



**formation.tech**

# Formation Git

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





**formation.tech**

# Git

# Git - 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 des différences incrémentales au sein d'un dossier/projet
- 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
- Outil en ligne de commande
- Outil de sauvegarde manuel

# Git - Documentation

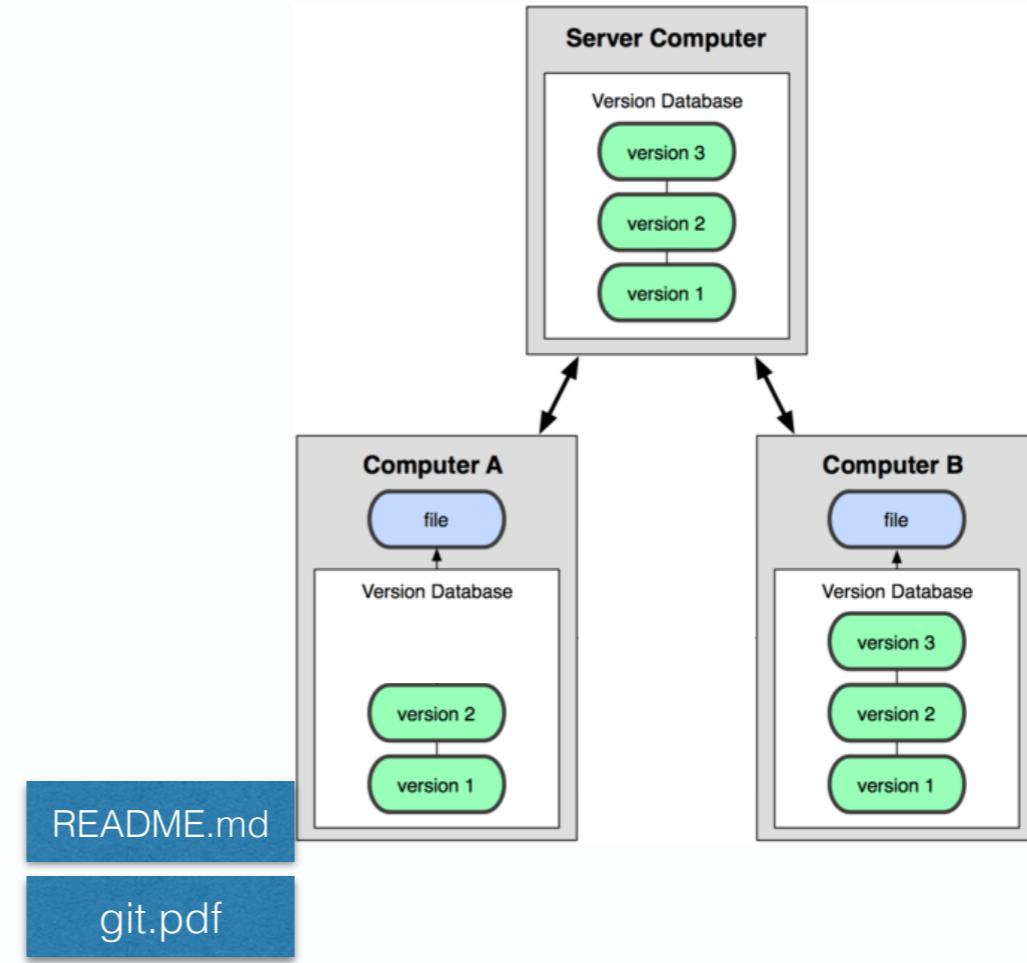
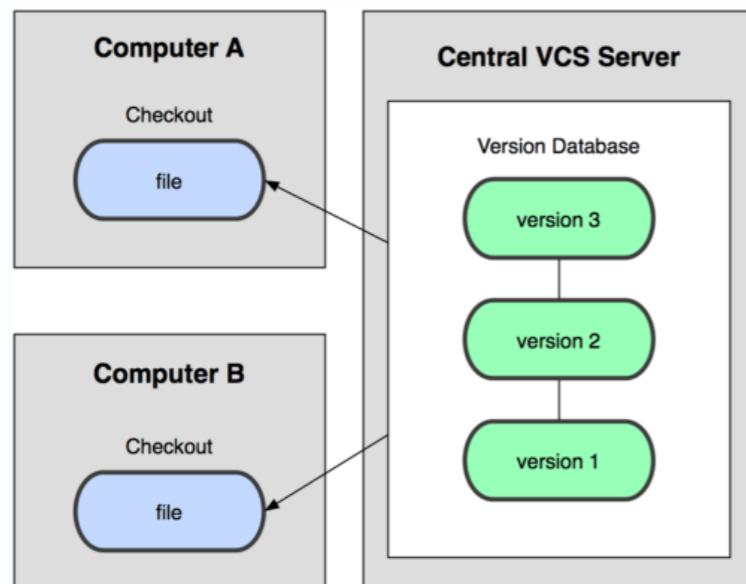


- Git Book : Pro Git  
<https://git-scm.com/book/fr/v2>
- Tutoriels d'Atlassian (JIRA, Trello, SourceTree, BitBucket...)  
<https://www.atlassian.com/git/tutorials/>
- Git Interactive Cheatsheet  
<http://ndpsoftware.com/git-cheatsheet.html>
- Learn Git Branching  
<https://learngitbranching.js.org/>
- Github Blog pour les nouveautés de Git  
<https://github.blog/2022-04-18-highlights-from-git-2-36/>
- Awesome git  
<https://github.com/dictcp/awesome-git>

# Git - DVCS vs CVCS



- Système de gestion de version centralisé (CVCS)
- Ex : CVS, Subversion (SVN)
- 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 ou revenir en arrière
  - plus rapide (accès locaux)
  - permet de collaborer à un projet et obtenir l'autorisation des mainteneurs à postériori (plutôt plateforme)

# Git - Qu'est-ce qu'une modification ?

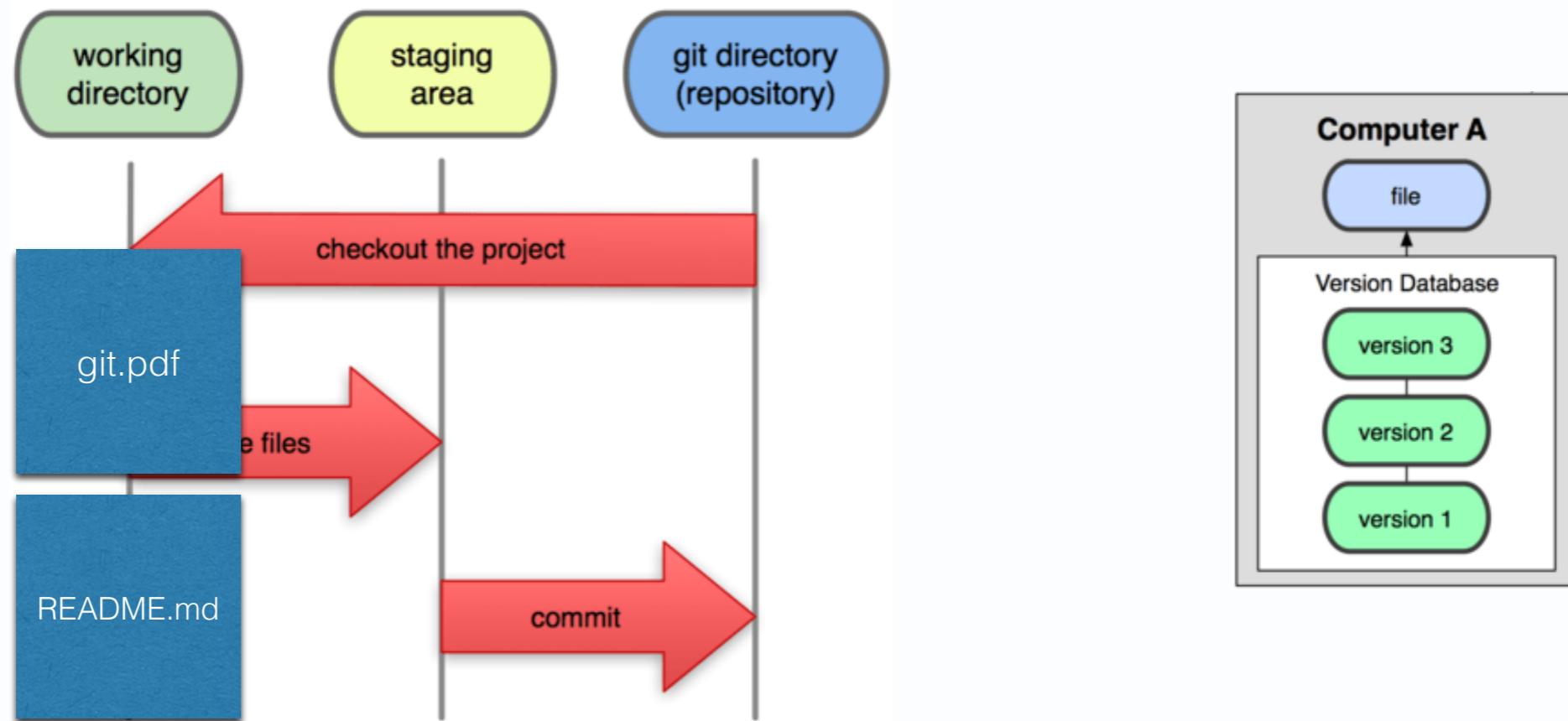


- Modifier quelque chose dans le projet, peut être :
  - créer un nouveau fichier (précédemment untracked / non-suivi)
  - supprimer un fichier
  - déplacer / renommer un fichier (ne pas confondre avec supprimer et créer un nouveau fichier)
  - modifier un fichier (texte) :
    - ajouter une ou plusieurs lignes
    - supprimer une ou plusieurs lignes
    - modifier (équivalent à ajouter et supprimer) une ou plusieurs lignes

# Git - 3 états locaux



- 3 états locaux pour les modifications
  - working directory / working tree / workspace (fichiers modifiés / nouveaux fichiers / fichiers renommés ou supprimés depuis la dernière sauvegarde)
  - staging area / index / cache (espace de sélection, le contenu de la prochaine sauvegarde)
  - git repository / git directory / local repository (sauvegardes)





# Git - Installation

- Linux
  - `yum install git`
  - `apt install git`
- Mac OS X
  - <http://sourceforge.net/projects/git-osx-installer/>
  - Via Homebrew :  
`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)  
Pour mettre à jour : `git update-git-for-windows`
  - Via Chocolatey :  
`choco install git`
  - Via Winget :  
`winget install -e --id Git.Git`



# Git - Installation

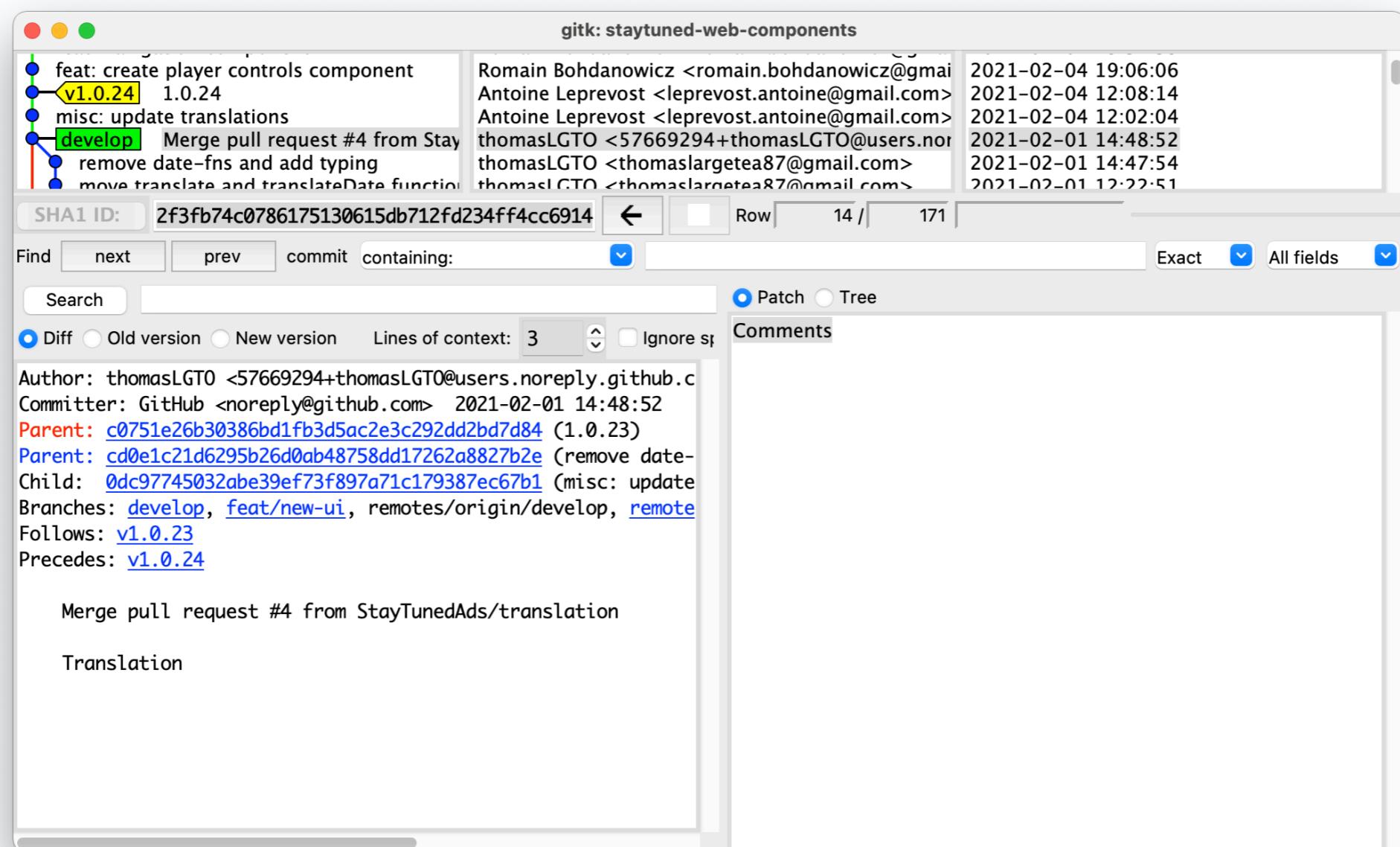
- Sous Windows
  - Git Bash (émulateur de Bash Unix qui utilise Mingwin pour l'émulation)
  - Powershell  
<https://github.com/dahlbyk/posh-git>  
Install-Module -Name posh-git



# Git - GUI

- gitk

- Installé avec git
- Assez basique

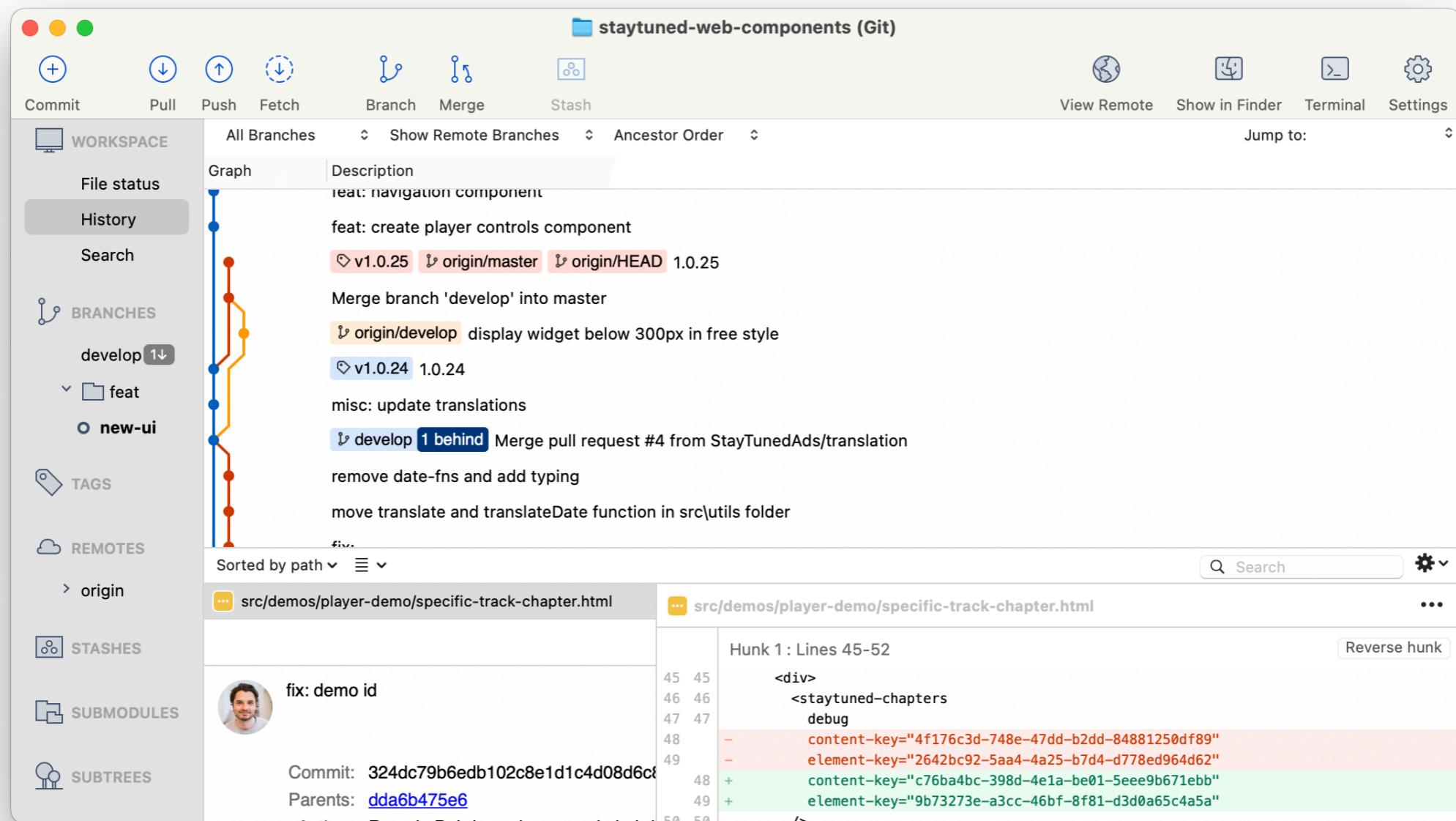




# Git - GUI

## ▸ 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 + user + locale au projet) :  
`git config --list`
  - Config locale (du projet courant)  
`git config --list --local`
  - Config utilisateur système (compte Windows/MacOS/Linux)  
`git config --list --global`
  - Config système (tous les comptes)  
`git config --list --system`
  - S'identifier:  
`git config --global user.name "Romain Bohdanowicz"`  
`git config --global user.email "romain.bohdanowicz@formation.tech"`
  - Configurer le proxy (si pas via les variables d'environnements HTTP\_PROXY...) :  
`git config --global http.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://docs.github.com/en/get-started/getting-started-with-git/associating-text-editors-with-git>
- Vérifier que son éditeur est bien configuré  
`git config --global --edit`

# Git - Aide

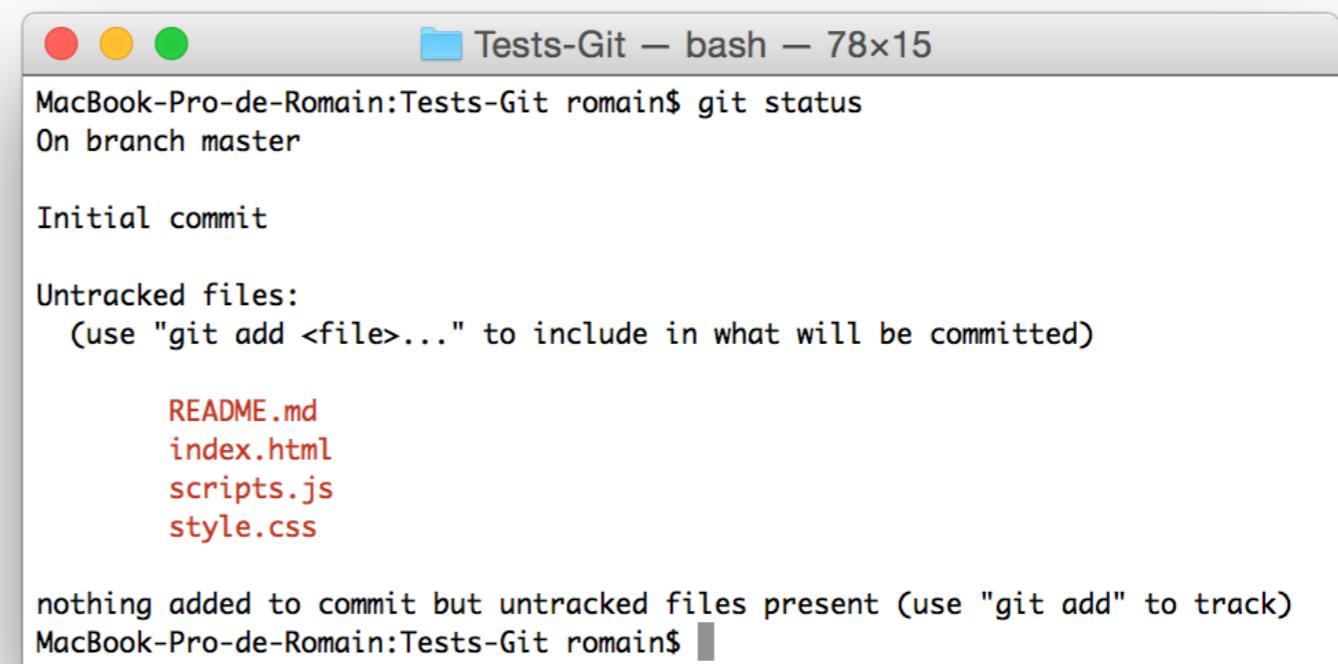


- › Obtenir de l'aide, aide sur une commande, sur un concept
  - `git help`
  - `git help config`  
`git config --help`
  - `git help tutorial`

# Git - Démarrage et Etats



- Créer un nouveau repository (en local)
  - `git init`
- Obtenir l'état des fichiers du projet (Working Dir / Staging Area / Git Repository)
  - `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-Romain:Tests-Git roman$
```



# Git - Déplacer vers l'index

- Ajouter des modifications à l'index (add ou son alias stage)
  - `git add README.md`
  - `git stage README.md`
- Pattern Glob
  - `git add *.html`
  - `git add *.{css,js,html}`
- Ajouter un répertoire (. le répertoire courant)
  - `git add .`
  - `git add src/`
- Ajout partiel / patch
  - `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](https://seesparkbox.com/foundry/semantic_commit_messages)
  - <http://karma-runner.github.io/3.0/dev/git-commit-msg.html>
- Vous pouvez passer un template de message avec l'option commit.template, ex :  
`git config --global commit.template /Users/romain/.gitCommitTemplate`



# 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 - 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 puis utiliser git add
  - Ou directement supprimer et git add :  
`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 - Ignorer

- Ignorer des fichiers
  - Créer un fichier `.gitignore` (partagé via git)
    - Le fichier `.gitignore` permet d'ignorer tout les fichiers ou dossier indiqués
    - Y compris dans les sous-répertoires

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

- Dans `.git/info/exclude` (uniquement pour ce projet, non-partagé)
- Dans un fichier `.gitignore` global à configurer via `core.excludesfile` (pour tout vos projets, non-partagé)



# Git - Afficher l'historique

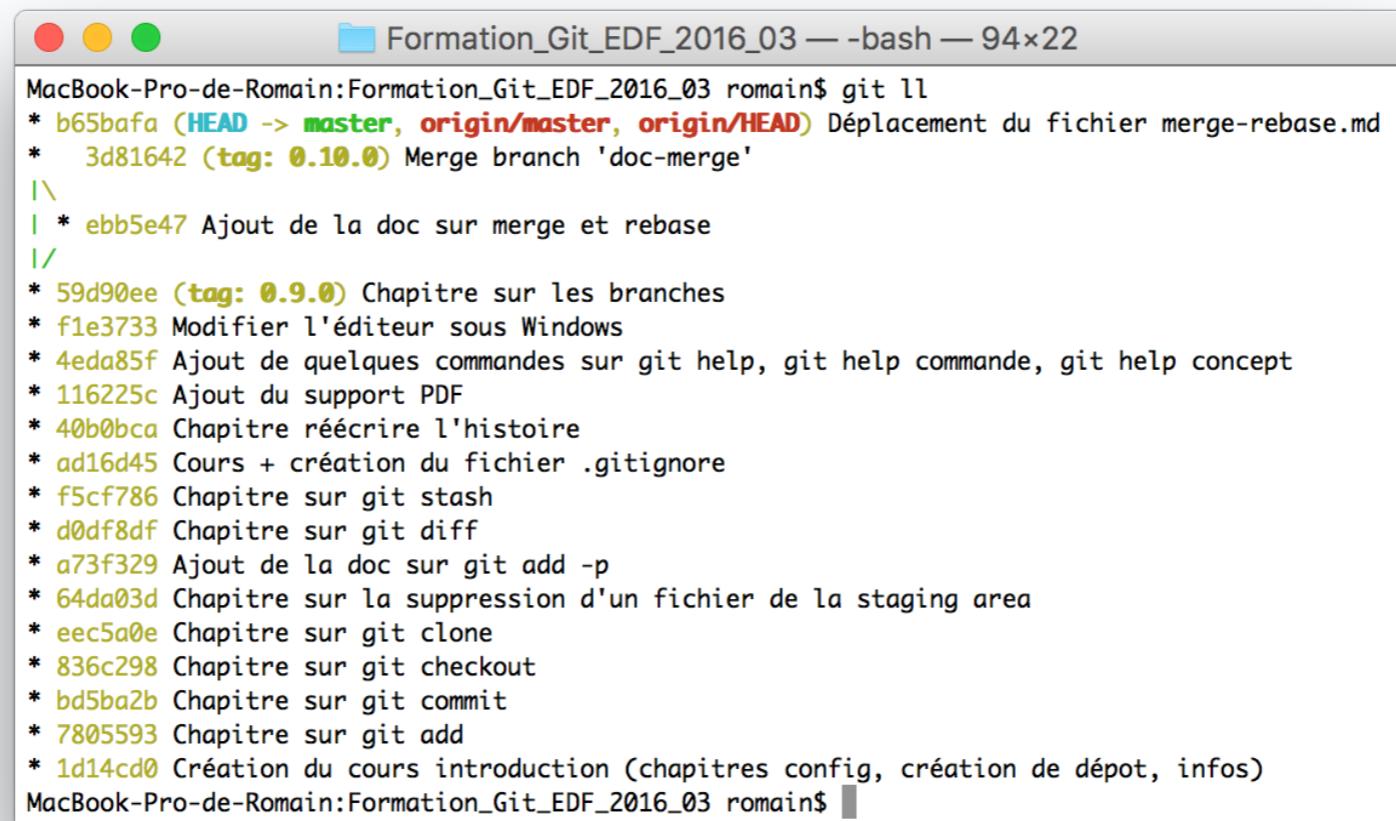
- Historique des modifications
  - git log
  - git log --pretty=format:"%h - %an : %s"
  - git log --oneline
  - git log --graph
  - git log --graph --all
  - git log --decorate
  - git log --pretty=format:"%h - %ar - %an : %Cgreen% s% Creset"

```
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 --graph --all --"
  - 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 un objet



- Les objets git (commit, tree, fichiers) sont identifiés par des hashes (algorithme SHA-1), exemple :  
`734713bc047d87bf7eac9674765ae793478c50d3`
- Pour faire référence à un objet on peut se contenter des premières lettres si pas d'ambiguïté (minimum 4) :  
`git show 734713bc047d87bf7eac9674765ae793478c50d3`  
`git show 734713bc047d`  
`git show 734713b`
- 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

- › Le parent 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
```

- › ATTENTION SOUS DOS, ^ ne fonctionne pas (utilisez ~1 à la place) :

```
git show HEAD^ fera un git show HEAD
```

# 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 les modifications en cours du working directory (-u pour garder les fichiers non surveillés -- untracked --)  
`git stash save -u "Description des changements"`
  - pour mettre de côté des modifications en cours de la staging area  
`git stash push -m "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 stash clear`



# Git - Identifier une version

- Reflog
  - git conserve un historique de toutes les références sur lequel portait le HEAD, y compris les commits supprimés (valable 90 jours par défaut)  
`git show HEAD@{10}`

```
front-end-scs-ang4 — less · git reflog — 95x20
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>
- Attention Gitlab n'est plus compatible avec Internet Explorer



**formation.tech**

# Réécrire l'historique

# Réécrire l'historique - Checkout



- 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").

# Réécrire l'historique - Revert



- Pour créer un nouveau commit opposé à celui à supprimer
- Dans pas d'incidence sur l'historique déjà partagé
- Ex:  
`git revert [commit]`

# Réécrire l'historique - Introduction



- Pourquoi réécrire l'historique ?
  - Parce que vous avez oublié un fichier ou des lignes 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 commits ne sont pas assez « atomiques », ils contiennent plusieurs fonctionnalités
  - Parce que vous avez plusieurs commits pour une même fonctionnalité
  - Parce que vous voulez changer le message, l'auteur ou la date du commit
- **Attention à ne pas réécrire des commits déjà partagés (poussés) !!!**
- Extrait de la doc officielle de git rebase :  
<https://git-scm.com/book/fr/v2/Les-branches-avec-Git-Rebaser-Rebasing>

*« 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. »*

# Réécrire l'historique - Amend

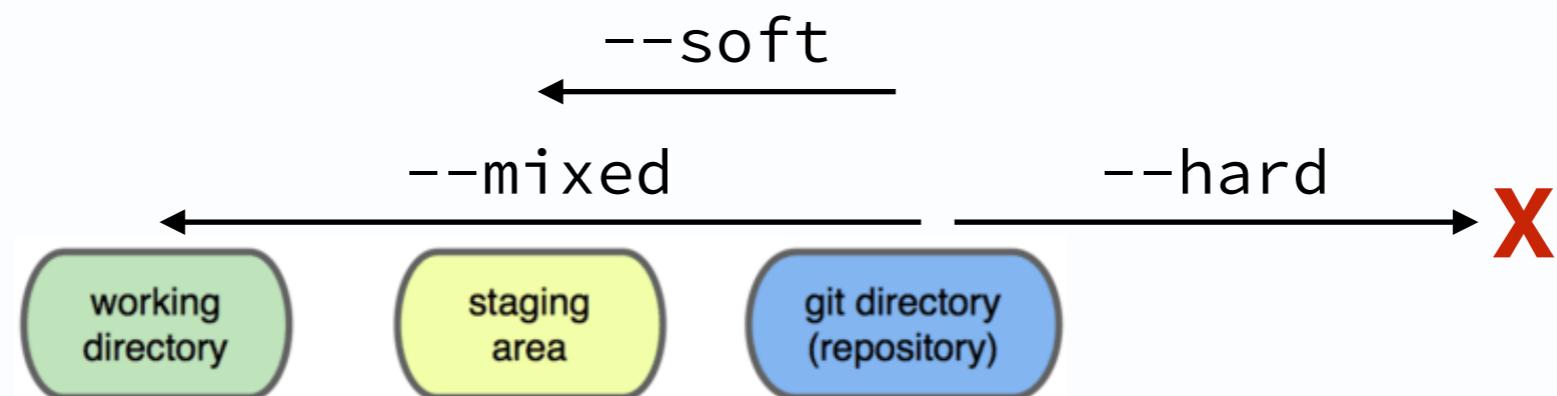


- Modifier le dernier commit de la branche courante
  - `git commit -m 'validation initiale'`
  - `git add fichier-oublie.ext`
  - `git commit --amend`  
Ou sans changer le message de commit:  
`git commit --amend --no-edit`
- Pourquoi pas un alias :
  - `git config --global alias.oops "commit --amend --no-edit"`
  - `git commit -m 'validation initiale'`
  - `git add fichier-oublie.ext`
  - `git oops`

# Réécrire l'historique - Reset



- › 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).



# Réécrire l'historique - Rebase interactif



- Rebase interactif
  - La commande `git rebase` permet de réécrire l'historique d'une branche
  - En lançant la commande `git rebase -i`, git nous propose de réécrire l'historique de la branche depuis le dernier push
  - On peut également lui demander de réécrire l'historique depuis un certain commit :  
`git rebase -i HEAD~4`
  - Pour réécrire depuis le premier commit :  
`git rebase -i --root`

# Réécrire l'historique - Rebase interactif



## ‣ Rebase interactif

```
pick 6c12a06 chore: initial commit
pick db857c9 docs: cheatsheet
pick bd43faa chore: install commitlint
pick 78bfc1d chore: .editorconfig

# Rebase 78bfc1d onto 78bfc1d (4 commands)
#
# Commands:
# p, pick <commit> = use commit
# r, reword <commit> = use commit, but edit the commit message
# e, edit <commit> = use commit, but stop for amending
# s, squash <commit> = use commit, but meld into previous commit
# f, fixup <commit> = like "squash", but discard this commit's log message
# x, exec <command> = run command (the rest of the line) using shell
# b, break = stop here (continue rebase later with 'git rebase --continue')
# d, drop <commit> = remove commit
# l, label <label> = label current HEAD with a name
# t, reset <label> = reset HEAD to a label
# m, merge [-C <commit> | -c <commit>] <label> [<oneline>]
# .      create a merge commit using the original merge commit's
# .      message (or the oneline, if no original merge commit was
# .      specified). Use -c <commit> to reword the commit message.
#
# 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
```

# Réécrire l'historique - Rebase interactif



- 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
#
# ...
```

# Réécrire l'historique - Rebase interactif



- 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
```



# Réécrire l'historique - Cherry pick

- 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
$
```



# Réécrire l'historique - Cherry pick

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

```
$ 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

```
$ 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
```



# Réécrire l'historique - Cherry pick

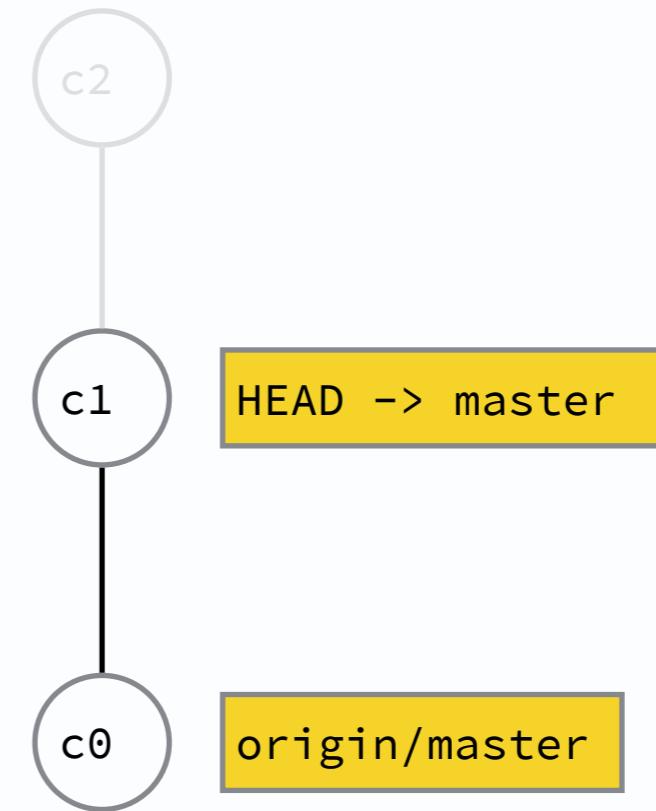
- Pour récupérer un commit du reflog, on peut utiliser la commande *cherry-pick*

```
$ 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
```

# Réécrire l'historique - Cherry pick



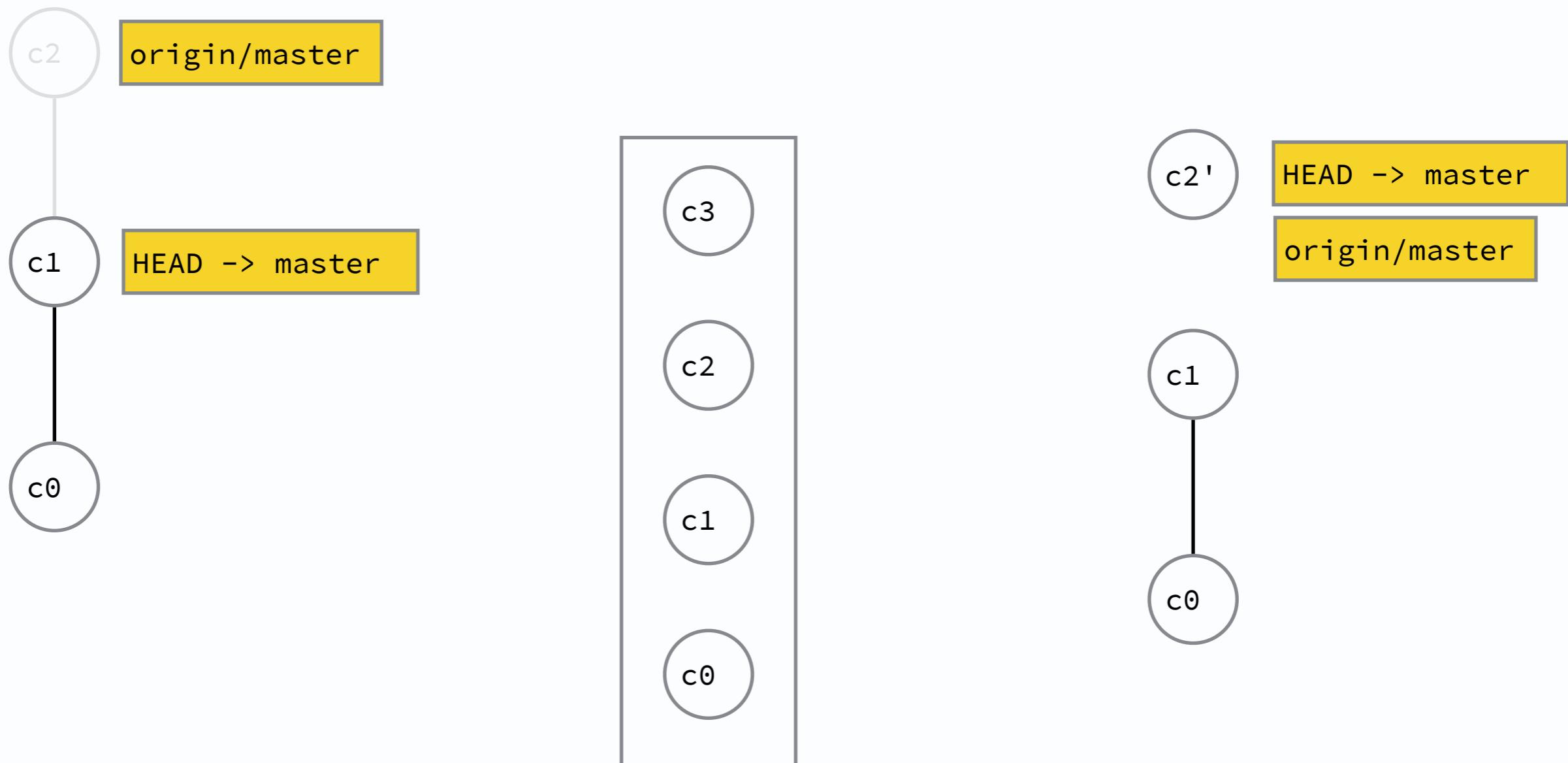
- `git reset HEAD^ / git reset master^ / git reset master~1 / git reset c1`



# Réécrire l'historique - Cherry pick



- `git reset HEAD^ / git reset master^ / git reset master~1 / git reset c1`





So you have a mess  
on your hands

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



# Réécrire l'historique - Exercice



- Enoncé

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



**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. Par défaut la branche principale de git s'appelle master.

- Exemple de branche :

- Commune : la version stable de votre code (master / main / develop / trunk)
- Environnement : correspond à des déploiement automatique (dev / prod / staging)
- Nouvelle fonctionnalité : ajout d'un module d'actualité → feat/actualite
- Correction de bug : bug dont l'identifiant est 234 sur Github/Gitlab → issue-234
- Migration : passage de Bootstrap 3 à Bootstrap 4 → mig-bootstrap4
- Release : version 2.0 → rel-2.0
- Backup : avant de réécrire l'historique d'une branche
- Version spécifique : un client qui aurait une fonctionnalité différente des autres



# Git - 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 switch feat/actu` (à partir de git 2.23)  
`git checkout feat/actu`
- › Crée et changer de branche  
`git switch -c feat/actu` (à partir de git 2.23)  
`git checkout -b feat/actu`
- › Renommer la branche courante  
`git branch -m <newname>`
- › Renommer une branche  
`git branch -m <oldname> <newname>`
- › Supprimer une branche (avec le warning si les commits n'existent pas ailleurs)  
`git branch -d feat/actu`
- › Supprimer une branche (forcer le delete)  
`git branch -D feat/actu`

# Git - Branches locales

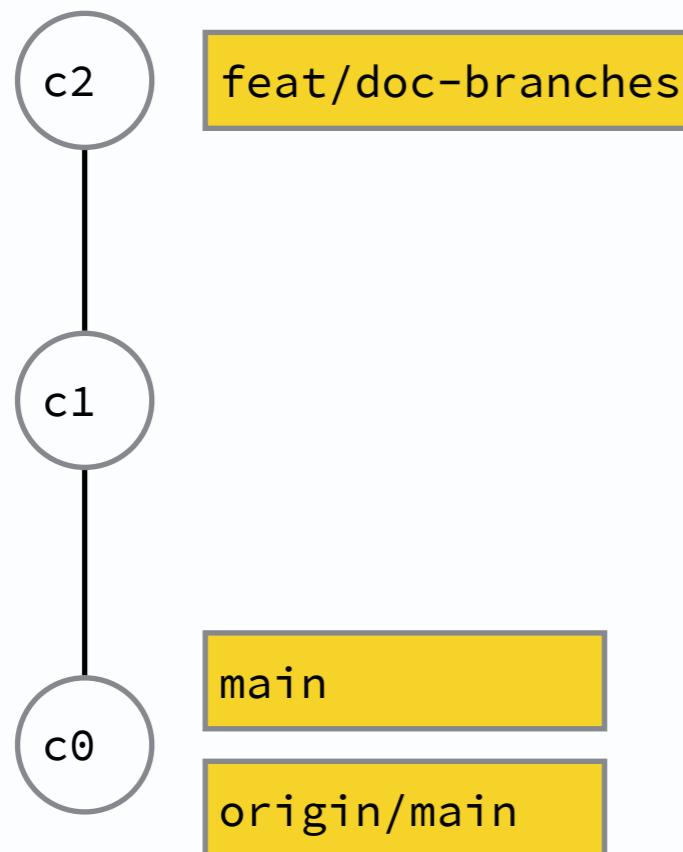


- Fusionner des branches  
Il y a 3 commandes de fusionner une branche : git merge, git rebase et git cherry-pick
- git merge  
« Merger » une branche signifie que vous souhaitez, si une opération de Fast-Forward est possible (pas de modifications entre temps sur la branche d'origine), l'historique ne 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.
- git cherry-pick  
Récupère le contenu de la branche commit par commit
- Avant de fusionner !
  - Se placer sur la branche qui reçoit les commits  
`git switch main`
  - Ne pas avoir de modifications en cours (tous les commits ont été faits)

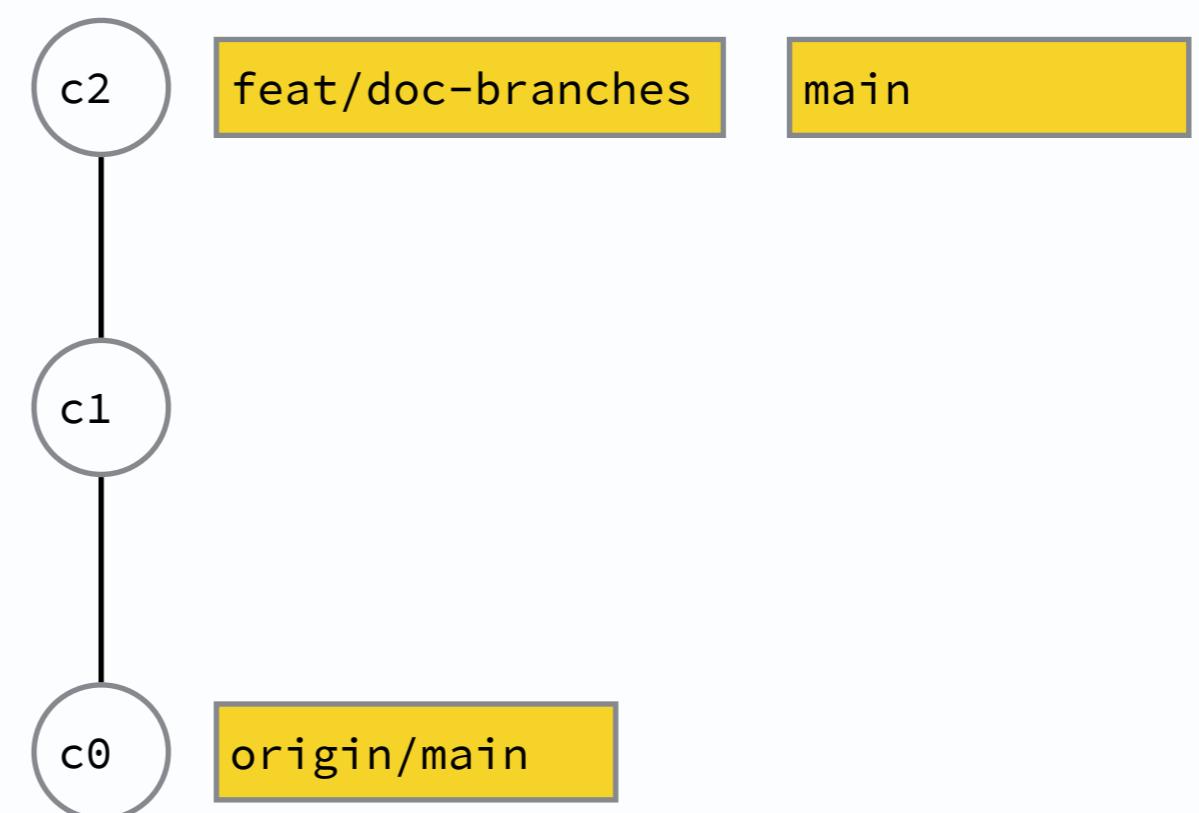


# Merge - Fast-forward

- Avant



- Après

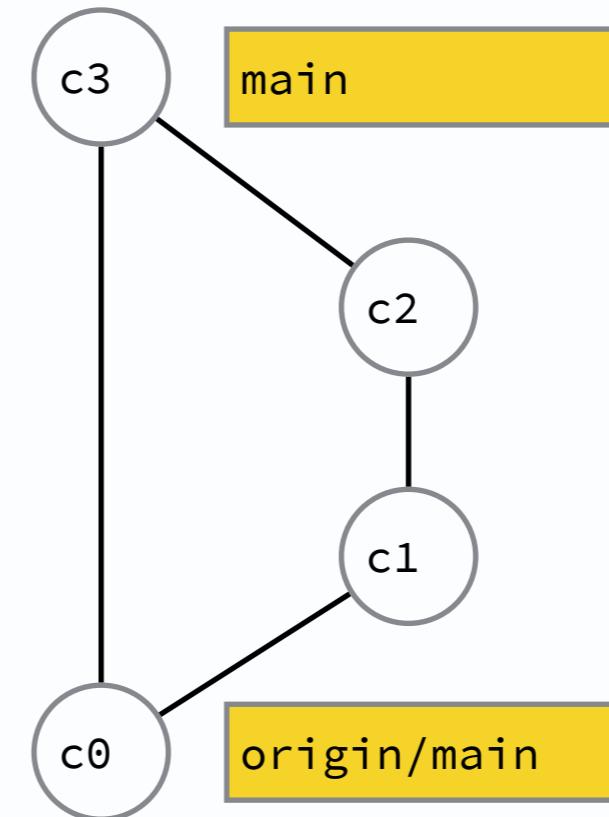
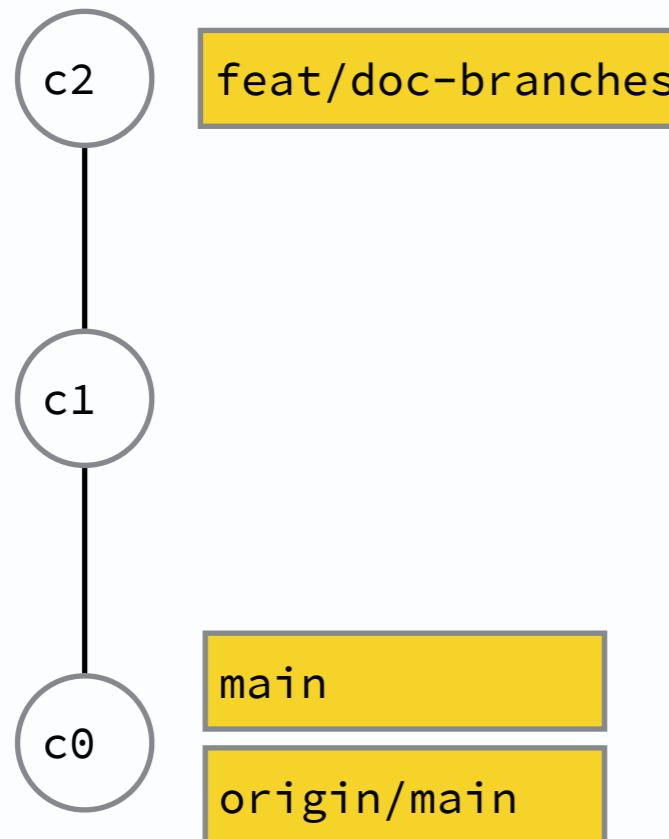


- `git switch main`
- `git merge --ff feat/doc-branches`  
`git merge --ff-only feat/doc-branches`
- (`--ff` est l'option par défaut)



# Merge - Recursive / ORT

- Avant

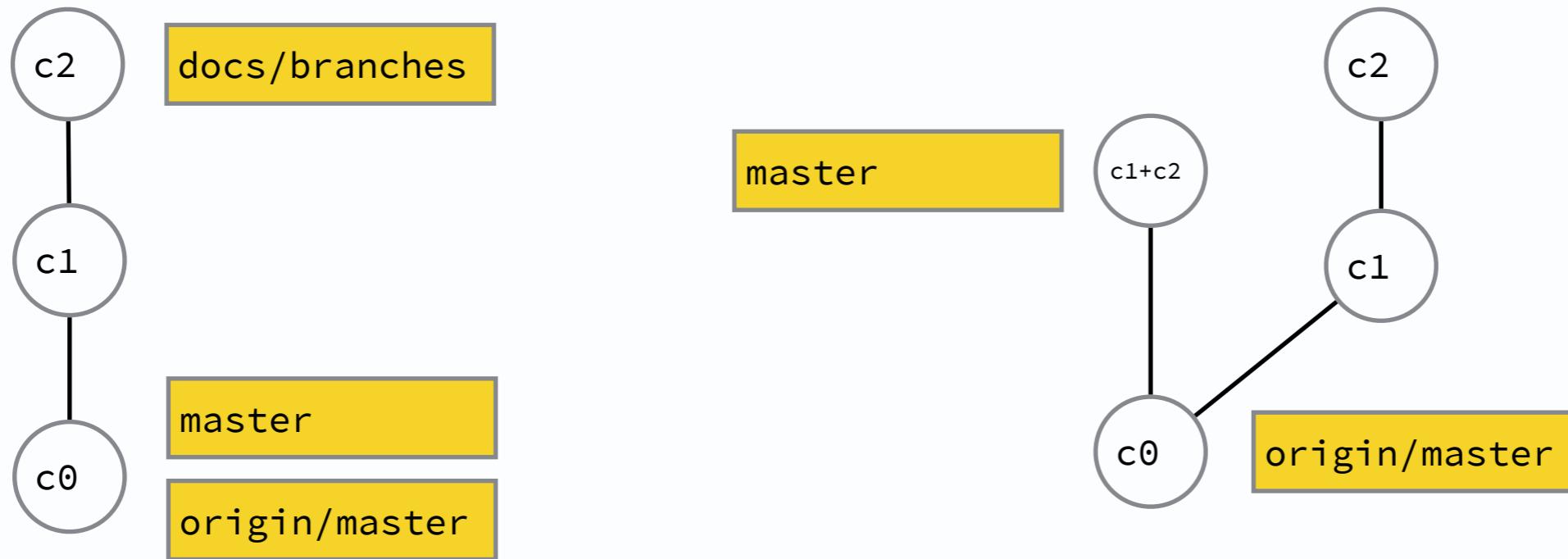


- `git switch main`
- `git merge --no-ff feat/doc-branches`
- Après



# Merge - Squash

- › Avant

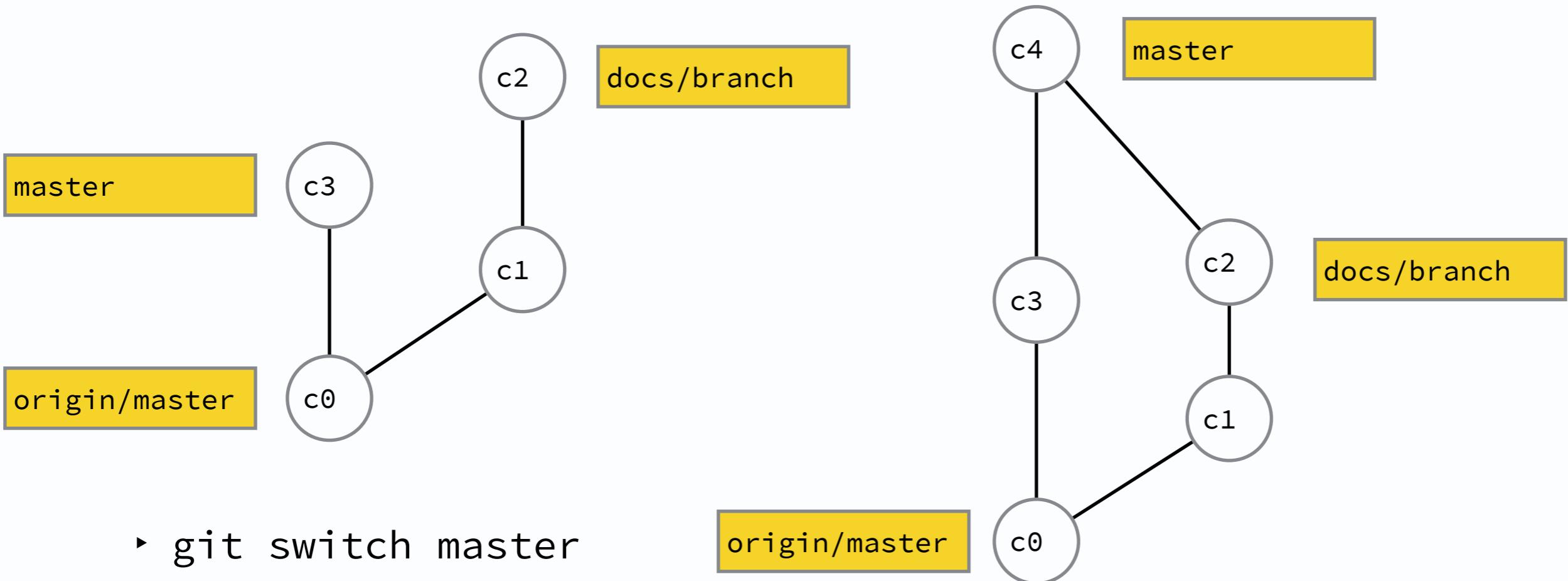


- › `git switch master`
- › `git merge --ff --squash docs/branches`
- › `git commit`



# Merge - Recursive / ORT

- Avant

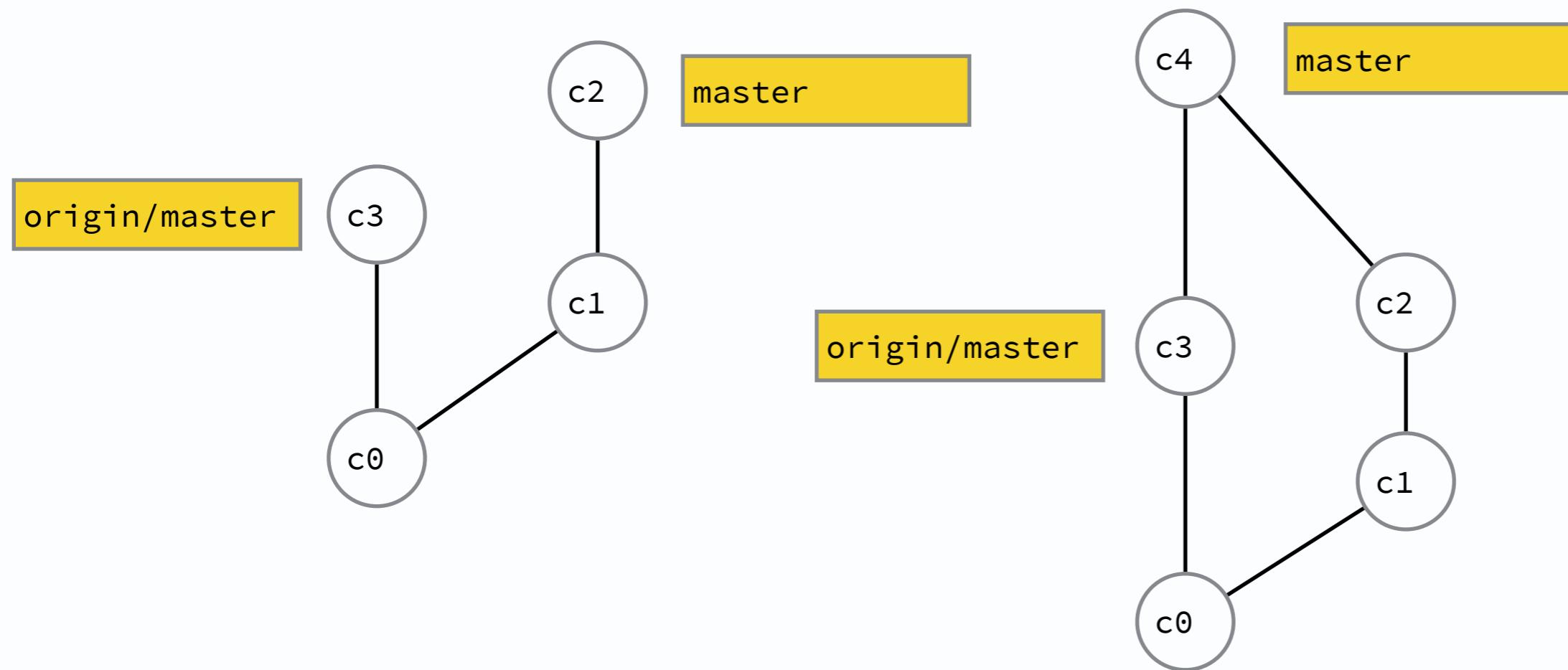


- git switch master
- git merge --no-ff docs/branch
- (ici --ff ou --no-ff ont le même résultat, --ff-only n'aurait rien fait dans ce cas)
- Après



# Merge - Appliquer aux remotes

- Avant



# Merge - Options

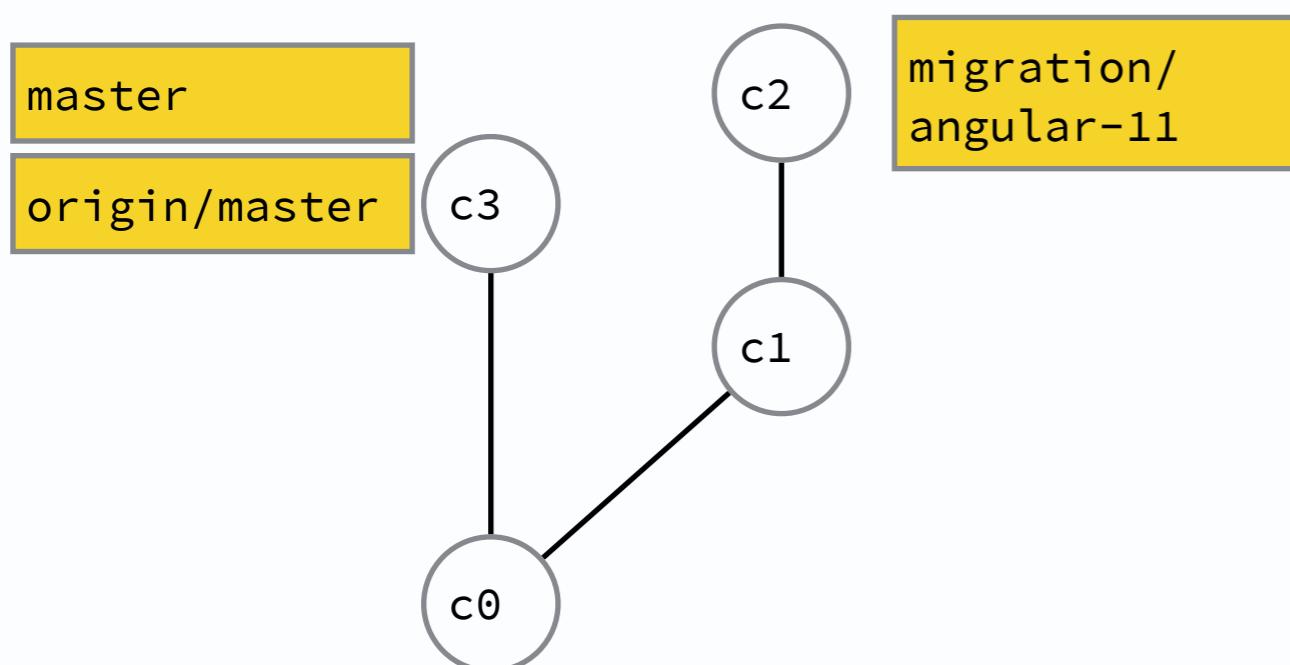


- --ff : Fast-Forward si possible; sinon recursive (commit de merge) - Par défaut
- --ff-only: Fast-Forward si possible; sinon erreur
- --no-ff: toujours recursive (commit de merge)
  
- Pour changer la valeur par défaut :
  - --no-ff  
`git config --global merge.ff false`
  - --ff-only  
`git config --global merge.ff only`

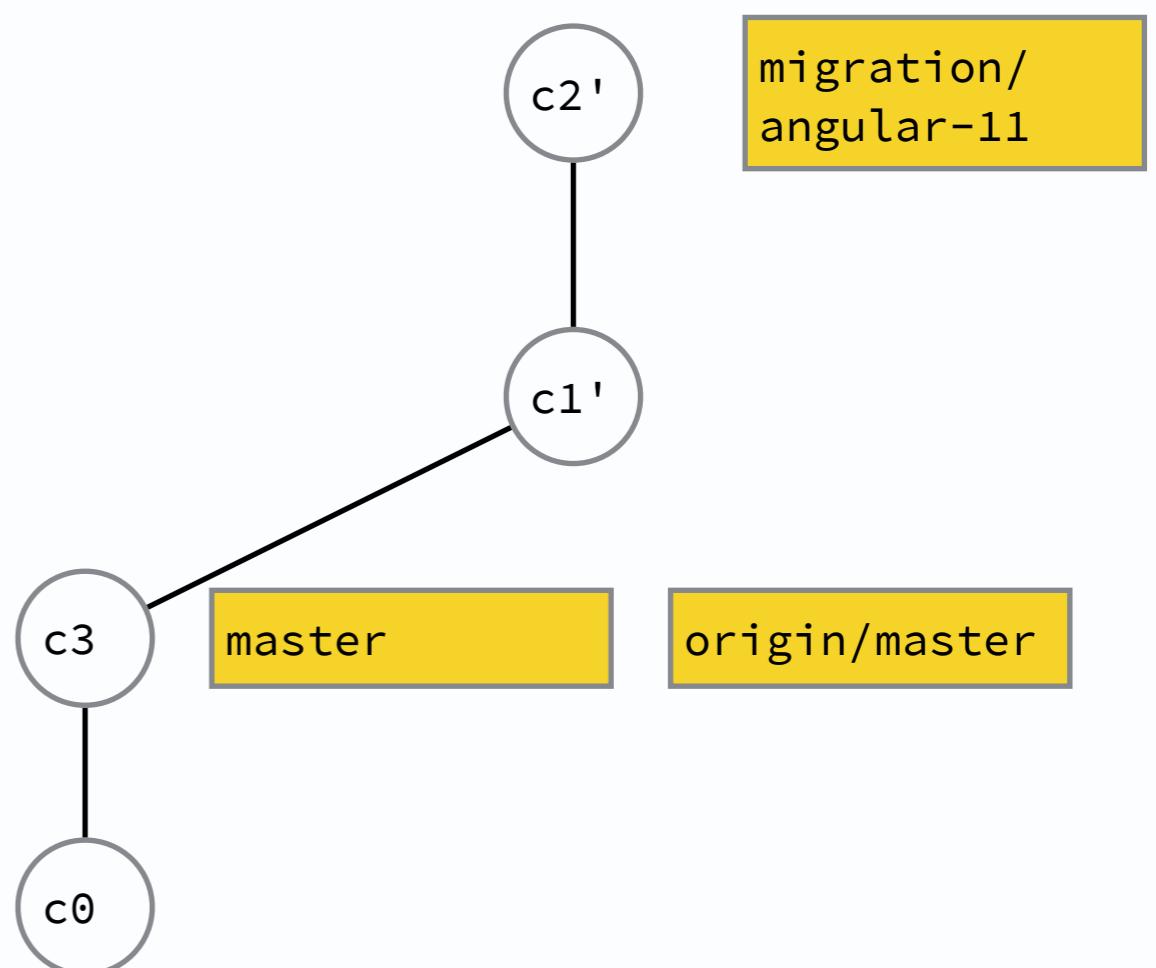


# Rebase

► Avant



► Après

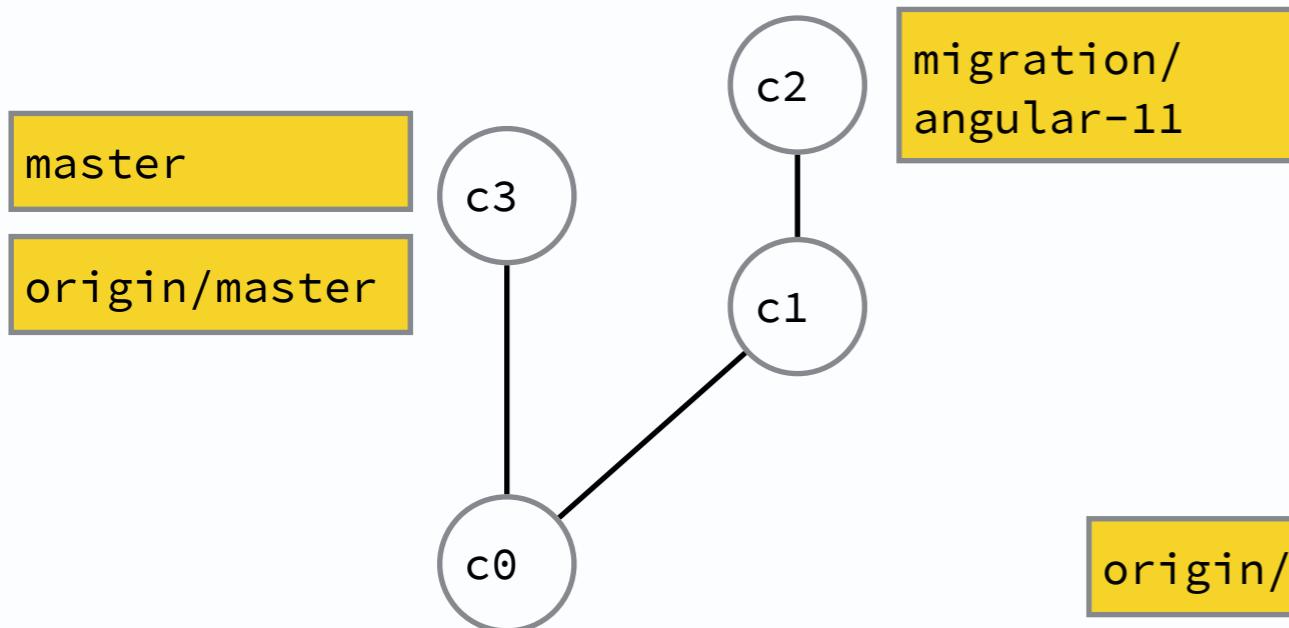


- `git switch migration/angular-11`
- `git rebase master`

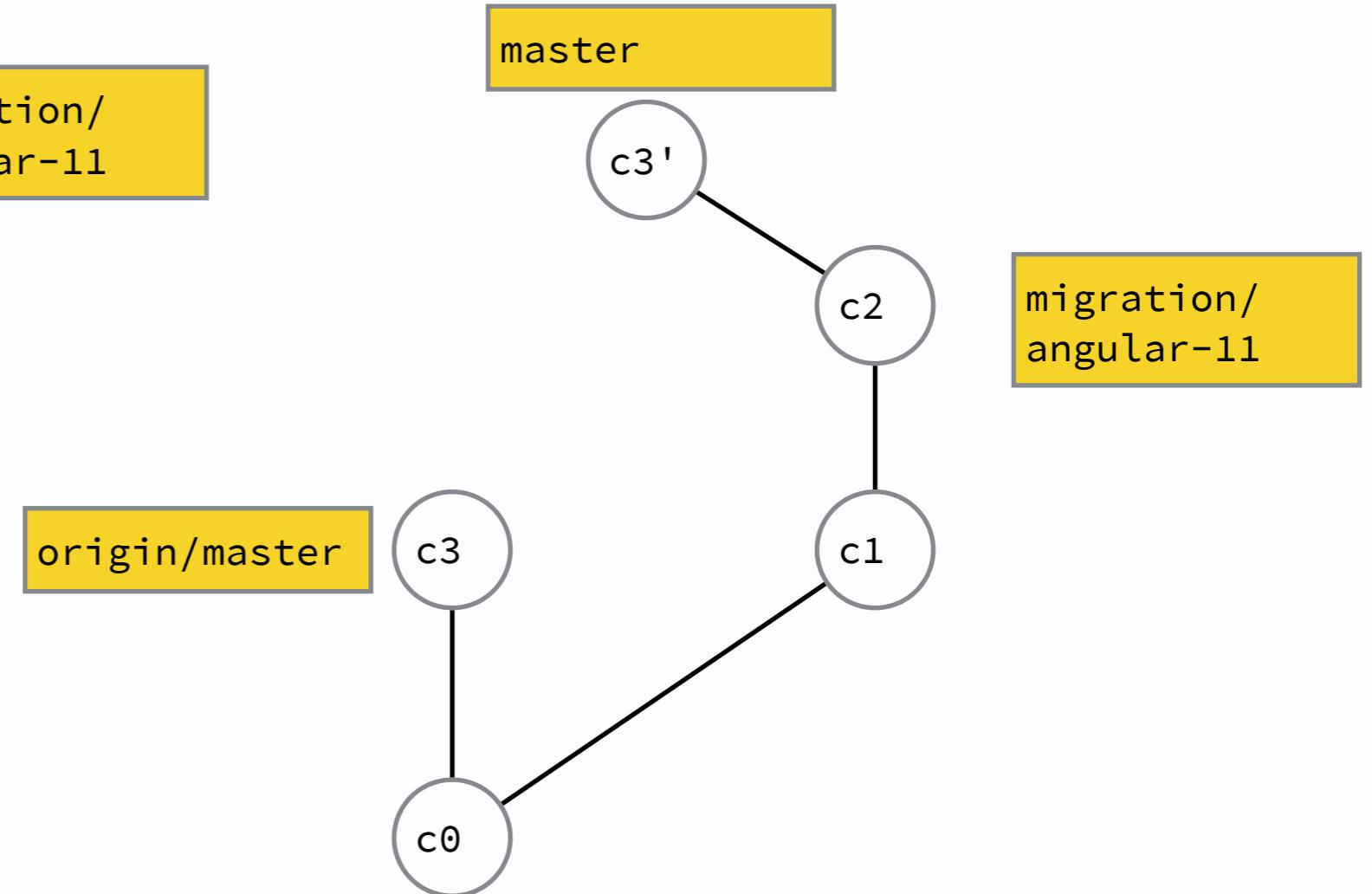


# Rebase - A NE PAS FAIRE

- Avant



- Après



- `git switch master`

- `git rebase migration/angular-11`

- A NE PAS FAIRE car nouveau c3 qui avait déjà été partagé

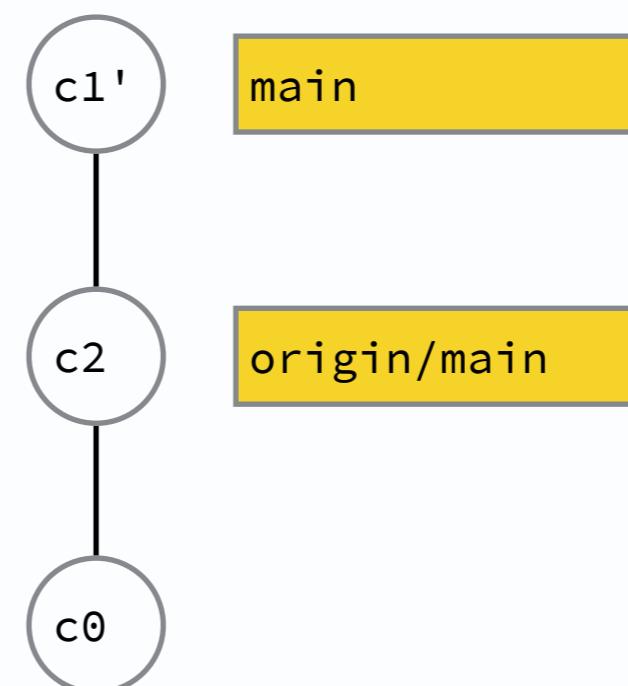


# Rebase (remote)

► Avant



► Après

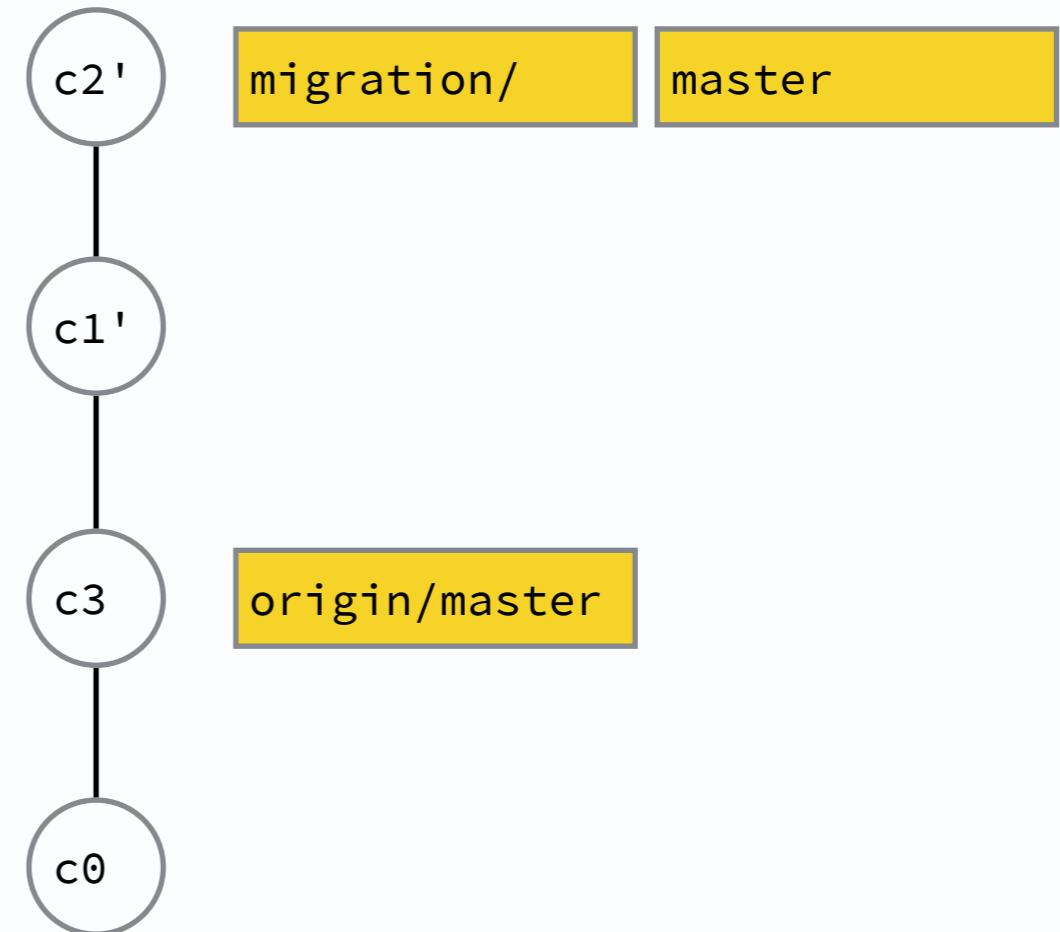
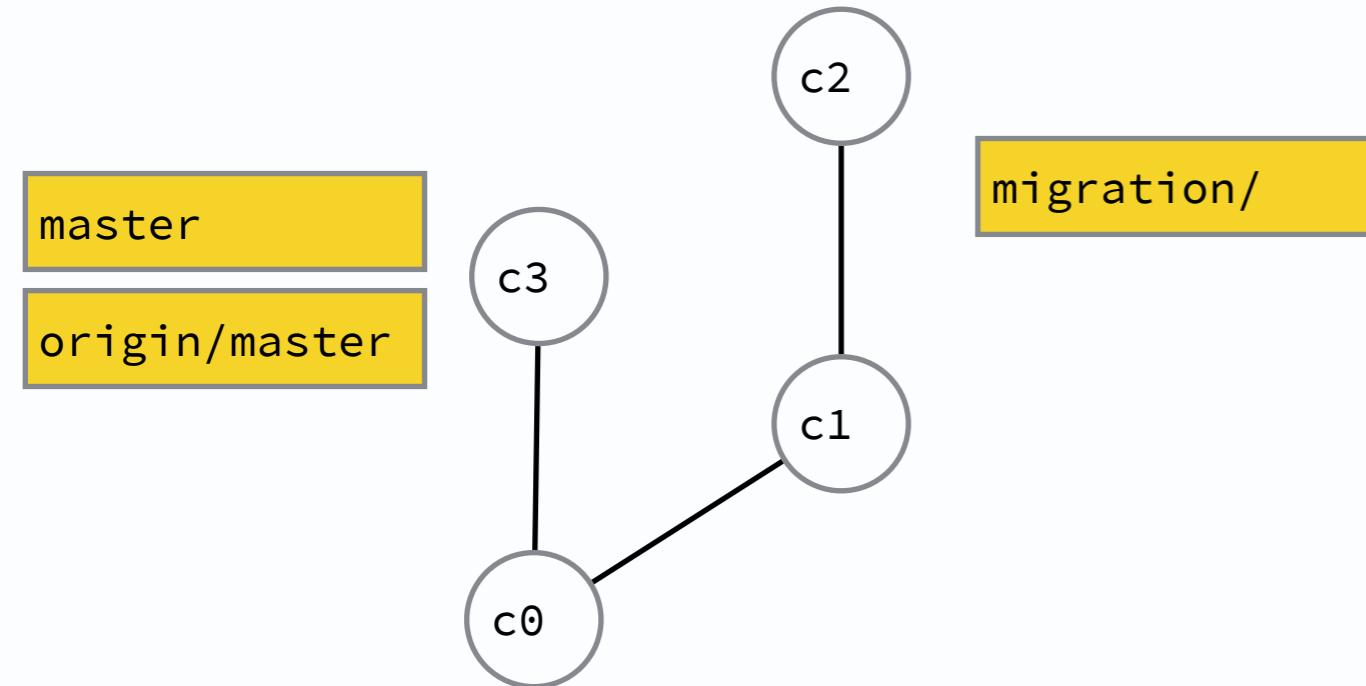


- `git switch main`
- `git rebase origin/main`



# Rebase + Merge Fast-Forward

- › Avant

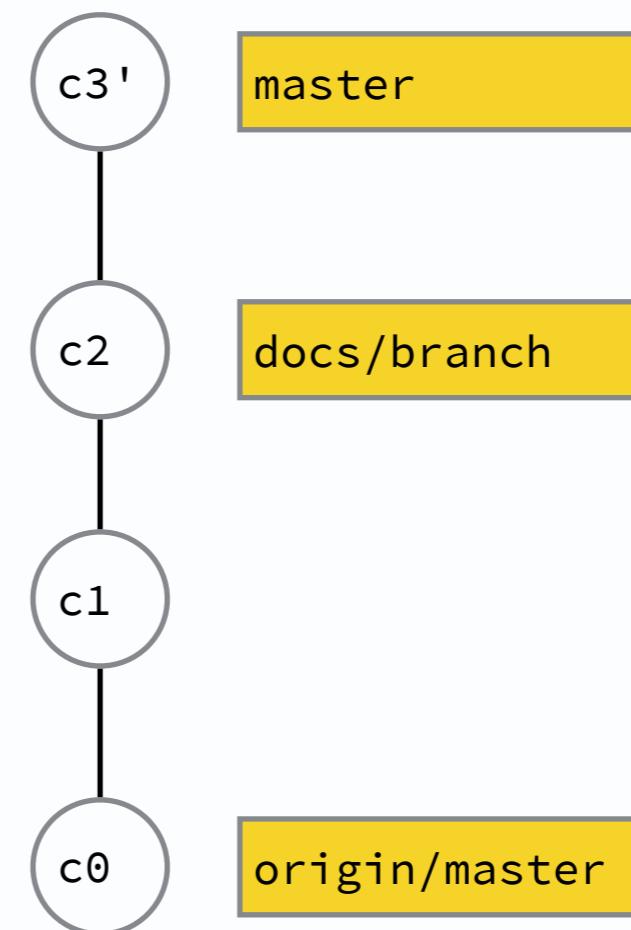
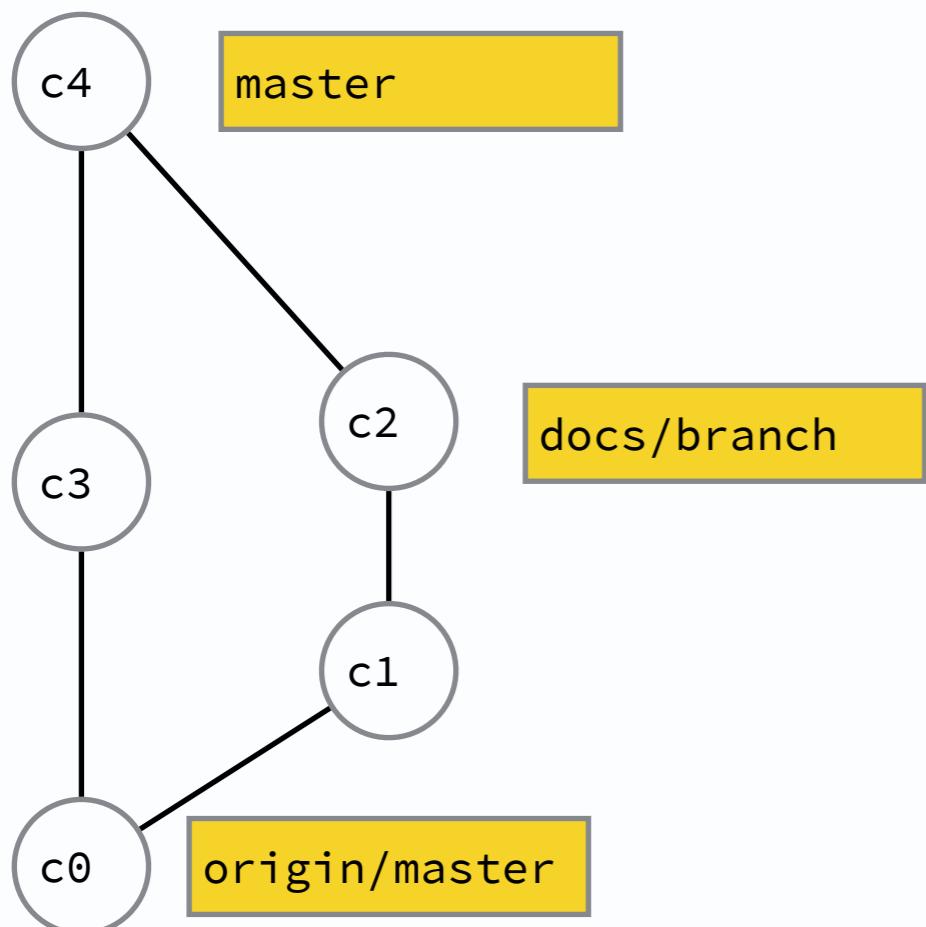


- › git checkout docs/branch
- › git rebase master
- › git checkout master
- › git merge --ff docs/branch
- › Après



# Rebase - Détruit les commits de merge

- Avant

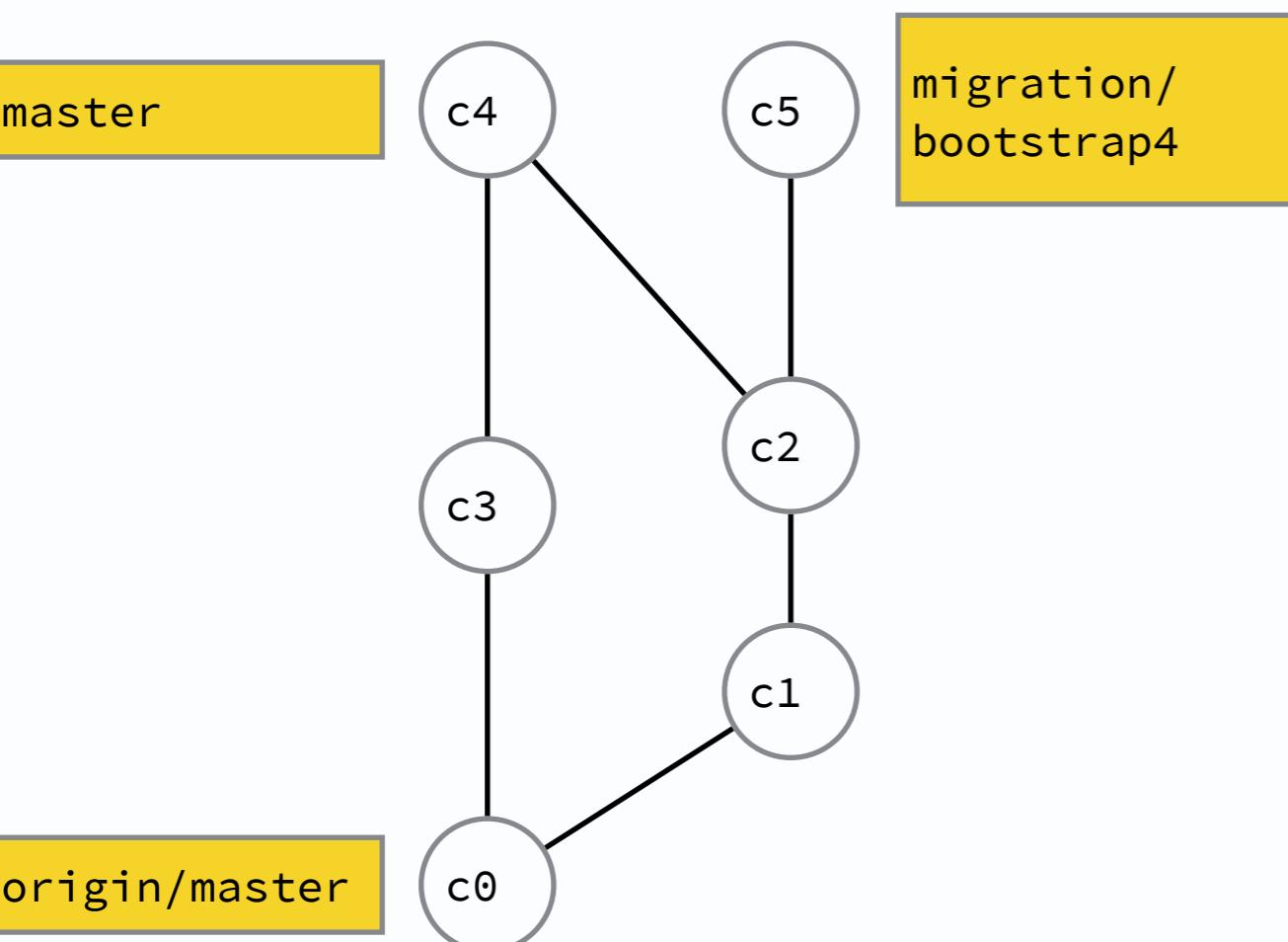


- git checkout master
- git rebase docs/branch
- Après

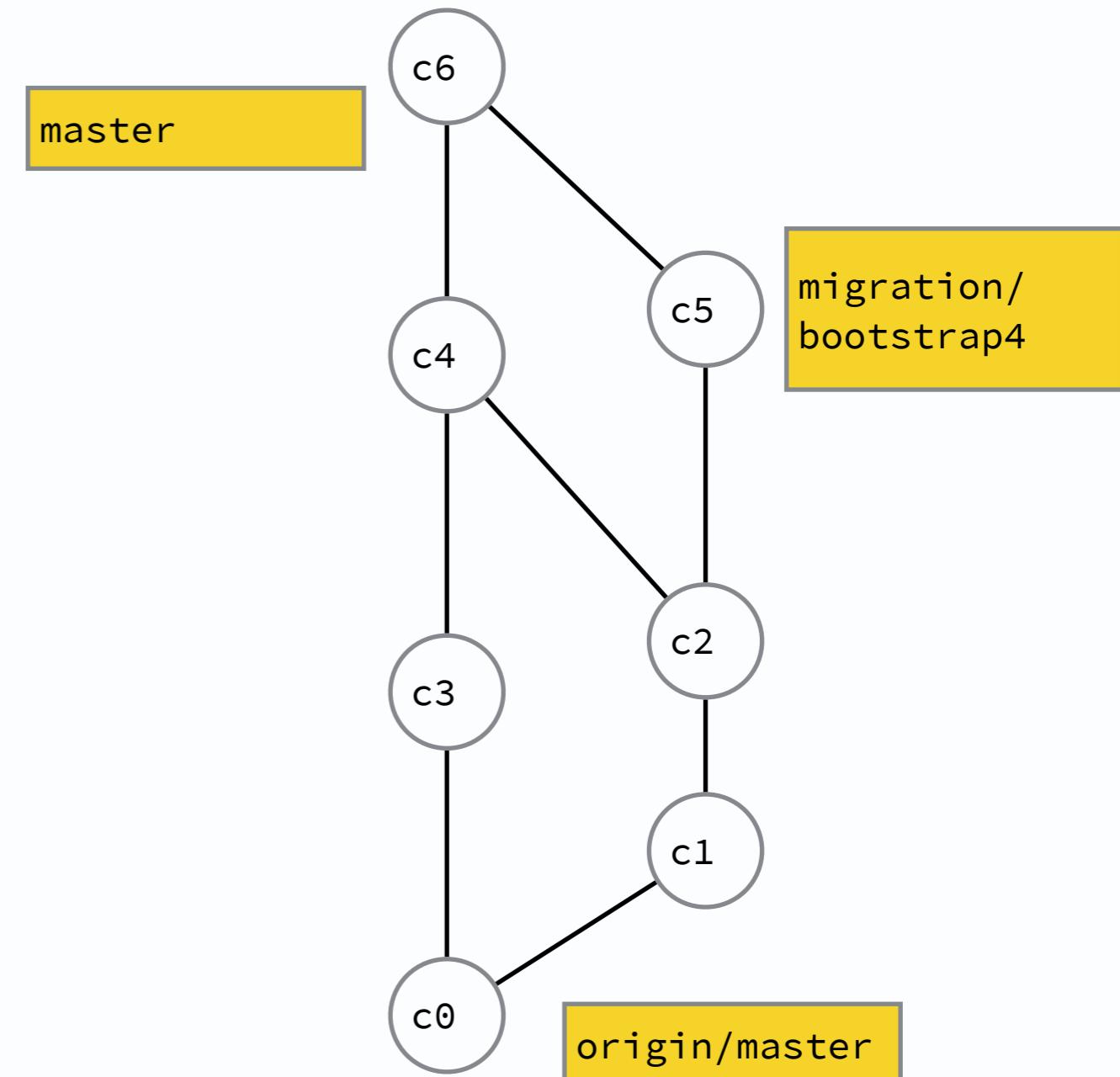


# Rebase - Détruit les commits de merge

- Avant



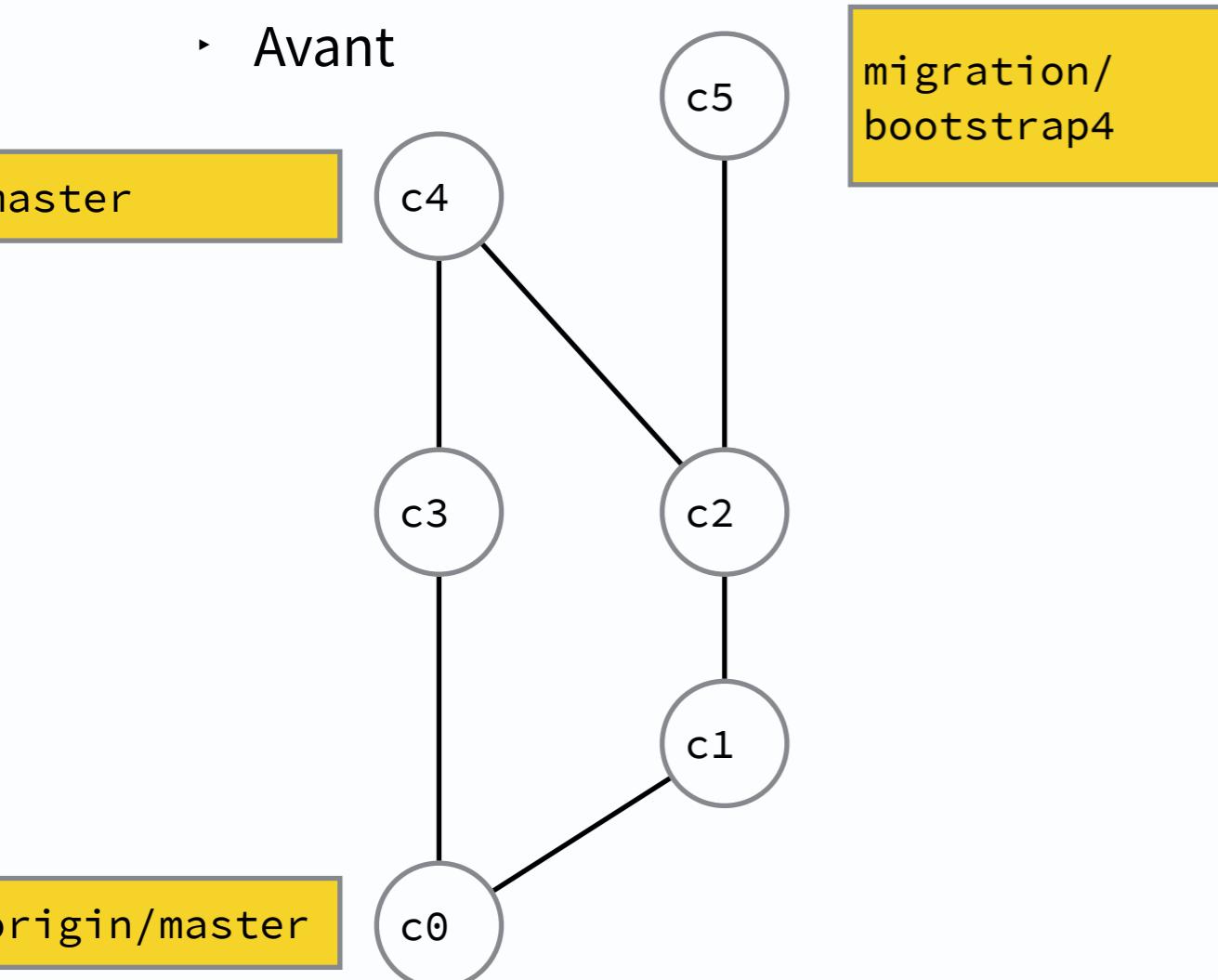
- git checkout master
- git merge docs/branch
- Après



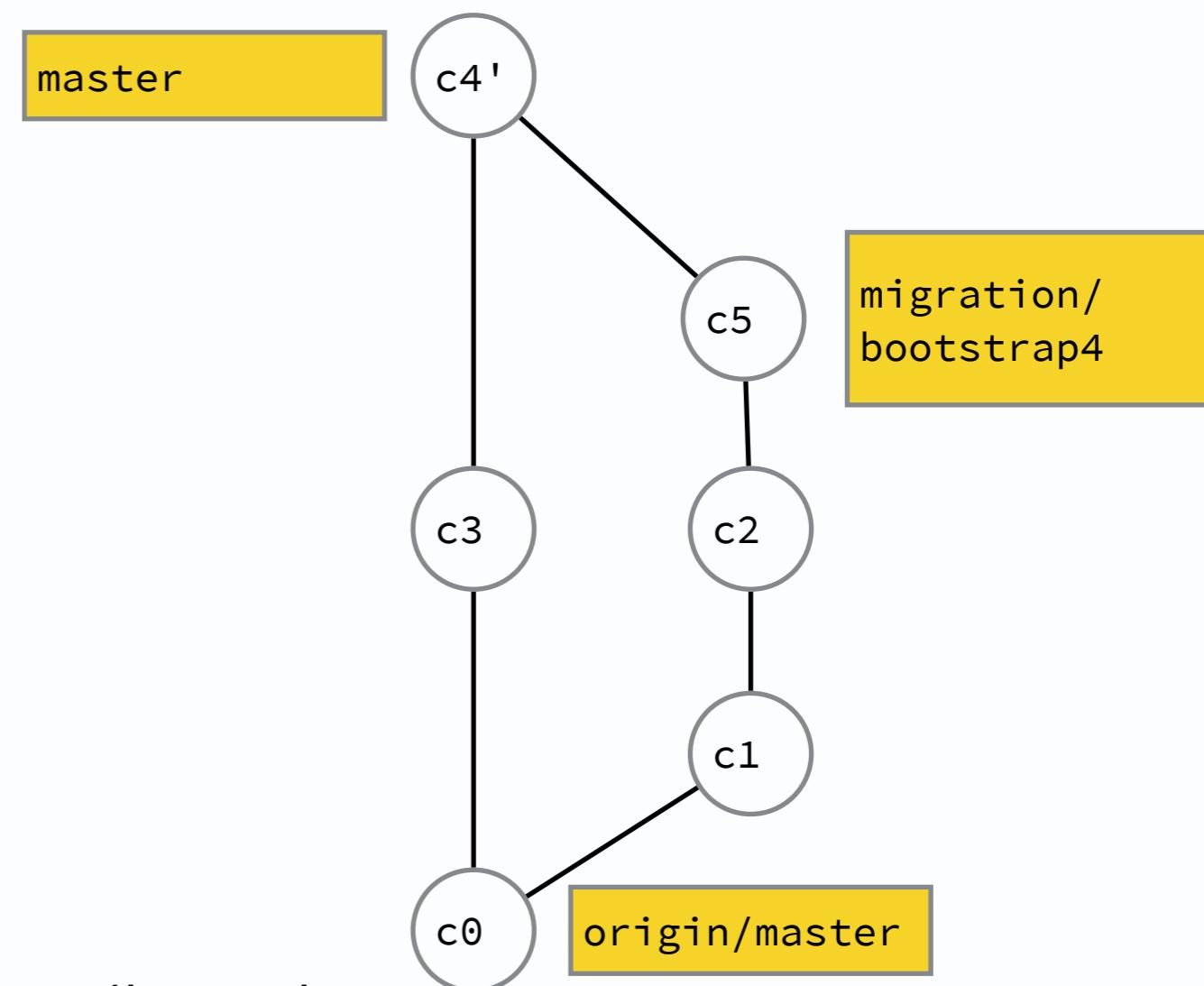


# Rebase - Détruit les commits de merge

- ▶ Avant



- ▶ `git checkout master`
- ▶ `git rebase --rebase-merges docs/branch`
- ▶ Après



- ▶ `git rebase --preserve-merges docs/branch` (avant 2.18)

# 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

- Connaitre le nom de la branche sur laquelle vous vous trouvez :  
`git symbolic-ref HEAD --short`



**formation.tech**

# Conflits



# Conflits - Introduction

- Certaines commandes peuvent mener à des conflits :
  - git merge
  - git rebase
  - git cherry-pick
  - git stash
  - ...



# Conflits - Introduction

- Certaines commandes peuvent mener à des conflits :
  - git merge
  - git rebase
  - git cherry-pick
  - git stash
  - ...



**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)
  - `git push origin master:autre-nom-distant`  
(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**

# Workflows



# Git - Centralized/Basic Workflow

- Centralized Workflow

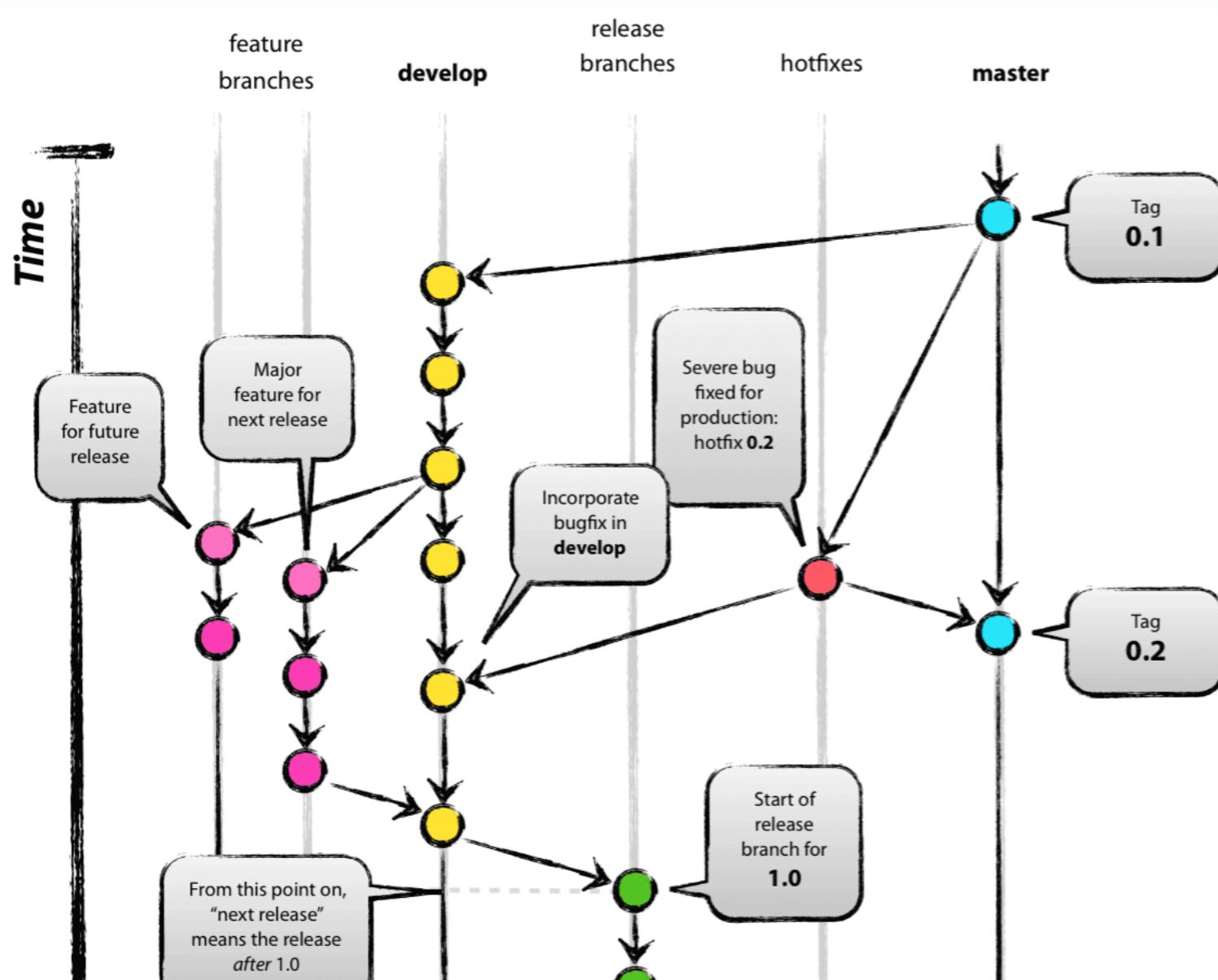
Travailler directement sur master/main (ok seul ou à plusieurs mais pas en même temps, ou avec rebase à plusieurs en même temps mais risqué en cas de conflit)  
Attention aux "plans de métro"

```
* d7d52f2 Update contrib modules
* d8ae64c Update core from 7.35 to 7.43
/
* c58337e NIW-162: Added maxlength and number validation for phone field on user form and customer profile form
* f90c091 Hidden "Offres Spéciales" menu link
/
* 56505bd Merge environment 'recette' into staging
/
* 3afc827 Merge environment 'recette' into staging
/
* 4000382 Merge environment 'recette' into staging
/
* ad23e15 Merge environment 'recette' into staging
/
* 5f39f31 Merge environment 'recette' into staging
/
* c4ef37e Merge environment 'recette' into staging
/
* f3c7e50 Merge environment 'recette' into staging
/
* df85d63 Merge environment 'recette' into staging
/
* b5e41e7 Merge environment 'recette' into staging
*
* b1b1267 Merge environment 'recette' into staging
/
* b90526d Merge environment 'recette' into staging
/
* a83b460 Merge environment 'recette' into staging
/
* 471703f Merge environment 'recette' into staging
/
* 0506115 Merge environment 'recette' into staging
/
* 67671ba Merge environment 'recette' into staging
```



# Git - Git Flow Workflow

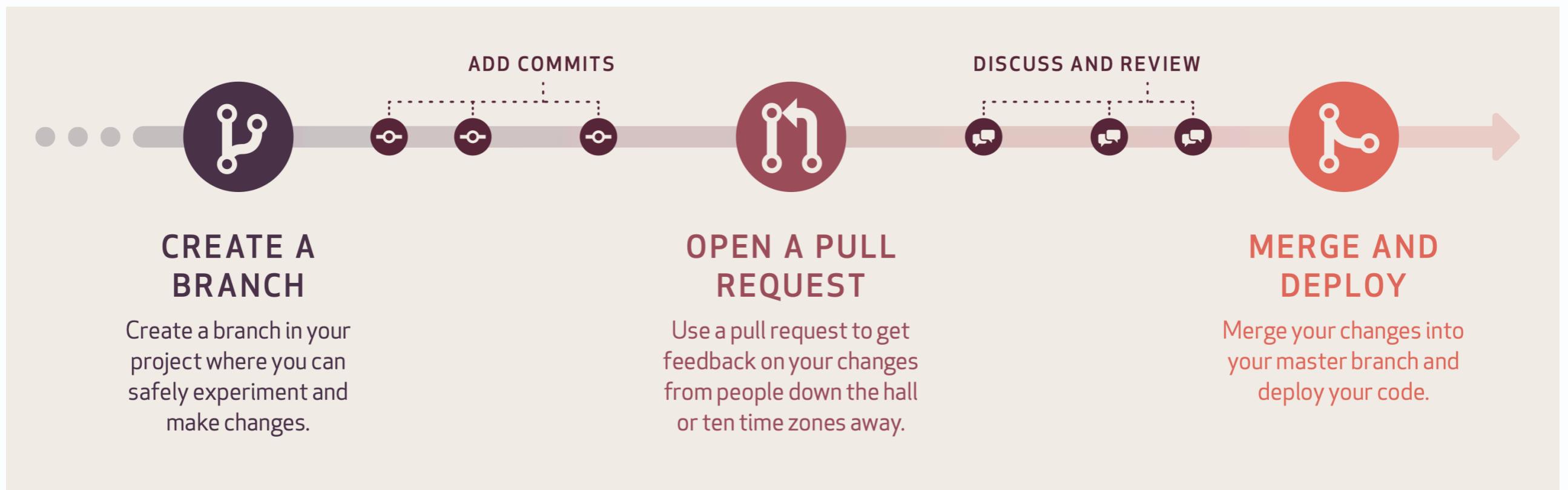
- Git Flow
  - <http://nvie.com/posts/a-successful-git-branching-model/>
  - [http://danielkummer.github.io/git-flow-cheatsheet/index.fr\\_FR.html](http://danielkummer.github.io/git-flow-cheatsheet/index.fr_FR.html)
- "Déprécié" depuis Mars 2020



# Git - Github Flow Workflow



- Github Flow
  - Créer une nouvelle branche
  - Faire ses commits
  - Passer par une Pull Request (Merge Request sur GitLab et BitBucket)
  - Merger via la plateforme





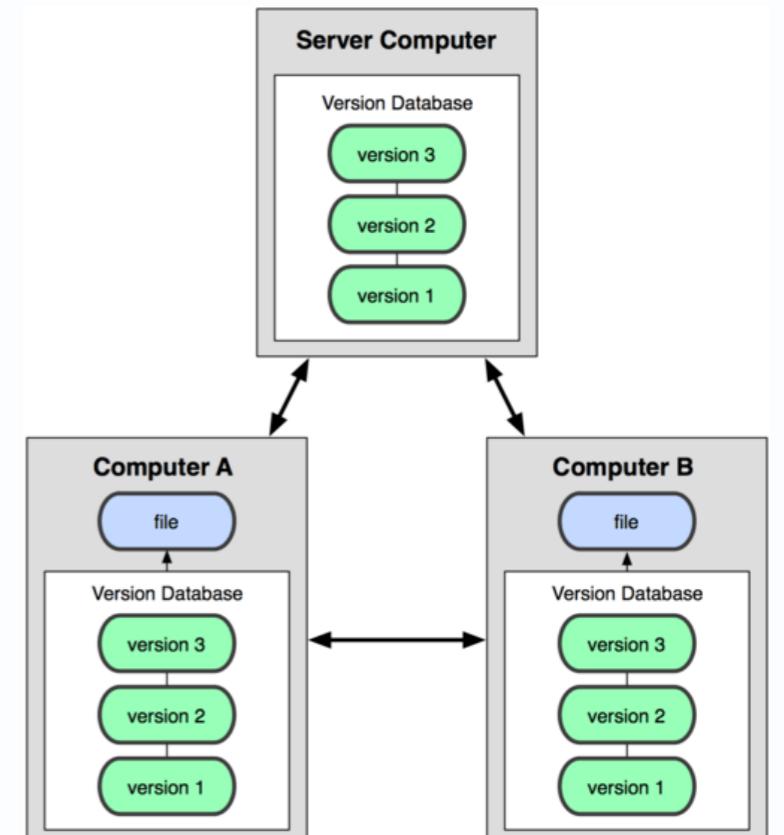
**formation.tech**

# 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 https://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 Procolol



- 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
  - Echanges réseau un peu plus lourds qu'en 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
  - Valider le message de commit
  - Lancer des tests automatisés avant un commit
  - Créer un build/compilation 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`