Git training

Nicolas Barrier Witold Podlejski
Criscely Lujan-Paredes

Presentation of the Git Software

What is Git?

- The most popular **Version Control Software** (VCS)
- **Free** and **open source**
- **Light** and **local** use (without internet)
- Manages and **tracks versions** of a project (code, manuscript, data)
- Can be linked with **remote server** (GitHub, Gitlab)

What is Git for?

- **Track changes** (*commits*) over time with information

  - **who**, **when** and **what** are the changes

- Eventually go **back in time**
- **Highlight a specific version** of the project (*tags*).
  - For example, new software versions.
- **Synchronize** the project in the **cloud** with remote servers (GitHub, Gitlab)
- **Resolve version conflict** when simultaneous changes
- Create **derivates** of a project (*branches*):
  - production, development, feature
- **Publish** the project (open science)

In short...

Installation and configurations

Installing Git

Windows and Mac

Download and install Git from https://git-scm.com/downloads.

When done, open `Git Bash`

Linux

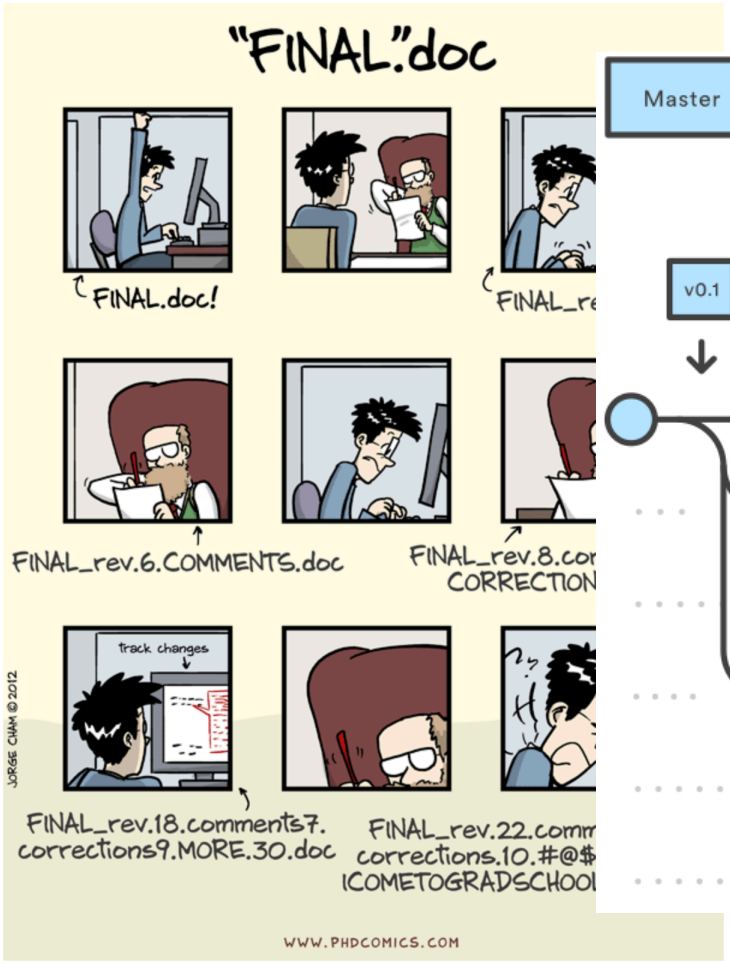


Figure 1: Without Git

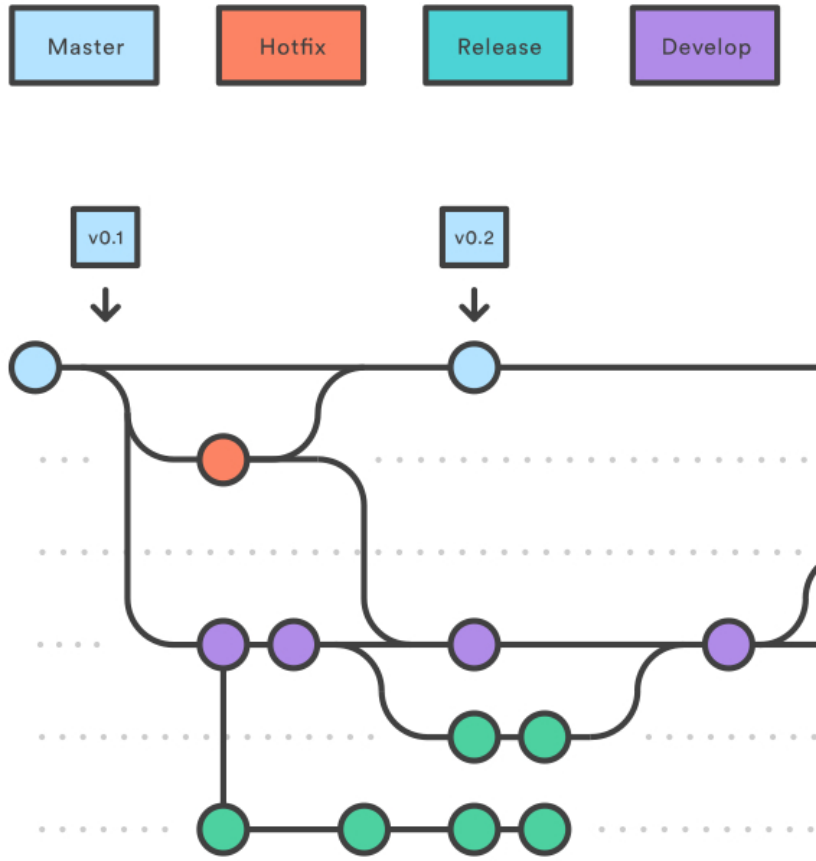


Figure 2: With Git

Open a `Terminal` window and type:

```
sudo apt install git
```

Git configuration

On `Git Bash` or in the `Terminal`:

- Type `git config --global user.name "Firstname Lastname"`
- Type `git config --global user.email "myadresse@ird.fr"`

> **Note**
>
> These two lines identify you in the history of a project.

- Type `git config --global --list` to see the global git configuration.

Git configurations: aliases

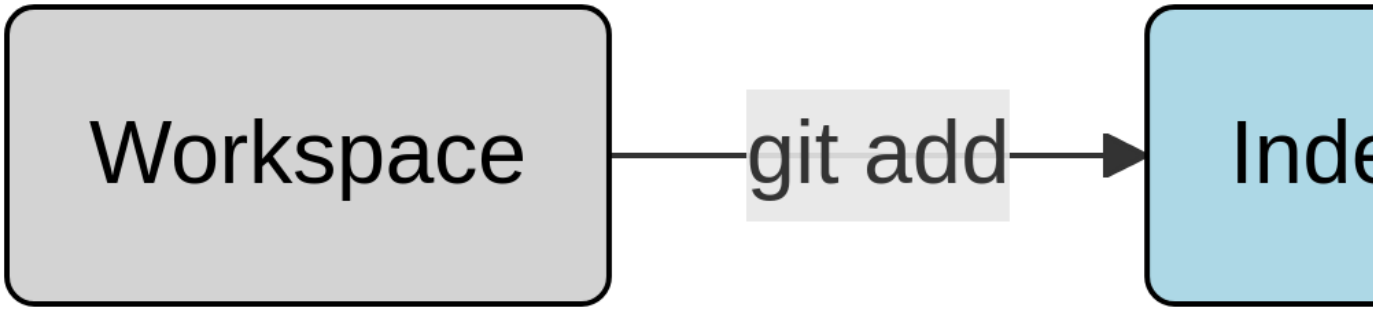
To create Git aliases (i.e. shortcuts):

- Type `git config --global alias.tree "log --graph --decorate --pretty=oneline --abbrev-commit"`
- Type `git config --global alias.br "branch -vv"`
- Type `git config --global alias.re "remove -vv"`

You can now call the `git tree`, `git br` and `git re` commands.

Getting started with Git in local

Git architecture



- `Workspace`: your working directory → your computer
- `Local`: the local repository → contains the history of your project
- `Index`: a buffer between `Workspace` and `Local` → list of the files that will be sent from `Workspace` to `Local`
- `git add` : the command to add the file(s) in the `Index`
- `git commit`: the command to validate the changes (moves the files from `Index` to `Local`)

> Tip
>
> To help understand, think of moving house. Workspace=old house; Local repository=new house; Changes=boxes; Index=moving truck

Getting started

- Move to the folder where you want to work by using the `cd` command

- Create a folder called `training-git` by typing `mkdir training-git`
- Move to the folder by typing `cd training-git`
- Type `ls -alrt`. What do you see?
- Type `git init`. Read the console output
- Type again `ls -alrt`. What is new?

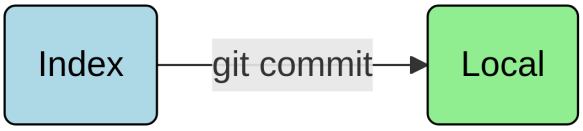
> **Note**
>
> A `.git` folder has appeared. It contains the full history of your project (`Local` repository)

- Type `git status` and `git log`. What does it tell you?

First commit

- Create a `README.md` file by typing `touch README.md`
- Type `git status` → what does it tell you about `README.md` ?
- Type `git add README.md` and `git status` → what is the new status of the file?

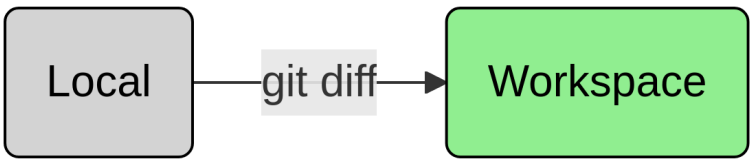
- Type `git commit -m "First commit"` and type `git status` and `git log`



> **Note**
>
> `0f0e96a` is a short version of the identifier of the commit

Second commit

- Open the `README.md` file, add `# Git training` and save
- Type `git status` → what is the status of the file?
- Type `git diff` → what does this command do?



- Type `git add README.md` and `git commit -m "Second commit"`
- Type `git log`

Creating tags

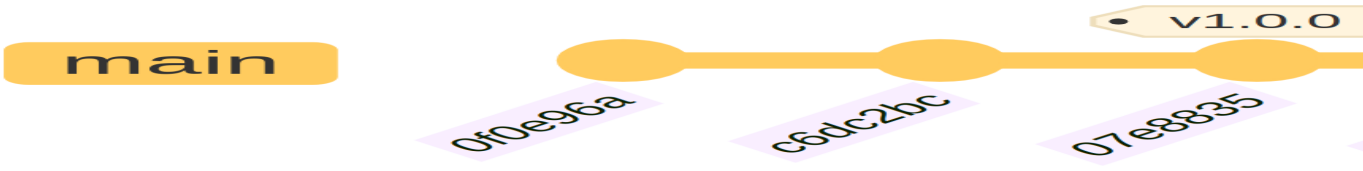
- Open the `README.md` file and add `## Version v1.0.0`.
- Type `git add README.md`
- Type `git commit -m "Third commit"`
- Type `git tag v1.0.0` and `git log`

- Type `git tag` to list all existing tags

Reverting a commit

To revert a commit, i.e. to cancel changes done in a previous one:

- Type `git revert -n c6dc2bc` (replace `c6dc2bc` by your commit id)
- Type `git status`. What happens?
- The `README.md` file is now modified and staged (i.e. in the `Index`)
- To see what has been changed, type `git diff --staged`
- To commit this modification, type `git commit -m "Revert commit"`



> **Note**
>
> The `-n` option is to prevent an automatic commit. Therefore you need to commit yourself

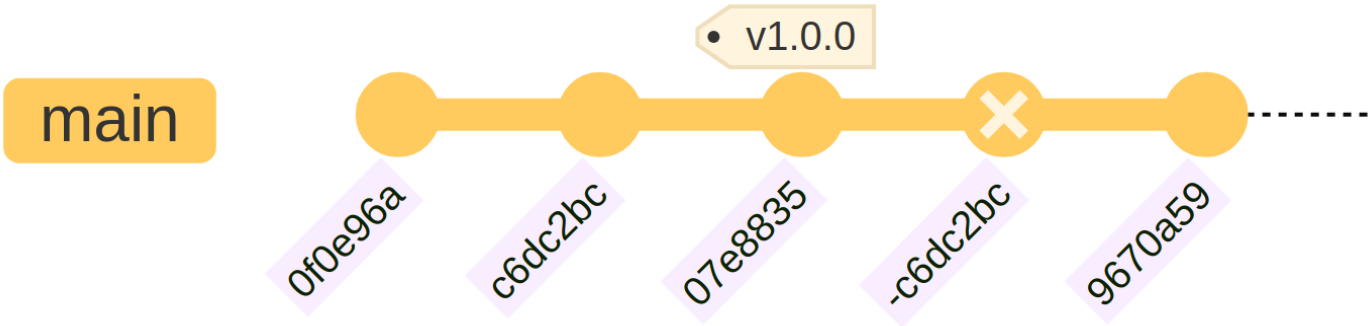
Ignoring files

It is possible to tell Git to ignore some files by using a .gitignore file (for instance, `.Rdata` or `.tmp` files).

- Create an empty `output.Rdata` file and type `git status`
- Create a `.gitignore` file and write `*.Rdata`. Type again `git status`

The `output.Rdata` file does not appear as an `Untracked file` anymore

- Type `git add .gitignore` and `git status`
- Type `git commit -m "Fourth commit"`



> **Tip**
>
> To list the ignored files, type `git ls-files --others --ignored --exclude-from=.gitignore`

Moving in the history

- Type `git checkout v1.0.0` → move to a tag

- Type `git checkout 0f0e96a` → move to a specific commit

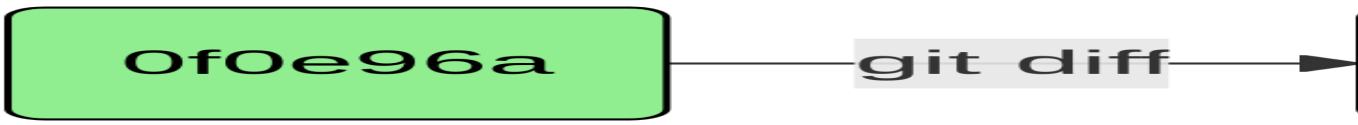
- Type `git checkout main` → move at the latest commit

> **Tip**
>
> `HEAD` is a symbolic reference pointing to wherever you are in your commit history, as shown in `git log`

Display differences

- Type `git diff 0f0e96a v1.0.0` → compares a commit and a tag.

> **Warning**
>
> Order matters when using `git diff`. Differences are shown with the reference state considered to be the first argument.



- Type `git diff 0f0e96a c6dc2bc` → compares two commits.
- Type `git diff 0f0e96a HEAD` → compares where you are in the history (`HEAD`) with a given commit.

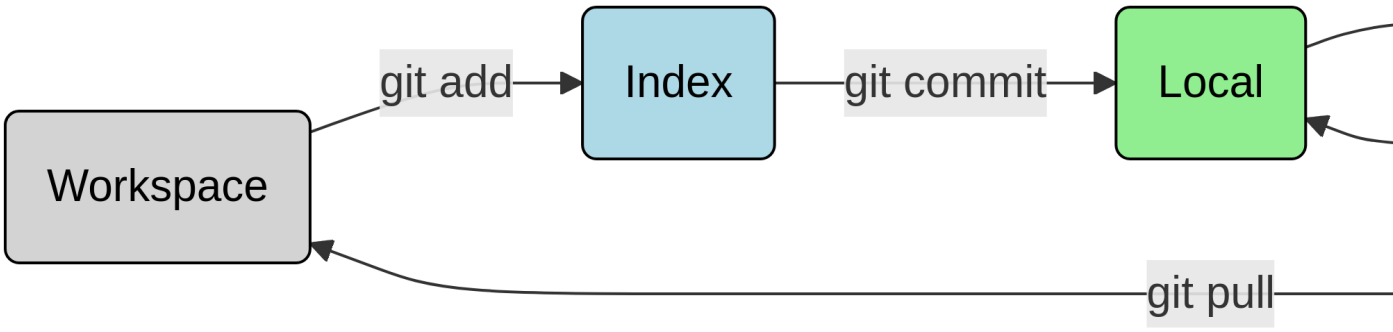
Using Git with remote server

Using remotes

In addition of saving the history, Git has other advantages. It allows to:

- Save a project remotely
  - Synchronization with different computers (laptop, HPCs)
- Share a project (codes, packages) with the community
  - Reproducible science

To do so, a 4^{th} component in the Git architecture must be considered: the `Remote` repository. It contains a **remote** version of the history of your project



Remote hosts

There are several remote hosting possibilities:

Commercial hosts:

- GitHub: https://github.com/
- GitLab: https://gitlab.com/

Institutional hosts:

- GitLab IRD: https://forge.ird.fr/
- GitLab Ifremer https://gitlab.ifremer.fr/

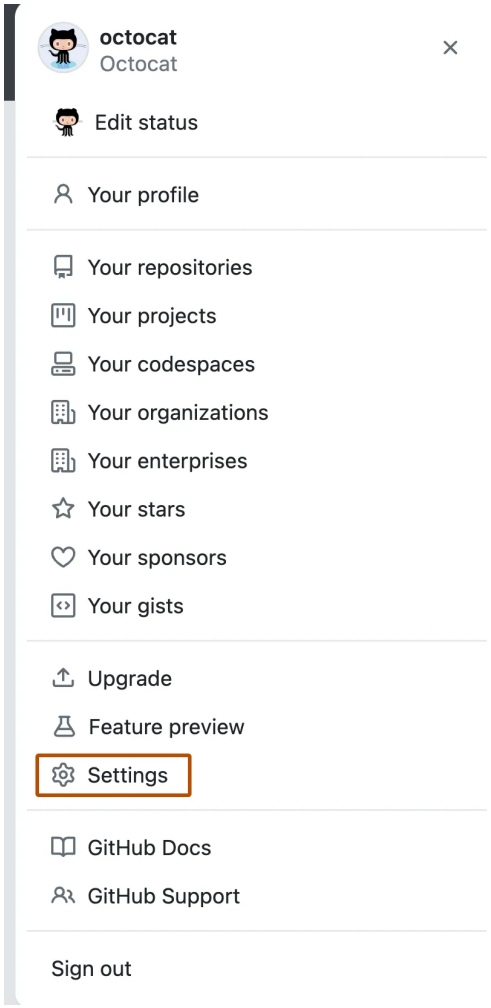
In the following, we will use GitHub.

> Tip
>
> GitHub proposes extra-features for students, teachers and researchers. Visit https://education.github.com/benefits for more informations

Creation of a personal access token

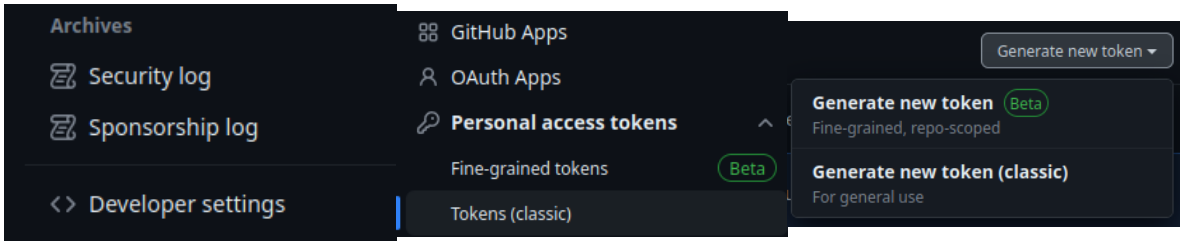
To authentificate, you need to create an authentification token (see GitHub authentification of details).

To do so, click on your profile photo and then on `Settings`:



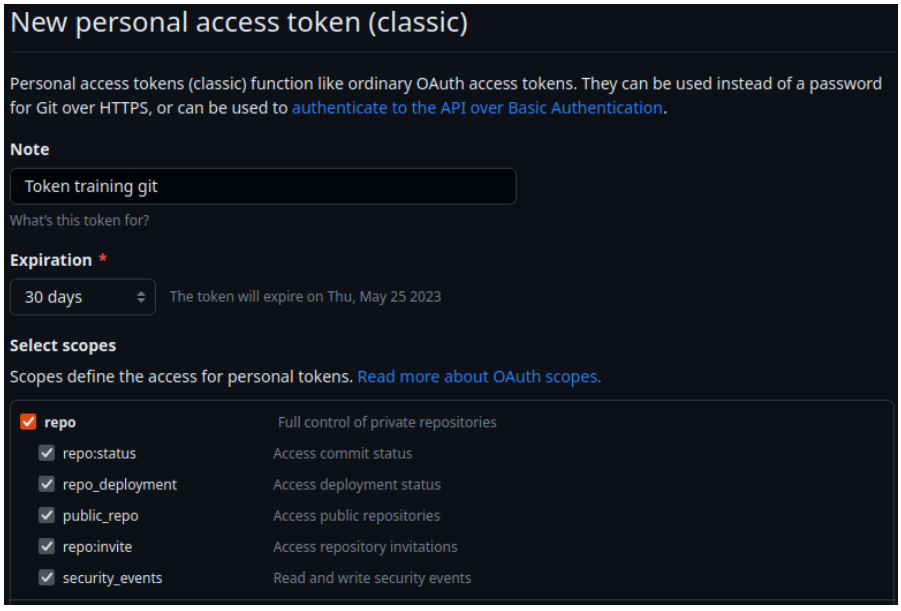
Creation of a personal access token

- In the left sidebar, click on `Developer settings`.
- Under `Personal access tokens`, click `Tokens (classic)`.
- Select `Generate new token` and `Generate new token (classic)`.



Creation of a personal access token

- Add a description note and **click on the "repo" box**, as shown below:

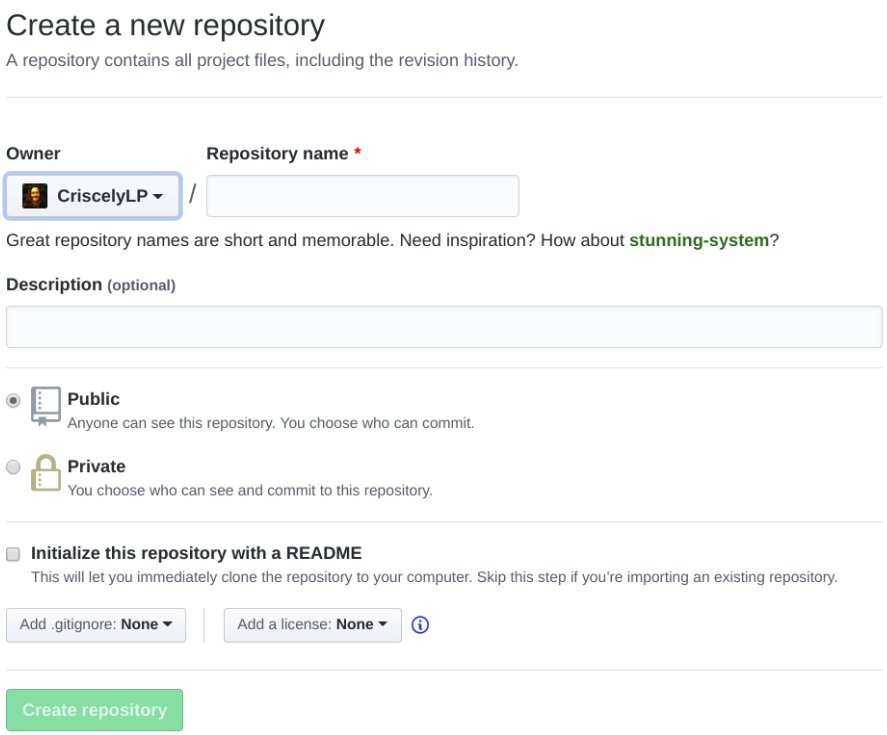


- Click on the `Generate token` box button.
- Copy and save in a `.txt` file or in a Password manager tool (for example KeePassXC) your token: **this is your password to publish codes!** It should look like something like this:

`ghp_****************************************`

Creation of a GitHub repository

- On your GitHub page, click on `Repositories`
- Click on the the green `New` button
- Set the name of your remote repository. **Leave the other fields empty**
- Click on `Create repository`



Linking Git local and remote

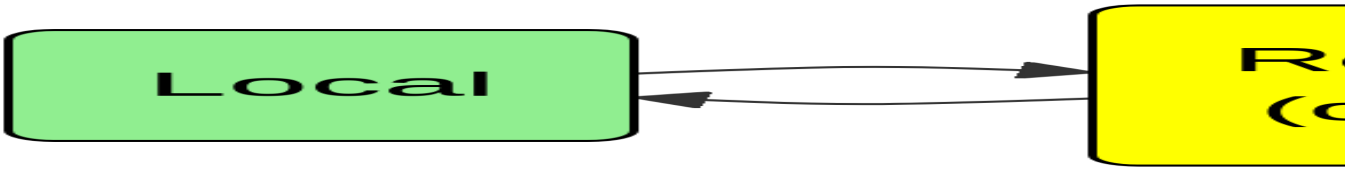
- In `Terminal` or `Git Bash`, type the following line:

```
git remote add origin https://github.com/barriern/git-train.git
```

> ⚠ Warning
>
> Replace `barriern` by your GitHub login and `git-train` by the name of your GitHub repository.

> ❗ Important
>
> The URL must end by `.git`!

- It connects your `Local` repository with the `Remote` repository, called *origin*



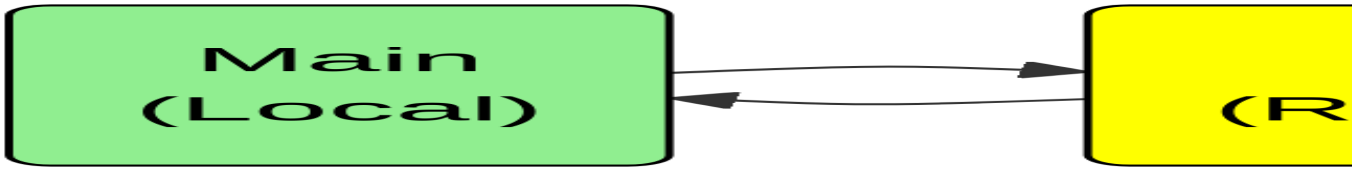
- Type `git remote -vv`

Linking Git local and remote

Now that the local and remote repositories are linked, the same thing must be done with the branches.

- First, rename your local branch with the name expected by GitHub (either `main` or `master`):
  - Type `git branch -M main`
- Push your branch on the remote server:
  - Type `git push -u origin main`

It connects the *local* and *remote* main branches (`-u` option stands for `--set-upstream`) and sends the local commits to the remote branch



- Type `git branch -vv`
- Refresh your repository webpage: what do you see? **what is missing?**

Linking Git local and remote: tags

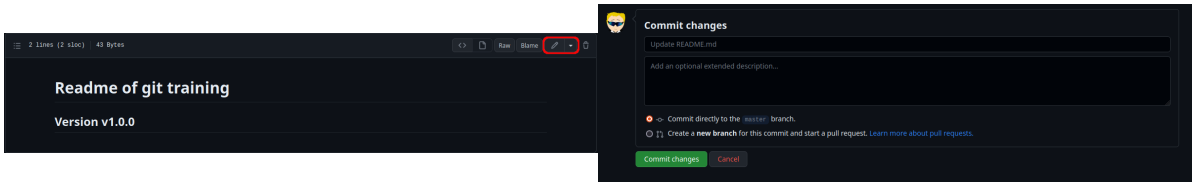
- To push tags to the remote, type `git push --tags`
- Refresh your repository webpage. Check whether the tags are properly pushed

> **Note**
>
> No need to specify the `-u origin main` arguments here

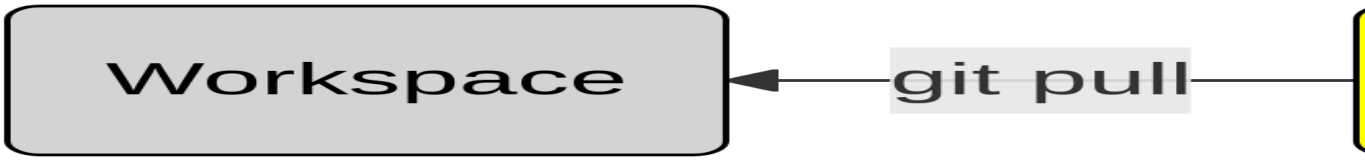
Synchronization from the remote

- In GitHub, click on the `README.md` file and then on the edit button

- Add a `Update from Github` line and click on `Commit changes`



- The `Remote` change of `README.md` is not yet visible in `Workspace`!
- In `Git Bash` or `Terminal`, type `git pull`



- Look again at the `README.md` file on your computer. You should see the update.

Synchronization: conflicts

- On GitHub, add `x = 1` at the end of the `README.md` file. **Do not pull the changes!**
- On your computer, edit the `README.md` and add `x = 2`.
- Type `git add README.md`
- Type `git commit -m "Fifth commit"`
- Type `git push`. What do you see?

- Type `git pull` and `git status`. An error occurs because there is a conflict in the `README.md` file that cannot be solved by Git.

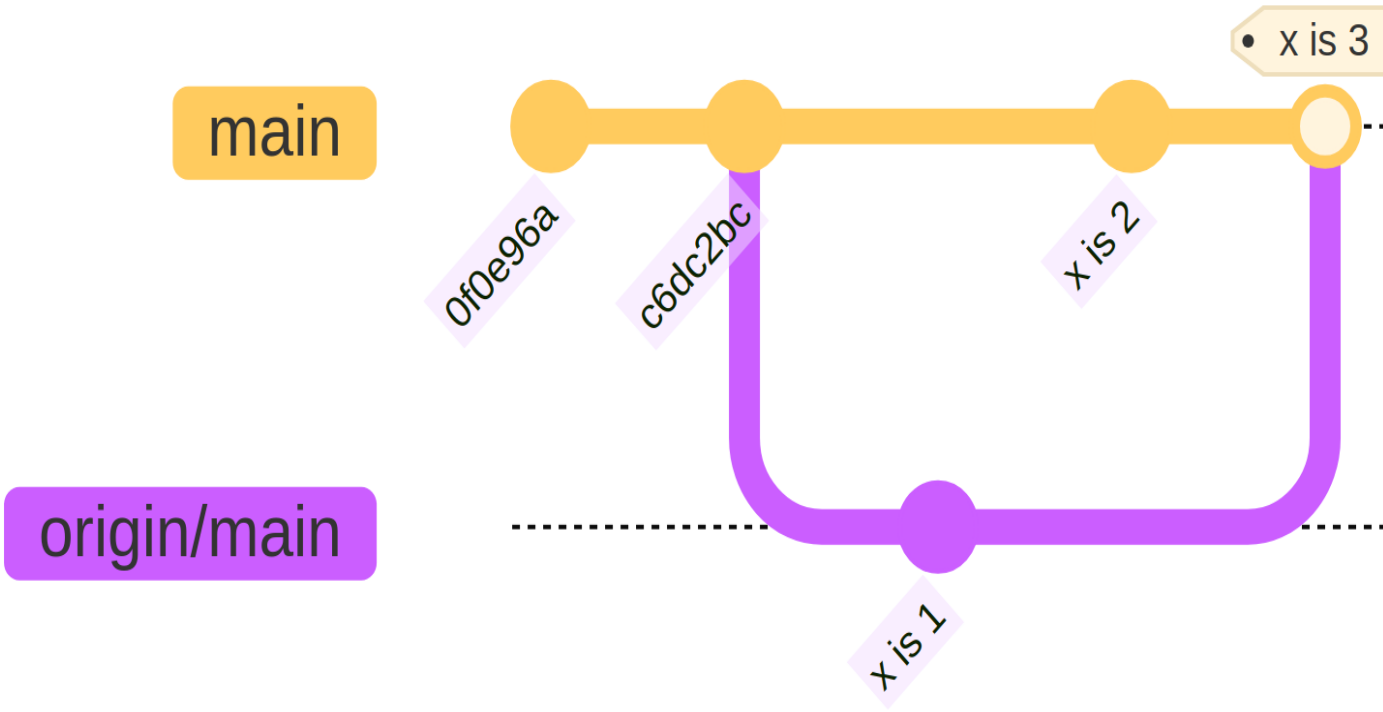
Synchronization: conflicts

- Open the `README.md` file. You should see:

```
<<<<<<< HEAD
x = 2
=======
x = 1
>>>>>>> 70a4c105e377db282c0769606960f0afcccdd071
```

> **!** Important
>
> These are conflicts markers. `Git` does't know whether to chose `x = 1` or `x = 2`. This is your job

- Open the file, remove the conflict markers and solve the value of x by setting `x = 3`
- Add, commit and push the changes
- Refresh the GitHub page and look at the file



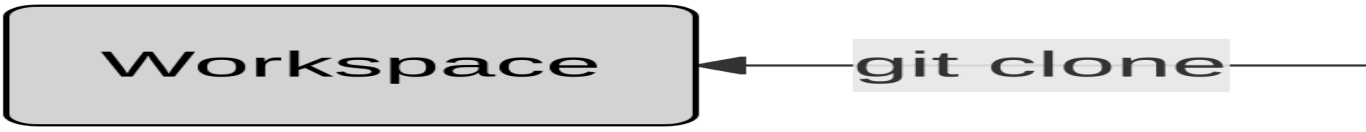
> **Tip**
>
> To avoid conflicts, use `git pull` and `git push` extensively

Cloning an existing repository

The `git clone` command allows to synchronize locally an existing remote repository.

- On GitHub, create a new repository as done previously.
- **This time, include a `README.md` and eventually a `LICENCE` file**
- Copy the link to the new GitHub repository

- On Git Bash or Terminal, type `git clone https://github.com/barriern/new-repo.gi` (replace the URL by the proper name)



- Move to the cloned folder by typing `cd new-repo` (replace with the name of your cloned folder)
- Type `git branch -vv`, `git remote -vv` and `git log` to see the full history.

Cloning an existing repository

> **Tip**
>
> To create a Git repository from scratch the easy way, create the repository on GitHub with a `README.md`.
> Local and remote repositories and branches are automatically synchronized!
> Then add your files on your local folder, add, commit, push

> **Important**
>
> **Do not clone or initialize a Git repository in another Git repository!**

Conclusion: good practice

- Before starting working on a project, do a `git pull`
- Commit very often using `git commit` extensively

- Push often as well using `git push`
- Use `git status` extensively to know what you have done

Git clients

Git clients: what is it?

Git Clients are softwares that facilitate the use of Git (see Git Guis for a list).

Beside, most code editors include Git functionalities

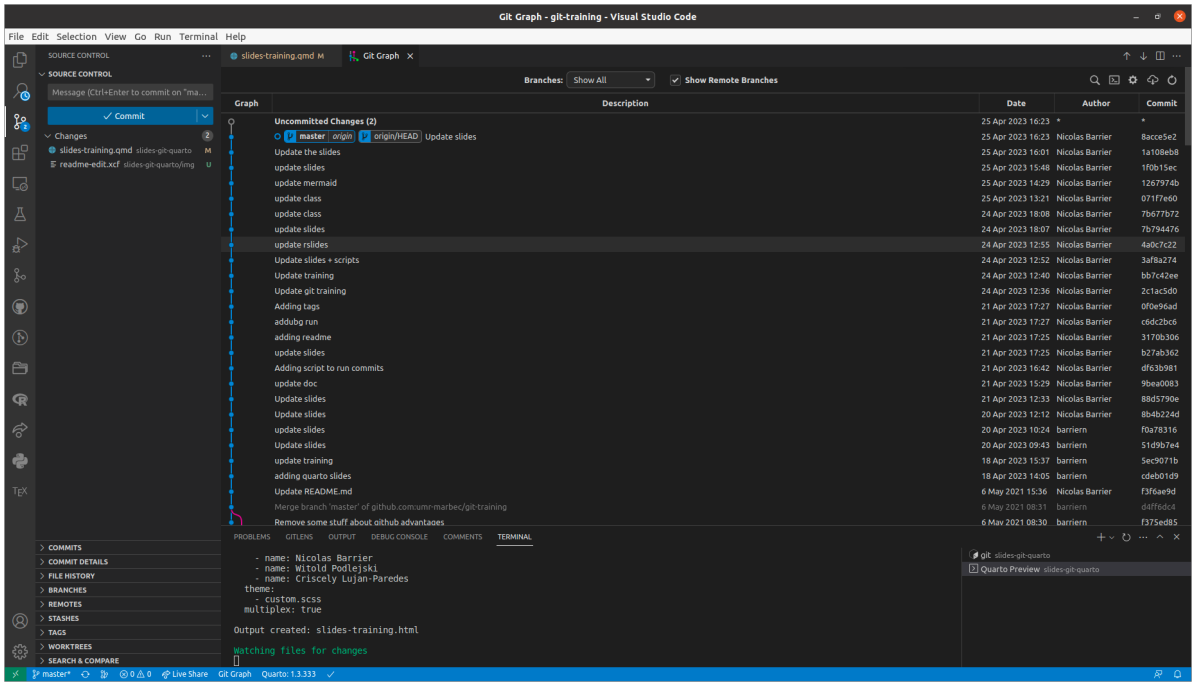


Figure 3: VSCode

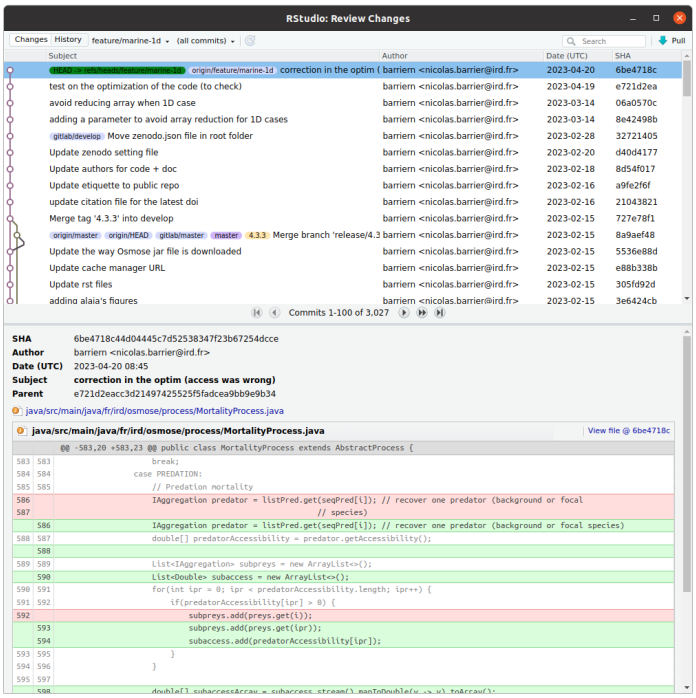


Figure 4: RStudio

Git clients: VSCode

Git clients: RStudio

Git clients: Netbeans

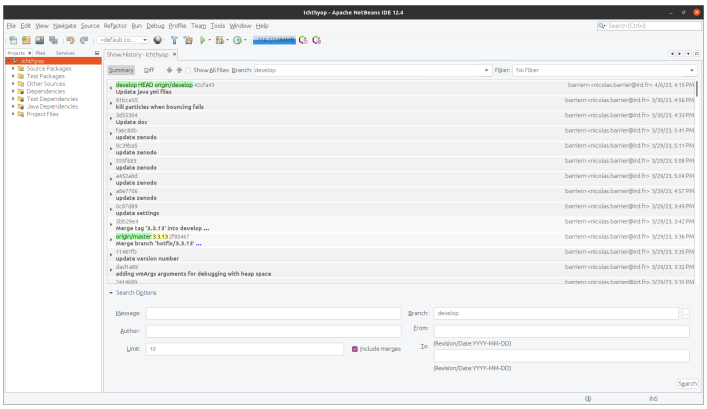


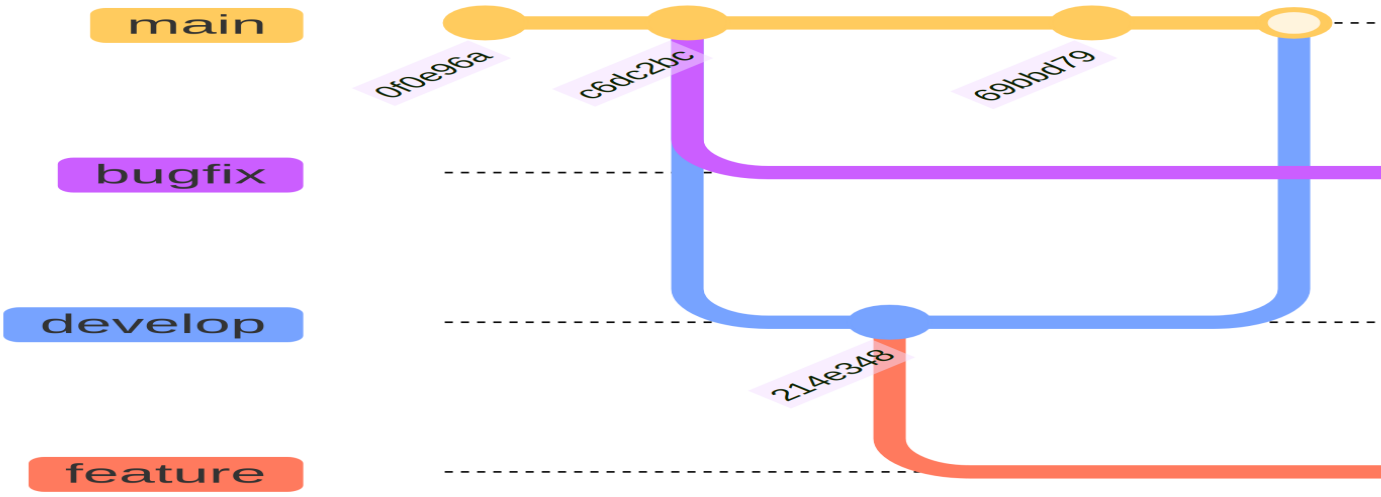
Figure 5: Netbeans

Git branches

Git branches

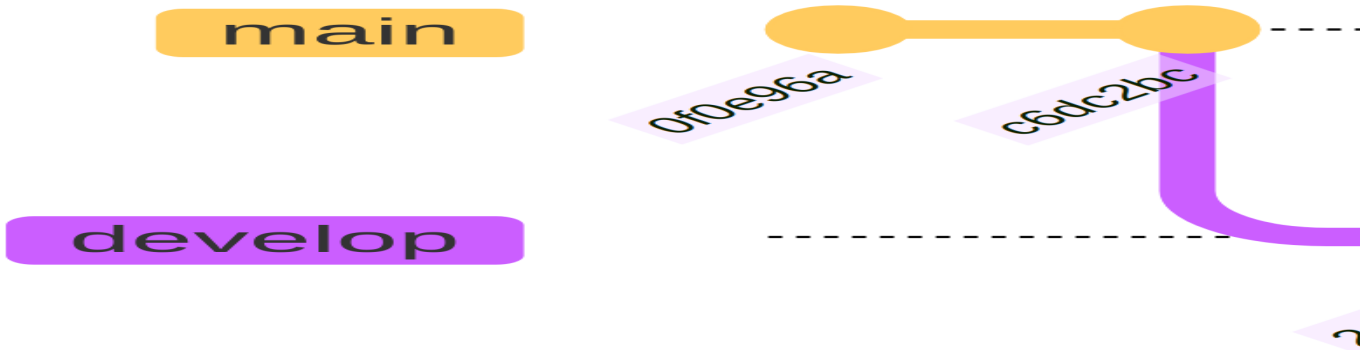
Working with branches allows to create derivates for a project. For example:

- A `main` branch for the production version
- A `develop` branch for preparing the next release
- A `feature` branch for testing new features



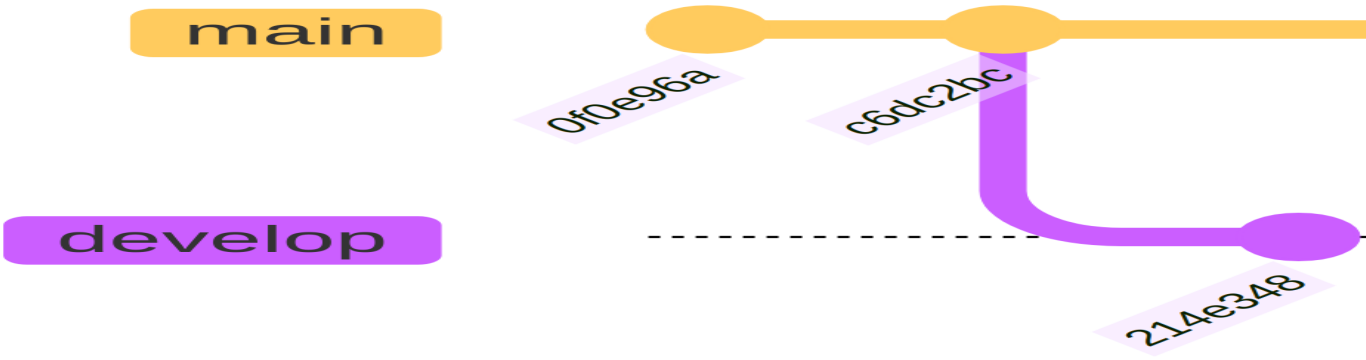
Creating branches

- Type `git checkout -b develop`
- Type `git status`, `git br` and `git tree`
- Open the `README.md` file, add some text and save.
- Type `git add README.md`
- Type `git commit -m "3rd commit"`
- Type `git br` and `git tree`



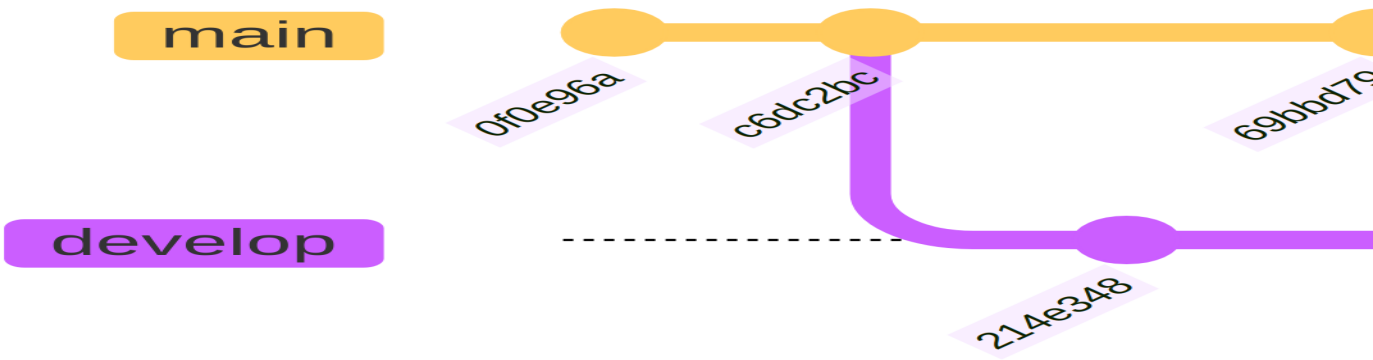
Switching branch

- Type `git checkout main` (or `git checkout master`)
- Type `git br`
- Open the `LICENCE` file and add some text in it
- Type `git add LICENCE`
- Type `git commit -m "Third commit"`
- Type `git tree`



Merging branches

- On the `main` branch, type `git merge develop -m "merge-develop"`
- Type `git log` and `git tree`

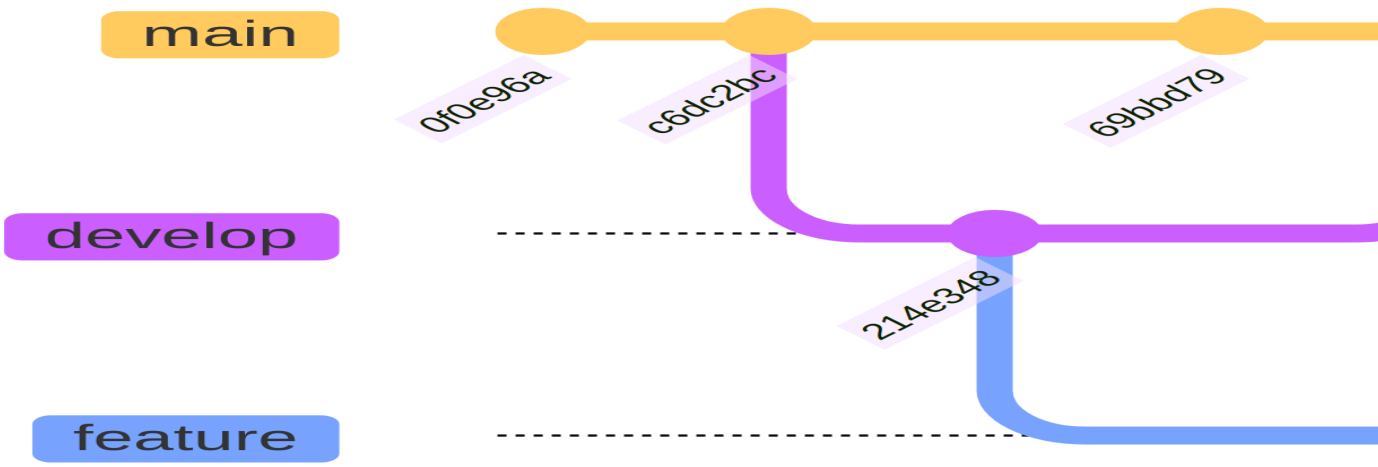


The `merge` command puts the commits from the argument branch (here `develop`) and puts them into the current branch (here `main`).

> **Note**
>
> During the merging process, another commit is created

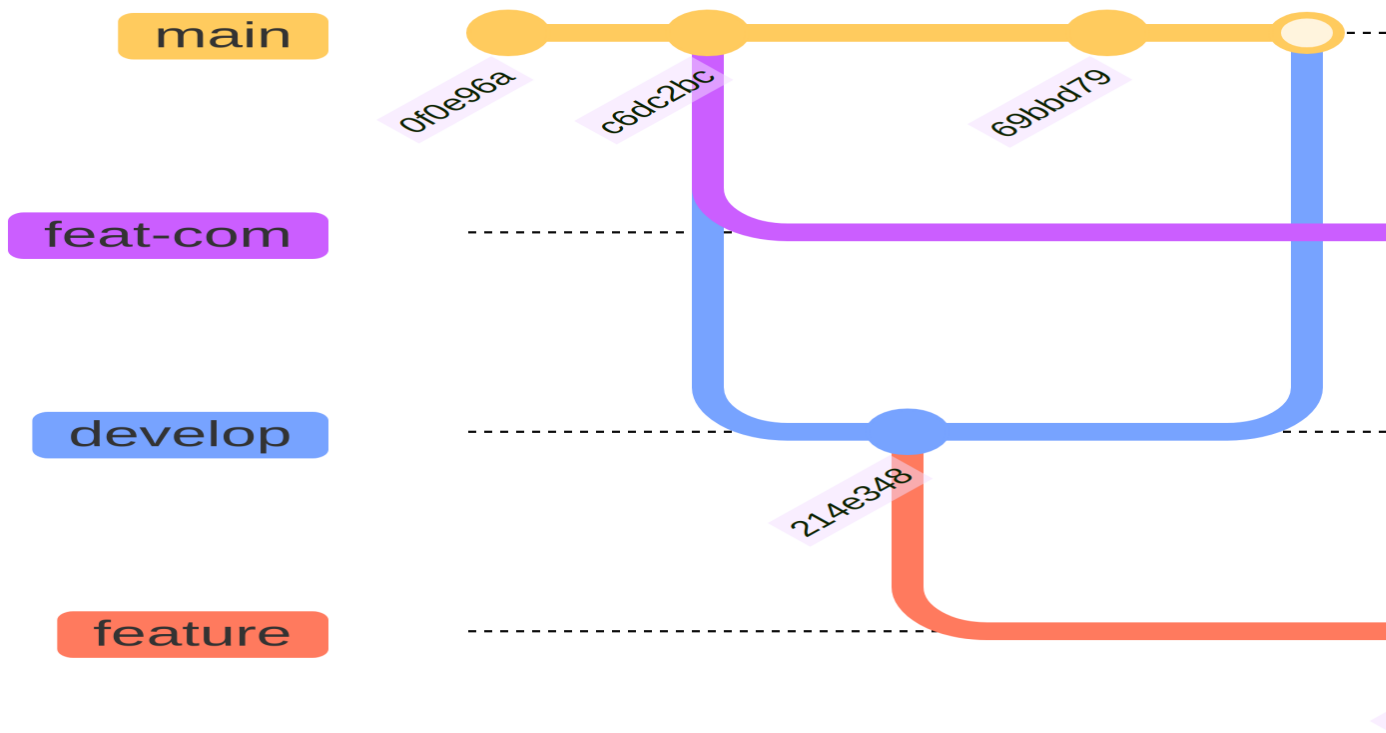
Creating branch from another branch

- Type `git checkout -b feature develop`
- Create a `script.R` file
- Type `git add script.R`
- Type `git commit -m "Fourth commit"`



Creating branch from a commit

- Type `git checkout -b feat-com 1831e4e` replacing `1831e4e` by an actual commit ID.
- Create a `script.py` file
- Type `git add script.py` and `git commit -m "Sixth commit"`

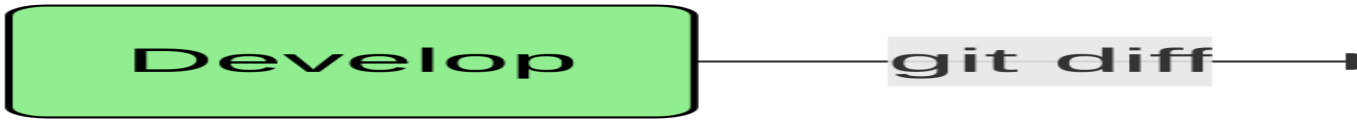


Differences between branches

- Type `git diff develop main`

You will see the text that has been added to the LICENCE file (69bbd79 commit)

> **Warning**
>
> Order matters: it shows what has been added to `main` branch compared to the `develop` branch



Using branches on remote servers

To push a branch on a remote server

- Switch to the branch you want to push: `git checkout develop`
- Push the branch as follows: `git push -u origin develop`.

> **!** Important
>
> On the `push` command, the last argument is the name branch on the remote server. Make it consistent with the local branch

Use `git branch -vv` extensively to check the links between local/remote branches.

Deleting a branch (locally)

- Type `git checkout main`
- Type `git branch -d develop`
- Type `git br`
- Type `git branch -d feat-com`

An error occurs! The suppression of `feat-com` implies the loss of the `d9d02608` commit. To force the suppression, use `-D` instead of `-d`.

- Type `git branch -D feat-com`

> **Note**
>
> The suppression of `develop` was ok because the content of commit `3rd` is included in the merge.

Deleting a branch (remotely)

Deletion from the remote branch is not automatic. To delete branch remotely:

- `git push origin --delete develop`

> **Important**
>
> Make sure that the branch should be removed or has been merged. Delete locally first and then remotely

> **Tip**
>
> For the lazy people, the Git Flow extension, managing branches is very easy! Everything is almost done automatically!

Large file storage (LFS)

What for?

To version (reasonably) large files (images, data samples) → Git with LFS extension.

> ⚠ Warning
>
> Make sure that the remote host is compatible with LFS (GitHub is compatible)

> ❗ Important
>
> Do not use for data that change extensively. It can be used for example to store toy datasets to show how a package work (vignettes, documentation)

LFS install

- LFS is automatically installed with Git on Windows.
- On Linux, it can be installed using `sudo apt install git-lfs`
- When it is installed, you need to activate it. To do so, type `git lfs install`

Tracking files with LFS

- Create a `data.csv` file and add `Year,Size,Species`
- Type `git lfs track "*.csv"`

A `.gitattributes` file has appeared, which list all the file extensions managed by Git LFS.

- Type `git add .gitattributes data.csv`
- Type `git commit -m "Using LFS"`
- Type `git push`

- On GitHub, click on your file `data.csv` file.

Remainder

Basic commands

- `git init`: initialise a git project (create .git folder)
- `git add [files]`: add files to list of tracked files
- `git commit -m "message"`: validate locally a version of the project
- `git status`: see the unvalidated and untracked changes
- `git checkout [commit]`: load the project version corresponding to the index
- `git pull`: import the changes from remote project to local
- `git push`: export the changes from local project to the remote server

Git configuration (mandatory)

- Configure your identity: `git config --global user.name "Firstname Lastname"`
- Configure your e-mail: `git config --global user.email "myadresse@ird.fr"`

Branch handling

- `git branch [branch_name]`: create a new branch (but you remain on the previous branch)

- `git branch -b [branch_name]`: create a new branch and move to this newly created branch
- `git checkout [branch_name]`: move to the corresponding branch
- `git merge [branch_name1] [branch_name2]`: merge two different branch, you may need to resolve version conflict.
- `git branch -d [branch_name1]`: delete a branch (safe mode)
- `git branch -D [branch_name1]`: delete a branch (unsafe mode)

Linking with remote

- `git clone [URL]`: Import an existing project from remote server.
- `git remote add origin [URL]`: link directly the local repository with a remote

Authentication of your computer and the remote server

- SSH key: easy way on Linux distributions
  - Tuto here: https://jdblischak.github.io/2014-09-18-chicago/novice/git/05-sshkeys.html
- Authentication Token
  - Tuto here: https://docs.github.com/en/authentication/keeping-your-account-and-data-secure/creating-a-personal-access-token

Good practices

- Pull before any work on the project
- Commit as frequently as possible
- Write explicit commit message
- Push regularly

IDE (graphical user interface) with Git

- R
  - RStudio
  - Visual Studio Code
- Python
  - Spyder
  - Visual Studio Code
  - Pycharm (all JetBrain softwares)

Sources

- Plateau bioinformatique, Montpellier: Formation Git(Lab) (05/04/2018)
- UMR AMAP (Atelier MAIA P3M), Montpellier: Introduction à GIT (04/04/2019)