Développement de logiciels libres

Pablo Rauzy
pablo.rauzy@univ-paris8.fr
pablo.rauzy.name/teaching/ddll

UFR MITSIC / L3 informatique

Séance 2

Gestion de versions avec Git

Gestion de versions avec Git

- Un logiciel de gestion de versions permet de stocker des fichiers en conservant une chronologie de modifications.
- Il sert aussi à la collaboration entre plusieurs développeur·e·s.

- Git est le logiciel de gestion de version décentralisé le plus répandu.
- Il a été écrit à l'origine par Linus Torvalds pour gérer le développement collaboratif du noyau Linux.
- *dépôt :* ensemble de fichiers gérés par Git, ainsi que les données de Git lui même concernant l'historique de leurs modifications.

- Il existe des interfaces graphiques pour Git mais il est la plupart du temps utilisé en ligne de commande.
- Nous allons commencer par les commandes permettant de l'utiliser seul.

Initialisation d'un dépôt

- `git init`

État du dépôt

- `git status`
- *Staging area :* endroit où vivent les modifications pas encore enregistrées dans l'historique.

Ajout de fichiers

- `git add <files>`
- `.gitignore`

Enregistrement de modifications

- `git commit [--all]`
- *commit :* enregistrement d'un ensemble de modifications apportées au projet.

- `git log [--oneline] [--name-status] [-n] [--author] [--since]`

Différence

- `git diff [--cached] [--color-words] [<file> | <commit> | <commit-range>]`
- *diff :* différences entre deux versions d'un (ensemble de) fichier(s).

- `git revert [--no-commit] <commits>`

Remonter le temps

- `git reset [--soft|--mixed|--hard] <commit>`
- `git checkout <commit> <file>`

- En plus de voyager dans le temps, il est possible de se promener dans des univers parallèles avec Git.

Point de divergence

- `git branch [<new-branch>]`

Changement d'univers

- `git checkout <branch>`

Retour vers le futur

- `git checkout <commit> -b <new-branch>`

- `git merge <branch>`

Récupération spécifique

- `git cherry-pick <commit>`

Gestion de conflits

- Parfois Git ne s'en sort pas tous seul et il a besoin qu'on l'aide à gérer des *conflits* de modifications parallèles.
- Ça peut se faire à la main, ou, si on sait qu'on veut garder les modifications d'un côté spécifique :
- `git checkout --ours|--theirs <files>`

Gestion des univers

- `git branch [-d|-D|-m|-M] <branch>`

- Voyons maintenant comment on peut travailler à plusieurs avec Git.

Initialiser un dépôt “serveur”

- `git init --bare`

Initialisation

- `git clone <remote>`

Récupération des modifications

- `git pull [<remote> [<branch>]]`

Envoie des modifications

- `git push [<remote> [<branch>]]`

Gestion des dépôts distants

- `git remote`
- `git remote add <name> <remote>`
- `git remote remove <name>`

- Une bonne idée est d'avoir un historique *sémantique*.
- Cela permet de mieux retrouver l'origine d'un problème, ou de récupérer des modifications précises et de manières utiles.

Petits commits

- Faire des petits commits qui ont un sens (pas tout commiter d'un coup à la fin de la journée).
- `git add --patch [files]`
- `git checkout --patch [files]`

Modifier le dernier commit

- Il arrive souvent qu'on fasse des petites erreur, comme oublier d'ajouter un nouveau fichier au dépôt par exemple.
- Dans ce cas ou peut faire ce qu'il faut (ajouter le fichier) puis faire un nouveau commit qui va remplacer le dernier en fusionnant avec lui.
- `git commit --amend`

Garder la branche principale propre

- Toujours garder la branche principale propre.
- C'est à dire développer des nouvelles fonctionnalités ou faire de gros changement dans des branches, et quand c'est prêt et que ça marche, fusionner dans la branche principale.
- ```
  git branch new_feature
  git checkout new_feature
  git commit …
  git commit …
  …
  git checkout master
  git merge new_feature
  ```
  ```

- On ne va pas voir cela dans ce cours mais Git permet aussi :
  - de réécrire l'histoire (avec `git rebase`),
  - de retrouver le commit responsable d'un comportement du logiciel (avec `git bisect`),
  - de déclencher des scripts lors de certains évènements (avec les hooks),
  - etc.