



Formation git - basique

Comprendre git pour l'utiliser tous les jours



34 AVENUE DE L'OPERA 75002 PARIS > FRANCE > WWW.OCTO.COM



■ *Se connaître*

- ▷ Quel *source control* utilisez-vous ?
- ▷ Qu'aimez-vous sur celui-ci ?
- ▷ Qu'est ce que vous aimez moins ?
- ▷ Qu'est-ce-qui vous manque et que vous aimeriez avoir ?



Agenda

Basique

- 1 - Découverte de git
- 2 - Hands-on : premier pas avec git
- 3 - De SVN vers git
- 4 - Hands-on : comprendre les basiques de git
- 5 - Hands-on : travailler avec un repo. Distant
- 6 - Hands-on : git stash, blame
- 7 - Hands-on : utiliser un workflow
- 8 - Hands-on : ré-écrire l'histoire

Avancé

- 1 - Git sous le capot
- 2 - Hands-on : rebaser pour changer l'histoire
- 3 - Merge vs rebase
- 4 - Hands-on : git cherry-pick, bisect
- 5 - Hands-on : revenir en arrière
- 6 - Hands-on : git hook
- 7 - Hands-on : travailler avec gitlab





Découverte de git

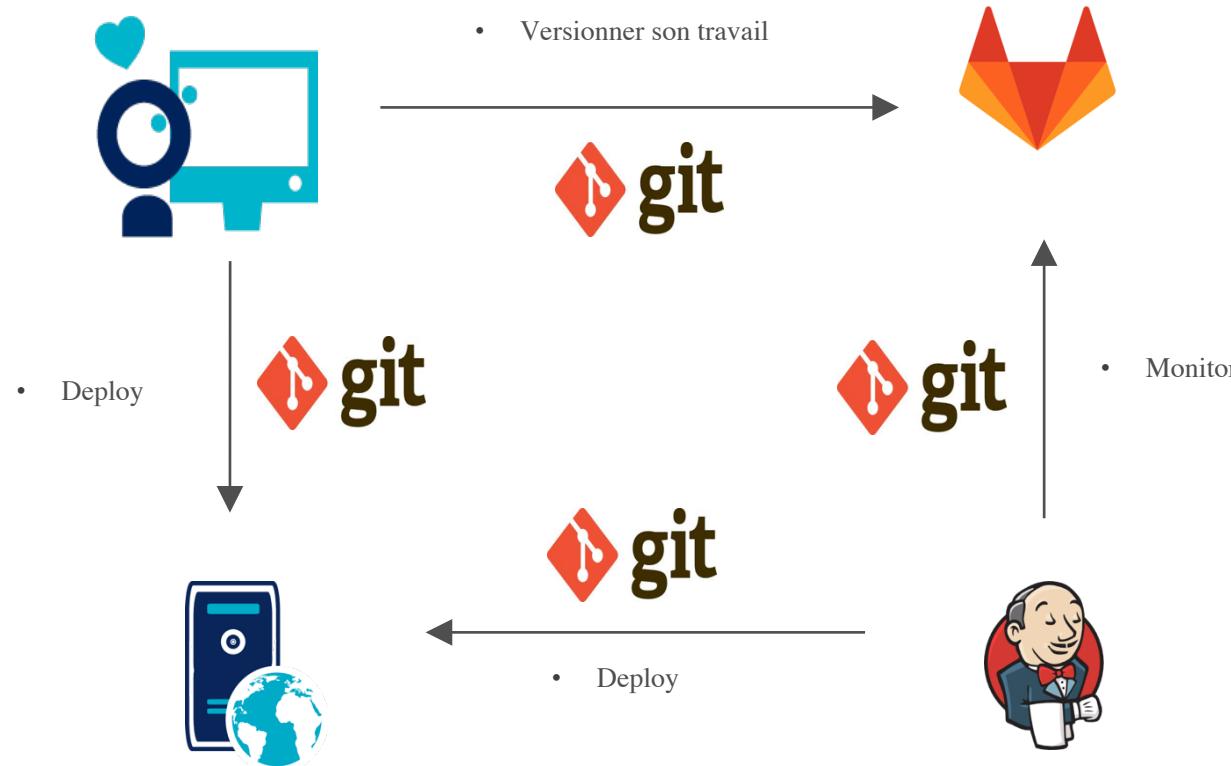
git from the trenches ...



• [@xkcd](https://xkcd.com/351)



git, un protocole et un outillage versatile



- Le protocole git permet de synchroniser votre travail avec d'autres développeurs
- Des actions peuvent être déclenchées lors de certaines commandes (hook)
- Vous pouvez utiliser des forges (gitlab, github, bitbucket) comme repo. central
- Vous pouvez déployer à l'aide de git (jetez un oeil à heroku)



Introduction

- git a été créé en 2005, par Linus Torvalds pour gérer le noyau linux
 - BitKeeper ne pouvait plus être utilisé gratuitement
 - De très nombreuses branches vivant en parallèle
 - Une gouvernance très distribuée
 - Besoin de pouvoir forkér
 - Besoin de rapatrier des *features* choisies
- git est un outil très puissant, très flexible et très permissif ...
 - ...mais qui peut vite devenir complexe
- Néanmoins, il peut être utilisé de deux façons
 - À la « SVN » : commit/push/pull, quelques branches, tags
 - On y gagne alors surtout en collaboration grâce à des plateformes comme gitlab, et en capacité de merge
 - On a moins de conflits
 - Le moteur de merge de git est remarquable
 - A la « git » : features branch, stash, tags, cherry picking, bissect, dépôts distribués...
 - Gestion de source adaptable à quasi n'importe quel workflow
 - Plein de *features* très utiles pour les projets complexes



■ Introduction

- Et donc, oui, git c'est compliqué...
- Mais oui, git possède plein d'outils qui peuvent améliorer votre quotidien de développeur (promis, demandez à tous ceux qui maîtrisent git)
- Attention, il ne faut pas seulement apprendre des commandes mais surtout comprendre les concepts sous-jacents
- Pour avoir les bases en git, il faut bien comprendre :
 - Les différents états d'un fichier
 - La différence entre une branche locale et distante
 - La synchronisation entre les dépôts



Hands-On

Premier pas avec git



Pour les utilisateurs sous Windows

- Utiliser git bash avec ConEmu pour utiliser la ligne de commande plus facilement

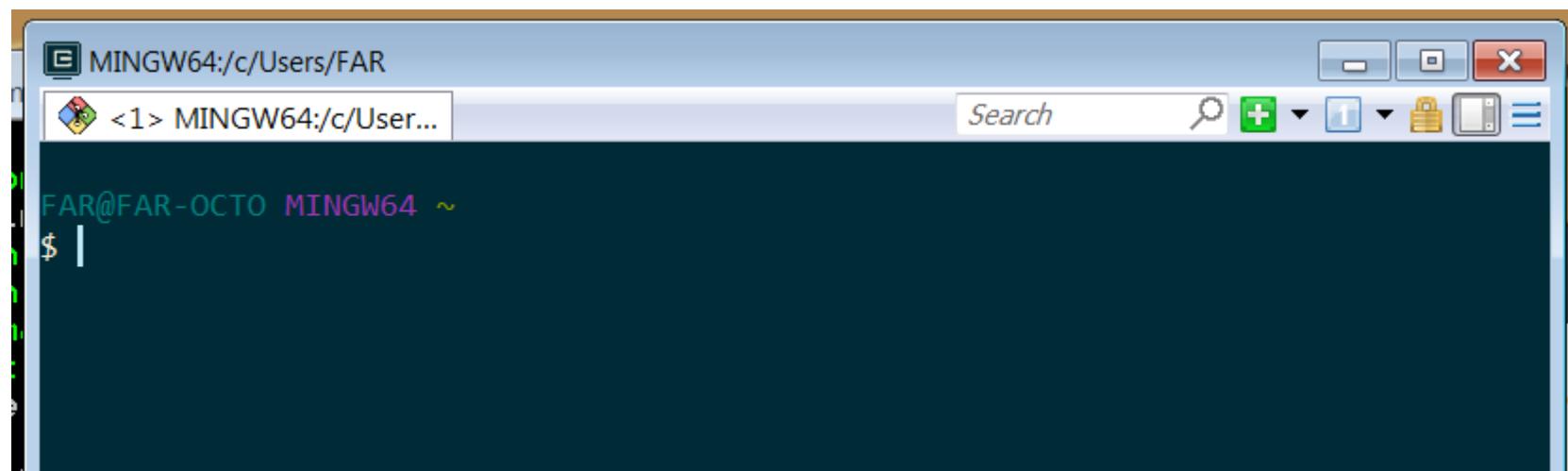
Depuis une invite de commande lancée en administrateur :

```
> @powershell -NoProfile -ExecutionPolicy Bypass -Command "iex ((New-Object System.Net.WebClient).DownloadString('https://chocolatey.org/install.ps1'))" && SET "PATH=%PATH%;%ALLUSERSPROFILE%\chocolatey\bin"  
> choco install git  
> choco install conemu
```

Vous trouverez cette commande sur le site de chocolatey, un gestionnaire de package pour Windows. Elle installe chocolatey

installe git et git bash

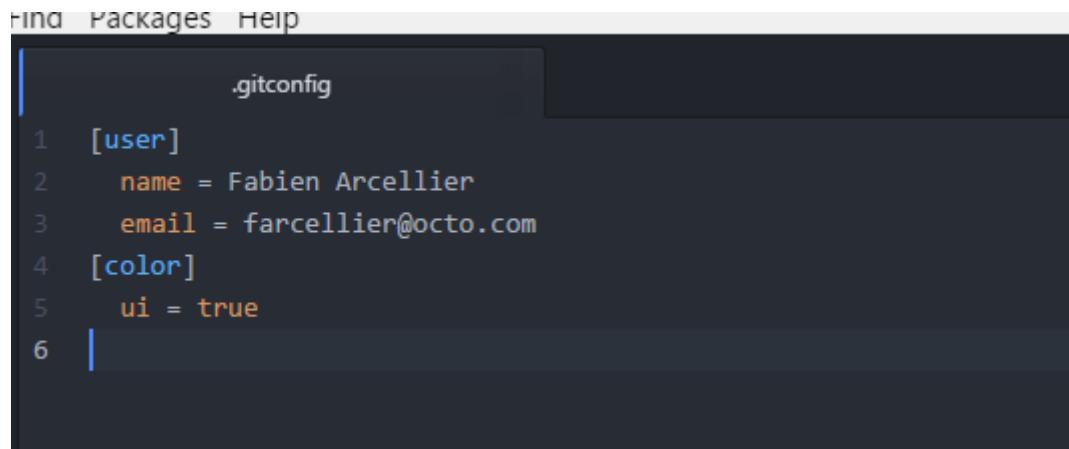
installe conemu



■ Premières commandes et prise en main

- Mettre en place son environnement local pour annoter les commits
 - git config --global user.name "Fabien Arcellier"
 - git config --global user.email farcellier@octo.com
 - git config --global color.ui true

Pour en savoir plus : [Démarrage rapide - Paramétrage à la première utilisation de git](#)



```
Find Packages Help
.gitconfig
1 [user]
2   name = Fabien Arcellier
3   email = farcellier@octo.com
4 [color]
5   ui = true
6 
```

A screenshot of a terminal window titled ".gitconfig". The window shows a configuration file with the following content:
1 [user]
2 name = Fabien Arcellier
3 email = farcellier@octo.com
4 [color]
5 ui = true
6
The file is displayed in a dark-themed terminal.

Le fichier de config est un fichier ini créé par défaut dans votre dossier utilisateur ~/.gitconfig



Créer un premier repo. en local et ajouter un fichier

● Création d'un premier repo git :

```
> mkdir monrepo  
> cd monrepo  
> git init
```

git add . / git add -
all aurait aussi marché
pour ajouter l'ensemble
des fichiers du dépôt

● On ajoute un fichier et on le commit...

```
> touch readme.txt  
> git add readme.txt  
> git status  
> git commit -m "added read me"  
> git status  
> git log
```

```
$ git log  
commit 6e24abc225a93fca44c64c245e93540d08c4834f  
Author: Fabien Arcellier <farcellier@octo.com>  
Date:   Sun Dec 18 14:20:14 2016 +0100  
  
        added read me
```

Question avancée :

- La commande git init --bare crée un repository central. A quoi sert ce type de repo. ? ([réponse](#))
- Qui a le même hash de commit que moi ? ([explication](#))



Cheat sheet vi & shell

- Deux commandes à connaître.

- Mode insertion :

- > a
- > i

- Sauver/Quitter :

- > ESC : wq!

- Tips pour les commandes en Shell ajoutez :

- > Git HUD : <https://github.com/gbataille/gitHUD>
- > Ou Git-Radar : <https://github.com/michaeldfallen/git-radar>

```
13:56 gbataille ~/Doc..Perso/sampleGit ↵ m 1↔1 [b2] 1↑ 1M 1M 1A $ █
```



Supprimer un fichier du repo.

● On ajoute un fichier et on le commit...

```
> touch myfile.txt  
> git add myfile.txt  
> git status  
> git commit -m "added myfile"  
> git status  
> git log
```

● Pour supprimer un fichier du repo. : git rm à la place de git add puis commit etc.

```
> git rm myfile.txt  
> git status  
> git commit -m "remove myfile"  
> git status  
> git log
```

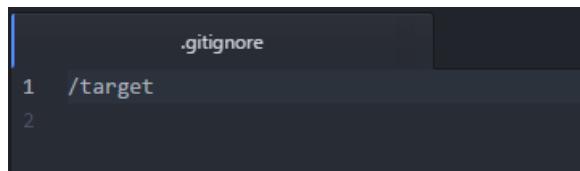
Question avancée :

- Comment supprimer un fichier du repository git (pas de l'historique) sans le retirer du dossier de travail ? (réponse)



— Ignorer un fichier ou un dossier pour ne pas l'envoyer dans le repo.

- Ajoutez un fichier .gitignore à la racine du repo. pour ignorer le dossier target lors d'un build maven



```
.gitignore
1 /target
2
```

- Vous pouvez employer les syntaxes suivantes :

● **.pyc	Ignorer les fichiers *.pyc dans tout le repo
● *.pyc	Ignorer les fichiers *.pyc dans le dossier racine
● /src/data	ignorer tous les fichiers ou dossier sous le dossier src/data
● /app/*.pyc	ignorer tous les fichiers ou dossier appellé app
● app	ignorer tous sauf target/.gitkeep (écrase les précédents)
● !/target/.gitkeep	

Question avancée :

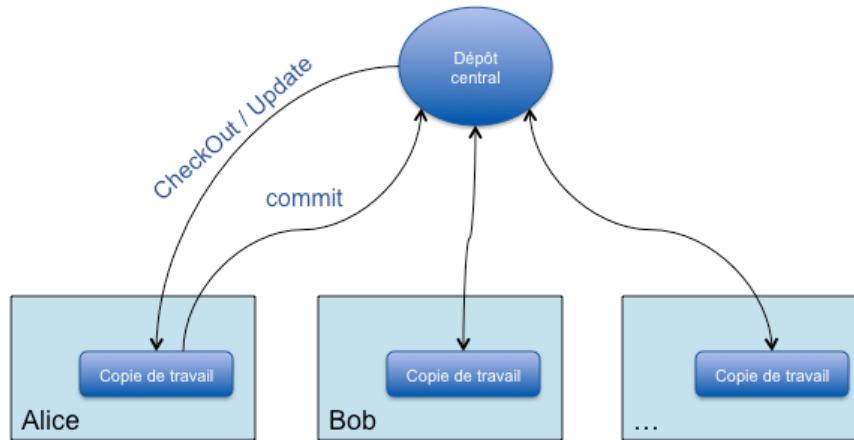
- Comment commiter un dossier vide sous git ? [\(réponse\)](#)



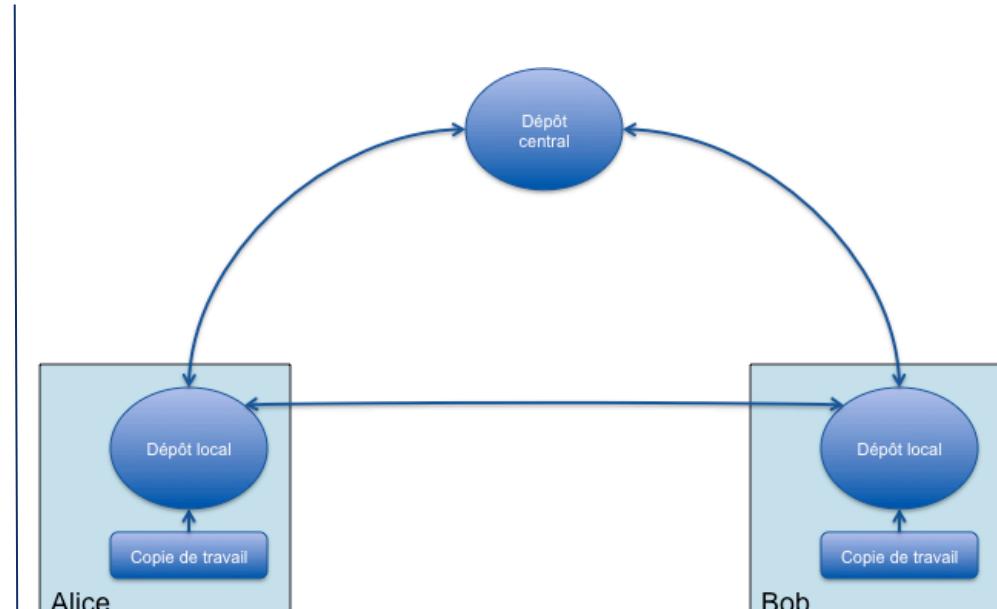


De SVN vers git

VCS et DVCS : dépôt central et dépôts distribués



dépôt central et copies de travail (SVN, CVS, TFS, ...)



dépôts distribués(git, Mercurial, ...)

* VCS : Version Control System

* DVCS : Distributed Version Control System



— De SVN vers git : glossaire

SVN	git
trunk	master
branch	branch
tag	tag
revision	hash
revert	checkout
commit	commit
update	-
log	log
checkout	clone

Même si des concepts sont communs et portent le même nom, il y a des subtilités sur leur usage. Il est dangereux de s'appuyer sur les concepts de SVN pour exécuter certaines commandes. Il faut réapprendre à les utiliser.



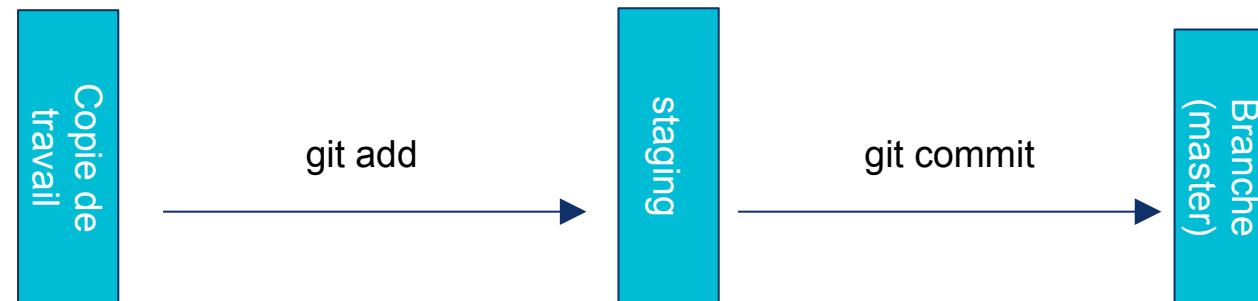
De SVN à git

- Avec git, une branche se crée au niveau du dépôt et ne coûte rien. Sur SVN, elle est créée dans un sous-dossier et se comporte comme un dossier normal.
- Un changement de branche est rapide également
- Faire des commits atomiques au maximum, vous pouvez réécrire l'historique tant que vous n'avez pas mergé votre branche dans le *master*.
- Moins de merges manuels à faire avec git
 - Moteur de merge plus puissant
- git consomme beaucoup d'espace disque
- git a un système de dépôt local, quand on commit, on commit en local. Sur SVN, on commit directement sur le central
 - Un commit ne demande pas d'être connecté
- L'identifiant d'un commit est plus difficile à comprendre (hash sur 40 caractères) qu'un numéro de révision SVN



— État d'un fichier

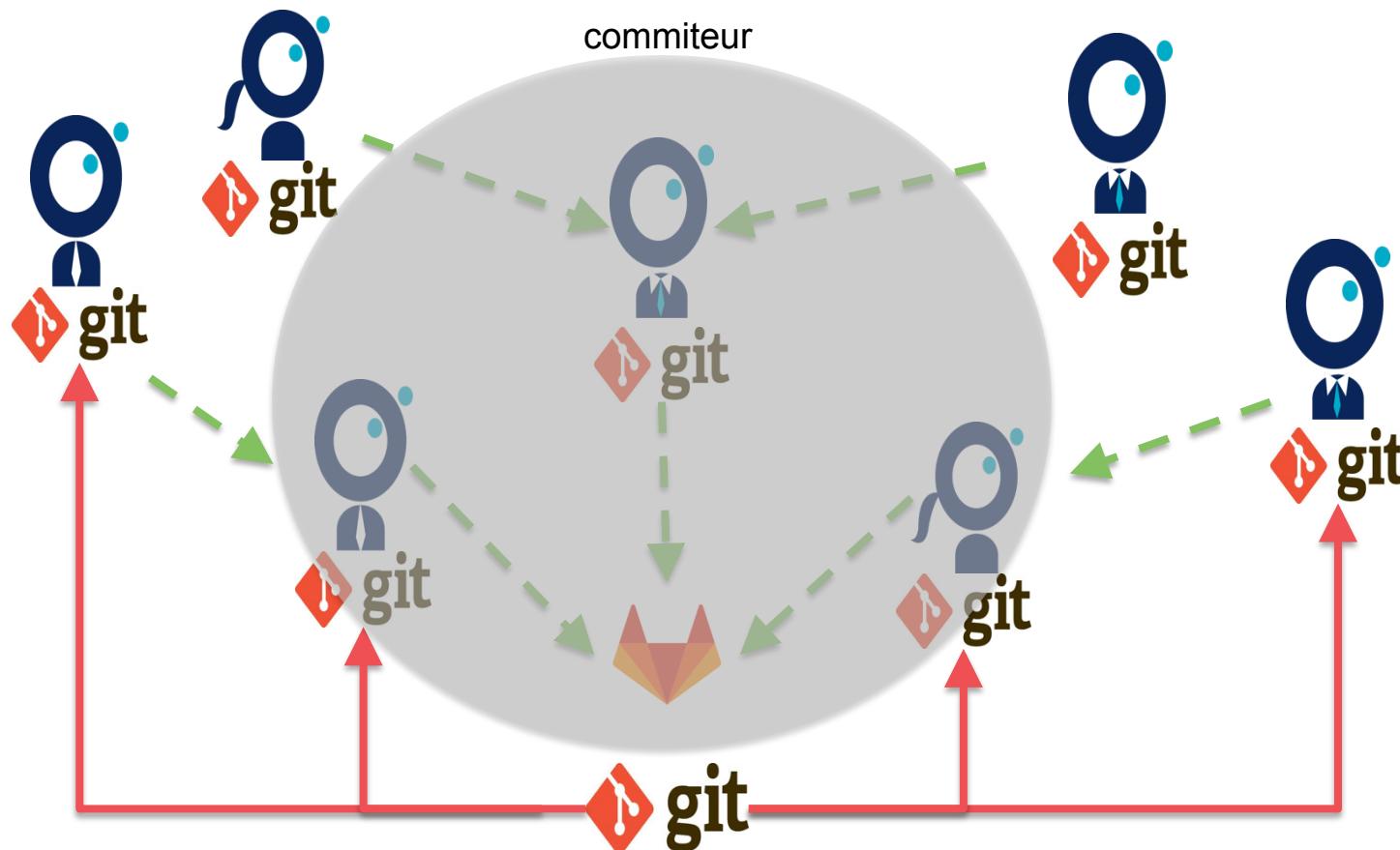
- Dans un VCS type SVN, un fichier peut avoir trois états :
 - Non versionné (untracked dans git)
 - Modifié
 - A jour
- Dans git, il existe 1 autre état dû à une zone intermédiaire appelé index :
 - staged
 - Indique qu'il a été sélectionné pour le prochain commit (git add)



Plusieurs opérations en ligne de commande permettent de manipuler le staging, nous les découvrirons dans les hands-on.



Un modèle décentralisé qui vient de l'open source

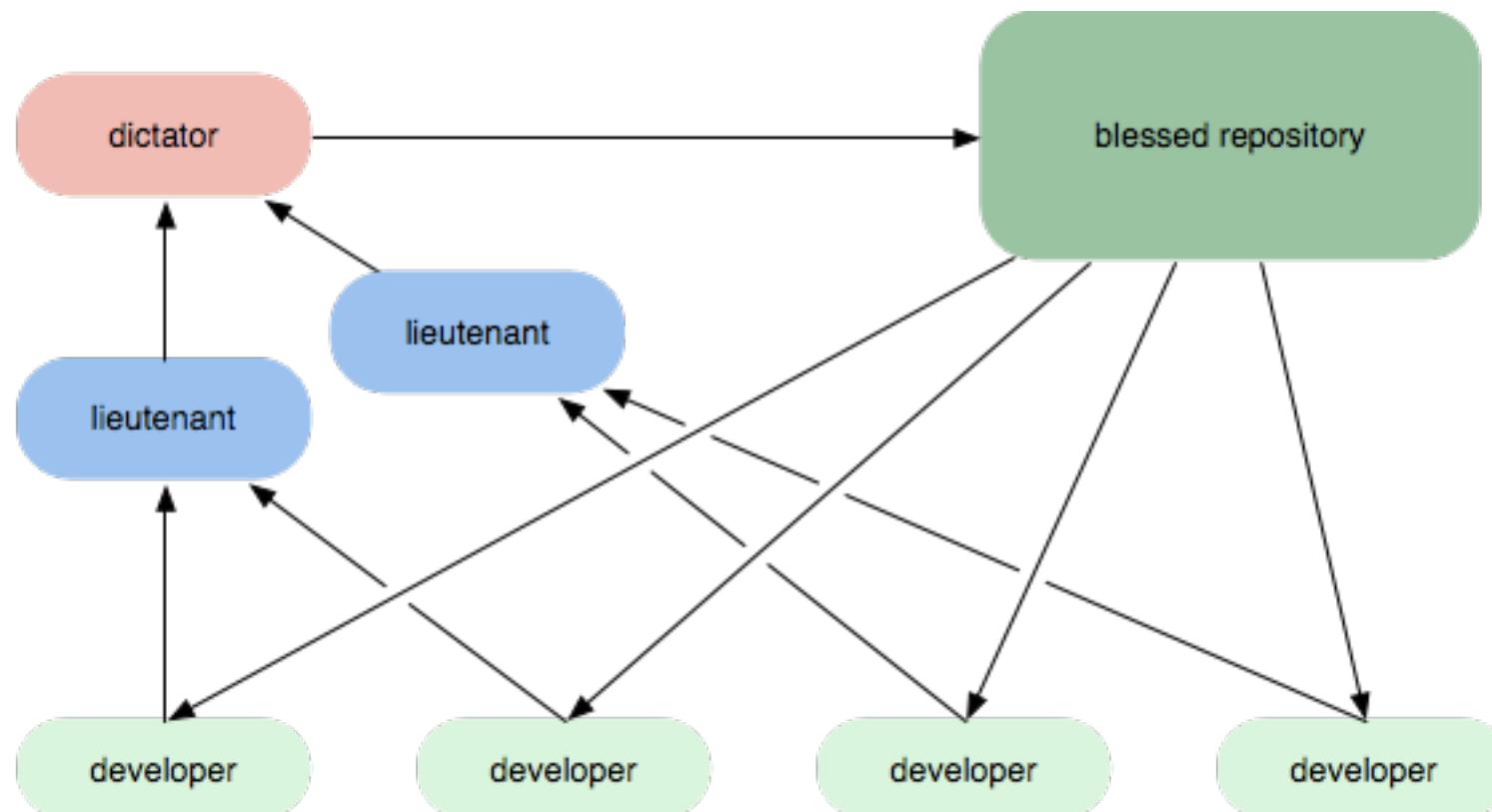


- Le modèle distribué autorise des workflows de travail originaux. Un développeur ne peut pas publier une modification directement sur le master
- En pratique, nous émulons ce workflow au travers de branche et de merge request dans git

- Push
[dashed green arrow icon]
- pull
[red arrow icon]



— Un modèle décentralisé qui vient de Linux



Hands-On

Comprendre les basiques de git



git help

A tout moment, vous pouvez obtenir de l'aide sur une commande git.

- git help commit
- git help push
- git help <command>

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

These are common Git commands used in various situations:

start a working area (see also: git help tutorial)
clone Clone a repository into a new directory
init Create an empty Git repository or reinitialize an existing one

work on the current change (see also: git help everyday)
add Add file contents to the index
mv Move or rename a file, a directory, or a symlink
reset Reset current HEAD to the specified state
rm Remove files from the working tree and from the index
[...]



Commits & Diff

- git commit -m « message » → Commit tout ce qui a été stagé (git add) avec le message message
- git commit -am « message » → Ajoute les changements de tous les fichiers déjà versionnés
- git commit --amend → Ajoute les modifications au dernier commit (attention à ce qu'il n'ait pas déjà été propagé à d'autres repo. où vous aurez un conflit)
- Comparer ce qui a été fait :
 - git diff
 - git diff -> montre ce qui a été modifié depuis le dernier commit mais non stagé~ git diff HEAD
 - git diff --cached -> crée un diff entre le staging et le dernier commit de la branche actuelle
 - git diff --shortstat -> afficher une liste des fichiers modifiés avec un compteur sur les lignes

Hands-on

- > Modifiez le fichier *readme.txt* une première fois en ajoutant du texte à l'intérieur
- > Affichez les changements
- > Créez un fichier *phpinfo.php* avec `<?php phpinfo(); ?>` à l'intérieur
- > Commitez vos changements avec l'option -am (sans utiliser git add)
Que se passe t-il ?
- > commitez le fichier *phpinfo.php* en amendant le dernier commit



Log

- git log
- git log --graph
- git log -p {filename}
 - Voir tous les commits sur un fichier
- git log --stat
 - lister les fichiers et leurs modifs (insertion / suppression)

Hands-on

- > Affichez l'historique de votre repository git.
- > Formatez différemment votre historique git -> git help log
- > Récupérez le hash de votre dernier commit en une ligne

```
| * | commit 4d316e96093bcb65e8a6bc73ef0f8dc925456800
|   | Author: Arnaud MAZIN <amazin@octo.com>
|   | Date: Tue Dec 11 23:18:02 2012 +0100
|   |
|   |     added smtp relay on app-vitrine
|   |
* | | commit 57e1e423d97b60f60510ff64c981dbe7adfb6025
| | | Author: Bertrand Paquet <bpaquet@octo.com>
| | | Date: Wed Dec 12 14:10:18 2012 +0100
| | |
| |     Remove useless file
| |
* | commit f6b0c5dfb2c8baa40abd5dea95912475c1df0f8d
| \\ Merge: 28e07d0 191a2a0
| | | Author: Bertrand Paquet <bpaquet@octo.com>
| | | Date: Wed Dec 12 11:15:39 2012 +0100
| |
|     Merge branch 'master' of git.cloudwatt:cw_deploy
| |
* commit 191a2a00118e7a98c21b662a93c2fb12e9037965
| | Author: Arnaud MAZIN <amazin@octo.com>
| | Date: Tue Dec 11 18:05:22 2012 +0100
| |
|     mails must be writable
```



Log

- git log --pretty=oneline --abbrev-commit --decorate
 - voir un log raccourci avec une ligne par commit
- git log -n1 --format="%h"
 - Voir le short hash du dernier commit
- git diff 50eb85e HEAD
 - voir la différence entre un commit et le HEAD du master

```
22dc9b0 (HEAD -> master) remove myfile  
50eb85e added myfile  
adc8a42 added read me
```

```
22dc9b0
```

```
diff --git a/myfile.txt b/myfile.txt  
deleted file mode 100644  
index e69de29..0000000
```

Hands-on

- > Listez les différences entre 2 commits
- > Affichez le nombre de lignes modifiées et supprimées par fichier et par commit



the log(s) - shortcuts

- HEAD~x
 - > x commits before HEAD
- HEAD@{x}
 - > x operations before HEAD



Alias

- Un alias est un raccourci vers une commande.
- Définition :
 - `~/.bash_profile`
- Syntaxe :
 - `alias name="command"`
 - Exemple :
 - `alias gls="git log --oneline --decorate"`
- Utilisation :
 - `gls`

```
$ gls
5c353dd (HEAD -> master) .gitignore
4c5fd21 add file
```

Hands-on

> Ajoutez un alias nommé 'gls' pour afficher le dernier hash de commit



— Alias Type

```
alias gaa="git add -A"
alias gba="git branch -a"
alias gco='git commit'
alias gcd='git diff --cached'
alias gfp='git fetch --prune'
alias gis='git status'
alias gpo='git push origin'
alias gpom='git push origin master'
alias gl='git log --stat'
alias gls='git log --pretty=oneline --abbrev-commit --decorate'
alias glo='git log --pretty=oneline --abbrev-commit --decor -4'
```



— Reset

- Si je modifie readme.txt et que je veux le réinitialiser à l'état du dernier commit :
 > git checkout readme.txt
- Pour annuler toutes les opérations de staging :
 > git reset
- Pour annuler une opération de staging sur un fichier
 > git reset readme.txt
- Pour revenir à l'état du dernier commit mais garde tous les fichiers non trackés :
 > git reset --hard



Hands-On

Travailler avec un dépôt distant



Clone & Pull

- Cloner un dépôt distant pour l'avoir en local

```
> git clone https://gitlab.com/octoformationgit/repodistant
```

- Voir le dépôt distant qui a été cloné

```
> git remote -v
```

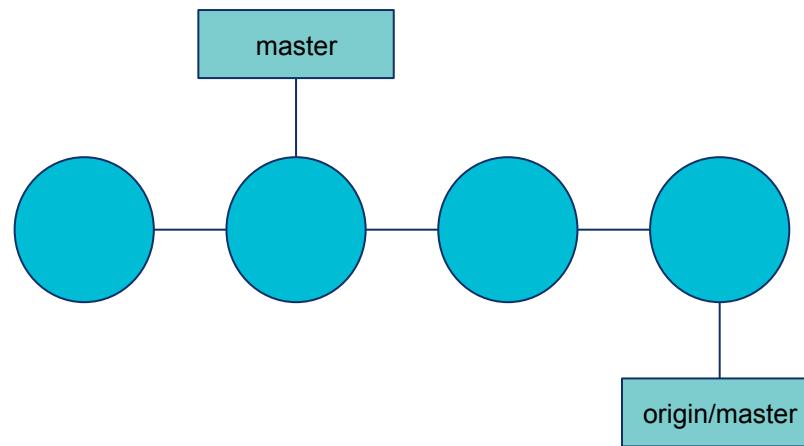
- Mettre à jour votre branche master du dépôt local depuis le dépôt distant (et la merger si nécessaire)

```
> git pull origin master
```

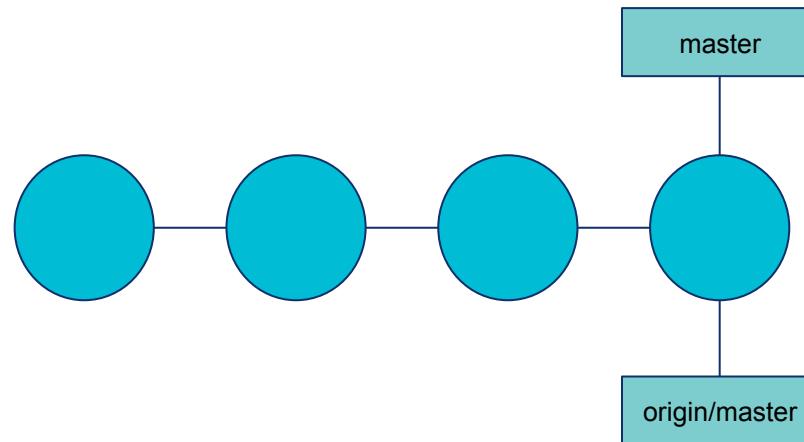
> Quel problème risquez-vous de rencontrer avec `git pull origin master` ?
> Comment récupérer les informations sans effectuer le merge ?



— Pull par l'exemple



```
?> git pull origin master
```



Fetch

- Synchroniser les informations sur le repo. distant

```
> git fetch origin
```

- Voir toutes les branches

```
> git branch -a
```

- Voir le retard de votre branche locale sur la branche remote

```
> git status
```

Quel est l'impact de la commande git fetch sur votre dépôt local ?



Configurez votre compte gitlab

Pour ce challenge, vous aurez besoin de configurer votre clé ssh :

- ssh-keygen
- cat ~/.ssh/id_rsa.pub

The screenshot shows the 'Profile Settings' page of a GitLab account. The top navigation bar includes 'Profile', 'Account', 'Applications', 'Chat', 'Access Tokens', 'Emails', 'Password', 'Notifications', 'SSH Keys' (which is underlined, indicating it's the active tab), 'Preferences', and 'Audit Log'. A search bar and a notifications icon are also present. The 'SSH Keys' section has a heading 'SSH Keys' and a sub-heading 'SSH keys allow you to establish a secure connection between your computer and GitLab.' Below this is a button 'Add an SSH key'. A note says 'Before you can add an SSH key you need to [generate it](#)'. A large text input field is labeled 'Key' with the instruction 'Don't paste the private part of the SSH key. Paste the public part, which is usually contained in the file '~/.ssh/id_rsa.pub' and begins with 'ssh-rsa''. Below the input field is a 'Title' input field and a green 'Add key' button. At the bottom, a section titled 'Your SSH keys (0)' displays a message 'There are no SSH keys with access to your account.'

Lorsque vous accédez à un dépôt authentifié, que ce soit en protocole git://... ou git@..., le système SSH sous-jacent va passer en revue vos clés privées au regard des clés publiques qu'il connaît. Privilégiez ce mode SSH au mode HTTPs



Initier votre propre dépôt gitlab

hands-on

- créez un repo os_corp en local
- créez un projet gitlab os_corp sur un espace gitlab
- enregistrez le repo gitlab os_corp comme origine

```
> git remote add origin git@gitlab.com:{user}/os_corp.git
```

- ajoutez un fichier helloworld.java
- commitez le fichier helloworld.java dans votre repo local
- poussez votre fichier vers le dépôt distant

```
> git push origin master
```



— Remote

- Pour ajouter une référence vers un dépôt remote :

```
> git remote add monrepo git@gitlab.com:xxxx/xxxx.git  
> git remote add monrepo https://gitlab.com/xxxx/xxxx.git
```

- Pour supprimer une référence d'un dépôt remote

```
> git remote remove monrepo
```

> Quel protocole peut utiliser git ? ([réponse](#))



Push

- Transférer les commits d'un dépôt local vers un dépôt distant :

```
> git push <remote> <branch>
```

git vous empêchera de push si vous risquez d'endommager les commits distants (un non-fast-forward merge sur le dépôt distant)

```
Cannot write over Jane's commit  
To git@github.com:xxxx/xxxx.git  
! [rejected]      master -> master (non-fast-forward)  
....
```

- Pour forcer le transfert des commits d'un dépôt local vers un dépôt distant

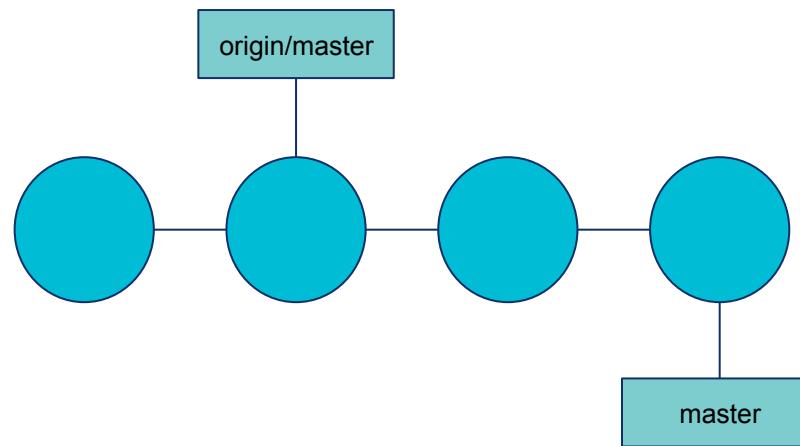
```
> git push --force <remote> <branch>
```

> Dans quel cas ne pas employer cette commande ?
> Dans quel cas l'employer ?

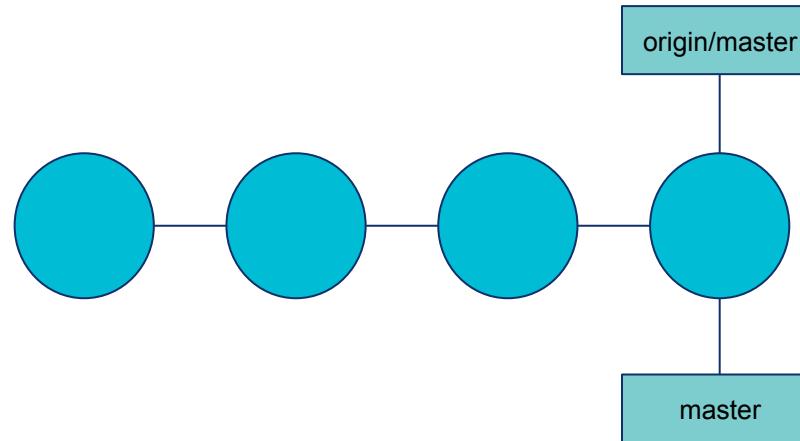
En cas de doute, demander l'avis d'un collègue, n'expérimentez pas sur une branche publique d'un dépôt central !



Push par l'exemple



```
?> git push origin master
```



— Push par l'exemple



Collaborer sur le même repository

- enregistrez vous sur le repository <https://gitlab.com/octoformationgit/page-web-participative>
- clonez le dépôt
- sur le modèle ci-dessous, ajoutez votre nom et prénom
- commitez puis publiez votre code sur le dépôt

At Your Service



Sturdy Templates

Our templates are updated regularly so they don't break.



Ready to Ship

You can use this theme as is, or you can make changes!



Up to Date

We update dependencies to keep things fresh.



Made with Love

You have to make your websites with love these days!



Philippe Kernevez

Developper



Jérôme Van Der Linden

Developper



— Branch

- Pour créer une branche locale :
 - `git branch mabranche`
- Passer votre environnement de travail sur mabranche
 - `git checkout mabranche`
- On fait des modifications, on commit etc. puis pousser sa branche en remote
 - `git push origin mabranche`
- Administrer ses branches
 - montrer les branches locales
 - `git branch`
 - montrer les branches remotess (pensez à fetch avant)
 - `git branch -r`
 - supprimer la branche sur le repo. distant (définitif)
 - `git push origin :mabranche`
 - supprimer la branche mabranche en local
 - `git branch -D mabranche`

> Comment créer une branche et passer dessus en une seule commande ? ([réponse](#))
> Pourquoi je ne peux pas supprimer la branche master sur gitlab ? ([réponse](#))



Travailler en bonne intelligence

- adresse du repository <https://gitlab.com/octoformationgit/page-web-participative>
- créer une branche de travail add_name-{user}
 - > git branch add_name-{user}
 - > git checkout add_name-{user}
- sur le modèle ci-dessous, ajoutez votre nom et prénom

At Your Service



Sturdy Templates
Our templates are updated regularly so they don't break.



Ready to Ship
You can use this theme as is, or you can make changes!



Up to Date
We update dependencies to keep things fresh.



Made with Love
You have to make your websites with love these days!



Philippe Kernevez
Développer



Jérôme Van Der Linden
Développer

- commitez et pushez votre branche de travail add_name-{user}
 - > git push origin add_name-{user}
- créez une merge request sur le gitlab



Merge & Conflits

- Par défaut git est excellent en merge et il y a beaucoup moins de conflits qu'avec SVN
- Néanmoins ça arrive... (cas de deux développeurs qui bossent en même temps sur le même fichier)
- Pas de panique !

```
here is my readme  
<<<<<< HEAD  
the cake is a lie.  
=====  
the cake is telling the truth!  
>>>>>  
4e76d3542a7eee02ec516a47600002a90a4e4b48
```

The diagram illustrates a merge operation. On the right, the text "Votre fichier" is positioned above a blue arrow pointing towards the center. On the left, the text "Fichier distant" is positioned above another blue arrow pointing towards the center. The central area contains a conflict resolution, showing the保留本地更改 (keep local changes) and丢弃本地更改 (drop local changes) markers, along with the combined text from both sources.

- Modifiez les fichiers en conflit, supprimez les marqueurs de merge
- Puis commitez -les...
- C'est mergé !



Hands-On

git stash, blame



— Quand utiliser git stash ?

- Vous souhaitez mettre votre travail de côté pour travailler sur une autre branche et le reprendre plus tard.
- J'ai un environnement de travail propre avec un commit initial.
- git stash est le bon candidat pour conserver du travail temporaire, et évite :
 - > git commit -m "my temp feature"
 - > git commit –amend
- *Peut-être remplacé avantageusement par un branche locale temporaire*



git stash

- git stash
 - mettre son espace de staging de côté
- git stash list
 - voir l'ensemble des stashes sauvées
- git stash apply
 - restaurer un stash
- git stash apply stash@{0}
 - pour appliquer uniquement le stash 0
- git stash drop
 - supprimer le dernier stash
- git stash pop
 - correspond à apply+drop

Saved working directory and index
state WIP on master: 77c3d0e first
commit

HEAD is now at 77c3d0e first commit

stash@{0}: WIP on master: 77c3d0e
first commit

stash@{1}: WIP on master: 77c3d0e
first commit

Hands-on

- > Créez un nouveau dépôt en local
- > Ajoutez un fichier “stashme.txt” à votre repo. et l’ajoute au staging
- > Faire un premier commit
- > Ajoutez un autre fichier “stashmeagain.txt” à votre repo. et le stasher.
- > Ajoutez un autre fichier “stashmeagain2.txt” à votre repo. et le stasher.
- > Listez les stashes, puis appliquez le 1er stash.
- > Comment créer une branche avec le 2ème stash ?



Git blame

- Permet de connaître l'auteur et le commit de chaque ligne d'un fichier

Hands-on

- Cloner <https://gitlab.com/octoformationgit/page-web-participative-rebase>
- Aller sur la branche `feat-new_button-farcellier`
 - `git co feat-new_button-farcellier`
- Constater l'origine des lignes (L78)
 - `git blame index.html`

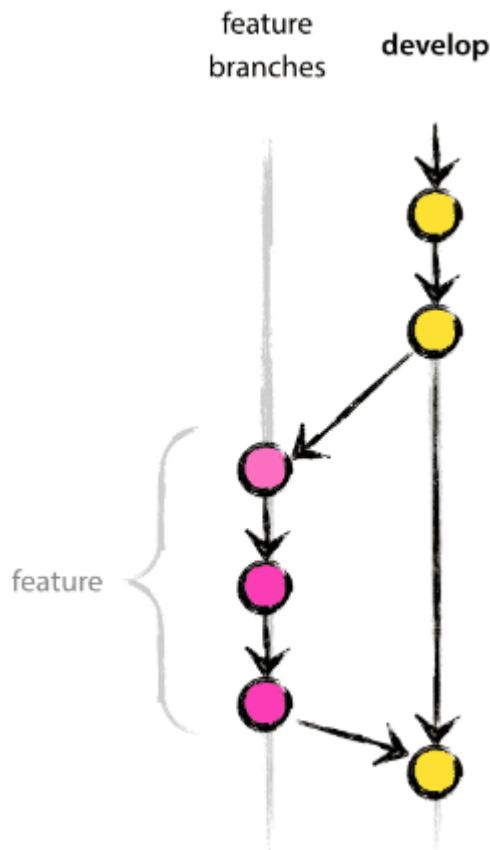


Hands-On

Utiliser un workflow



Feature branch - au quotidien



- créer une branche de travail
 > `git checkout -b ma_branche`
- faire vos commits régulièrement
- synchroniser votre branche régulièrement
 > `git pull --rebase origin master`
- pousser vers gitlab régulièrement
 > `git push origin ma_branche`

Si vous avez ce message sur votre branche lors d'un push
(après un pull –rebase)

```
Cannot write over Jane's commit
To git@github.com:mikrob/testgit.git
! [rejected]      ma_branche -> ma_branche (non-fast-
forward)
....
```

```
git push -f origin ma_branche
```



Feature branch

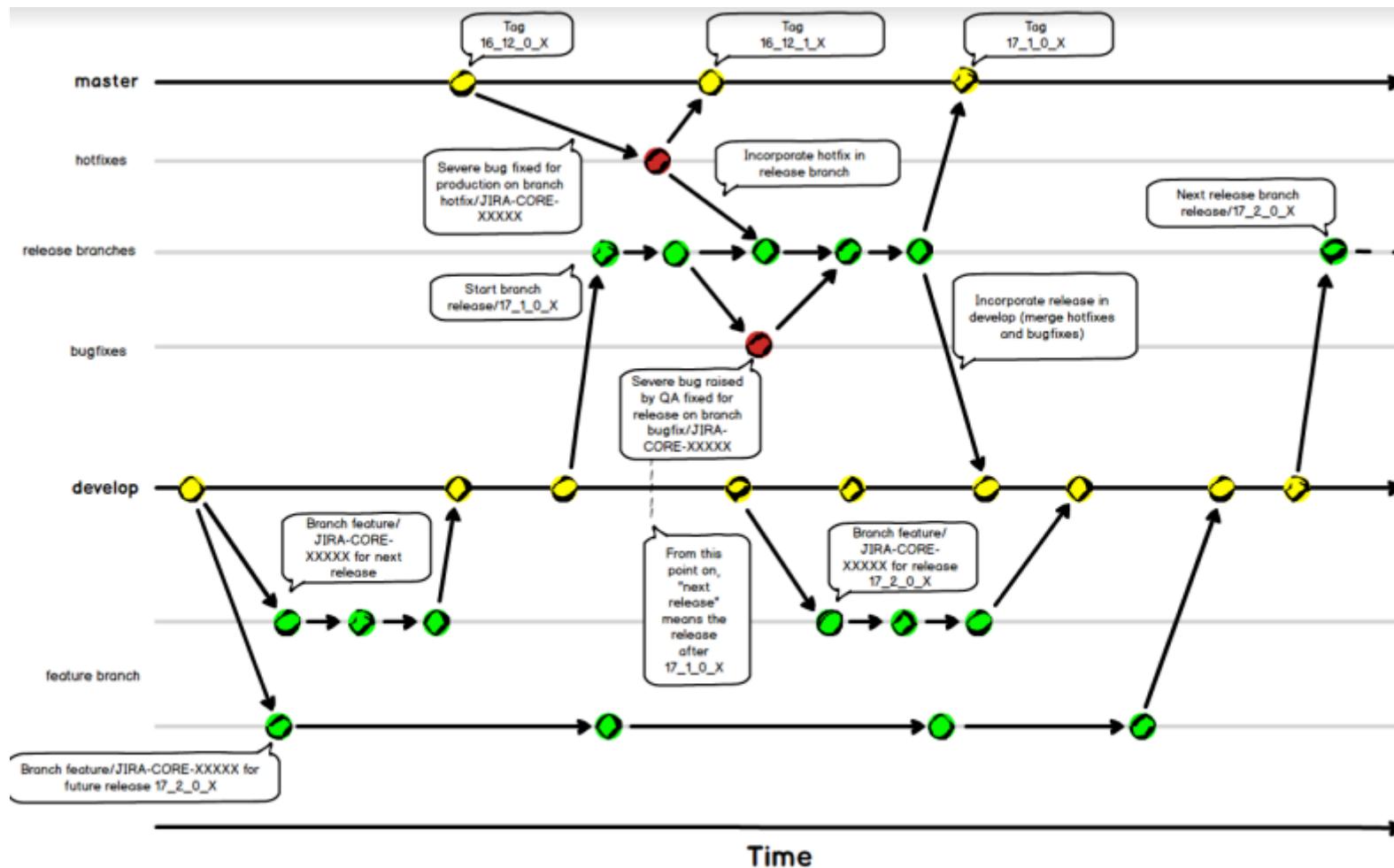
- Impact d'utilisation de Merge Request (-no-ff) combiné avec des features branches 'légères' :
 - > Les commits sont doublés dans l'historique



?	Nicholas C. Za...	8740984	Merge pull request #303 from ilyavolodin/eslint_errors
?	Ilya Volodin	477b420	Fixing ESLint errors
?	Nicholas C. Za...	03090fd	Merge pull request #297 from ilyavolodin/upgrade-escape
?	Ilya Volodin	7913cbe	Upgrading escape version and fixing related bugs
?	Nicholas C. Za...	4b42681	Merge pull request #299 from ilyavolodin/no-unused-vars
?	Ilya Volodin	0cfb18b	Fixing assignment during initialization issue



Comprendre le workflow de GitFlow

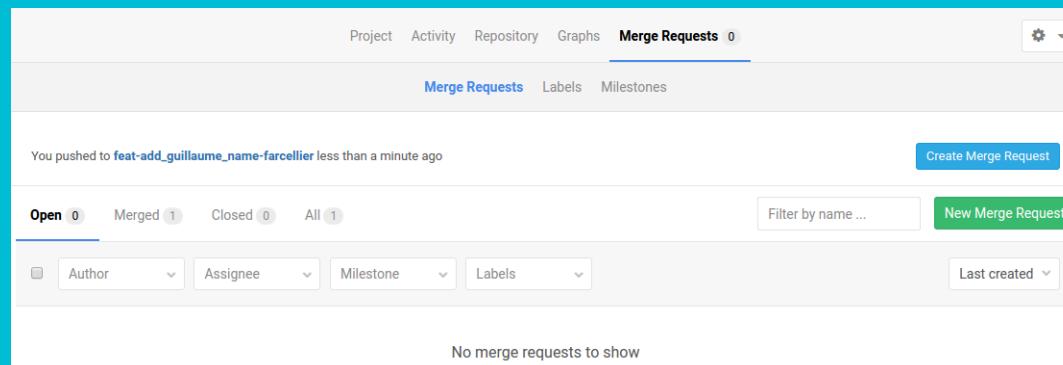


Code Review sur une merge request / pull request

Hands-on

Pour ce hands-on, vous aurez besoin d'un ami.

- > Forkez le dépôt <https://gitlab.com/octoformationgit/page-web-participative>
- > Invitez votre ami sur votre repo en tant que "master"
- > Soumettez une merge request de *feat-add_guillaume_name* vers *master* (*dans votre propre dépôt, pas dans celui d'origine que vous avez forké*)



- > Assignez là à votre ami
- > Votre ami met un commentaire sur la ligne mal indenté et demande la correction
- > Corrigez l'indentation, republier votre branche en la forçant si vous avez amendé le commit
- > Votre ami valide et merge la branche



Hands-On

Ré-écrire l'histoire...



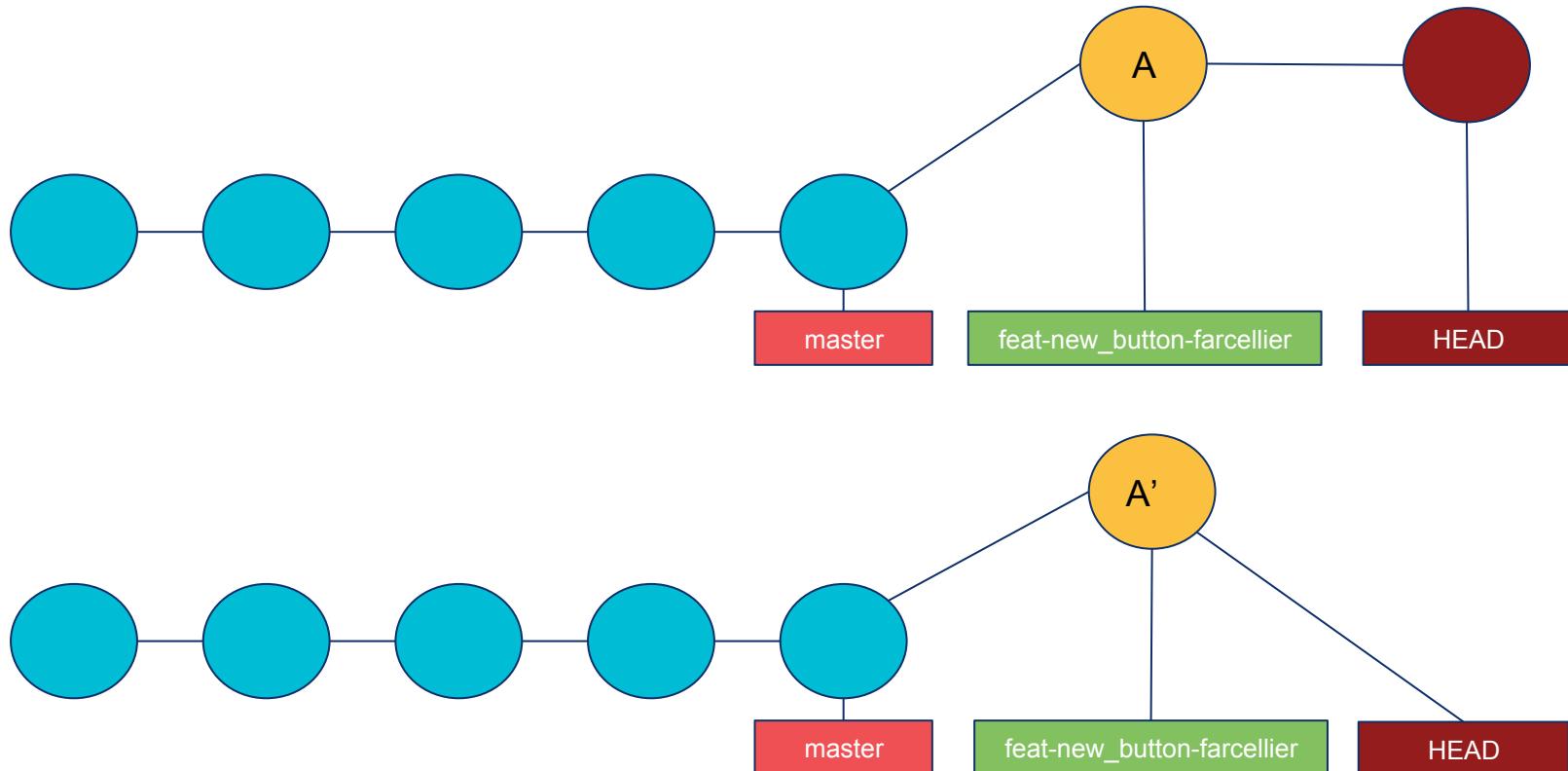
— Refaire son historique

Hands-on

- > Forkez le dépôt <https://gitlab.com/octoformationgit/page-web-participative>
- > Faites 2 commits sur des fichiers quelconque



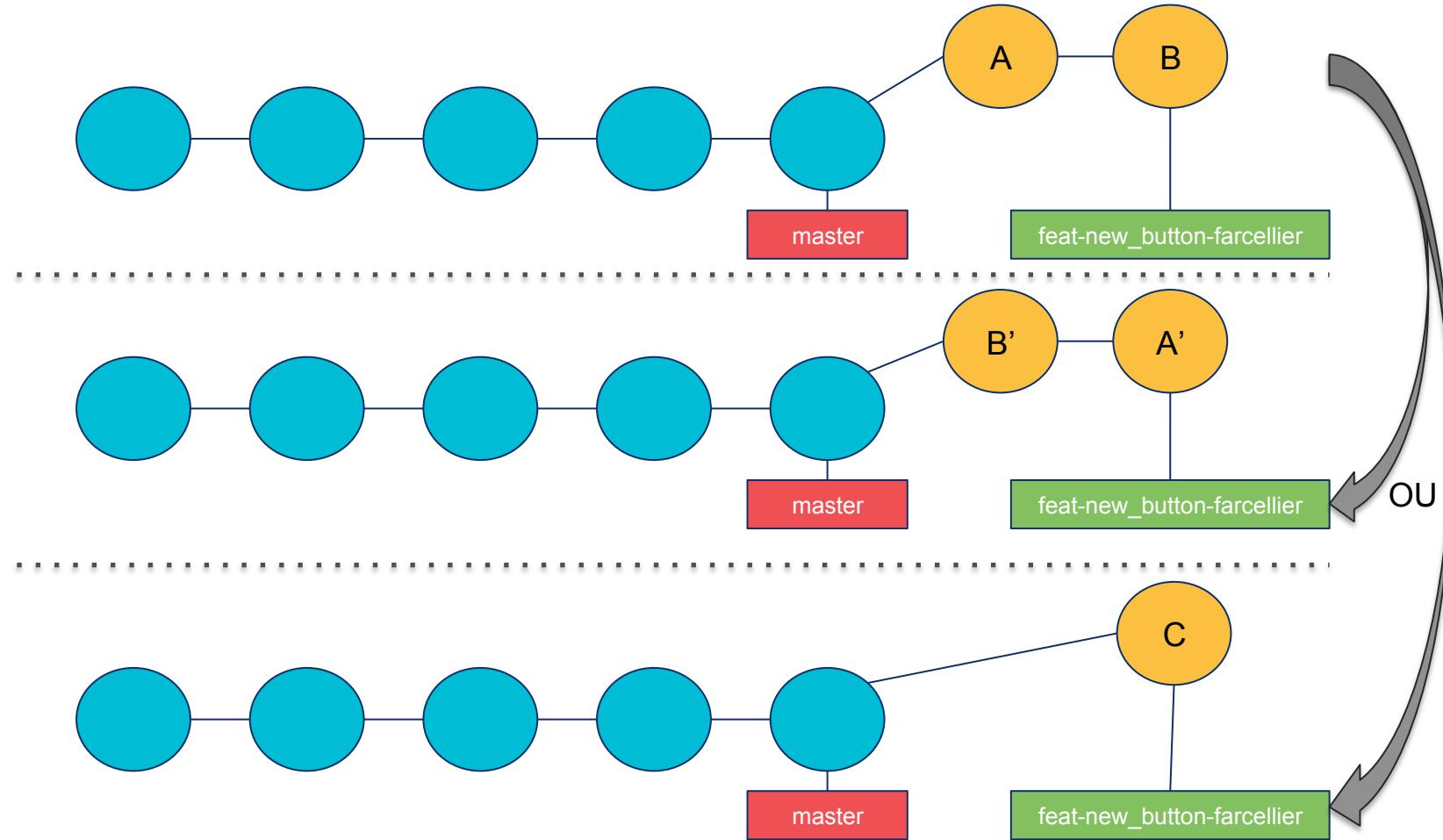
— Refaire son historique : ‘git amend’



Hands-on

- 'Amendez' le dernier commit avec votre index
 - `git commit --amend`

Refaire son historique : Rebase interactif



Hands-on

> Fusionnez les 2 derniers commits avec un rebase interactif
`git rebase -i {short-hash-du-commit-master}`

Comment récupérer une erreur locale (pas encore ‘pushée’)

- En ré-écrivant l'histoire : git reflog

```
pke:~/S/2/D/F/a/page-web-participative-rebase-conflict[m 4<2 [feat-new_button-farcellier] 2>6> git reflog
f2243ef (HEAD -> feat-new_button-farcellier) HEAD@{0}: rebase finished: returning to refs/heads/feat-new_button-farcellier
f2243ef (HEAD -> feat-new_button-farcellier) HEAD@{1}: rebase: feat: the marketing request we rename the button
3f4ccad HEAD@{2}: rebase: feat: the marketing requests we add a button to send an email
a64ed59 (origin/master, origin/HEAD, master) HEAD@{3}: pull --rebase origin master: checkout a64ed59add1507f238698e92ca7969bc4d2106b3
200637f (origin/feat-new_button-farcellier) HEAD@{4}: checkout: moving from master to feat-new_button-farcellier
a64ed59 (origin/master, origin/HEAD, master) HEAD@{5}: clone: from https://gitlab.com/farcellier/page-web-participative-rebase-conflict
```

Hands-on

- Restaurez l'état avant le rebase interactif : retrouvez l'état précédent dans les logs
git reflog
- Remettez la branche sur ce commit
git reset --hard XXXX



— A cet après-midi...

