

## Suivre et partager ses sources avec GitLab

Matthieu Boileau, Alexis Palaticky

CNRS - Université de Strasbourg

20 novembre 2017



&



GitLab

# Outline

1 Intérêt et applications de git

2 Conclusion

## Pourquoi utiliser un suivi de version ?

- **Enregistrer** les modifications d'un jeu de fichiers au cours du temps
- Rester **réversible** :
  - pouvoir retourner à une version antérieure,
  - comparer avec une version antérieure
- **Documenter** les modifications (date, auteur et message d'accompagnement)
- Un logiciel de suivi de version (VCS pour *Version Control System* en anglais) comme Git gère très bien tout projet qui se présente sous la forme de **fichiers sources**
- C'est le meilleur moyen de **collaborer** sur des sources !

## Pourquoi utiliser un suivi de version ?



### Ce que Git gère très bien :

- ✓ les scripts ou code de calcul
- ✓ les documents  $\text{\LaTeX}$  (cet exposé est sur [GitLab](#) !)
- ✓ les fichiers texte de configuration
- ✓ les sources html
- ✓ etc.

### Ce que Git gère mal :

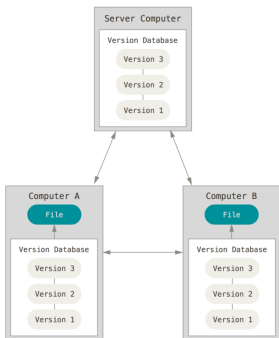
- ✗ les gros fichiers binaires
- ✗ les documents Microsoft Office ou OpenOffice
- ✗ le texte formaté en général
- ✗ les bases de données (type mysql)
- ✗ etc.

## Comparaison avec un système de partage de type Owncloud ou Seafile

	Owncloud/Seafile	Git
<b>Type de fichiers</b>	<ul style="list-style-type: none"> <li>✓ tous types</li> </ul>	<ul style="list-style-type: none"> <li>✓ suivi pour les fichiers sources</li> <li>✗ pas de suivi pour les binaires</li> <li>✗ pas adapté aux gros fichiers (sauf avec git-lfs)</li> </ul>
<b>Suivi de version</b>	<ul style="list-style-type: none"> <li>✗ très limité</li> </ul>	<ul style="list-style-type: none"> <li>✓ outil avancé</li> </ul>
<b>Partage</b>	<ul style="list-style-type: none"> <li>✗ modèle centralisé uniquement</li> <li>✗ synchronisations essentiellement automatiques</li> </ul>	<ul style="list-style-type: none"> <li>✓ modèle distribué</li> <li>✓ on contrôle les synchronisations</li> </ul>
<b>Prise en main</b>	<ul style="list-style-type: none"> <li>✓ très simple</li> </ul>	<ul style="list-style-type: none"> <li>✗ demande un apprentissage</li> </ul>

## Un suivi de version distribué

- les clients possèdent un miroir complet de la base de données du serveur
- on peut travailler en mode déconnecté et synchroniser quand on le souhaite
- indirectement, on crée des sauvegardes multiples



(Source: Pro Git book <http://git-scm.com/book>)

# Git en pratique

## Git en ligne de commande dans le Terminal :

```
git add
git commit -m "My commit message"
git status
git log
git push
git pull
git checkout
git diff
etc.
```

```
boiteau@boiteau: ~$ git help --all
usage: git [--version] [--help] [-C <path>] [-c name=value]
          [--exec-path<path>] [--html-path] [--man-path] [--info-path]
          [-p | --paginate | --no-pager] [--no-replace-objects] [--bare]
          [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
          <command> [<args>]

available git commands in '/Applications/Xcode.app/Contents/Developer/usr/libexec/git-core'

add                diff-files          merge-recursive    reset
add-interactive    diff-index          merge-one-file     rev-list
am                 diff-tree           merge-ours         rev-parse
annotate           difftool            merge-recurse      revert
apply              difftool--helper   merge-resolve      rm
archimport         fast-export         merge-subtree      send-email
archive            fast-import        merge-tree         send-pack
bisect             fetch               mergetool          sh-libs--envsubst
bisect--helper     fetch-pack          aktag              shell
blame              filter-branch      mv                 shortlog
branch             for-each-ref        name-rev           show
bundle             format-patch       notes              show-branch
cat-file           fsck                pack-objects       show-index
check-attr         get-tar-commit-id  pack-redundant    show-ref
check-ignore       gc                  pack-refs          stage
check-mailmap      grep                patch-id           stash
check-ref-format  gui--askpass       prune              status
checkout           hash-object        prune-packed       stripspace
checkout-index     help                pull              submodule
cherry             http-backend       quitasport        svn
cherry-pick        http-fetch         read-tree          symbolic-ref
citool             http-push          rebase             tag
clean              imap-send          receive-pack       unpack-file
clone              index-pack         reflog            unpack-objects
column             init                remote             update-index
commit             init-db            remote-ext         update-ref
commit-tree        instaweb           remote-fd          update-server-info
config             interpret-trailers remote-ftp         upload-archive
count-objects      log                 remote-ftp        upload-pack
credential         ls-remote          remote-ftp        var
credential-cache   ls-tree            remote-https     verify-commit
credential-cache--daemon ls-tree            remote-https     verify-pack
credential-store   mailinfo           remote-https     verify-tag
cvsimport          mailsplit          remote-testsvn    web--browse
cvsimport          merge              repack            whatchanged
cvsimport          merge-base         request-pull      worktree
cvsimport          merge-file         rerere            write-tree
cvsimport          merge-index

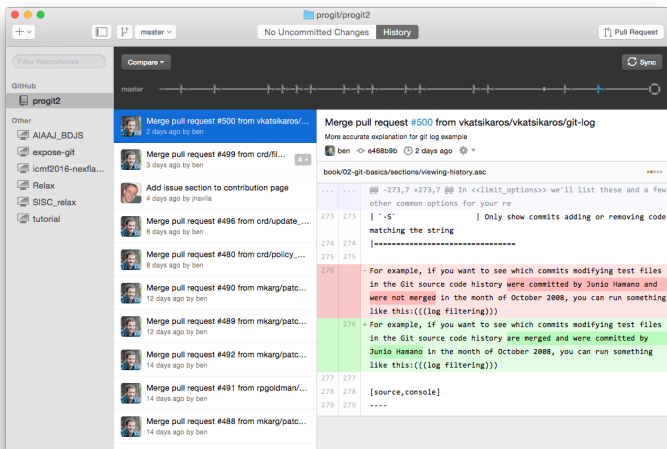
git commands available from elsewhere on your $PATH

diff-cmd.sh latexdiff lfs

'git help -a' and 'git help -g' list available subcommands and some
concept guides. See 'git help <command>' or 'git help <concept>'
to read about a specific subcommand or concept.
boiteau@boiteau: ~$
```

# Git en pratique

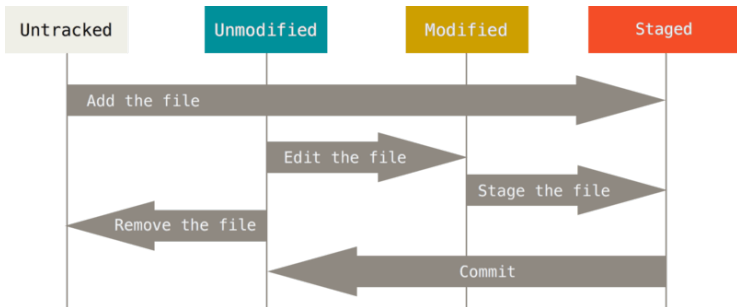
Avec une interface graphique très simple comme **GitHub Desktop**, on couvre  $\approx 90\%$  de l'utilisation courante de git :





## Les quatre statuts des fichiers suivis

- Le cycle de vie d'un fichier suivi avec Git

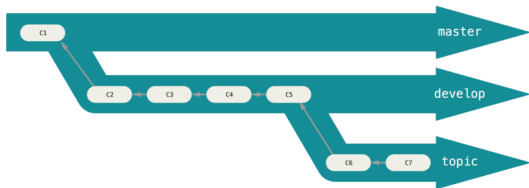


(Source: Pro Git book <http://git-scm.com/book>)

- Les fichiers qui ne sont pas des sources (fichiers objets, fichiers de compilations, exécutables, etc.) peuvent être ignorés.

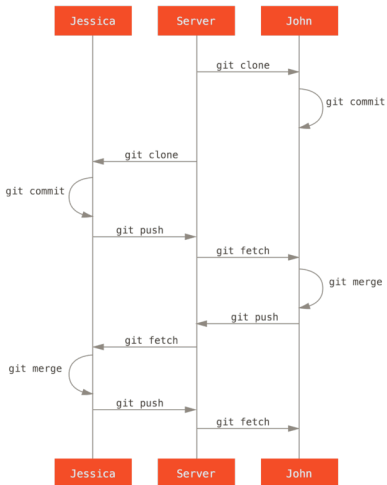
## Git et le système des branches

- Git permet de créer et fusionner très facilement des branches
- un système de branches permet de préserver une version stable (branche *master*) sans limiter les développements (branche *develop*)
- les branches sont particulièrement utiles pour le travail collaboratif et par sujet (*topic*).



(Source: Pro Git book <http://git-scm.com/book>)

# Travail collaboratif avec Git : le workflow typique d'une petite équipe



En pratique :

- Côté serveur : **GitLab**
- Côté clients (John et Jessica) : ligne de commande ou client graphique (GitHub Desktop, par exemple)

(Source: Pro Git book <http://git-scm.com/book>)

## Travail collaboratif avec Git

- Même si **Git** est basé sur un modèle distribué, le partage se fait essentiellement via un **serveur Git**.

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

Github	Bitbucket	GitLab
✓ très gros projet (73+ millions dépôts, 718 employés) [Ref.]	✓ très gros projet	✓ projet pérenne (204 employés, 39 pays) [Ref.]
✓ Grande interopérabilité avec d'autres outils	✓ Git & Mercurial	✓ peu de limites dans la version hébergée
✗ dépôts publics uniquement	✗ 5 utilisateurs max/dépôt	✗ certaines limitations en taille
✗ pas d'instance privée	✗ pas d'instance privée	✓ instance privée opensource
		✓ outils d'intégration continue natifs

(décembre 2017, d'après <http://comparegithosting.com>)

Parmi ces 3 géants, **GitLab** est la seule solution qui permet d'héberger une instance privée (problème de l'hébergement des données de recherche sur des clouds privés).

## Installer son serveur GitLab ?

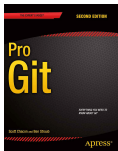
Pour accéder à un serveur GitLab, de plus en plus de solutions institutionnelles existent comme `https://gitlab.unistra.fr` alors...

### pourquoi déployer un serveur GitLab local ?

- pour maîtriser l'accès au service (en particulier pour les collaborateurs extérieurs)
- pour maîtriser la maintenance (mise à jour, etc.)
- pour maîtriser les fonctionnalités additionnelles : Mattermost, GitLab Pages, etc.
- assez peu gourmand en ressources informatiques
- effort modéré pour l'administration (mais augmente avec les fonctionnalités)
- politiquement : on identifie le contenu à l'entité qui l'héberge

# Documentation et bonnes pratiques

## Documentation



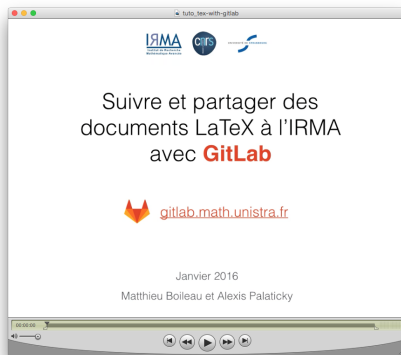
ProGit book : une référence (libre)

- Pro Git book (version française) : <https://git-scm.com/book/fr>
- la documentation officielle de Git : <http://git-scm.com/documentation>
- un manuel concis : <http://gitref.org/index.html>
- la doc GitLab et son aide contextuelle : <https://gitlab.math.unistra.fr>

## Quelques recommandations

- Dans les documents textes ( $\text{\LaTeX}$ ), ne pas écrire plus d'**1 phrase par ligne**
- Faire de **nombreux commits** (Git est fait pour ça) contenant des modifications **petites et cohérentes** plutôt que l'inverse
- Utiliser les branches, les *merge requests*, la revue de code, les *issues*
- Faire de **l'intégration continue** (facile avec **GitLab CI !**)

# Documentation et bonnes pratiques



Un **tutoriel vidéo** de 15 min pour suivre des sources  $\text{\LaTeX}$  avec GitLab.