



Formation Git

Romain Bohdanowicz

Twitter : @bioub

<http://formation.tech/>





Introduction



- Système de gestion de version distribué (DVCS)
- Créé par Linus Torvalds en 2005 pour gérer le code source du noyau Linux
- Permet de sauvegarder les différences entre plusieurs versions de fichiers (plutôt texte), principalement du code source
- Facilite la collaboration entre plusieurs développeurs

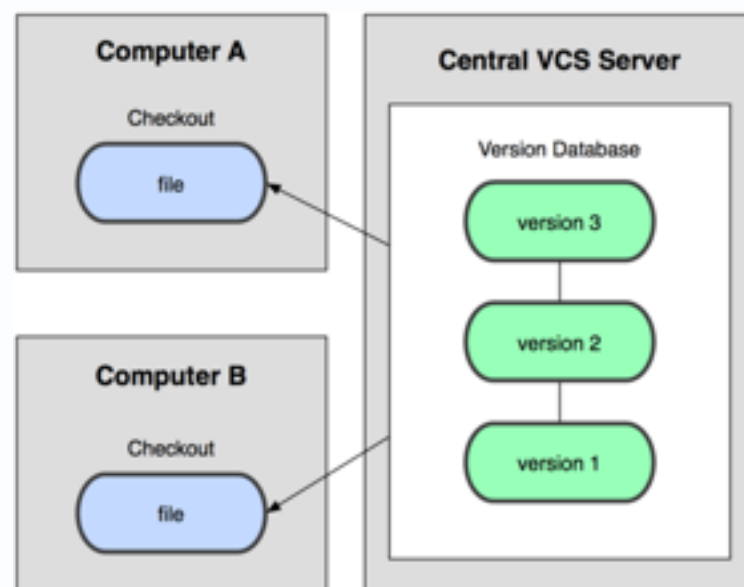


- ▶ Git Book
<https://git-scm.com/book/fr/v2>
- ▶ Tutoriels d'Atlassian (SourceTree, BitBucket...)
<https://www.atlassian.com/git/tutorials/>
- ▶ Git Cheatsheet
<http://ndpsoftware.com/git-cheatsheet.html>

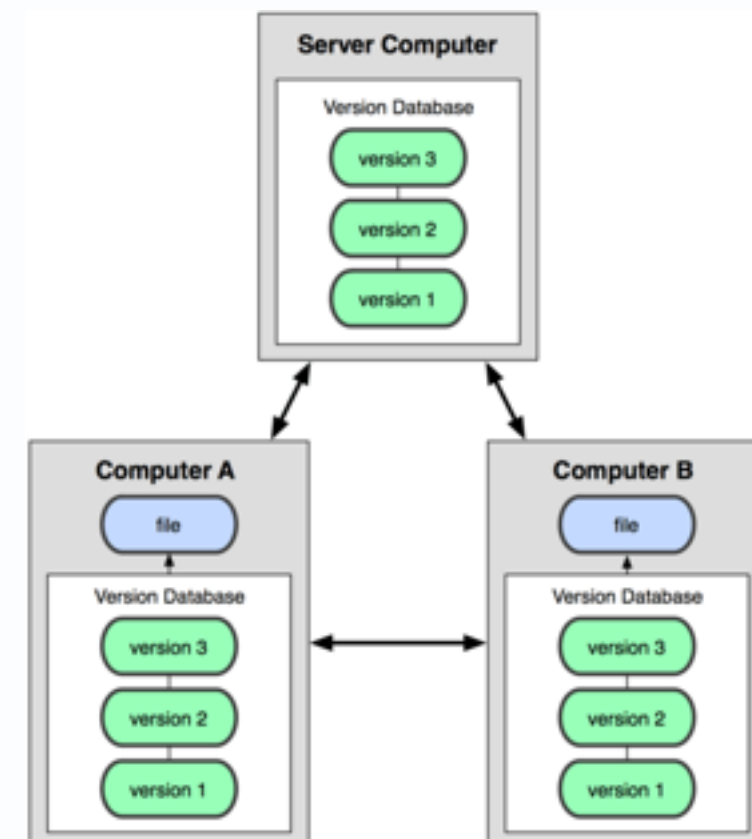
Git - DVCS vs CVCS



- Système de gestion de version centralisé (CVCS)
- Ex : CVS, Subversion



- Système de gestion de version distribué (DVCS)
- Ex : Git, Mercurial





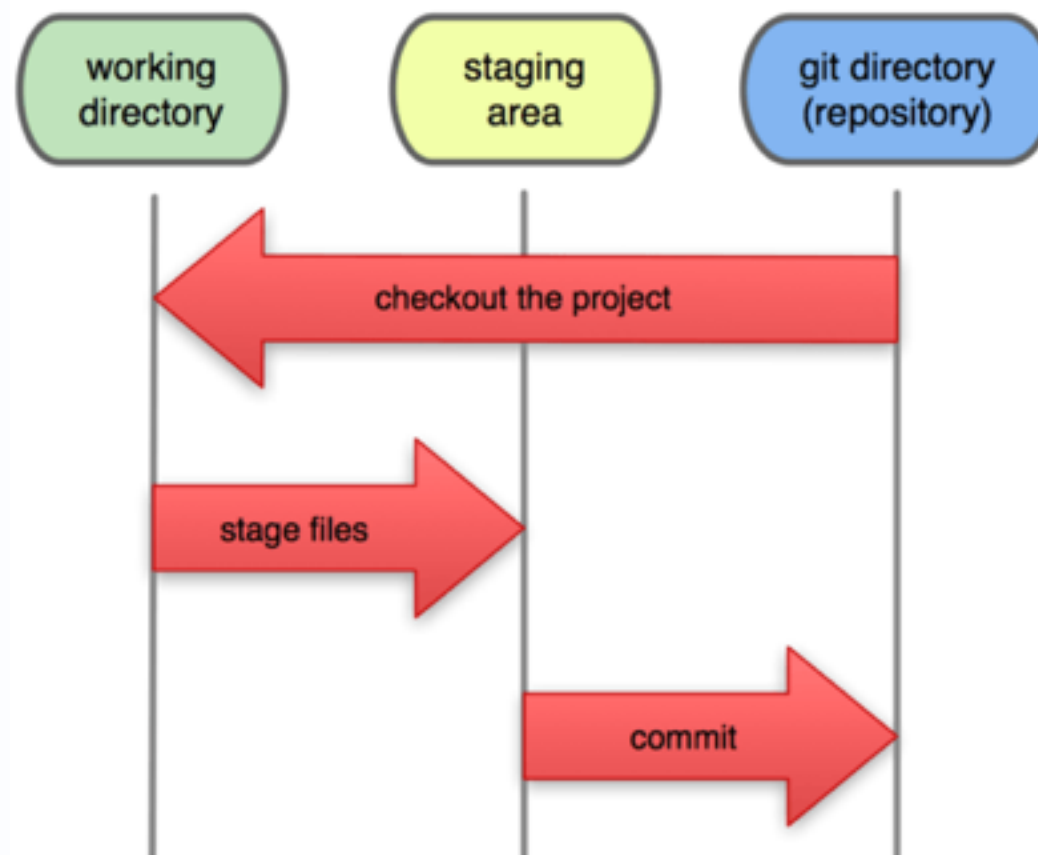
▸ Avantages du DVCS

- l'historique des sources est présent sur plusieurs machines (crash de disque...)
- ne pas avoir à être connecté au réseau pour versionner
- permet de collaborer à un projet et obtenir l'autorisation des mainteneurs à posteriori
- plus rapide (accès locaux)



▸ 3 états pour les fichiers

- working directory (changement non-versionnés et fichiers non-surveillés : untracked)
- staging (à publier lors d'un prochain commit, on parle aussi d'index ou de cache)
- git repository (modifications enregistrées)





▸ Linux

- `yum install git`
- `apt-get install git`

▸ Mac OS X

- <http://sourceforge.net/projects/git-osx-installer/>
- `brew install git`

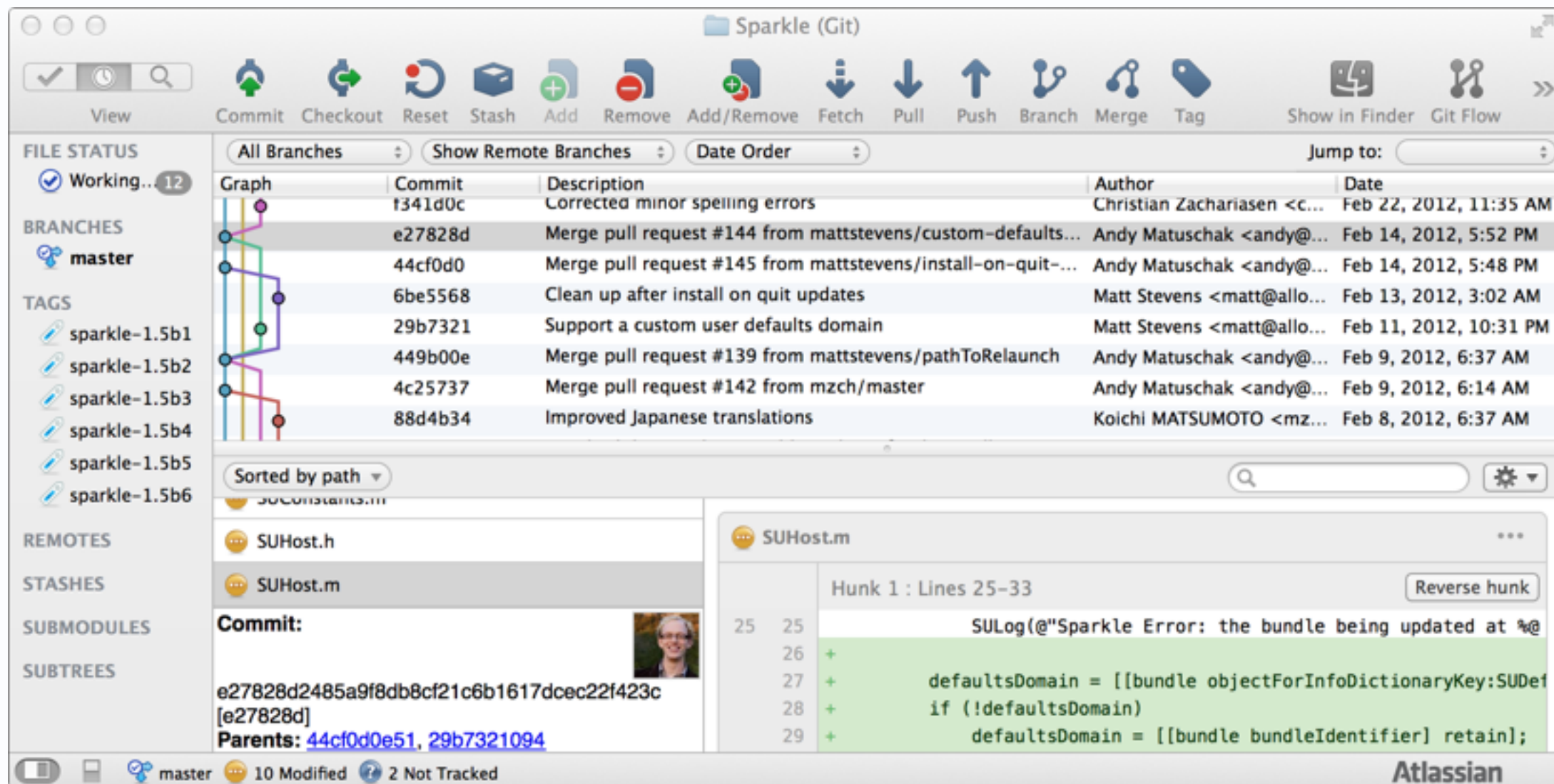
▸ Windows

- <https://git-for-windows.github.io> (penser à ajouter Git au PATH pour pouvoir s'en servir sous DOS, valeur par défaut dans le dernier installeur)

Git - GUI



- ▶ Mac OS X / Windows
 - SourceTree (open-source) : <https://www.sourcetreeapp.com>





Commandes de base



- Vérifier la version de git installée

- `git --version`

- Configurer son environnement

- Afficher la configuration (globale à la machine + locale au projet) :

- `git config --list`

- S'identifier :

- `git config --global user.name "Romain Bohdanowicz"`

- `git config --global user.email "romain.bohdanowicz@gmail.com"`

- Configurer le proxy :

- `git config --global http.proxy http://user:pass@proxyhost:proxyport`

- `git config --global https.proxy http://`

- `user:pass@proxyhost:proxyport`

- Modifier l'éditeur

- `git config --global core.editor notepad`

- `git config --global core.editor "'C:/Program Files/Notepad++/notepad++.exe' -multiInst -notabbar -nosession -noPlugin"`



- Obtenir de l'aide, aide sur une commande, sur un concept
 - `git help`
 - `git help config`
 - `git help tutorial`

Git - Démarrage et Etats



- ▶ Créer un repository

- `git init`

- ▶ Obtenir le status du projet

- `git status`

Pour voir le contenu des répertoires untracked

`git status -u`

A screenshot of a macOS terminal window titled "Tests-Git — bash — 78x15". The terminal shows the output of the `git status` command. The output indicates an initial commit on the master branch and lists four untracked files: README.md, index.html, scripts.js, and style.css. It also provides instructions on how to track these files using `git add`.

```
MacBook-Pro-de-Romain:Tests-Git romain$ git status
On branch master

Initial commit

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        README.md
        index.html
        scripts.js
        style.css

nothing added to commit but untracked files present (use "git add" to track)
MacBook-Pro-de-Romain:Tests-Git romain$
```



- Ajouter des sources (nouvelles ou modifiées) à l'index (add ou son alias stage)
 - `git add README.md`
 - `git stage README.md`
 - `git add *.{css,js,html}`
 - `git add .`



▸ Versionner

- `git commit -m "Version initiale du projet"`
Le message et l'utilisateur paramétrés sont indispensables pour obtenir un historique clair

```
MacBook-Pro-de-Romain:Tests-Git — bash — 84x8
MacBook-Pro-de-Romain:Tests-Git romain$ git commit -m "Version initiale du projet"
[master (root-commit) f7bcc2b] Version initiale du projet
4 files changed, 0 insertions(+), 0 deletions(-)
create mode 100644 README.md
create mode 100644 index.html
create mode 100644 scripts.js
create mode 100644 style.css
MacBook-Pro-de-Romain:Tests-Git romain$
```



- ▶ Supprimer des modifications de l'index (mais les conserver dans le working directory = unstage)
 - S'il n'y a jamais eu de commit :
`git rm --cached README.md`
 - S'il y a déjà eu des commits :
`git reset HEAD README.md`
- ▶ Supprimer un fichier dans le working directory
 - Le supprimer simplement sur le disque
 - Ou en ligne de commande :
`git rm README.md`
- ▶ Annuler les modifications d'un fichier dans le working directory
 - `git checkout HEAD README.md`

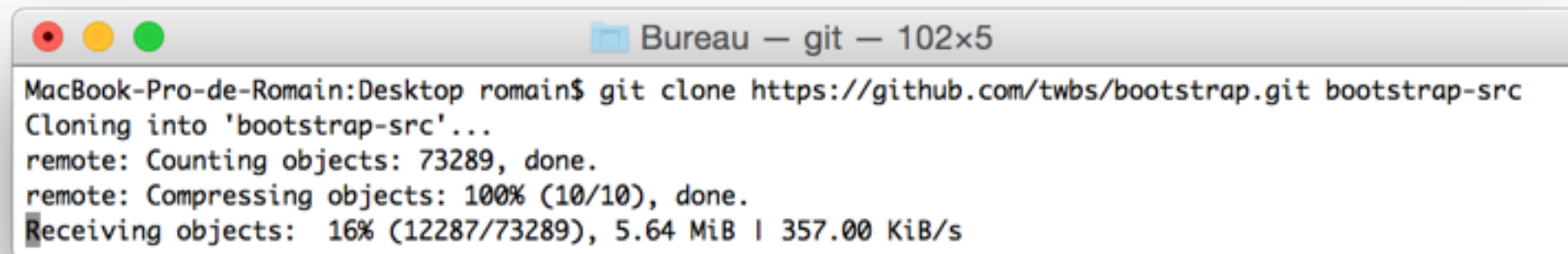


▸ Cloner un repository existant

Opération lente car il faut télécharger tout l'historique.

- `git clone https://github.com/twbs/bootstrap.git chemin_vers_le_dossier_destination`

Le nom du répertoire est facultatif.

A screenshot of a macOS terminal window titled "Bureau — git — 102x5". The terminal shows the execution of the command `git clone https://github.com/twbs/bootstrap.git bootstrap-src`. The output indicates the cloning process is in progress, with progress bars for counting, compressing, and receiving objects. The window has standard macOS window controls (red, yellow, green buttons) in the top left corner.

```
MacBook-Pro-de-Romain:Desktop romain$ git clone https://github.com/twbs/bootstrap.git bootstrap-src
Cloning into 'bootstrap-src'...
remote: Counting objects: 73289, done.
remote: Compressing objects: 100% (10/10), done.
Receiving objects: 16% (12287/73289), 5.64 MiB | 357.00 KiB/s
```

- On peut indiquer une profondeur si on ne souhaite pas récupérer tout l'historique (plus rapide)

`git clone --depth 1 https://github.com/twbs/bootstrap.git`



- Renommer (ou déplacer) un fichier
 - `git mv nom_origine nom_cible`
Eviter de faire un renommage sur en passant par le système de fichier (git serait perdu).



► Ignorer des fichiers

Créer un fichier .gitignore

(peut également se faire dans .git/info/exclude)

- Le fichier .gitignore permet d'ignorer tout les fichiers ou dossier indiqués
- Y compris dans les sous-répertoires

A screenshot of a code editor window showing a file named ".gitignore". The file contains five lines of text, each representing a directory to be ignored: ".idea", "node_modules", "bower_components", "dist", and "maquette". The lines are numbered 1 through 5 on the left margin. The sixth line is highlighted in yellow.

```
1 .idea
2 node_modules
3 bower_components
4 dist
5 maquette
6
```

Git - Afficher l'historique



▸ Historique des modifications

- `git log`
- `git log --pretty=format:"%h - %an : %s"`
- `git log --oneline`
- `git log --graph`
- `git log --decorate`
- `git log --pretty=format:"%h - %ar - %an : %Cgreen %s"`

▸ Pourquoi pas un alias ?

- `git config --global alias.ll "log --oneline --decorate --graph --"`
- `git ll`

A screenshot of a macOS terminal window titled 'Formation_Git_EDF_2016_03 -- bash -- 94x22'. The terminal shows the command 'git ll' being executed, which displays a compact, color-coded history of commits. The output includes commit hashes, branch names (like HEAD, master, origin/master, origin/HEAD), and commit messages, such as 'Déplacement du fichier merge-rebase.md', 'Merge branch 'doc-merge'', 'Ajout de la doc sur merge et rebase', and 'Chapitre sur les branches'. The terminal window has standard macOS window controls (red, yellow, green buttons) at the top left.

```
MacBook-Pro-de-Romain:Formation_Git_EDF_2016_03 romain$ git ll
* b65bafa (HEAD -> master, origin/master, origin/HEAD) Déplacement du fichier merge-rebase.md
* 3d81642 (tag: 0.10.0) Merge branch 'doc-merge'
| \
| * ebb5e47 Ajout de la doc sur merge et rebase
| /
* 59d90ee (tag: 0.9.0) Chapitre sur les branches
* f1e3733 Modifier l'éditeur sous Windows
* 4eda85f Ajout de quelques commandes sur git help, git help commande, git help concept
* 116225c Ajout du support PDF
* 40b0bca Chapitre réécrire l'histoire
* ad16d45 Cours + création du fichier .gitignore
* f5cf786 Chapitre sur git stash
* d0df8df Chapitre sur git diff
* a73f329 Ajout de la doc sur git add -p
* 64da03d Chapitre sur la suppression d'un fichier de la staging area
* eec5a0e Chapitre sur git clone
* 836c298 Chapitre sur git checkout
* bd5ba2b Chapitre sur git commit
* 7805593 Chapitre sur git add
* 1d14cd0 Création du cours introduction (chapitres config, création de dépôt, infos)
MacBook-Pro-de-Romain:Formation_Git_EDF_2016_03 romain$
```



▸ Voir les différences

- Différence entre working directory et le dernier commit
`git diff`
- Différence entre la staging area et le dernier commit
`git diff --cached`
- Pour voir les changements mot par mot
`git diff --word-diff`
- Pour voir les changements lettre par lettre
`git diff --color-words=.`
- Pour voir les différence entre le dernier commit et le précédent
`git diff HEAD^ HEAD`



- Pour mettre de côté des modifications sans les placer dans l'historique
 - pour voir les changements déjà mis de côté
`git stash list`
 - pour mettre de côté des modifications en dehors de l'historique (-u pour garder les fichiers non surveillés -- untracked --)
`git stash save -u "Description des changements"`
 - pour rappatrier le dernier stash sauvegardé
`git stash pop`
 - pour rappatrier le stash `stash@{1}` (voir `git stash list`)
`git stash pop stash@{1}`
** Attention : ** si conflit il faut éditer manuellement le fichier et retirer les lignes `<<<<<<`, `=====`, `>>>>>>` puis faire un commit.
 - pour supprimer (à faire manuellement en cas de conflit sur `git stash pop`)
`git stash drop`



Réécrire l'histoire



► Pourquoi réécrire l'histoire ?

- Parce que vous avez oublié un fichier dans un commit
- Parce que un bug subsiste ou qu'un nouveau bug à été introduit dans un commit et vous ne souhaitez par revenir à des bugs dans votre historique
- Parce que vos commit ne sont pas assez « atomiques », il contient 2 ou 3 fonctionnalités à la fois

► Attention à ne pas réécrire une histoire déjà partagée

- Extrait de la doc officielle de git rebase :

« Ne rebasez jamais des commits qui ont déjà été poussés sur un dépôt public. Si vous suivez ce conseil, tout ira bien. Sinon, de nombreuses personnes vont vous hair et vous serez méprisé par vos amis et votre famille. »

<https://git-scm.com/book/fr/v2/Les-branches-avec-Git-Rebaser-Rebasing>



▸ Modifier le dernier commit

- `git commit -m 'validation initiale'`
- `git add fichier-oublie.ext`
- `git commit --amend`
Ou sans changer le message de commit :
`git commit --amend --no-edit`

▸ Pourquoi pas un alias :

- `git config --global alias.oops "commit --amend --no-edit"`
- `git commit -m 'validation initiale'`
- `git add fichier-oublie.ext`
- `git oops`



- Juste pour voir :
 - `git checkout HEAD~2`
 - `git checkout HEAD^^`
 - `git checkout 795d220`
- Pour remettre le curseur à son état initial
 - `git checkout nom-de-la-branche`
- Pour créer de nouveaux commits :
 - Défait les 2 derniers commit mais laisse les modifs dans l'index
`git reset --soft HEAD~2`
 - Défait les 2 derniers commit, retire de l'index mais conserve les modifs dans le working directory (--mixed est le comportement par défaut)
`git reset --mixed HEAD~2`
 - Défait les 2 derniers commit, retire de l'index et du working directory
`git reset --hard HEAD~2`



▸ Rebaser

- La commande git rebase permet entre autre de réécrire complètement l'historique de nos commits
- Dans l'historique suivant, nous souhaiterions des versions plus atomiques (un seul commit pour le Chapitre sur Rebaser au lieu de 3)

```
MacBook-Pro-de-Romain:Test-Rebase romain$ git ll
* d1c2cce (HEAD -> master) Fin du chapitre sur Rebaser
* 17e5db5 Commande git rebase -i dans le chapitre Rebaser
* a03caf8 Ajout de la LICENSE
* b656fc5 Chapitre sur Rebaser
* 659bf61 Commit Initial
```



► Rebaser (suite)

Faisant intervenir les 4 derniers commits de la master, nous pouvons écrire :

```
git rebase -i master~4
```

```
pick b656fc5 Chapitre sur Rebaser
pick a03caf8 Ajout de la LICENSE
pick 17e5db5 Commande git rebase -i dans le chapitre Rebaser
pick d1c2cce Fin du chapitre sur Rebaser
```

```
# Rebase 659bf61..d1c2cce onto 659bf61 (4 command(s))
#
# Commands:
# p, pick = use commit
# r, reword = use commit, but edit the commit message
# e, edit = use commit, but stop for amending
# s, squash = use commit, but meld into previous commit
# f, fixup = like "squash", but discard this commit's log message
# x, exec = run command (the rest of the line) using shell
# d, drop = remove commit
#
# These lines can be re-ordered; they are executed from top to bottom.
#
# If you remove a line here THAT COMMIT WILL BE LOST.
#
# However, if you remove everything, the rebase will be aborted.
#
# Note that empty commits are commented out
```



► Rebaser (suite)

pick : pas de changement

reword : changement du message

edit : changement complet du commit

squash : fusion du commit avec le précédent

fixup : idem que squash en conservant le message précédent

exec : exécuter une commande sur le commit précédent (penser commande de test qui retournerait un code 0 ou un code d'erreur)

drop : supprimer ce commit

► Exemple :

pick *b656fc5* Chapitre sur Rebaser

fixup *17e5db5* Commande git rebase -i dans le chapitre Rebaser

fixup *d1c2cce* Fin du chapitre sur Rebaser

reword *a03caf8* Ajout de la LICENSE

```
# Rebase 659bf61..d1c2cce onto 659bf61 (4 command(s))
```

```
#
```

```
# Commands:
```

```
# p, pick = use commit
```

```
# r, reword = use commit, but edit the commit message
```

```
# e, edit = use commit, but stop for amending
```

```
# s, squash = use commit, but meld into previous commit
```

```
# f, fixup = like "squash", but discard this commit's log message
```

```
# x, exec = run command (the rest of the line) using shell
```

```
# d, drop = remove commit
```

```
#
```

```
# ...
```



► Rebaser (suite)

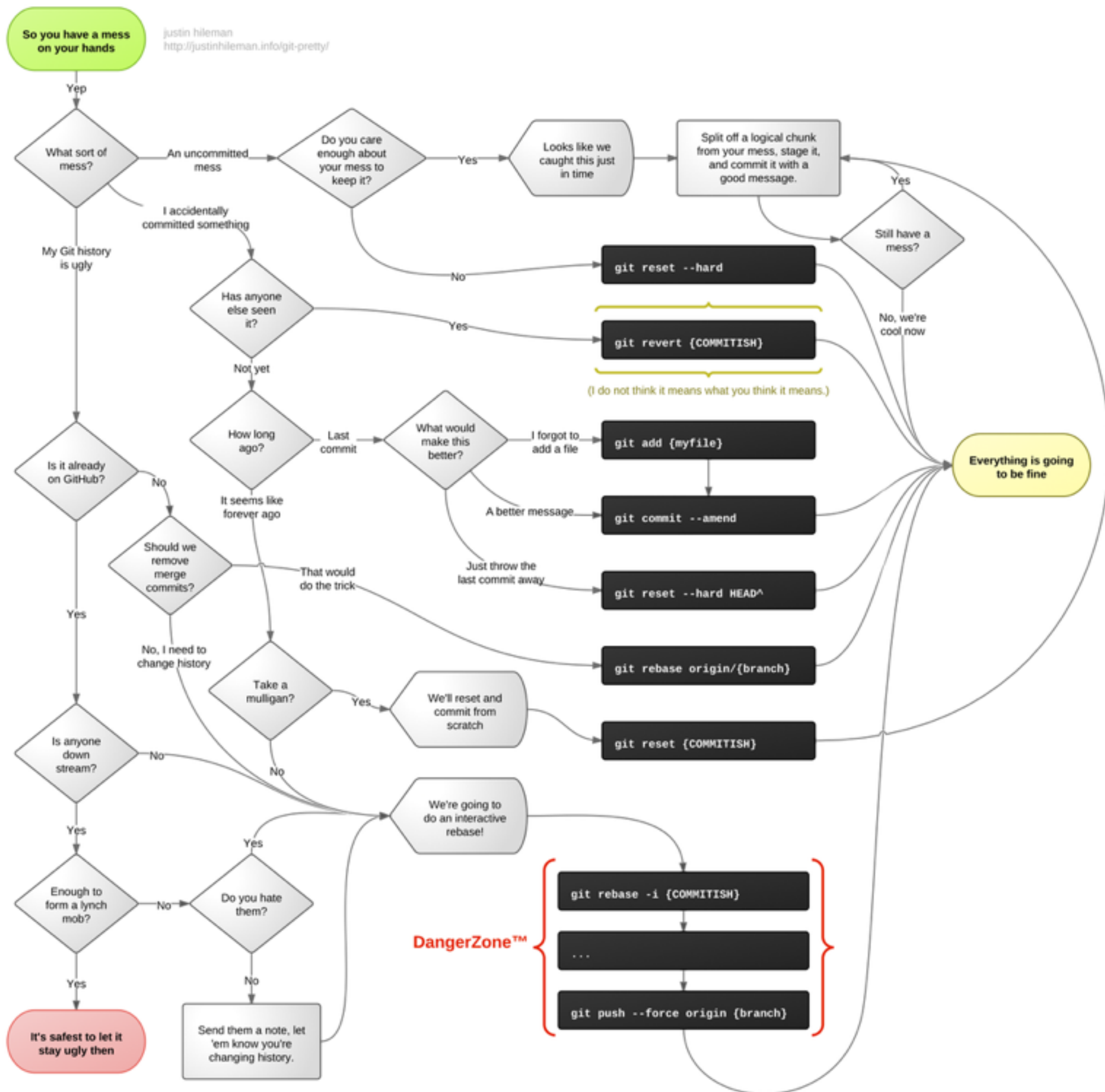
Les reword et squash vont ouvrir l'éditeur pour permettre de changer les messages

Ajout de la licence

```
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# Date:      Fri Mar 18 19:14:50 2016 +0100
#
# interactive rebase in progress; onto 659bf61
# Last commands done (4 commands done):
#   fixup d1c2cce Fin du chapitre sur Rebaser
#   reword a03caf8 Ajout de la LICENSE
# No commands remaining.
# You are currently editing a commit while rebasing branch 'master' on '659bf61'.
#
# Changes to be committed:
#   new file:   LICENSE
#
```

► Une fois terminé :

```
MBP-de-Romain:Test-Rebase romain$ git ll
* 56714a3 (HEAD -> master) Ajout de la licence
* 7f1760c Chapitre sur Rebaser
* 659bf61 Commit Initial
```





Branches locales



▸ Branches locales

Les branches vous permettent de pouvoir travailler sur une copie de votre historique, sans risquer d'altérer un code déjà testé et validé. Par défaut la branche principale de git s'appelle master.

▸ Exemple de branche :

- Nouvelle fonctionnalité : ajout d'un module d'actualité → `fonc-actualite`
- Migration : passage de Bootstrap 3 à Bootstrap 4 → `mig-bootstrap4`
- Correction de bug : bug dont l'identifiant est 234 sur Github/Gitlab → `issue-234`
- Nouvelle version : `version 2.0` → `rel-2.0`



▸ Branches locales

- Lister les branches existantes
`git branch`
- Créer une branche
`git branch fonc-actu`
(fonc-actu : le nom de la nouvelle branche « fonctionnalité actu »)
- Changer de branche
`git checkout fonc-actu`
- Créer et changer de branche
`git checkout -b fonc-actu`
- Renommer la branche courante
`git branch -m <newname>`
- Renommer une branche
`git branch -m <oldname> <newname>`
- Supprimer une branche
`git branch -d fonc-actu`



▸ Fusionner des branches

Il y a 2 manières de fusionner une branche : `git merge` et `git rebase`.

▸ `git merge`

« Merger » une branche signifie que vous souhaitez, si une opération de Fast-Forward est possible (pas de modifications entre temps sur la branche d'origine), l'historique ne fera pas apparaître un nouveau chemin.

▸ `git rebase`

« Rebaser » une branche signifie réécrire l'histoire en fusionnant les modifications qui ont été faite depuis la dernière synchro de la branche sur un ancêtre commun.

▸ Avant de fusionner !

- Ne faire apparaître dans le graphe que les fusions qui correspondent à une nouvelle fonctionnalité (pas pour une simple correction de bug).
- Se placer sur la branche dans laquelle on souhaite fusionner
`git checkout master`
- Ne pas avoir de modifications en cours (tous les commits ont été faits)



▸ Git merge

- `git merge fonc-actu`

▸ Merge Fast-forward

`git merge` va réaliser un fast-forward si aucun nouveau changement n'a été fait sur la branche d'origine, vous pouvez essayer de l'obliger avec :

`git merge --ff-only fonc-actu`

Dans ce cas privilégiez `git rebase`

```
MacBook-Pro-de-Romain:TestGit romain$ git merge fonc-actu
Updating c19571e..0c8f13c
[Fast-forward
 file2-actu | 0
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 file2-actu
MacBook-Pro-de-Romain:TestGit romain$ git log --oneline --graph
* 0c8f13c file2-actu
* c19571e file1
MacBook-Pro-de-Romain:TestGit romain$
```



► Merge Recursive

git merge va créer un nouveau commit de fusion si un changement a été fait sur la branche d'origine entre temps

git merge fonc-actu

```
MacBook-Pro-de-Romain:TestGitMergeFF romain$ git merge fonc-actu
Merge made by the 'recursive' strategy.
 file2-actu | 0
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 file2-actu
MacBook-Pro-de-Romain:TestGitMergeFF romain$ git log --oneline --graph
* 6bd8b57 Merge branch 'fonc-actu'
| \
| * 0c8f13c file2-actu
| * | a6a67c9 file2-master
| /
* c19571e file1
MacBook-Pro-de-Romain:TestGitMergeFF romain$
```



- ▶ Merge Recursive

Vous pouvez également le forcer même s'il n'y a pas eu de changement sur la branche d'origine (souhaitable, sinon on aurait utilisé git rebase)

`git merge --no-ff fonc-actu`

```
MacBook-Pro-de-Romain:TestGitMergeNoFF romain$ git merge --no-ff fonc-actu
Merge made by the 'recursive' strategy.
 file2-actu | 0
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 file2-actu
MacBook-Pro-de-Romain:TestGitMergeNoFF romain$ git log --oneline --graph
* 9ba5b51 Merge branch 'fonc-actu'
| \
| * 0c8f13c file2-actu
| /
* c19571e file1
MacBook-Pro-de-Romain:TestGitMergeNoFF romain$
```

- ▶ Peut devenir le comportement par défaut en éditant la config
`git config --global --add merge.ff false`



► Git rebase

Si les modifications apportées par la branche n'ont pas intérêts à être vu comme tel dans l'historique (nouveau chemin), par exemple s'il s'agit d'une simple correction de bug, on privilégiera git rebase qui va fusionner les arbres en leur détectant leur ancêtre commun

`git rebase fonc-actu`

```
MacBook-Pro-de-Romain:TestGitRebase romain$ git rebase fonc-actu
First, rewinding head to replay your work on top of it...
Applying: file2-master
MacBook-Pro-de-Romain:TestGitRebase romain$ git log --oneline --graph
* 182fa45 file2-master
* 0c8f13c file2-actu
* c19571e file1
MacBook-Pro-de-Romain:TestGitRebase romain$
```



- Il est possible de rebaser après un git merge
 - `git merge fonc12`
 - `git rebase fonc12`

```
TestBranches — -bash — 80x14
[MBP-de-Romain:TestBranches romain$ git log --oneline --graph
* 582398b Merge branch 'branch1'
| \
| * 1399b70 File2
* | 2068138 File3
|/
* 16cce97 M1
[MBP-de-Romain:TestBranches romain$ git rebase branch1
First, rewinding head to replay your work on top of it...
Applying: File3
[MBP-de-Romain:TestBranches romain$ git log --oneline --graph
* c92e4e8 File3
* 1399b70 File2
* 16cce97 M1
```




► Conflits

S'il y a eu des changements apportés au 2 branches sur les mêmes ligne, il sera alors impossible de les fusionner sans corriger le conflit au préalable.

```
MacBook-Pro-de-Romain:Tests-Git romain$ git merge fonc12
Auto-merging style.css
CONFLICT (content): Merge conflict in style.css
Automatic merge failed; fix conflicts and then commit the result.
MacBook-Pro-de-Romain:Tests-Git romain$
```

Corriger le fichier qui pose problème (penser à retirer les lignes <<<<, ===== et >>>>), puis commiter la version définitive.

```
style.css
1 body {
2   background-color: #eee;
3   <<<<<< HEAD
4   color: #222;
5   =====
6   color: #000;
7   >>>>>> fonc12
8 }
9 |
```

```
MacBook-Pro-de-Romain:Tests-Git romain$ git log --pretty=format:"%h - %an : %s" --graph
* 0ecc82e - bioub : Résolution des conflits (texte finalement en gris)
| \
| * b7c733a - bioub : Texte en noir
* | ef1500c - bioub : Texte en gris
* | 32ab820 - bioub : Merge branch 'fonc12'
| \
| \
| * e8eaf16 - bioub : Ajout d'un log dans scripts.js
* | 4e6fac9 - bioub : Ajout d'un fond pour body
| /
* 795dca3 - bioub : Ajout d'une ligne dans scripts.js
* 0cefc8f - Romain Bohdanowicz : Ajout d'une ligne depuis github
* bfbde8b - bioub : Suppression du dossier dist
* dfef550 - bioub : Ajout du dossier dist
* f7bcc2b - bioub : Version initiale du projet
```



- Bonnes pratiques de gestion de branches
<http://nvie.com/posts/a-successful-git-branching-model/>
- Mise en place simplifiée de ces pratiques
http://danielkummer.github.io/git-flow-cheatsheet/index.fr_FR.html



Branches distantes (Remotes)



▸ Remotes

Les remotes sont des copies contenant tous l'historique de vos modifications. Il va falloir les synchroniser avec vos repository locaux.

▸ Exemple de remotes :

- Serveur de sources communs à l'équipe / plate-forme d'intégration continue
- Sauvegarde du repository
- Serveur de préproduction/production



- ▶ Ajouter un dépôt distant

Par défaut lors d'un git clone, le repository s'appelle origin

- `git remote add origin https://github.com/bioub/tests-git.git`

- ▶ Listes les dépôts distants

- `git remote -v`



► Publier des sources sur un dépôt distant

- `git push origin master`
(origin : nom du dépôt distant, master : branche locale)
- Pour enregistrer la branche distante en Upstream (branche distante par défaut)
`git push -u origin master`
- On pourra alors simplement écrire (cf chapitre Configuration ci dessous)
`git push`

► Erreur

`git push` vous affichera un message d'erreur s'il y a eu des changements entre temps sur le remote, il faudra synchroniser la branche au préalable avec `git pull`

► Configuration

Privilégiez un `push.default` simple qui va pousser sur le serveur que la branche courante (matching ou lieu de simple pousserait toutes les branches)

```
git config --global push.default simple
```



- Récupérer les sources d'une branche distante
 - Récupérer les sources depuis un dépôt distant
`git pull origin master`
 - Ou bien si l'upstream a déjà été défini sur cette branche
`git pull`
 - Par défaut `git pull` va réaliser 2 opérations
 - `git fetch` qui va récupérer les sources sur une nouvelle branche locale
 - `git merge` de cette nouvelle branche dans votre branche ce qui aboutira à un nouveau chemin dans le graph
 - Il est possible et préférable de faire un `git rebase` pour éviter le nouveau chemin avec
`git pull --rebase=preserve`
 - Ou bien en éditant la config une fois pour toute
`git config --global pull.rebase preserve`



- Lister les tags
 - `git tag`
- Créer un nouveau tag
 - `git tag -a 0.9.0 -m "Version 0.9.0"`
- Tagger un précédent commit
 - `git tag -a 0.1.0 -m "Version 0.1.0" f7bcc2b2d`

```
MacBook-Pro-de-Romain:Tests-Git romain$ git log --pretty=oneline
0cefc8f807bb14adb362f55f3a85a396cb205b9a Ajout d'une ligne depuis github
bfbde8b99ab138dfe92b1adb7e9ca1007840d91d Suppression du dossier dist
dfef5500fc89517127944f33edb28a74dde6895a Ajout du dossier dist
f7bcc2b2d485db6011bb375e38a18d019a9bd17d Version initiale du projet
MacBook-Pro-de-Romain:Tests-Git romain$ git tag -a 0.1.0 -m "Version 0.1.0" f7bcc2b2d
```

- Partager les tags au serveur distant
 - `git push origin --tags`



Sous-modules



▸ Sous-modules

Les sous-modules vous permettent d'inclure une branche d'un repository en tant que sous répertoire d'un autre repository.

▸ Exemple de sous-module :

- Inclusion et mise à jour d'une bibliothèque comme Bootstrap dans votre projet
- Inclusion et mise à jour d'un thème Wordpress

▸ Bonnes pratiques

- S'il existe un système de gestion de dépendance dans votre langage de développement utilisez-le (exemple : composer en PHP ou npm en JavaScript)
- Evitez si possible de mettre à jour les sources depuis le sous-modules (plus difficile à maintenir)



- ▶ Ajouter un sous-module à votre projet

```
git submodule add https://github.com/twbs/bootstrap.git
lib/bootstrap
git commit -m "Ajout de Bootstrap en Submodule"
```
- ▶ Revenir à une version plus ancienne (3.3.6) (attention à ne pas mettre à jour les sources suite au checkout)

```
cd lib/bootstrap/
git checkout tags/v3.3.6
cd ../..
git add lib/bootstrap
git commit -m "Passage de Bootstrap en 3.3.6"
```
- ▶ Mise à jour d'un sous-module (si aucun changement apporté)

```
git submodule update --remote
```