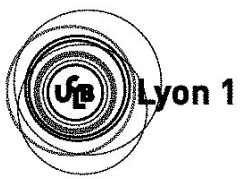
# Aide mémoire sur Git et Gitlab

Anne Cadiou, Laurent Pouilloux

Laboratoire de Mécanique des Fluides et d'Acoustique

Ateliers et Séminaires Pour l'Informatique et le Calcul Scientifique
PMCS2I - LMFA
Vendredi 8 novembre 2019








# À quoi ça sert ?

Systèmes de gestion de version (*en anglais Version System Control*)

**Sert à :**

- enregistrer et sauvegarder les développements au cours du temps
- revenir en arrière sur une version spécifique
- collaborer sur un même document
- enregistrer les modifications avec auteur et date

# Historique des principaux opensources (1/2)

- **Système local**
  - RCS (Revision Control System) 1982
    les développeurs partagent le même système de fichiers
- **Système client-serveur**
  - CVS (Concurrent Versions System) 1990
    dérivé de RCS, les développeurs partagent un dépôt unique
  - Subversion (SVN) 2000
    dérivé de CVS, fusions facilitées

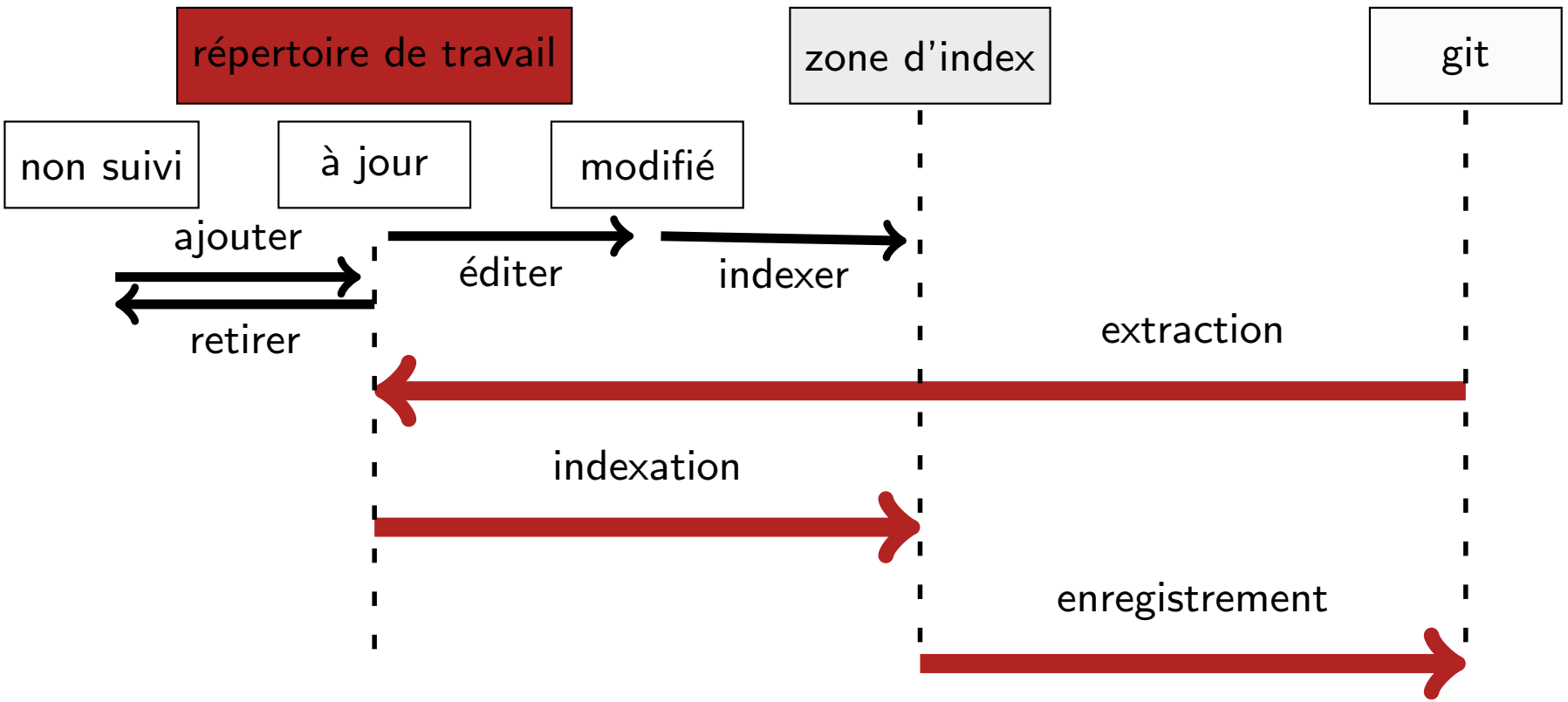
# Historique des principaux opensources (2/2)

Distributed Verson Control System (DVCS)

- **Système décentralisé**
  - Git
    conçu par Linus Torvalds pour le projet du noyau Linux en 2005
  - Mercurial
    en Python, évolution depuis 2005 de l'opensource de BitKeeper, anciennement utilisé pour le développement du noyau Linux (2002-2005)
  - Bazaar
    en Python, depuis 2005
  - Fossill
    en C, depuis 2006
  - Veracity
    depuis 2011
    intègre le suivi des bug et les outils de développement agile
    pas de rebase

# Principe de base

Cycle de vie d'un fichier



# Principales commandes

## Création d'un dépôt

À partir de rien

```
$ git init
```

Récupération d'un dépôt local existant

```
$ git clone /path/to/repository
```

Récupération d'un dépôt distant

```
$ git clone username@host:/path/to/repository.git
```

## Observer l'état du dépôt

Lister les fichiers par rapport à l'index

```
$ git status
```

Voir les changements pour un `commit` ou un fichier

```
$ git show [commit]:[file]
```

Afficher l'historique des enregistrements

```
$ git log
```

Lister par date et auteur les changements dans un fichier

```
$ git blame [file]
```

Afficher les différences sur les fichiers en attente (resp. déjà indexés)

```
$ git diff (resp. --cached)
```

Afficher les différences sur tous les fichiers

```
$ git diff HEAD
```

Afficher les différences entre deux commits

```
$ git diff commit1 commit2
```

## Effectuer des modifications

Inscription dans l'index (stage), prêt pour l'enregistrement (commit)

```
$ git add [file]
```

Tout mettre dans l'index

```
$ git add .
```

Supprimer ou renommer des fichiers

```
$ git rm file
$ git mv file1 file2
```

Enregistrer dans l'historique les fichiers nouvellement indexés

```
$ git commit -m "commit message"
```

Enregistrer tous les fichiers de l'index l'historique

```
$ git commit -m "commit message" -a
```

## Revenir en arrière

Supprimer le messsage qui vient d'être mis à l'historique

```
$ git commit --amend
```

Désindexer un fichier en gardant les modifications

```
$ git reset [file]
```

Revenir au dernier `commit` (dernier enregistrement dans l'historique)

```
$ git reset --hard [SHA1]
```

Récupérer une version antérieure dans détruire les plus récentes

```
$ git checkout SHA1
```

Mettre de côté un travail inachevé

```
$ git stash
```

## Travailler avec des branches

Lister les branches locales [et distantes]

```
$ git branch [-av]
```

Créer la `dev`

```
$ git branch dev
```

Passer dans la branche `dev` et mettre à jour le répertoire de travail

```
$ git checkout dev
```

Créer une nouvelle branche (`new`) et travailler dedans

```
$ git checkout -b new
```

Détruire la branche (`new`)

```
$ git branch -d new
```

Fusionner la branche `branch1` dans la branche `branch2`

```
$ git checkout branch2
$ git merge branch1
```

## Synchroniser

Récupérer les dernières modifications du dépôt (`origin`) sans les intégrer localement (no merge)

```
$ git fetch
```

Récupérer les dernières modifications du dépôt et les fusionner localement

```
$ git pull
```

Récupérer les dernières modifications du dépôt, fusionner et construire un historique linéaire

```
$ git pull --rebase
```

Pousser les modifications locales dans le dépôt (`origin`)

```
$ git push
$ git push origin master
```

# Configurer git

```
$ git config --global user.name "Anne Cadiou"
$ git config --global user.email "anne.cadiou@ec-lyon.fr"
```

```
$ git config --global core.editor vim
$ git config --global diff.external vimdiff
$ git config --global color.ui true
$ git config --global merge.tool vimdiff
...
$ git config --list
```

Stocké dans

```
$ ls ~/.gitconfig
```

# Exemple de configuration

```
[user]
  name = Anne Cadiou
  email = anne.cadiou@ec-lyon.fr
[core]
  editor = vim
[color]
  ui = true
[merge]
  tool = vimdiff
[alias]
  lg = log --pretty=format:\"%h%x09%an%x09%ad%x09%s\"
```

# Limitations

Ce que Git gère très bien :

- ✓ les scripts ou code de calcul
- ✓ les documents LaTeX
- ✓ les fichiers texte (configuration, scripts etc.)
- ✓ les sources html

Ce que Git gère mal :

- ✗ les gros fichiers binaires
- ✗ le texte formaté (documents de type Microsoft Office, OpenOffice etc.)
- ✗ les bases de données (type MySQL)

# Contournements et solutions possibles

Solutions pour gérer les gros fichiers binaires :

- git-annex
- git-fat, git-lfs
- git-bigfile
- git-bigstore
- git-sym
- etc.

Gestion des fichiers binaires de type Microsoft Office, PDF etc. :

✓ versionner les fichiers et combiner avec un outil exhibant les différences comme Word Diff, Pandoc etc.

Gestion des bases de données MySQL :

✓ versionner la sortie de mysql dump

⇒ Combiner les outils

# Service en ligne d'hébergement de projets

- Forge logicielle, gestion de projets de développement collaboratifs
  - Redmine, Tuleap, Trac,
  - Gitlab, Github, Gogs, Coding,
  - FusionForge (reprise opensource de GForge, devenue propriétaire),
  - SourceForge, SourceSup, ...

  Fonctions disponibles d'une forge (*d'après Wikipédia*)
  - **système de gestion des versions** (par exemple, via Git ou Mercurial)
  - gestionnaire de listes de discussion (et/ou de forums)
  - outil de suivi des bugs
  - gestionnaire de documentation (souvent sur le principe du wiki)
  - gestion des tâches

Services d'hébergement externes

```
https://sourceforge.net
https://www.github.com
```

ou interne (ex. Gitlab de votre labo, entité, établissement, ...)

```
https://gitlab.mecaflu.ec-lyon.fr
https://forge.p2chpd.univ-lyon1.fr
```

# Quelques comparaisons

## Comparaison de services d'hébergement Git en version gratuite

### Github

- ✓ très gros projet (138+ millions dépôts, 600 employés)
- ✓ Grande interopérabilité avec d'autres outils
- ✗ dépôts publics uniquement
- ✗ pas d'instance privée

### Bitbucket

- ✓ très gros projet
- ✓ Git & Mercurial
- ✓ intégration dans les produits Atlassian
- ✗ 5 utilisateurs max/dépôt
- ✗ pas d'instance privée

### GitLab

- ✓ projet pérenne et dynamique (132 employés, 33 pays)
- ✓ pas de limite dans la version hébergée
- ✗ des lenteurs sur le site gitlab.com
- ✓ instance privée opensource
- ✓ outils d'intégration continue natifs

(de Matthieu Boileau, nov. 2016, d'après `http://comparegithosting.com`)

# Hébergement et interfaces

| Github | GitLab | Gogs |
|---|---|---|
| ✓ Git, SVN, Hg, TFS | ✗ Git | ✗ Git |
| ✗ externe | ✓ interne/externe | ✗ interne (light) |
| ✓ très large communauté | ✓ très utilisé | ✓ relativement nouveau |
| ✓ prise en main intuitive | ✓ prise en main relativement rapide | ✓ sur le modèle de Github |
| prog. en Ruby | prog. en Ruby | prog. en Go |
| | ✓ extensions (Continuous Integration, Large File Storage, ...) | |

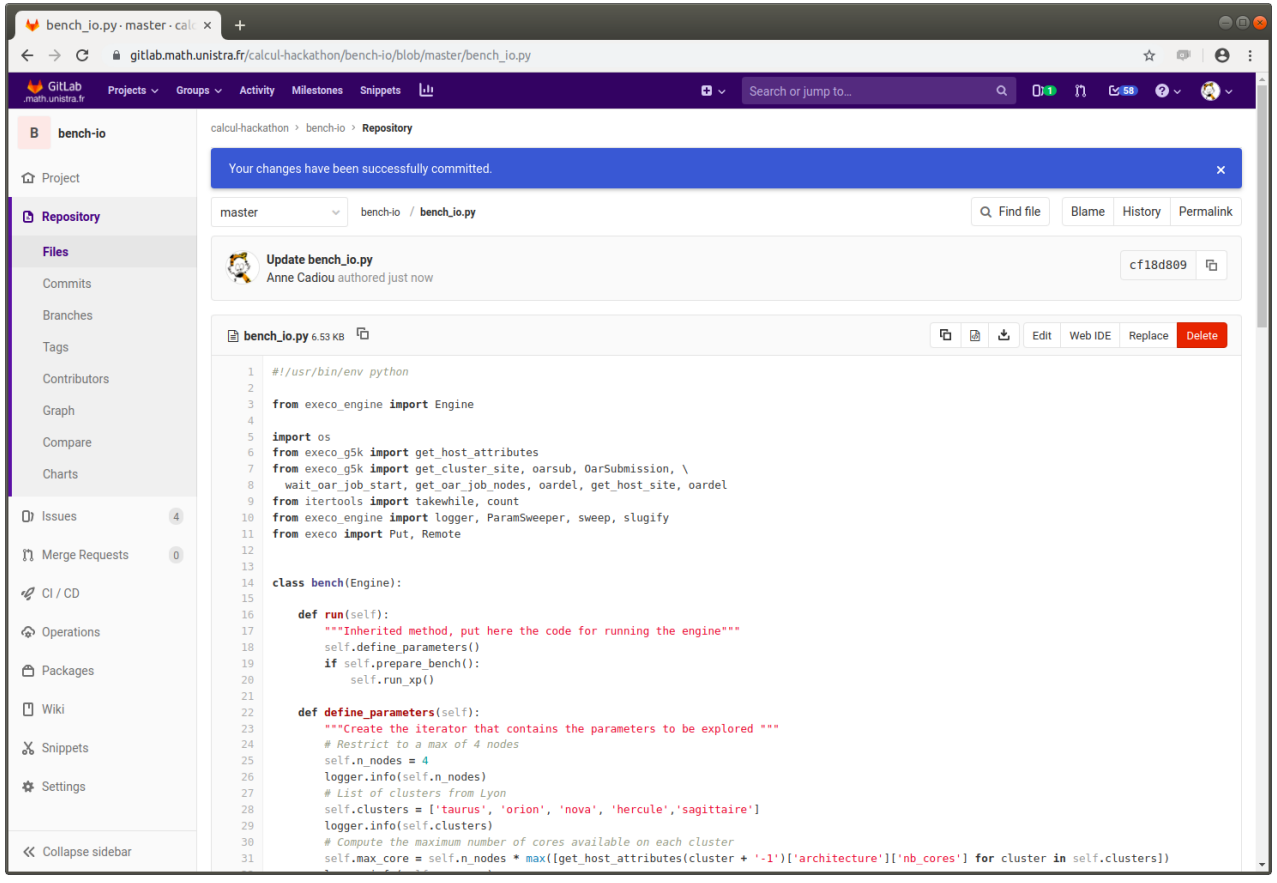
⇒ Combiner éventuellement suivant le mode de diffusion choisi

# Exploitation de Gitlab par l'utilisateur

## Les fonctionnalités de GitLab

- Création rapide de projets avec gestion des droits (public, privé,...)
- Groupes de projets
- Historique des commits
- Edition en ligne
- Outils annexes :
  - statistiques
  - wiki
  - gestionnaire de tickets
  - notifications par mail
- Intégration continue avec `gitlab-ci`

# Interface




Démo