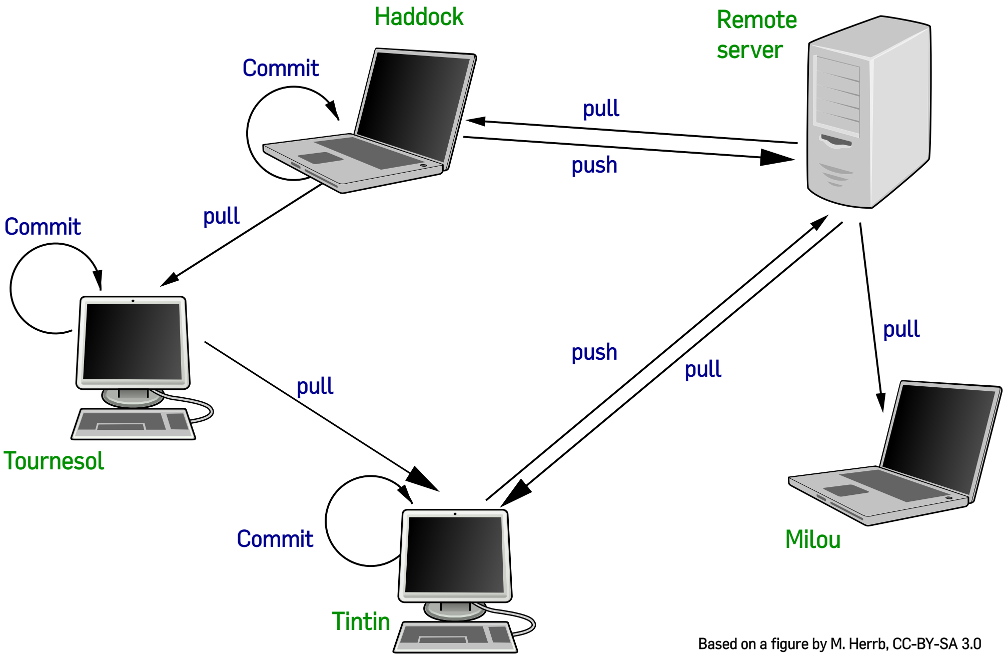
If you create a tag but then realize that there are still some commits you want to add into the tagged version, you can force the replacement of the tag in order to reference the most recent commit with the command `git tag <tagname> -f` or `git tag -a <tag name> -m "<new tag message>" -f`.

Part II

Going Online

Until now, you have only worked on a **local git** repository: this is a perfectly legitimate use of `git`. As a **distributed version control system** (DVCS), `git` is meant to support a wide range of code workflows, including purely local workflows: if you do not need to share your code over Internet, why would you need an Internet connection to benefit code versioning?

However, `git` is particularly powerful when working in groups: the core idea is that each participant own a full copy of a repository, and exchanges commits through **pushes** (to send commits to others) and **pulls** (to get commits from others). As you can see on the figure below (and contrary to traditional VCS like SVN), you **do not need to use a central server** (but you can!): `git` is distributed, each participant own a full, autonomous copy of the repository and can obtain (***pull***) commits from any other participant.



Based on a figure by M. Herrb, CC-BY-SA 3.0

Distant repositories can be on a remote Internet server like GitHub or `c4science.ch`, on your friends' computers, or even on a USB stick that you carry over with you. `git` calls them **remotes**. You can add as many remotes as you want to your local repository by giving them names.

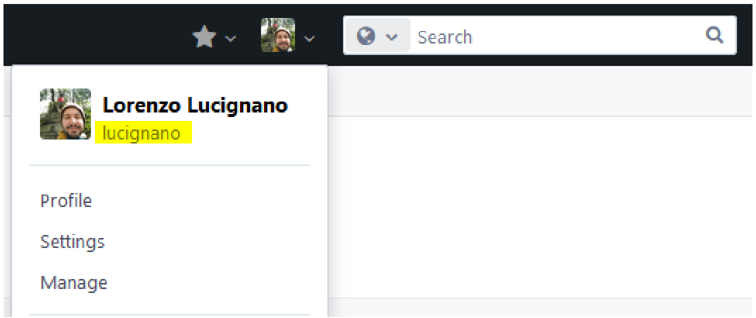
Often, you will have one main remote, which is traditionally called `origin` (but it's up to you to choose a different name!).

Step 1 – Adding a remote

You will add c4science as a remote repository to your local `git` repository.

First create an empty repository on c4science by following only the first paragraph "Creating a repository on c4science" at `https://c4science.ch/w/c4science/simplerepo/`. Name it after your local repository (not mandatory, but convenient). For now keep the default policies so that only you can access the repository. Now you can "Activate" your repository.

Make sure you setup a VCS password `https://c4science.ch/w/c4science/whatisvcs/`, since this password is not the same as the one used to access c4science. Your username is the profile name (see picture below).



Then, add this remote to your local repository, and push your changes online:

```
$ cd <REPO DIR> # for instance $HOME/cs211/first-repo
$ git remote add origin https://c4science.ch/diffusion/<xxxx>/<yyyy>.git # add a remote called origin
$ git push -u origin # push all your local commits to c4science
```

If the last command results in the fatal error ***fatal: The current branch master has no upstream branch***, it means that you need to type the name of the branch at the end, `git push -u origin master`.

Note

By default, the git push command doesn't transfer tags to remote servers. To push your tags use the command `git push origin --tags`. If you have replaced an existing tag (ex. running `git tag <tagname> -f`) that has been already pushed to the remote server, you have to run `git push origin --tags -f` to force the update on the remote server.

Note

If you are using the Git GUI, adding a Git remote is easy: just press the ***Remote*** button and ***Add***.

Taking it further (optional)

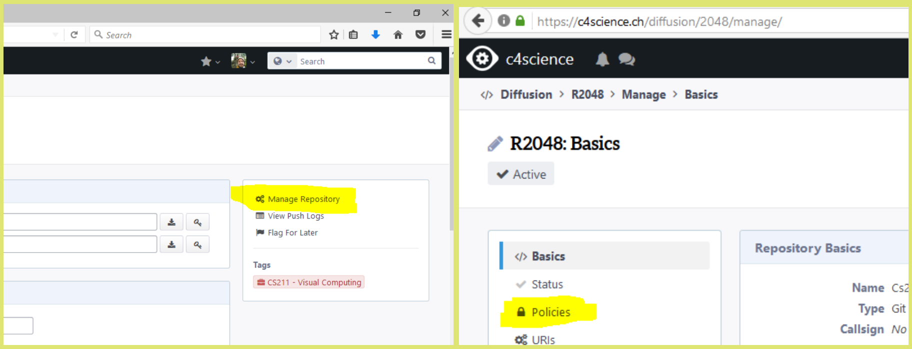
In this example, you use the `https` protocol as **transport** between the remote server and your local repository. This requires you to type your login and password every time.
`git` is however often used with `ssh` as transport. No password is required in that case (it transparently uses your `ssh` keys to establish an encrypted connection to the remote server). You can easily configure your account to use `ssh`. Read the documentation here:
`https://c4science.ch/w/c4science/sshkeys/`

Step 2 – Going Social

You will now team up with your neighbour to propose a fix for the issue he/she has opened in his/her project.

Note

The owner should give the rights to the colleagues in the remote repository policies by creating custom policies including the team members for the three fields **Visible To, Editable by, Pushable by**.



https://c4science.ch/diffusion/2048/manage/

R2048: Basics

Name Cs2
Type Git
Callsign No

Let's assume that one person has created a remote repo at `https://c4science.ch/diffusion/<xxxx>/<yyyy>.git)`, which will be the shared remote repo.

In order to contribute, you need to **clone** the repository to get a local copy.

```
$ cd <SOURCE DIR> # for instance $HOME/cs211
# Clone the repo <repo> inside directory <dir_name> (same as <repo> if omitted)
# To avoid confusion with your own repository, use smthg like 'first-repo-gerard' as dir_name
$ git clone https://c4science.ch/diffusion/<xxxx>/<yyyy>.git <dir_name>
```

Note

On the Git GUI you can click on "Clone Existing Repository".

You now have a full copy of the repository on your local machine in `<dir_name>`. Edit the Processing sketch of your teammate to correct the background color as he/she suggested in the issue he/she opened, and commit the result. In this case, a good commit message would be: “Fixing background color (issue #1)”. Finally, push the changes.

Your colleagues can retrieve these changes in their local repository by simply running `git pull` from their local repository (`first-repo`).

When working collaboratively, occasionally remote changes cannot be merged automatically in your local repository, causing a conflict. Please read this paragraph about basic merging and solving conflict in case you experience one `https://git-scm.com/book/en/v2/Git-Branching-Basic-Branching-and-Merging#_basic_merge_conflicts`.

Part III

The next steps

`git` is a large system that may appear complex at first sight. Take some time to get used to it: becoming familiar with `git` will be soon rewarding! and do not hesitate to ask questions during the coming weeks.

This tutorial did not introduce many concepts like branches, conflicts or rebase. If you want to learn more on `git`, here a few resources (besides your lovely teaching assistants!):

- Many `git` *cheatsheets* exist. Tower has a good one (`http://www.git-tower.com/blog/git-cheat-sheet/`), GitHub as well (`https://help.github.com/articles/git-cheatsheet/`, including a French version), and a nice interactive cheatsheet is there: `http://ndpsoftware.com/git-cheatsheet.html` (French translation also available)
- *git from the bottom up* is a great (and esay) reading to understand how `git` actually work. As a matter of fact, most `git` commands become evident once you know how they are built. `https://jwiegley.github.io/git-from-the-bottom-up/`