



formation.tech

Formation Git

Romain Bohdanowicz
Twitter / Github : @bioub
<http://formation.tech/>





formation.tech

Git

Git - Introduction



- Système de gestion de version distribué (DVCS)
- Crée par Linus Torvalds en 2005 pour gérer le code source du noyau Linux
- Permet de sauvegarder des différences incrémentales au sein d'un fichier
- Adapté principalement aux fichiers textes
- Permet de visualiser ou revenir en arrière dans un projet
- Plusieurs versions d'un même projet deviennent disponibles en simultané (pour l'ajout de nouvelles fonctionnalités, la migration...)
- Facilite la collaboration entre plusieurs développeurs



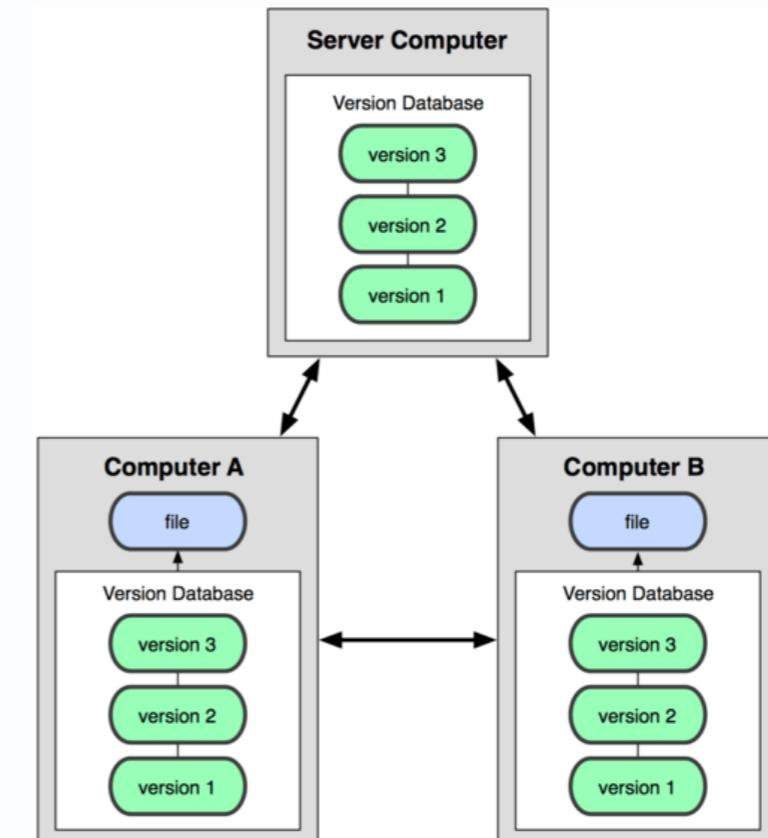
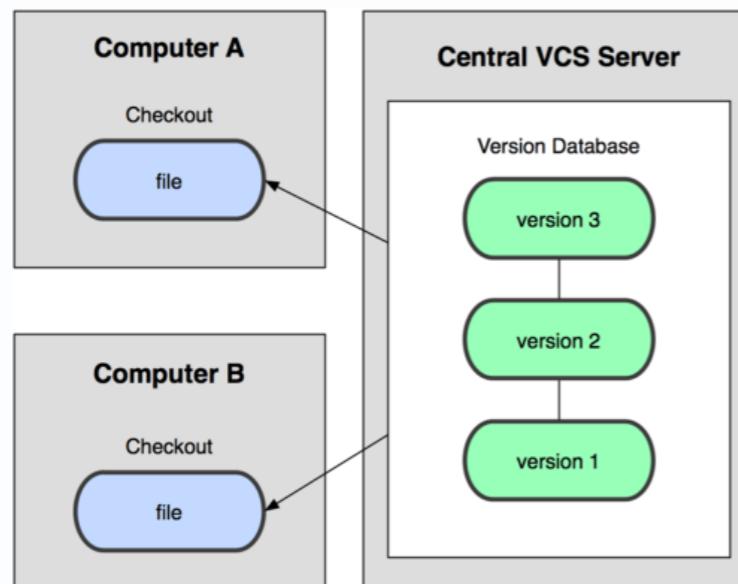
Git - Documentation

- Git Book
<https://git-scm.com/book/fr/v2>
- Tutoriels d'Atlassian (Trello, SourceTree, BitBucket...)
<https://www.atlassian.com/git/tutorials/>
- Git Cheatsheet
<http://ndpsoftware.com/git-cheatsheet.html>
- Learn Git Branching
<https://learngitbranching.js.org/>
- Github Blog pour les nouveautés de Git
<https://github.blog/2019-02-24-highlights-from-git-2-21/>

Git - DVCS vs CVCS



- Système de gestion de version centralisé (CVCS)
- Ex : CVS, Subversion
- Système de gestion de version distribué (DVCS)
- Ex : Git, Mercurial



Git - DVCS vs CVCS



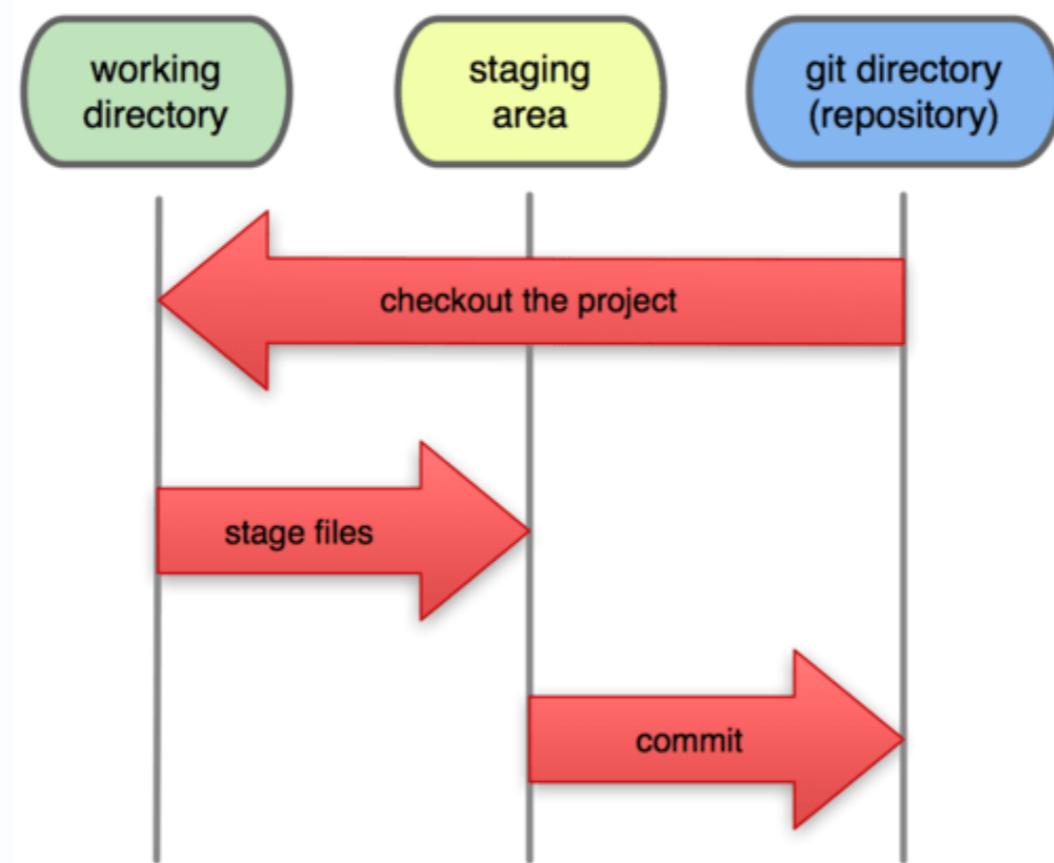
► Avantages du DVCS

- 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 postériori
- plus rapide (accès locaux)



Git - 3 états

- ▶ 3 états pour les modifications
 - 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)



Git - Installation



- ▶ Linux
 - `yum install git`
 - `apt-get install git`
- ▶ Mac OS X
 - <http://sourceforge.net/projects/git-osx-installer/>
 - `brew install git`
- ▶ Windows
 - <https://gitforwindows.org> (penser à ajouter Git au PATH pour pouvoir s'en servir sous DOS, valeur par défaut dans le dernier installateur)
 - `choco install git.install`



Git - GUI

► gitk

- Installé avec git
- Assez basique

Coaching_Fullstack_JS_2018_02: Tous les fichiers - gitk

Modifications locales non enregistrées

master — remotes/origin/master

Romain Bohdanowicz 2018-02-13 14:10:58
Romain Bohdanowicz 2018-02-13 09:48:49
Romain Bohdanowicz 2018-02-12 16:35:31

Id SHA1: 3f066d339d5bcc2681e5738ed96ee057401b3484

Recherche: commit contient : Exact Tous les champs

Rechercher Patch Arbre

Diff Ancienne version Nouvelle version Ligne

Auteur: Romain Bohdanowicz <romain.bohdanowicz>
Validateur: Romain Bohdanowicz <romain.bohdano>
Parent: f1c0b8ecb59e46fc7407d151201f5f15a720c6
Enfant: 00
Branche: master, remotes/origin/master
Suit:
Précède:

Services

```
----- AngularJS-Component/app
new file mode 100644
index 000000..5953916
@@ -0,0 +1,13 @@
+// Module IIFE
+// Immediately Invoked Function Expression
+(function() {
+ 'use strict';
+
+ var module = angular.module('app.module', [
+ 'hello/hello.component',
+ 'users-list/users-list.component',
+ ]);
+
+}());

```

Commentaires

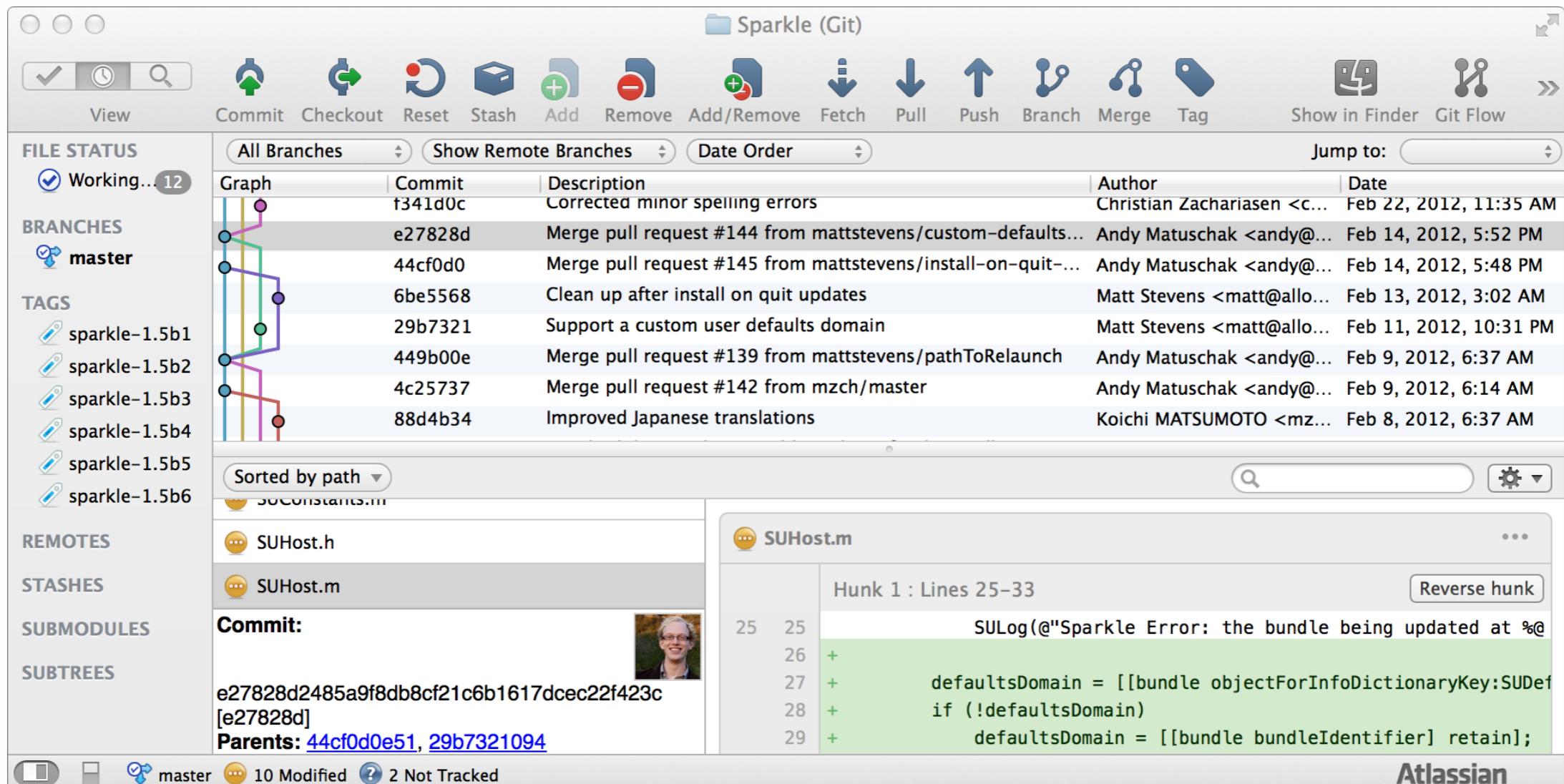
AngularJS-Component/app.module.js
AngularJS-Component/index.html
AngularJS-Component/services/user.service.js
AngularJS-Component/users-list/users-list.component.html
AngularJS-Component/users-list/users-list.component.js

Git - GUI



▶ SourceTree

- L'interface graphique (gratuite) pour git la plus complète
- Nécessite un compte Atlassian (éditeur de JIRA, Bitbucket, Trello...)





formation.tech

Commandes de base



Git - Configuration

- ▶ 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`



Git - Configurer son éditeur

- De nombreuses opérations git nécessitent l'ouverture d'un éditeur (commit sans message, merge, rebase -i...)
- Il faut choisir un éditeur qui puisse détecter les changements au sein d'un fichier
 - Notepad++
`git config --global core.editor "'C:/Program Files/Notepad++/notepad++.exe' -multiInst -notabbar -nosession -noPlugin"`
 - Visual Studio Code
`git config --global core.editor "code -w"`
 - vim
`git config --global core.editor vim`
 - TextMate
`git config --global core.editor "mate -w"`
- Configuration pour les principaux éditeurs
<https://help.github.com/en/articles/associating-text-editors-with-git>
- Vérifier que son éditeur est bien configuré
`git config --global -e`



Git - Aide

- 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`

```
MacBook-Pro-de-Romain:Tests-Git roman$ 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-Roman:Tests-Git roman$
```



Git - Ajout à l'index

- Ajouter des modifications à l'index (add ou son alias stage)
 - `git add README.md`
 - `git stage README.md`
 - `git add *.{css,js,html}`
 - `git add .`
- Ajout partiel
 - `git add -p README.md`
 - `git add -p .`

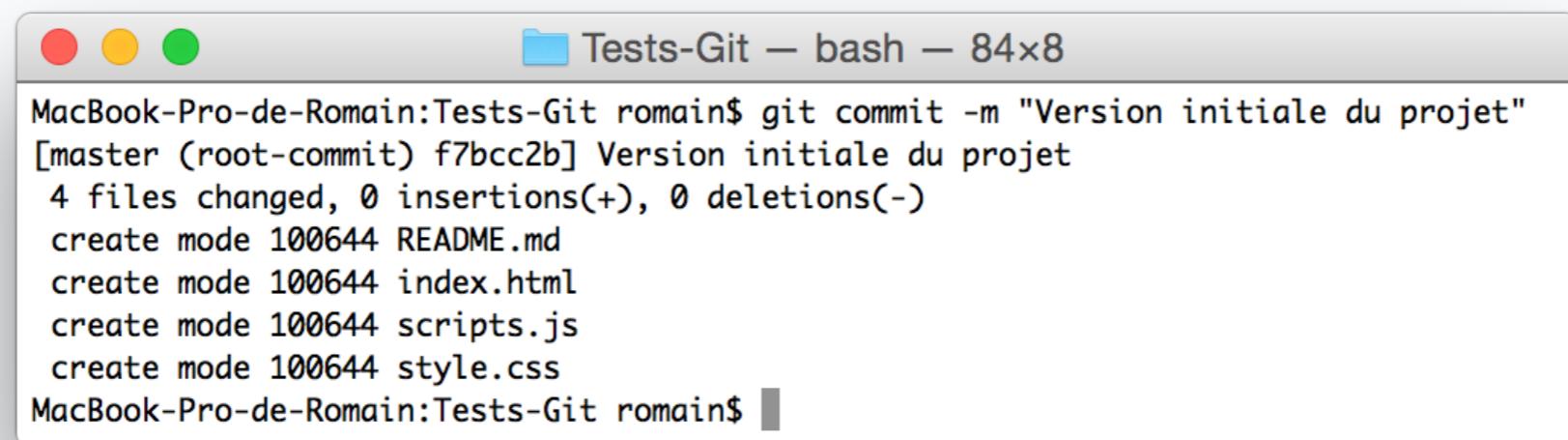


Git - Crédit de version

▶ 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 roman$ 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 roman$
```



Git - Conventions de messages

- Quelques conventions pour les messages de commit
 - <https://www.conventionalcommits.org/>
 - <https://github.com/angular/angular/blob/22b96b9/CONTRIBUTING.md#-commit-message-guidelines>
 - https://seesparkbox.com/foundry/semantic_commit_messages
 - <http://karma-runner.github.io/3.0/dev/git-commit-msg.html>



Git - Supprimer

- ▶ Supprimer des modifications de l'index (mais les conserver dans le working directory = unstaged)
 - 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`



Git - Cloner

- ▶ 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.

```
MacBook-Pro-de-Romain:Desktop roman$ 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
```



Git - Renommer un fichier

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



Git - Ignorer

▸ 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

```
❶ .idea
❷ node_modules
❸ bower_components
❹ dist
❺ maquette
❻
```



Git - Afficher l'historique

▸ Historique des modifications

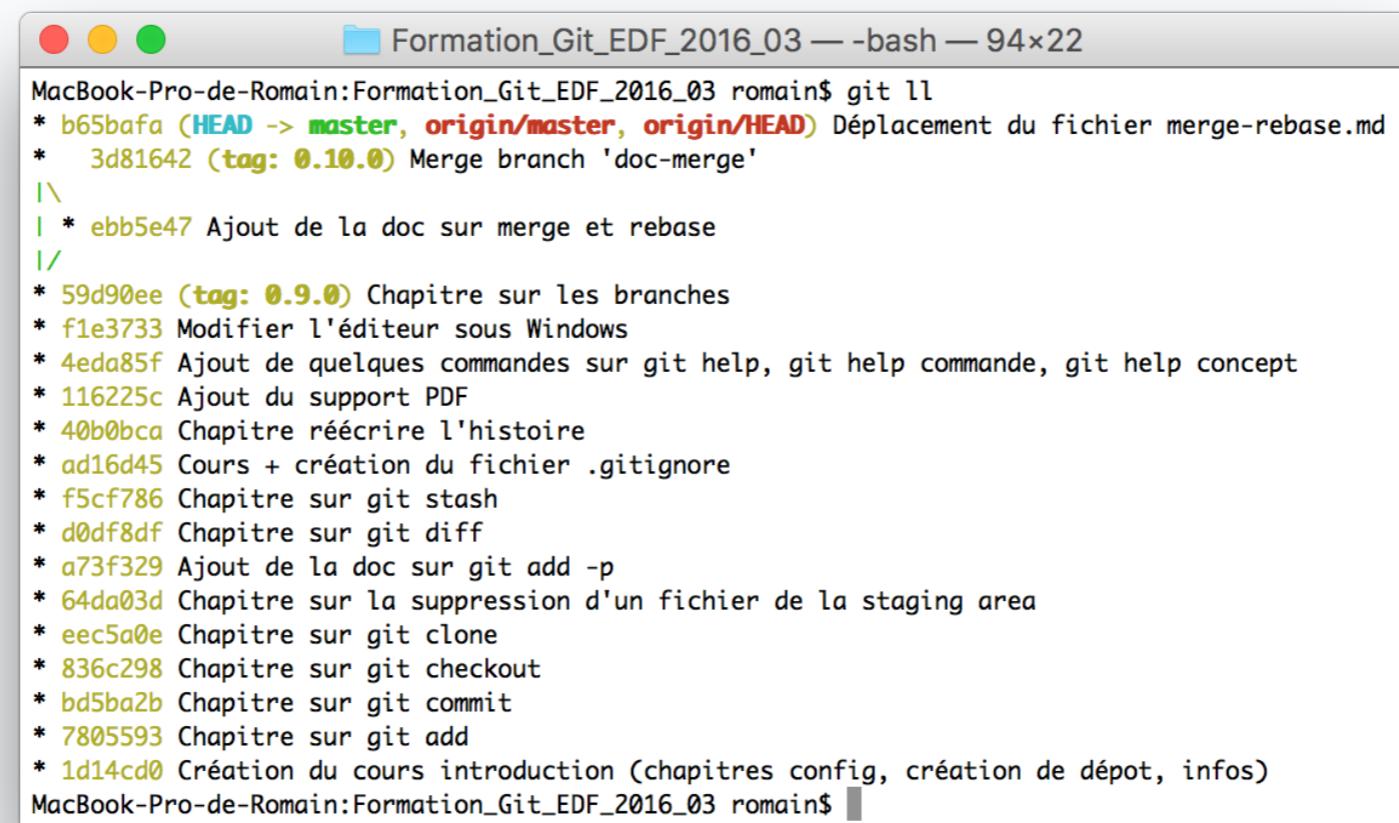
- git log
- git log --pretty=format:"%h - %an : %s"
- git log --oneline
- git log --graph
- git log --graph --all
- git log --decorate
- git log --pretty=format:"%h - %ar - %an : %Cgreen %s"

```
Formation_Git_EDF_2016_03 — bash — 94x22
MacBook-Pro-de-Romain:Formation_Git_EDF_2016_03 romain$ git log
* 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
* 4ed085f 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
* 64dd03d 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$
```



Git - Afficher l'historique

- ▶ Pourquoi pas un alias ?
 - git config --global alias.ll "log --oneline --decorate --graph --"
 - git ll
- ▶ Exemple d'alias :
<https://coderwall.com/p/euwpig/a-better-git-log>



```
Formation_Git_EDF_2016_03 — -bash — 94x22
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$
```

Git - Voir les différences



▶ Voir les différences

- Afficher le contenu d'un commit

```
git show 3f066d3
```

- Différence entre working directory et la staging area

```
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
```

```
git diff HEAD~1 HEAD
```



Git - Voir les différences

```
MacBook-Pro:Coaching_Fullstack_JS_2018_02 romain$ git show 3f066d3
commit 3f066d339d5bcc2681e5738ed96ee057401b3484 (HEAD -> master, origin/master)
Author: Romain Bohdanowicz <romain.bohdanowicz@gmail.com>
Date:   Tue Feb 13 14:10:58 2018 +0100

Services

diff --git a/angularJS-Component/app.module.js b/angularJS-Component/app.module.js
new file mode 100644
index 000000..5953916
--- /dev/null
+++ b/angularJS-Component/app.module.js
@@ -0,0 +1,13 @@
+// Module IIFE
+// Immediately Invoked Function Expression
+(function() {
+  'use strict';
+
+  var module = angular.module('app.module', [
+    'hello/hello.component',
+    'users-list/users-list.component',
+  ]);
+
+}());
+
+
diff --git a/angularJS-Component/index.html b/angularJS-Component/index.html
index 78b4109..de7e6d3 100644
--- a/angularJS-Component/index.html
+++ b/angularJS-Component/index.html
@@ -1,5 +1,5 @@
<!DOCTYPE html>
-<html lang="en" ng-app="hello/hello.component">
+<html lang="en" ng-app="app.module">
<head>
```



Git - Identifier une version

- Les versions git sont identifiés par des hashes, exemple :
`734713bc047d87bf7eac9674765ae793478c50d3`
- Pour faire référence à une version on peut se contenter des premières lettres si pas d'ambiguïté :
`git show 734713bc047d87bf7eac9674765ae793478c50d3`
`git show 734713bc047d`
`git show 734713`
- Sur le Github du noyau Linux qui contient 700000 commits (octobre 2017), il faut 11 caractères au minimum pour éviter toute ambiguïté
- HEAD : la version qu'on affiche dans le working directory (peut se déplacer avec la commande checkout)
`git show HEAD`
- Branche : pour faire référence au dernier commit d'une branche
`git show [branche]`
`git show master`



Git - Identifier une version

- ▶ L'ancêtre d'un commit

```
git show 734713^  
git show HEAD^  
git show master^  
git show 734713~1  
git show HEAD~1  
git show master~1
```

- ▶ 3 ancêtres en arrières

```
git show 734713^{***}  
git show HEAD^{***}  
git show master^{***}  
git show 734713~3  
git show HEAD~3  
git show master~3
```



Git - Mettre de côté des modifications

- ▶ 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 rapatrier le dernier stash sauvegardé
`git stash pop`
 - pour rapatrier 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`



Git - Identifier une version

▶ Reflog

- git conserve un historique de toutes les références sur lequel portait le HEAD, y compris les commits supprimés

```
git show HEAD@{10}
```

```
a7982fa (HEAD -> develop) HEAD@{0}: checkout: moving from master to develop
02cf53d (origin/master, master) HEAD@{1}: reset: moving to HEAD
02cf53d (origin/master, master) HEAD@{2}: checkout: moving from feat/data-collection to master
7736a59 (feat/data-collection) HEAD@{3}: checkout: moving from develop to feat/data-collection
a7982fa (HEAD -> develop) HEAD@{4}: commit: Fix todos
f6dad4f HEAD@{5}: commit: Show Components
98b20e3 HEAD@{6}: commit: Changes in Top bar and Content Alerting
9758c4e HEAD@{7}: commit: Content Alerting: simpler selection
5880079 HEAD@{8}: commit: Content Alerting add cache
158bc34 HEAD@{9}: commit: min-height: 100vh
dc61fbe HEAD@{10}: commit: Content-Alerting: history back update select
f14a6f2 HEAD@{11}: commit: Fix build problems
72a4fd3 HEAD@{12}: commit: WIP Content Alerting Component
f4f238d HEAD@{13}: commit: WIP Content-Alerting Component
7052cb3 HEAD@{14}: commit: feat: Clips Components
7a9e94c HEAD@{15}: commit (amend): CSS: body height 100%
3d65f05 HEAD@{16}: commit: CSS: body height 100%
0eec95e HEAD@{17}: commit: Cinemas Perfs: Export CSV filename
99998a2 HEAD@{18}: pull: checkout 9d8bba736c74cc07574aadeb4d85f0c1ae965b5b: returning to refs/h
:
```



Git - Exercice

- ▶ Exercice
<https://gitlab.com/exercices-git/introduction>



formation.tech

Réécrire l'histoire

Git - 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 pas 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 haïr et vous serez méprisé par vos amis et votre famille. »

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

Git - Réécrire l'histoire



- ▶ 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`

Git - Revenir en arrière



- ▶ 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`
- ▶ Attention : déplacer le curseur sur un commit plus ancien vous placera dans un état dit "detached HEAD", ce qui signifie que les nouveaux commits créés dans cet état n'appartiendront à aucune branche (et seront potentiellement perdus car pas "poussables").

Git - Revenir en arrière



- ▶ 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`
- ▶ Attention la commande `git reset --hard` ne vous laissera quasi aucune chance de retrouver le contenu de ces 2 derniers commits (seul le reflog vous permettra de les retrouver).



Git - Réécrire l'histoire

‣ 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 roman$ 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
```



Git - Réécrire l'histoire

▶ 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
```

Git - Réécrire l'histoire



- ▶ 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
#
# ...
```



Git - Réécrire l'histoire

‣ 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 roman$ git ll
* 56714a3 (HEAD -> master) Ajout de la licence
* 7f1760c Chapitre sur Rebaser
* 659bf61 Commit Initial
```



Git - Retrouver un commit perdu

- Soit le dépôt git suivant (contient 3 fichiers vides A, B et C)

```
Test-reflog — bash — 80x25
$ git init
Initialized empty Git repository in /Users/romain/Desktop/Test-reflog/.git/
$ touch A
$ git add A
$ git commit -m "A"
[master (root-commit) 27d6adf] A
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 A
$ touch B
$ git add B
$ git commit -m "B"
[master 58d151b] B
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 B
$ touch C
$ git add C
$ git commit -m "C"
[master 00de85b] C
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 C
$ git ll
* 00de85b (HEAD -> master) C
* 58d151b B
* 27d6adf A
$
```



Git - Retrouver un commit perdu

- ▶ A la suite d'un mauvais rebase, le fichier B est perdu

```
Test-reflog — bash — 91x5
$ git rebase -i HEAD~2
Successfully rebased and updated refs/heads/master.
$ git ll
* 4449d44 (HEAD -> master) C
* 27d6adf A
```

- ▶ Le reflog contient toujours le fichier

```
Test-reflog — bash — 92x17
$ git reflog
4449d44 (HEAD -> master) HEAD@{0}: rebase -i (finish): returning to refs/heads/master
4449d44 (HEAD -> master) HEAD@{1}: rebase -i (pick): C
27d6adf HEAD@{2}: rebase -i (start): checkout HEAD~2
00de85b HEAD@{3}: commit: C
58d151b HEAD@{4}: commit: B
27d6adf HEAD@{5}: commit (initial): A
$ git show HEAD@{4}
[commit 58d151b7cc160385a2340779f0f58728767c926f
Author: Romain Bohdanowicz <romain.bohdanowicz@gmail.com>
Date:   Fri Feb 23 19:16:01 2018 +0100

B

diff --git a/B b/B
new file mode 100644
index 000000..e69de29
```



Git - Retrouver un commit perdu

- ▶ Pour récupérer un commit du reflog, on peut utiliser la commande *cherry-pick*

```
Test-reflog — bash — 92x9
$ git cherry-pick HEAD@{4}
[master c8877b2] B
  Date: Fri Feb 23 19:16:01 2018 +0100
  1 file changed, 0 insertions(+), 0 deletions(-)
  create mode 100644 B
$ git ll
[* c8877b2 (HEAD -> master) B
 * 4449d44 C
 * 27d6adf A
```



Git - Exercice

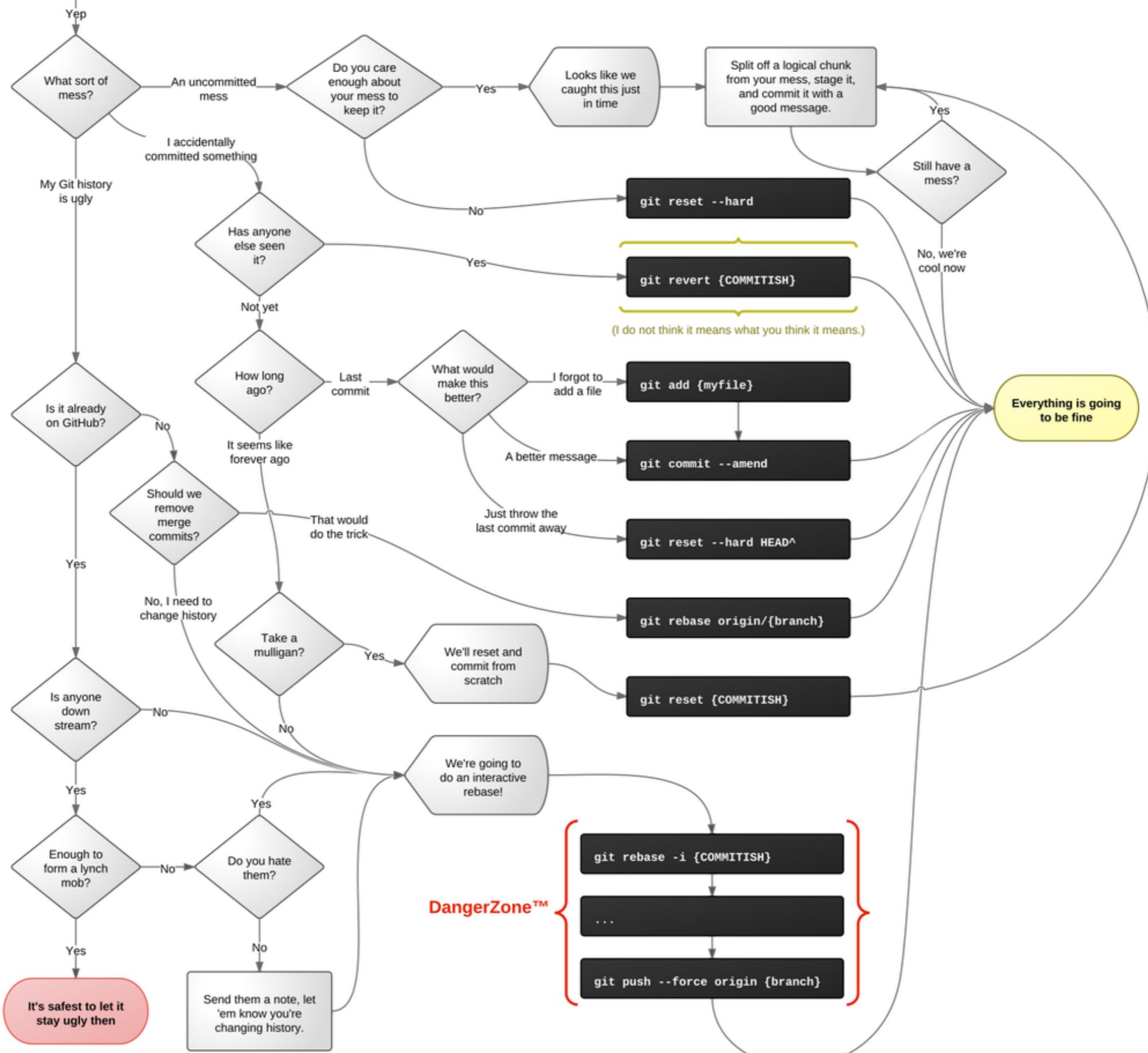
- ▶ Enoncé

<https://gitlab.com/exercices-git/rebase-interactif>



So you have a mess
on your hands

justin hileman
<http://justinhileman.info/git-pretty/>





formation.tech

Branches locales

Git - 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. Pas défaut la branche principale de git s'appelle master.

▶ Exemple de branche :

- Nouvelle fonctionnalité : ajout d'un module d'actualité → feat/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



Git - Branches locales

▶ Branches locales

- Lister les branches existantes

```
git branch
```

- Créer une branche

```
git branch feat/actu
```

(feat/actu : le nom de la nouvelle branche « fonctionnalité actu »)

- Changer de branche

```
git checkout feat/actu
```

- Créer et changer de branche

```
git checkout -b feat/actu
```

- Renommer la branche courante

```
git branch -m <newname>
```

- Renommer une branche

```
git branch -m <oldname> <newname>
```

- Supprimer une branche

```
git branch -d feat/actu
```



Git - Branches locales

- 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 faites 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 - Branches locales

- ▶ 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 privilégez git rebase

```
MacBook-Pro-de-Romain:TestGit roman$ 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 roman$ git log --oneline --graph
* 0c8f13c file2-actu
* c19571e file1
MacBook-Pro-de-Romain:TestGit roman$
```



Git - Branches locales

- ▶ 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 roman$ 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 roman$ git log --oneline --graph
*   6bd8b57 Merge branch 'fonc-actu'
|\ 
* | 0c8f13c file2-actu
* | a6a67c9 file2-master
|/
* c19571e file1
MacBook-Pro-de-Romain:TestGitMergeFF roman$
```



Git - Branches locales

- ▶ 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
```

The screenshot shows a terminal window titled "TestGitMergeNoFF — bash — 76x12". The command "git merge --no-ff fonc-actu" is run, resulting in a recursive merge. The log shows:

```
MacBook-Pro-de-Romain:TestGitMergeNoFF roman$ 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 roman$ git log --oneline --graph
*   9ba5b51 Merge branch 'fonc-actu'
|\ 
| * 0c8f13c file2-actu
|/
* c19571e file1
MacBook-Pro-de-Romain:TestGitMergeNoFF roman$
```

- ▶ Peut devenir le comportement par défaut en éditant la config

```
git config --global merge.ff false
```



Git - Branches locales

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



Git - Branches

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

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

Git - Branches



▶ 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.

```
Tests-Git — bash — 68x5
MacBook-Pro-de-Romain:Tests-Git roman$ 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 roman$
```

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

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

```
Tests-Git — bash — 90x16
MacBook-Pro-de-Romain:Tests-Git roman$ 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
```



Git - Branches locales

- 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



formation.tech

Branches distantes (Remotes)

Git - 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



Git - Dépôts distants

- ▶ 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`

Git - Dépôts distants



- ▶ 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`

Git - Dépôts distants



- ▶ 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=merges`
 - Ou bien en éditant la config une fois pour toute
`git config --global pull.rebase merges`

Git - Dépôts distants



- 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 roman$ git log --pretty=oneline
0cef8f807bb14adb362f55f3a85a396cb205b9a 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 roman$ git tag -a 0.1.0 -m "Version 0.1.0" f7bcc2b2d
```

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



Git - Dépots distants

- ▶ Enoncé
<https://github.com/bioub/Exercice-git-remote>



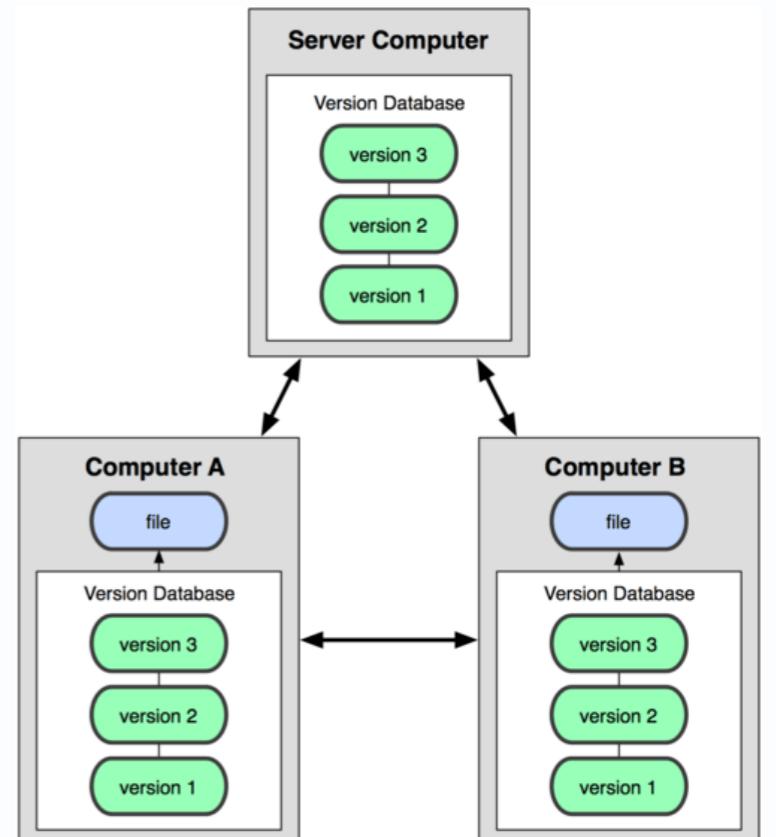
formation.tech

Git côté serveur

Git côté serveur - Introduction



- Git est un Gestionnaire de Version Distribué
- On peut l'installer côté serveur et effectuer des push/fetch/pull et assurer la collaboration ou le déploiement
- 4 protocoles :
 - Local
 - HTTP
 - SSH
 - Git





Git côté serveur - Bare Repository

- Côté serveur on utilise généralement des Bare Repositories
- Un bare repository est un dépôt qui n'a pas de working directory, uniquement le répertoire `.git`
- Pour le créer
`git init --bare`
`git clone --bare my_project my_project.git`

Git côté serveur - Local Protocol



- Il est tout à fait possible de cloner un dépôt local, ou via un partage réseau
- Exemple
`git clone /srv/git/project.git`
- Avantages
 - Pas de configuration
 - Utilise les droits du FileSystem
- Inconvénients
 - Nécessite de monter des disques réseau
 - Les fichiers .git sont accessible via le FileSystem et risque d'être modifiés de manière inappropriée

Git côté serveur - SSH Procotol



- Il est tout à fait possible de cloner un dépôt local, ou via un partage réseau

- Exemple

```
git clone ssh://[user@]server/project.git
```

- Avantages

- Assez simple à mettre en place et sécurisé
 - En général déjà installé sur la plupart des serveurs

- Inconvénients

- Pas d'accès anonyme possible
 - Nécessite d'avoir SSH sur le poste client
 - Nécessite d'avoir le port SSH ouvert sur un firewall ou proxy entreprise

- Configuration

<https://git-scm.com/book/en/v2/Git-on-the-Server-Setting-Up-the-Server>



Git côté serveur - HTTP Procotol

- Utilisé par la plupart des plateformes comme GitHub ou GitLab
- Exemple
`git clone https://example.com/gitproject.git`
- Avantages
 - Permet des accès anonymes ou via les mécanismes d'authentification HTTP
 - Les Firewalls / Proxies laissent généralement sortir les connexions sur les ports HTTP (80) et HTTPS (443)
- Inconvénients
 - Plus compliqué à mettre en place que Local ou SSH
- Configuration avec Apache
<https://git-scm.com/book/en/v2/Git-on-the-Server-Smart-HTTP>



Git côté serveur - Git Protocol

- Intégré dans git (qui écoutera sur le port 9418)
- Exemple
`git clone git://example.com/gitproject.git`
- Avantages
 - Le plus rapide
 - Accès anonymes
- Inconvénients
 - Le plus compliqué à mettre en place
 - Le port 9418 est rarement ouvert en entreprise
- Configuration
<https://git-scm.com/book/en/v2/Git-on-the-Server-Git-Daemon>



formation.tech

Hooks

Hooks - Introduction



- Les hooks ("crochets") permettent d'exécuter automatiquement des scripts avant ou après certaines opérations git
- Exemples
 - Vérifier les conventions de code avant un commit
 - Lancer des tests automatisés avant un commit
 - Créer un build avant un push
 - Lancer l'installation des dépendances après un checkout, merge ou rebase



Hooks - Créer un hook

- Pour créer un hook il suffit de créer un fichier nommé *pre-[operation]* ou *post-[operation]* dans le répertoire *.git/hooks*
- Sur les systèmes Unix (Mac/Linux) il faudra ajouter des droits d'exécution
- Coté serveur il est également possible de lancer les hooks
 - pre-receive
 - update
 - post-receive



Hooks - Partager des hooks

- Les hooks sont des scripts locaux, il n'est pas possible de les partager
- Il existe cependant des bibliothèques pour cela :
 - En JavaScript (installation via npm/Yarn)
 - husky → <https://github.com/typicode/husky>
 - lint-staged → <https://github.com/okonet/lint-staged>
 - En Python (installation via pip)
 - pre-commit → <https://pre-commit.com>



formation.tech

Sous-modules

Git - Sous-modules



▶ Sous-modules

Les sous-modules vous permettre 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)



Git - Sous-modules

- ▶ 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`