A (very) short introduction to git

Guillerme Duvillié
Henallux - UlB

2020 - 2021

Plan

A VCS?

A system that records changes to a file or set of files over time so that you can recall specific versions later.

`https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control`

In other words stores:

- files
- and all modifications on the latter.

Git is a distributed CVS

- Keep the files on **your machine**,
- **Synchronize** with a **server**,
- Any machine with correct rights can get the server version,
- And update it if needed.

Plan

Repository

```
/home/mfreeze/git_repo
|- .git/
|   |- config
|   `- ...
|- mywork/
|   |- workshops /
|   |   |- poema/
|   |   |   |- slides.tex
|   |   |   `- bash.sh
|   |   `-- future/
|   `- tsp/
|       `- poly_solver.jl
|- LICENCE
`- README.md
```

- **Directory** containing the tracked files,
- The directory must be “initialized”:
  - must contain a `.git` directory,
  - which contains git magic:
    - configuration files,
    - repository states,
    - …

Stage/Add

- Files **are not automagically** (nor automatically) tracked,
- Modifications on tracked file **are not automagically** (nor automatically) tracked,
- Add/stage is the act of (kindly) asking `git` to:
  - **Track** new files,
  - **Take modifications into consideration**.

Commit

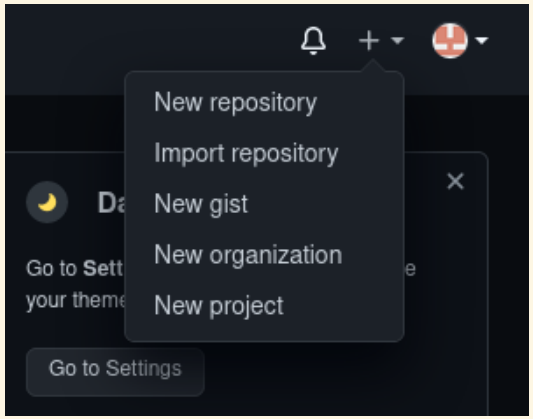
- A commit is a **set of modifications and/or additions**,
- Commits are **named**,
- They include **all** staged/added contents **since previous commit**,
- Each commit defines a snapshot of the repository.

Plan

1. A version control system (VCS)
2. Some vocabulary
3. First touch
4. Getting further

Clone the first repository

① Create an account on `https://github.com`,
② Create a new repository with `README` file:



③ Install `git`.
④ Change default git editor: `git config --global core.editor nano`.
⑤ Clone your newly created repository with `git clone` command.
⑥ You now have a local copy of the repository.

Simple workflow

Work on a git repository usually follows the same scheme:

1. **Retrieve** possible distant modifications,
2. Perform local modification,
3. **Stage** local modifications,
4. **Commit** local modifications,
5. **Retrieve** possible distant modifications,
6. Manage possible conflicts,
7. **Send** local modifications.

Get distant modifs

- Distant modifications can be **integrated** to local repository with: `git pull`
- Downloads distant commits,
- Try to integrate them.

Perform local modifs

① Create a `hello world` file,
② Edit the `README` file.

Stage modifs

- **Unstaged modifications** are modifications that are not tracked yet.
- They can be shown with `git status`.
- To stage a modification on a file (add it to the next commit), use `git add path/to/file`.

① List all modifications,
② Stage only the `README` modification.

Commit local

- To **validate** staged modification, we need to create a new commit.
- The `git commit` command is used:
  - without further options, editor is launched: write down commit name, save and quit
  - to avoid editor start, use `-m` option: `git commit -m "commit name"`
- Commit names are **very important**:
  - appear in logs, main github page, history, ...
  - they should give an **overview of the performed modifications**.

① Commit the `README` modification without `-m` option.
② Stage and commit the `hello_world` file with `-m` option.
③ Retrieve distant modifications.

Send local modifs

- At this point:
  - modifications are locally tracked,
  - but are not available for possible other user,
  - changes need to be **synchronized with distant repository**.
- Use `git push` command.

① Push local modifications to you repository.
② Use the same workflow to create a new file.
③ Create a new copy (`clone`) of the remote repository at another location.
④ Ensure that both repositories are identical.

Plan

1. A version control system (VCS)
2. Some vocabulary
3. First touch
4. Getting further

Conflict management

- When performing `git pull`:
  - Distant modifications are applied to local copy,
  - What if local copy also contains modifications?
    - Modifications occur at different location -> automatic merge
    - At the same location -> **CONFLICT**!
    - Conflicts need to be resolved by hand.
- When conflicts occur, no `push` is allowed before they are solved.

Conflict workflow

① Find conflict locations,
② Resolve conflicts,
③ Perform a full commit,
④ Retrieve distant modifications,
⑤ Push local modifications.

Seek. . .

- The `git diff` command can be used:
  - without argument `git diff` lists **pending modifications** compared to last commit,
  - when conflicts occur, conflicting files are **modified by git**,
  - thus they appear in `git diff` result,
  - to only get a list of conflicting files use: `git diff --name-only --diff-filter=U`.

. . . And destroy

A conflict **always looks like**:

```
<<<<<<< HEAD
    Local modification
=======
    Distant modification
>>>>>>> master
```

1. Chose the modifications you want to keep (can be a mix of both),
2. Remove all `<<<<<<<`, `=======` and `>>>>>>>` markers,
3. Save file,
4. Do it for all conflicts,
5. Perform a full commit `git commit -a`.

Working all together

1. Clone the newly created repository,
2. Wait until everyone has cloned,
3. Then in `id` file add your first and last name on a single line,
4. Validate changes and send them to server.
5. In the `institution` file, add your institute and send changes to the server. Ensure that each each institute appears only once.
6. Once everyone has finished, get distant modification.
7. Use `git log` command to see history of the repository.