

Démarrer avec GIT

Vincent Guigue – vincent.guigue@agroparistech.fr

Slides directement tirés des travaux d'Isabelle Blasquez

Les enjeux généraux

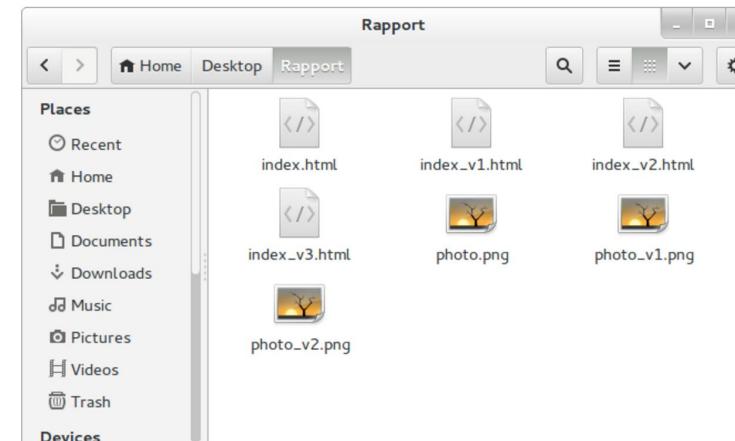
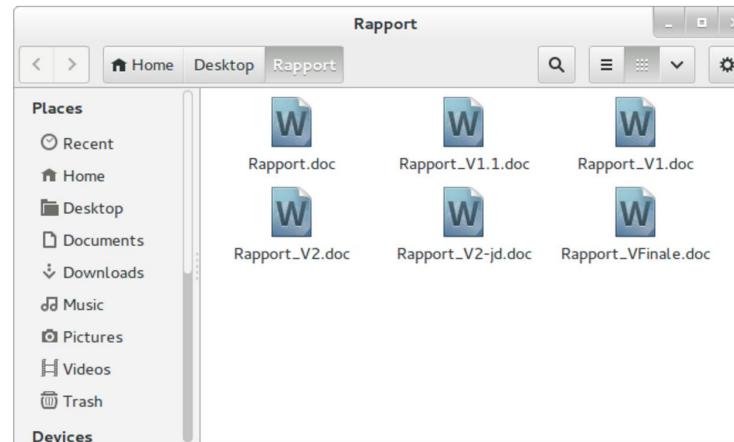
- Ne pas refaire ce qui a déjà été fait
- Travailler incrémentalement, mieux gérer les projets
- Travailler à plusieurs
 - Note:
codage + 1 semaine de break => vous êtes un étranger dans votre projet !
- Mettre à disposition votre travail pour les autres
 - + site web en option

Les enjeux *data*

- 90': on partage des articles et des algorithmes
 - On partage aussi des données, à envoyer par la poste ☺
- 2000': on partage des bouts de codes, des exécutables
 - UCI: base de jeux de données publiques
- 2010': la reproductibilité des résultats devient un enjeu majeur
 - Partage des données
 - Partage des codes (github)
 - **Partage des modèles entraînés**
- 2020': la reproductibilité est une obligation scientifique
 - Construire un repository git est impératif pour toute publication scientifique

Git = gestionnaire de version décentralisé

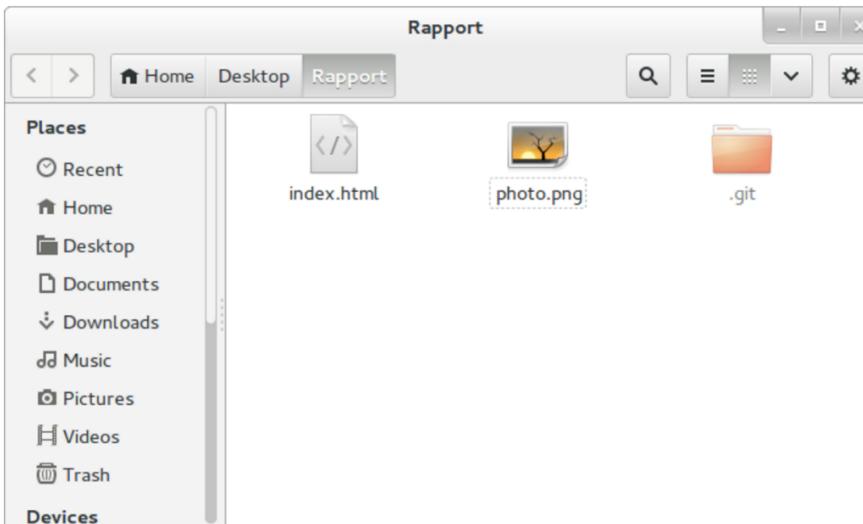
**Quelle est la dernière version ?
Quel est l'ordre des versions ?**



Gestionnaires de Versions à la rescousse !

Un logiciel de gestion de versions (ou **VCS** : **Version Control System**) est un logiciel qui permet de stocker un ensemble de fichiers en conservant la chronologie de toutes les modifications qui ont été effectuées dessus. Il permet notamment de retrouver les différentes versions d'un lot de fichiers connexes (**extrait Wikipedia**)

Bon exemple



Avantages de la gestion de versions

Faciliter la sauvegarde et le travail collaboratif en :

- disposant d'un **historique** sur les changements apportés
- permettant facilement le **retour en arrière**
- **Partageant** les changements
(documenter (messages de commit), fusionner,
récupérer, visualiser les modifications apportées,...)

Note: Le répertoire .git est un répertoire caché, qui contient tout l'historique des fichiers.

Historique des Gestionnaires de Version

✓ **Gestion de versions centralisé** (*un seul dépôt fait référence*)



(1986 : le premier : **Concurrent Versioning System**)



(2000 : **SVN** lancé pour remplacer CVS,
a suscité un réel intérêt pour les VCS,
le plus populaire avant l'arrivée des DVCS...)

✓ **Gestion de versions décentralisé** (*Distributed Version Contrôl System*)



(2005 : développé et optimisé pour le noyau Linux par Linus Thorval
⇒ **le plus populaire** ...)



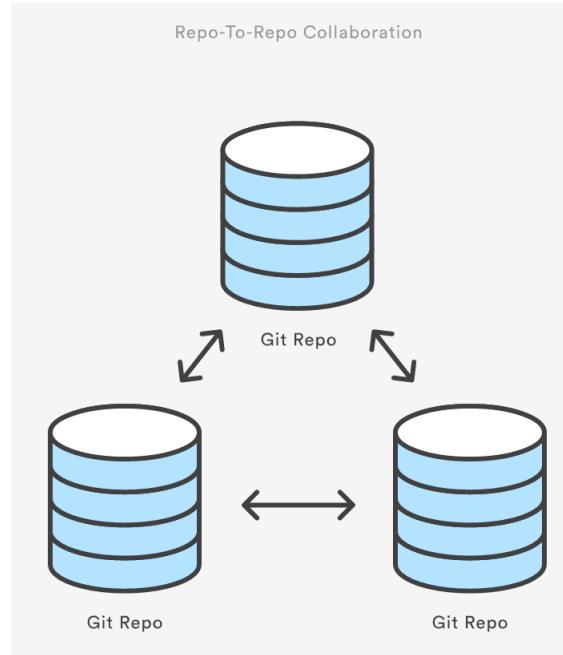
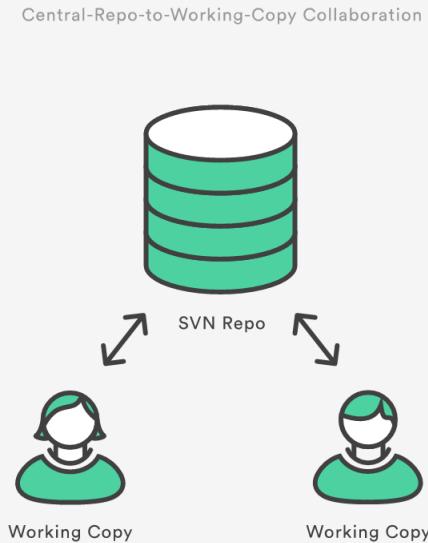
(2005)



(2008)

Plusieurs dépôts pour un même projet
⇒ Travail en local possible **sans accès réseau!**

Gestion Centralisée vs Gestion Décentralisée (pour la collaboration : copie de travail vs dépôt)



Joel Spolsky ([Stack Overflow](#), [Trello](#)) décrit la gestion de version **décentralisée** comme « probablement la plus grande avancée dans les technologies de développement logiciel dans ces années »

A lire : <https://www.joelonsoftware.com/2010/03/17/distributed-version-control-is-here-to-stay-baby>

Git ne fait aucune distinction entre la copie de travail et le dépôt centralisé.

Ce sont des **dépôts Git à part entière**.

Pourquoi préférer un DVCS ...

Avantages de la gestion décentralisée :

- **Ne pas être dépendant d'une seule machine** comme point de défaillance
- Travailleur **sans être connecté** au gestionnaire de version
- **Pas de permissions particulières** pour participer à un projet
(les droits de commit/soumission peuvent être donnés après avoir démontré son travail et non pas avant)
- **Opérations plus rapides** pour la plupart car réalisées en local (sans accès réseau)
- **Travail privé** (réalisation de brouillons sans devoir publier ses modifications et gêner les autres contributeurs)
- ... Avec **un dépôt de référence contenant les versions livrées d'un projet.**

Inconvénients :

- **cloner un dépôt est plus long** que récupérer une version car tout l'historique est copié
(ce qui est toutefois un avantage par la suite)
- **il n'y a pas de système de lock**
(ce qui peut poser des problèmes pour des données binaires qui ne se fusionnent pas).

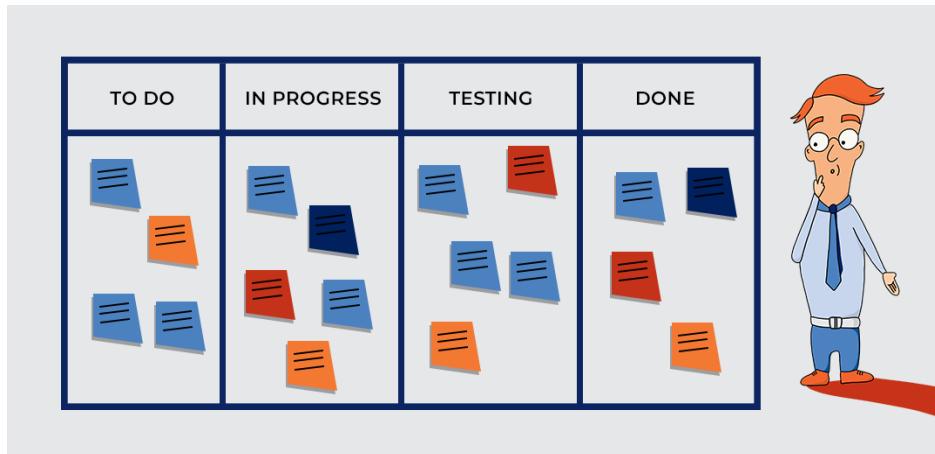
Nouvelle(s) méthodes de
développement

Méthodes agiles



Trouver une approche + adaptative que les cycles en V

- Listes de tâches
 - Indépendantes / parallélisable
 - Synchro = Réunions régulières + rapides
- Revues sur les tâches
- Travail collaboratif ++ / MAJ ++
 - Besoin d'outils
- Définitions de phases (e.g. sprint) et de roles (e.g. product owner)



RAD, Scrum, Kanban

Installation de l'outil de base

Installation

- ✓ **Installation:**
<https://git-scm.com/downloads>

Downloads



Older releases are available and the **Git source repository** is on GitHub.

```
$ git --help
usage: git [--version] [--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>]
           [-c name=value] [-c help]
           <command> [<args>]

The most commonly used git commands are:
add           Add file contents to the index
bisect        Find by binary search the change that introduced a bug
branch       List, create, or delete branches
checkout     Checkout a branch or paths to the working tree
clone        Clone a repository into a new directory
commit       Record changes to the repository
diff          Show changes between commits, commit and working tree, etc
fetch        Download objects and refs from another repository
grep          Print lines matching a pattern
init          Create an empty git repository or reinitialize an existing one
log           Show commit logs
merge        Join two or more development histories together
mv            Move or rename a file, a directory, or a symlink
pull          Fetch from and merge with another repository or a local branch
push          Update remote refs along with associated objects
rebase       Forward-port local commits to the updated upstream head
reset        Reset current HEAD to the specified state
rm            Remove files from the working tree and from the index
show          Show various types of objects
status       Show the working tree status
tag           Create, list, delete or verify a tag object signed with GPG
```

See 'git help <command>' for more information on a specific command.

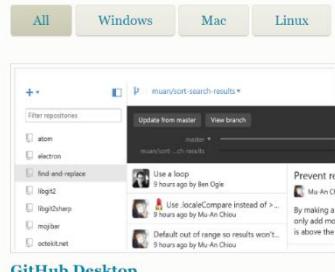
✓ Utilisation de git :

- **dans un terminal en ligne de commande**
(**git --help** : la commande à retenir !)

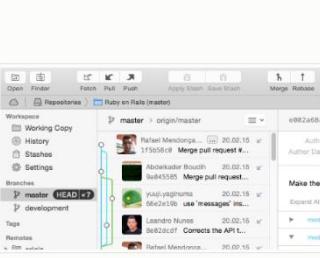
→ via un outil graphique

GUI Clients

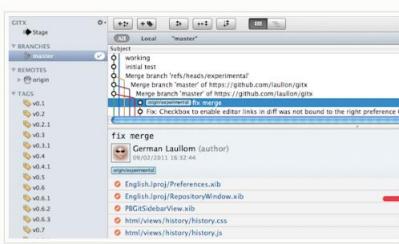
Git comes with built-in GUI tools for committing ([git-gui](#)) and browsing ([gitorious](#)), but there are several third-party tools for users looking for platform-specific experience.



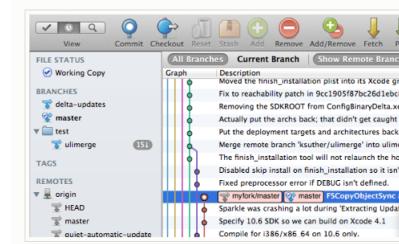
GitHub Desktop
Platforms: Windows, Mac
Price: Free



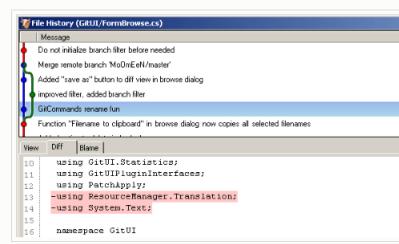
Tower
Platforms: Windows, Mac
Price: \$79/user (Free 30 day trial)



GitX-dev
Platforms: Mac
Price: Free



SourceTree
Platforms: Mac, Windows
Price: Free



Git Extensions
Platforms: Windows
Price: Free

... et bien d'autres sur <https://git-scm.com/downloads/guis> et
https://git.wiki.kernel.org/index.php/InterfacesFrontendsAndTools#Graphical_Interfaces

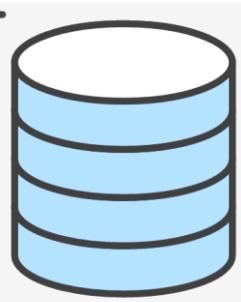
Les commandes de base

git + commande + arguments

On va aller vite sur la théorie...

... Et vous allez ensuite revenir sur tous ces aspects par la pratique

Le dépôt au cœur de Git (ou repository)



Créer un nouveau dépôt :

`git init`

(dans le répertoire courant)

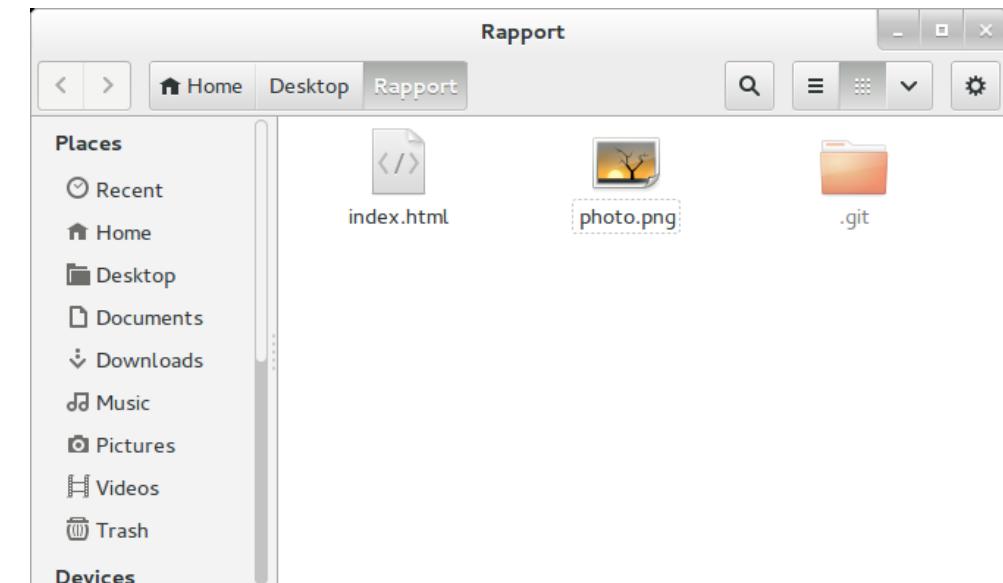
`git init <directory>`

Cloner un dépôt existant :

`git clone <repo>`

(dans le répertoire courant)

`git clone <repo> <directory>`



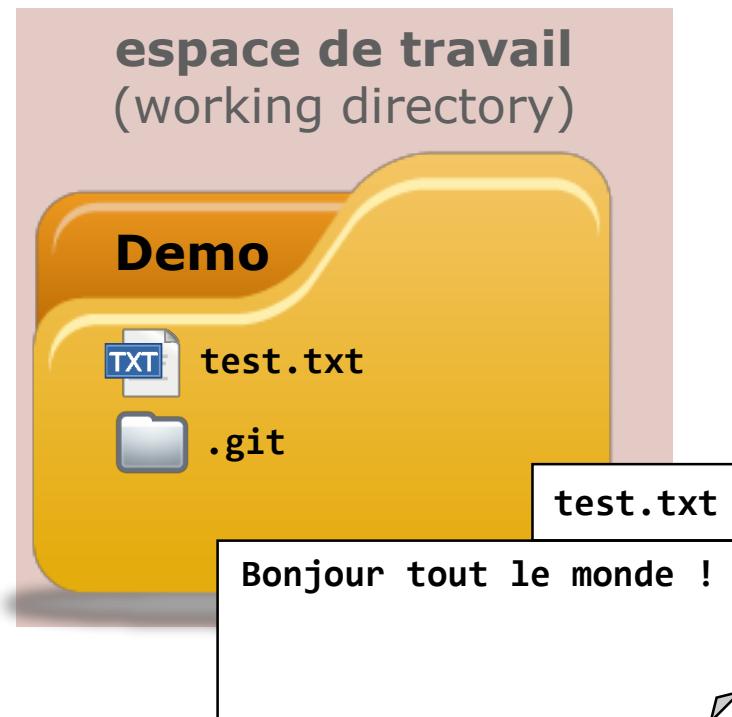
Dépôt : Un dépôt est un dossier qui porte le nom **.git**.

C'est là que Git va stocker l'historique du projet c-a-d toutes les informations en rapport avec votre projet comme la liste des commits effectués, la liste des branches, etc...

(En savoir plus sur le contenu réel de ce dépôt :

https://git-scm.com/book/fr/v2/Les-tripes-de-Git-Plomberie-et-porcelaine#_git_internals)

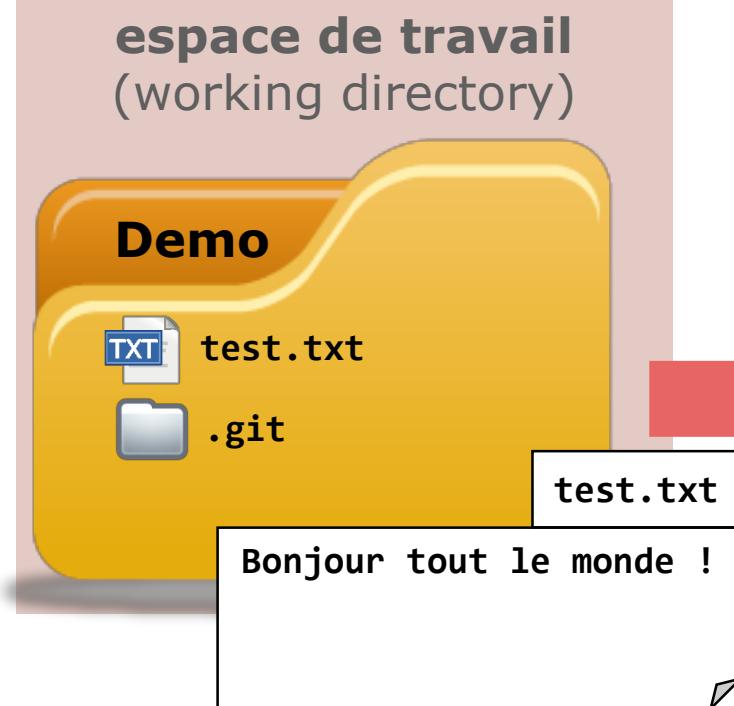
Un répertoire avec son fichier prêt à être versionné



git init

Ajout d'un fichier dans l'Index (zone en attente de commit)

git add test.txt



status

Liste, entre autres, les fichiers stagés (dans l'index en attente d'être commité), non stagés

```
$ git status
On branch master
Initial commit
Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

    new file:   test.txt
```



ls-files

Liste les fichiers dans l'index et leur hash associé (signature en SHA-1)

```
$ git ls-files --stage
100644 306c18deac1991cddc759c58a9098f3cb2eb602f 0      test.txt
```

Implication du *add*

- Certains fichiers ne sont pas partagés
 - Brouillons personnels
 - Essais non concluants
 - Corrections d'exercice ☺

Committer (le contenu de l'Index) dans le dépôt local

committer :

enregistrer un ensemble
de modifications

git commit

+ Message obligatoire
qui va être demandé

git commit -m "message"

log

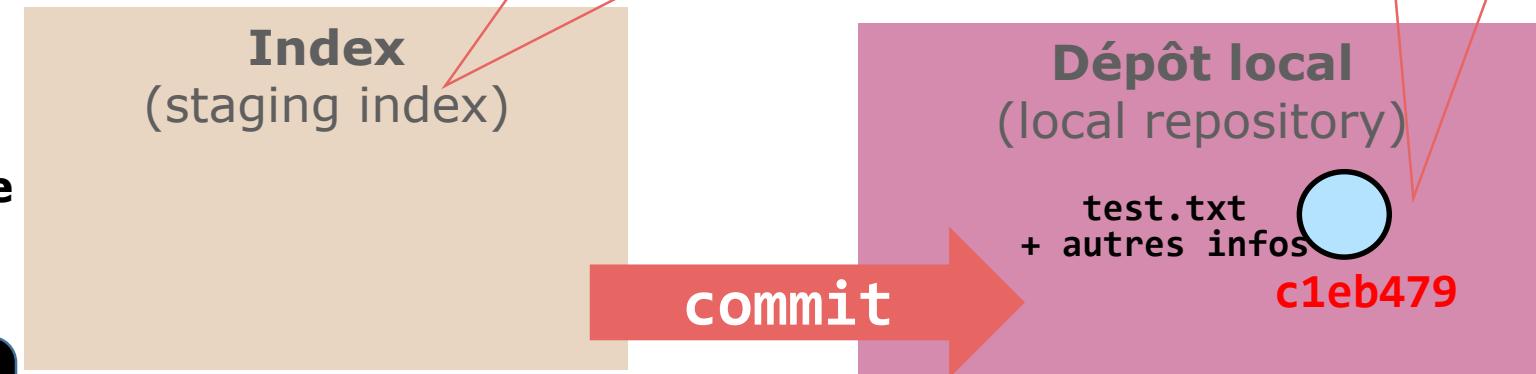
Liste des commits
(dans le dépôt local)

```
$ git log  
commit c1eb479524b7352604b2cf156d73d757b735f175  
Author: Isabelle BLASQUEZ <isabelle.bl[REDACTED]>  
Date:   Fri Feb 17 09:16:52 2017 +0100  
  
        Premier Commit
```

show

Détail d'un commit
(du dépôt local)

```
$ git show  
commit c1eb479524b7352604b2cf156d73d757b735f175  
Author: Isabelle BLASQUEZ <isabelle.bl[REDACTED]>  
Date:   Fri Feb 17 09:16:52 2017 +0100  
  
        Premier Commit  
  
diff --git a/test.txt b/test.txt  
new file mode 100644  
index 000000..306c18d  
--- /dev/null  
+++ b/test.txt  
@@ -0,0 +1 @@  
+Bonjour tout le monde !  
\ No newline at end of file
```

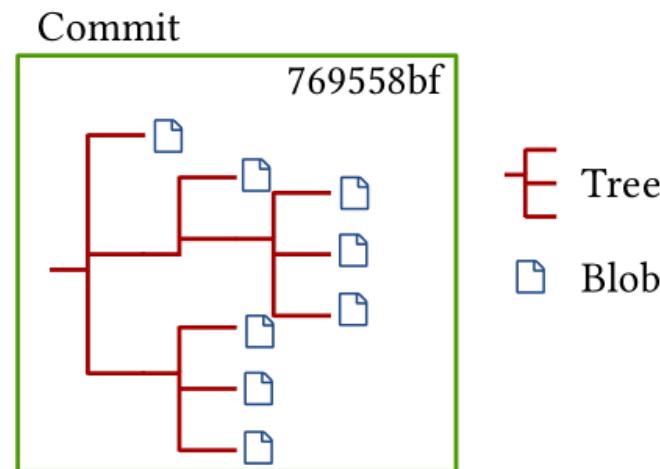


```
$ git commit -m "Premier Commit"  
[master (root-commit) c1eb479] Premier Commit  
1 file changed, 1 insertion(+)  
create mode 100644 test.txt
```

A propos du commit

Pour identifier une version, git s'appuie sur l'objet de type **commit**.

Un **commit** est l'unité de sauvegarder de git.
C'est une **image du répertoire de travail à l'instant t**
représentée à l'aide de **trees** et de **blobs**



Chaque objet est identifié
40 caractères hexadécimaux
(**somme de contrôle SHA-1** de son contenu)

Chaque commit est répertorié avec son **auteur**, sa **date**, son **message** et son **SHA-1**
ainsi que des **pointeurs vers un(des) commit(s) parent(s)**.

Parenthèse sur les controles de version

Exemples [[modifier](#) | [modifier le code](#)]

Voici la signature obtenue sur une phrase (codée en utf8) :

`SHA1("Wikipédia, l'encyclopédie libre et gratuite") = 6153A6FA0E4880D9B8D0BE4720F78E895265D0A9.`

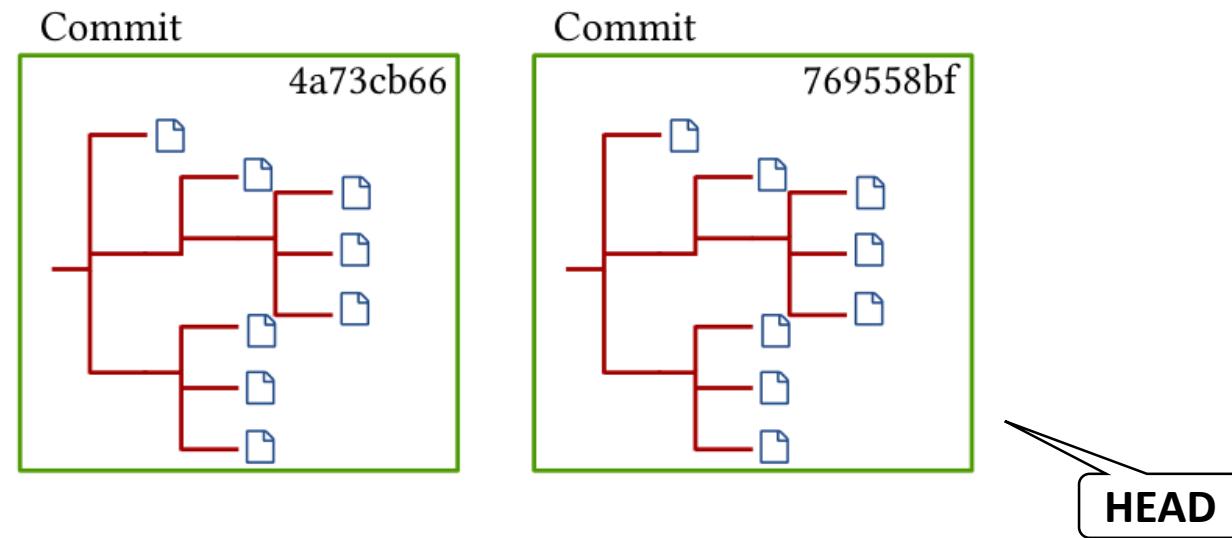
En modifiant un caractère, la signature change radicalement :

`SHA1("Wikipédia, l'encyclopédie libre et gratuitE") = 11F453355B28E1158D4E516A2D3EDF96B3450406.`

- Valider le contenu d'un fichier avec une clé
 - Vérifier l'intégrité
 - (Lien avec la cryptographie et les fonctions de hashage)

A propos du dépôt Git

Un dépôt Git peut être vu comme une collection d'objets liés entre eux.

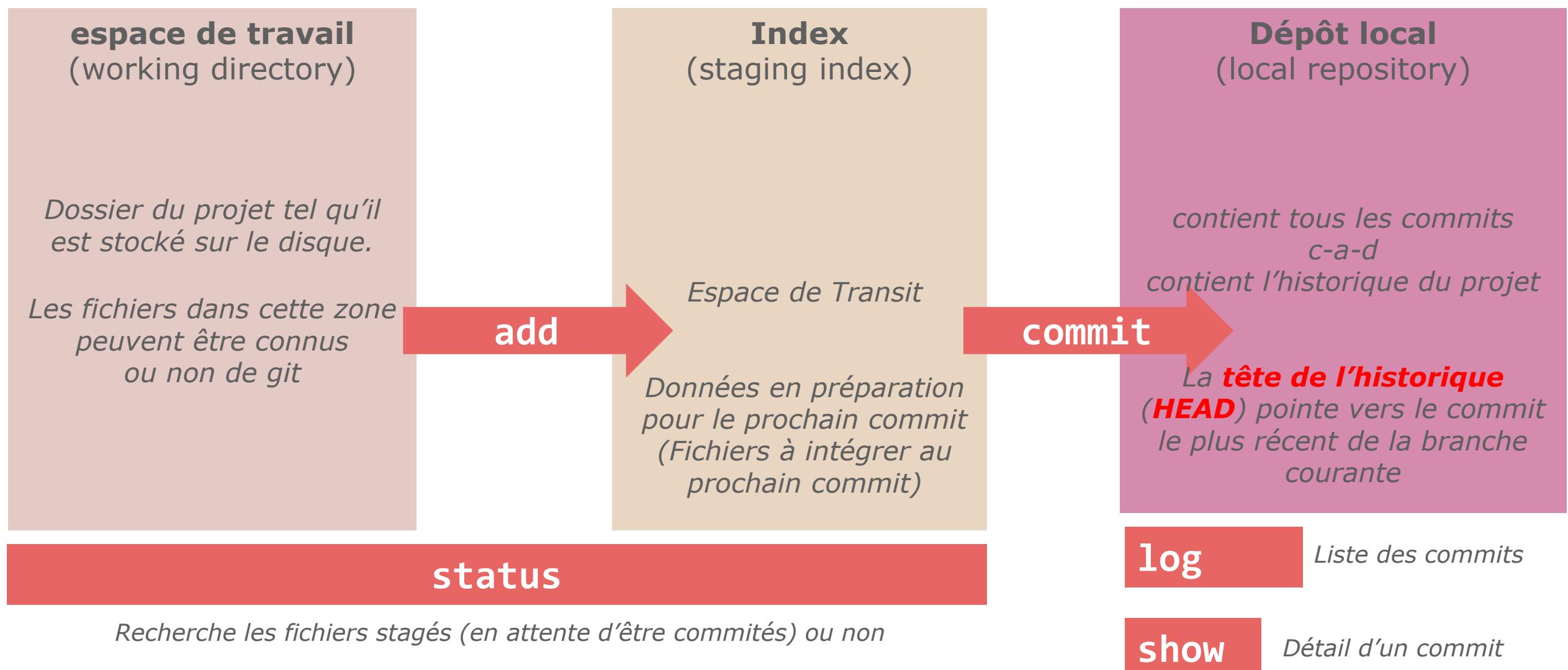


L'historique d'un dépôt Git est l'ensemble des versions du répertoire de travail (succession de commits)

La tête (HEAD) de l'historique est un pointeur vers le dernier commit (utilisé comme prochain commit parent)

En résumé : les espaces de travail

Git utilise **3 espaces différents** pour manipuler les données.



Enregistrer de nouvelles modifications dans un nouveau commit (en une seule ligne)



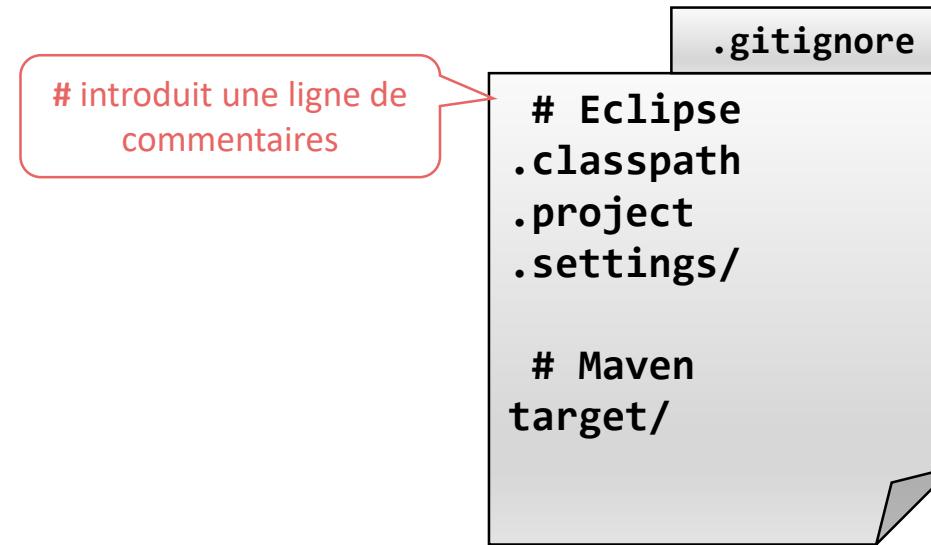
Ignorer des fichiers dans l'historique (.gitignore)

Certains **fichiers n'ont pas besoin d'être versionnés**, et ne doivent **pas être trackés**

(*fichiers de log, fichiers résultants d'une compilation, fichiers de configuration, ...*)

... à spécifier dans un fichier **.gitignore** à la racine du dépôt.

(*non trackés par git, ils ne seront pas détectés comme ayant subi un changement (modified) et potentiellement commitable*)



(Exemple pour un projet Maven sous Eclipse)

Quelques liens à consulter pour personnaliser son **.gitignore** :

https://git-scm.com/docs/gitignore#_pattern_format , <https://github.com/github/gitignore>

<http://gary-rowe.com/agilestack/2012/10/12/a-gitignore-file-for-intellij-and-eclipse-with-maven> , <https://fr.atlassian.com/git/tutorials/gitignore>

Isabelle BLASQUEZ

Historique des versions

espace de travail (working directory)

git status

Liste les fichiers stagés, non stagés et non trackés

git log

Dépôt local (local repository)

Affiche l'ensemble de l'historique (tous les commits)

git log -n <limite>

*Affiche les derniers n commits
où n=<limite>*

*git log-n-3
pour les 3 derniers commits*

git log -oneline

*Affichage de l'historique
où chaque commit est affiché sur une seule ligne*

git log -author="nom auteur"

*Affichage uniquement les commits
filtrés avec l'auteur spécifié*

git log <fichier>

*Affichage uniquement des commits
Contenant le fichier spécifié*

...

Visualiser les différences (git diff)

git diff commit1..commit 2

différences entre deux commits

(Exemple : modifications introduites par
le dernier commit)

```
$ git diff c1eb479..HEAD
diff --git a/test.txt b/test.txt
index 306c18d..9e77514 100644
--- a/test.txt
+++ b/test.txt
@@ -1 +1,2 @@
-Bonjour tout le monde !
\ No newline at end of file
+Bonjour tout le monde !
+J'utilise git !
\ No newline at end of file
```

git diff

différences entre espace de travail et index

git diff --cached

différences entre l'index et le HEAD

git diff HEAD

différences entre espace de travail et le HEAD

...

Se positionner sur un commit donné (git diff)

`git checkout commitparSonSHA`

Consulter (afficher) une ancienne version, sans toucher à votre espace de travail

Le checkout d'un commit transforme l'espace de travail pour afficher un ancien état de votre projet sans modifier son état actuel.

L'historique n'est pas modifié...

... ainsi pour retrouver l'état actuel de l'espace de travail, il suffit de ...

`git checkout master`

Revenir au commit le plus récent de la branche principale

Possible aussi pour un fichier en particulier :

`git checkout commitparSonSHA <fichier>`

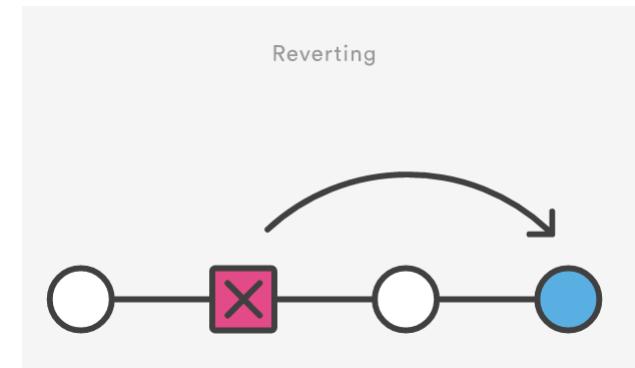
... suivi de ...

`git checkout HEAD <fichier>`

J'ai fait mon commit un peu trop vite... est-ce que je peux l'annuler ?

`git revert commitparSonSHA`

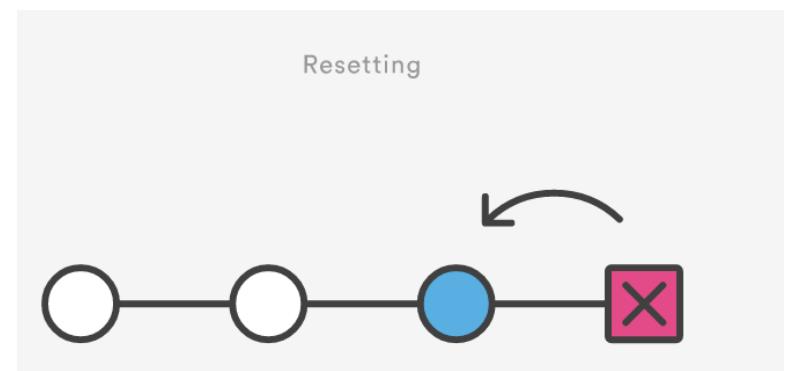
Revenir à un état antérieur en créant un nouveau nouveau commit (défait en créant un nouveau commit, mais ne revient pas en arrière à proprement parlé...)



`git reset HEAD`

Annuler les changements dans l'index

(remet l'index dans le même état que la version la plus récente du dépôt)



`git reset --hard HEAD`

Annuler les changements de l'espace de travail et de l'index

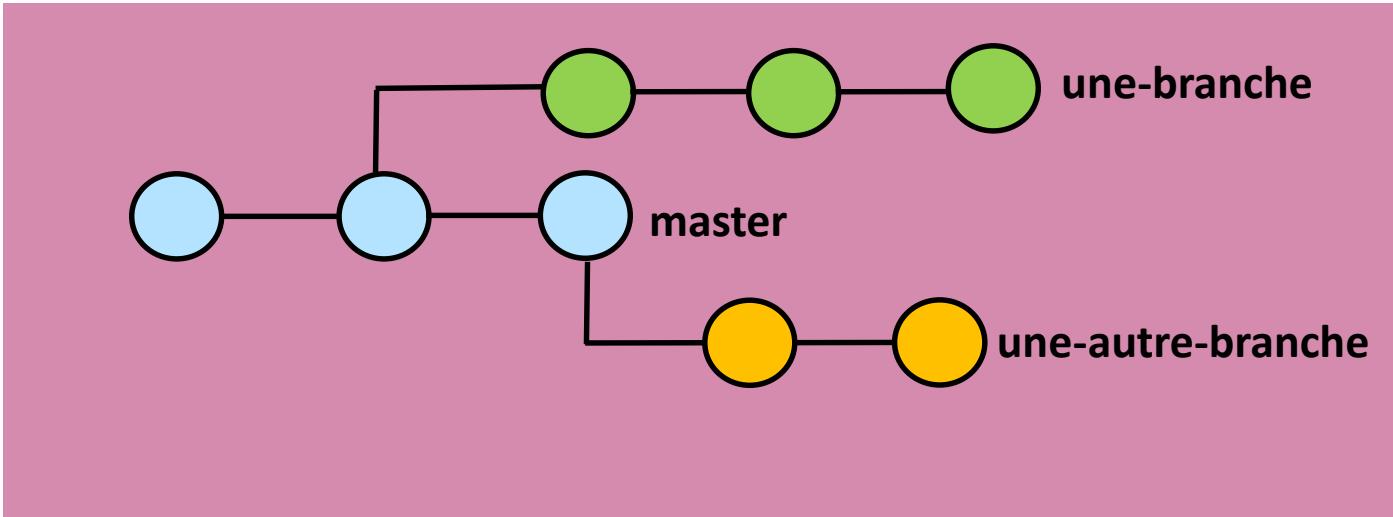
(Attention, **commande irréversible !!!**)

`git commit --amend -m "Votre nouveau message"`

Modifier le message du commit (possible uniquement si commit pas non pushé sur dépôt distant (origin))

Créer des branches

A propos des branches

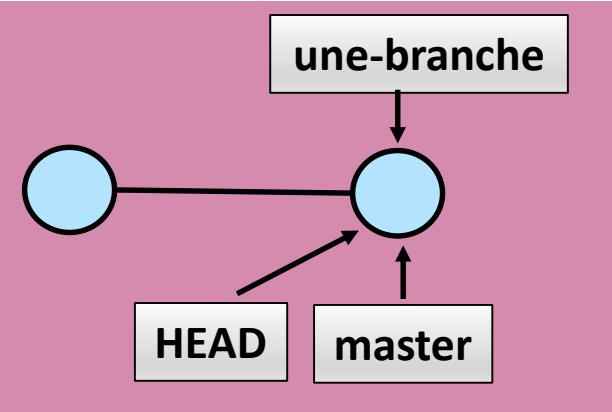


La branche par défaut est `master`

Une branche correspond à une version en parallèle de celle en cours de développement.

Une branche peut servir à développer de nouvelles fonctionnalités, à corriger des bugs sans pour autant intégrer ces modifications à la version principale du logiciel

Créer une branche et en faire la branche courante



```
git branch une-branche
```

Créer une branche

Un nouveau pointeur (`une-branche`) est pour l'instant simplement créé sur le commit courant.

L'historique ne change pas

Nom de branches : ne pas commencer par un tiret, ni deux points consécutifs, ne pas terminer par un slash



```
git checkout une-branche
```

Se positionner sur la branche fraîchement créée pour qu'elle devienne la branche courante (celle qui recevra le prochain commit)

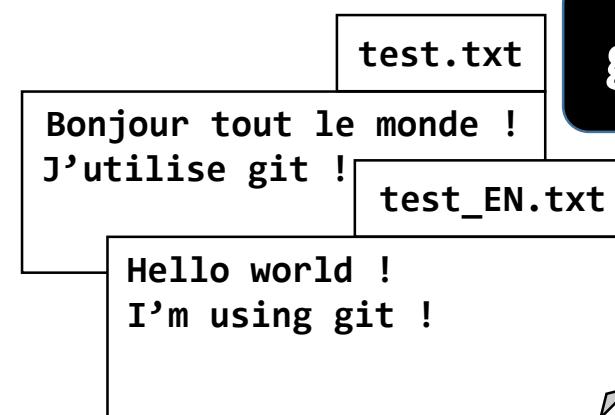
Ou en une seule ligne de commande :

```
git checkout -b une-branche
```

Que se passe-t-il lorsqu'on change de branche ?

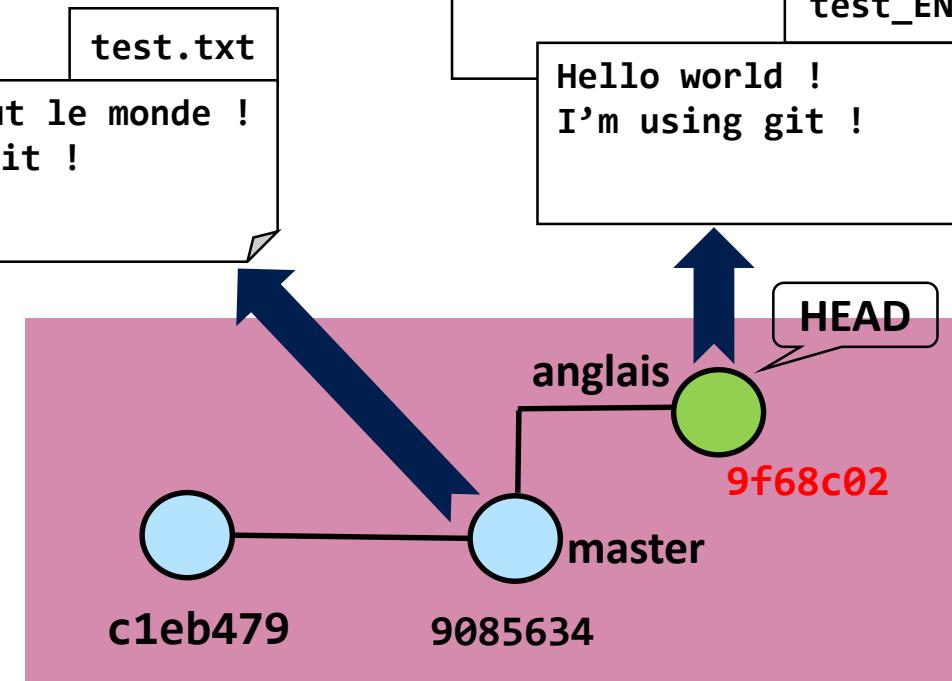
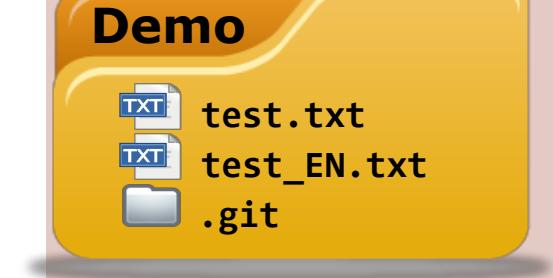
git checkout master

espace de travail



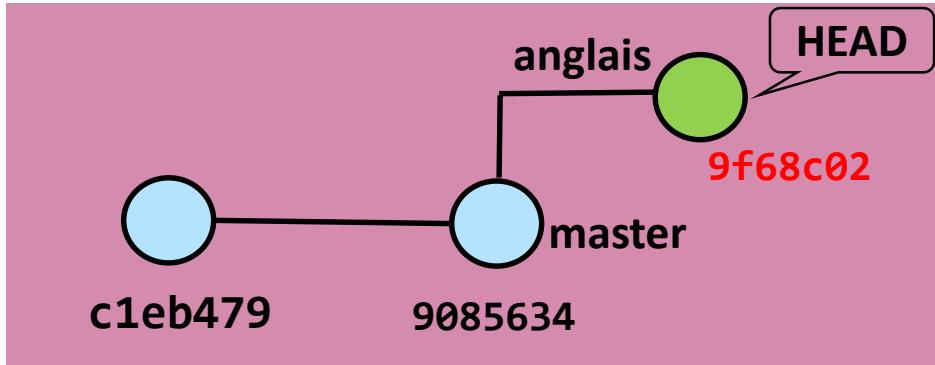
git checkout anglais

espace de travail

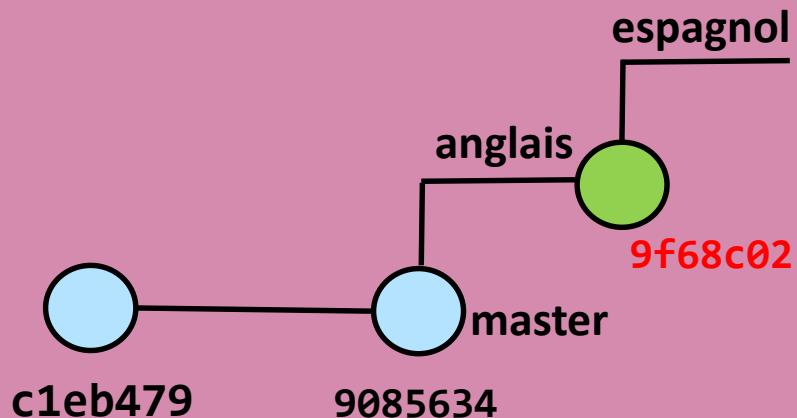


- Le répertoire de travail est modifié
- L'index reste intact
- La référence HEAD est mise à jour sur le commit le plus récent de la branche choisie (nouvelle branche courante)

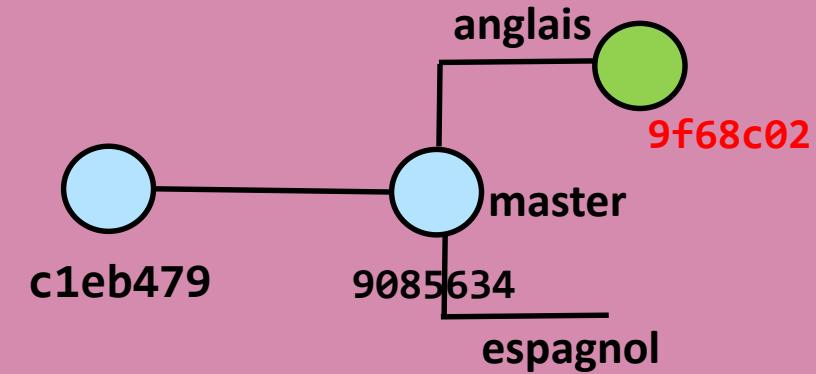
Créer (tirer) une autre branche ...



La branche est tirée depuis la branche courante ...



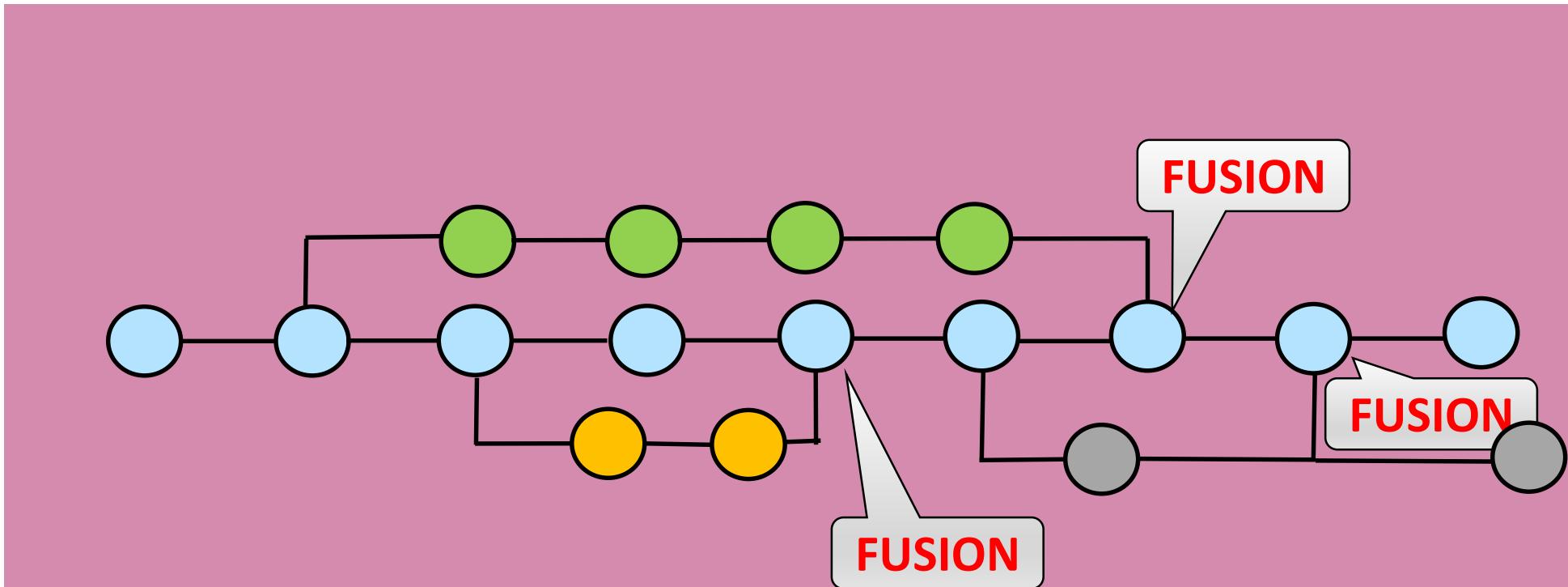
```
git branch espagnol
```



```
git checkout master
```

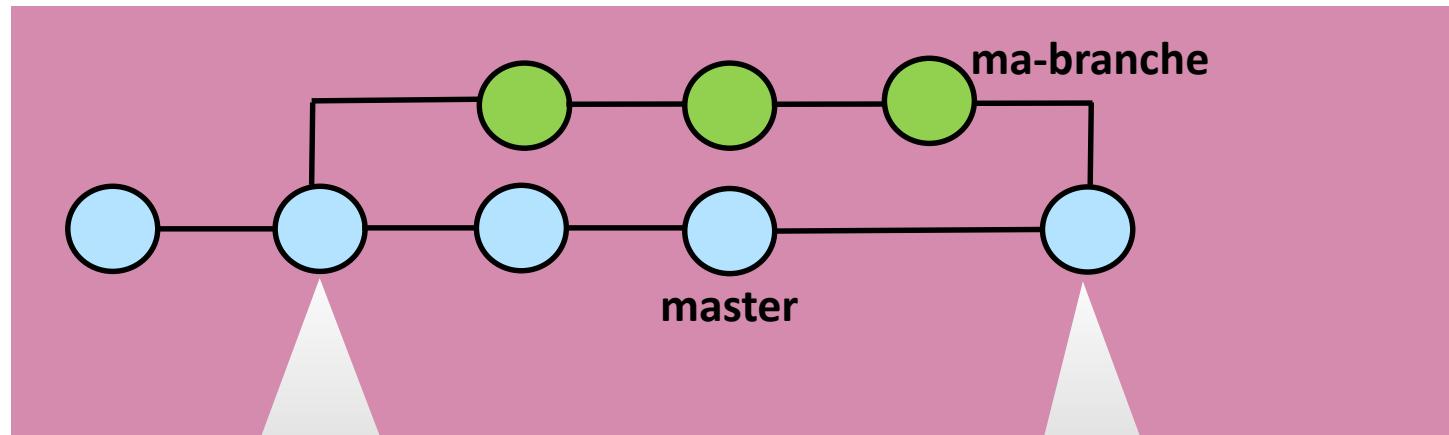
```
git branch espagnol
```

Fusionner des branches



Intégrer les modifications faites sur une branche dans une autre branche.
(Récupération et intégration des différences d'une branche dans une autre)

Fusionner une branche : Principe général (fusion avec commit)



Si on est sur master
et
qu'on souhaite fusionner
ma-branche dans master :

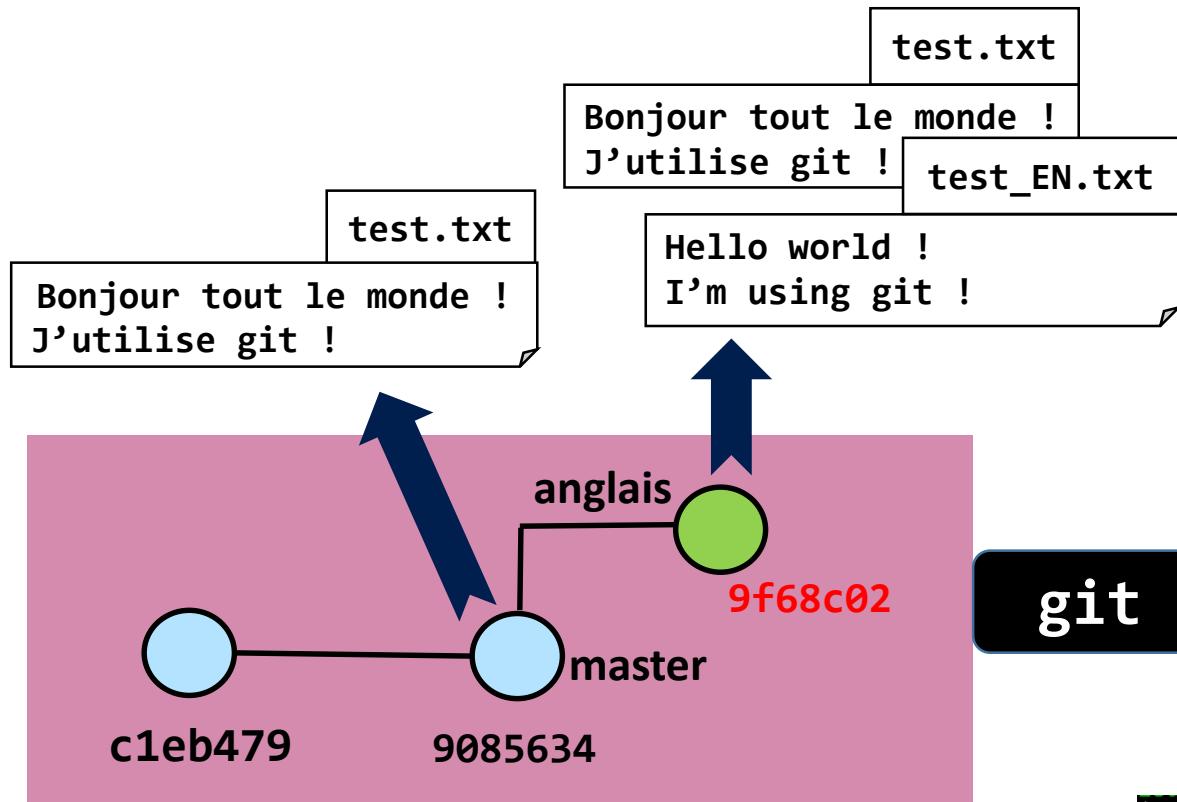
`git merge ma-branche`

→ git recherche le dernier commit en commun (**commit « ancêtre commun »**)

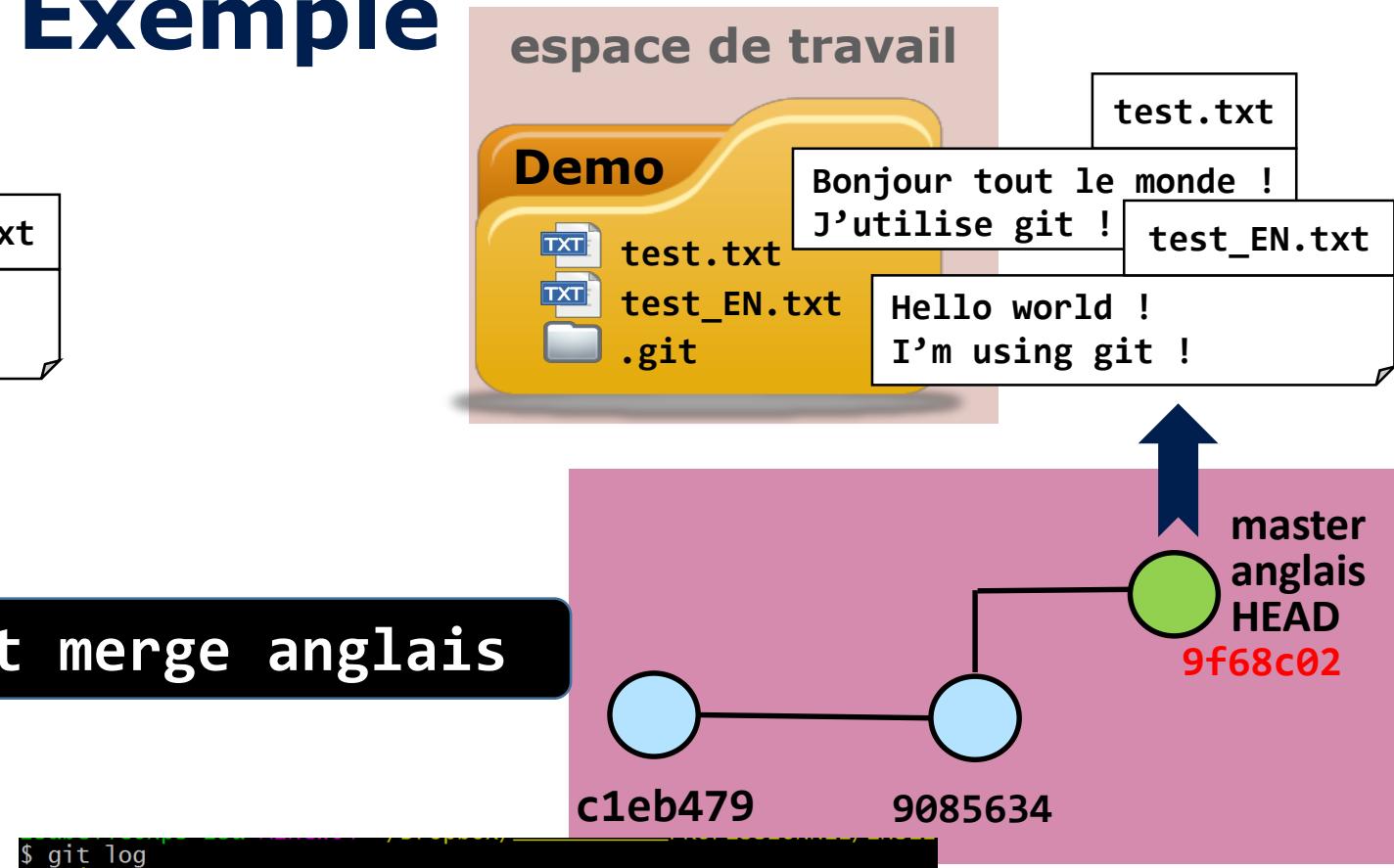
→ git crée un nouveau commit sur master qui contiendra les modifications apportées par ma-branche (**commit de fusion**)

Grâce au **commit de fusion**, les deux branches gardent leur **identité** dans le graphe de l'historique

Fusion Fast-Forward : Exemple



```
$ git merge anglais
Updating 9085634..9f68c02
Fast-forward
 test_EN.txt | 2 ++
 1 file changed, 2 insertions(+)
 create mode 100644 test_EN.txt
```



```
$ git log
commit 9f68c02c75df9b88d138ba7559f34991f0b1b264
Author: Isabelle BLASQUEZ <isabelle.blasquez@...>
Date: Tue Mar 7 16:18:47 2017 +0100

    message en anglais

commit 908563408714f417561a443e4766821c30b25ec6
Author: Isabelle BLASQUEZ <isabelle.blasquez@...>
Date: Fri Feb 17 11:10:45 2017 +0100

    ajout git

commit c1eb479524b7352604b2cf156d73d757b735f175
Author: Isabelle BLASQUEZ <isabelle.blasquez@...>
Date: Fri Feb 17 09:16:52 2017 +0100

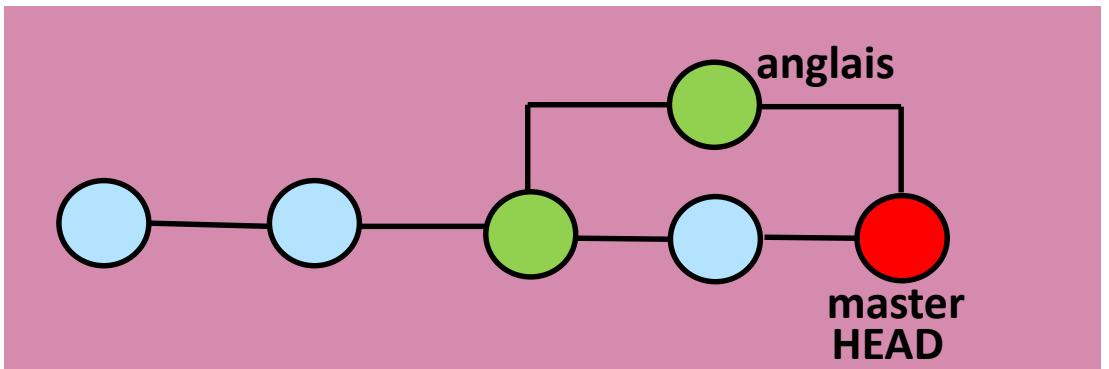
    Premier Commit
```

git log
confirme la
présence
d'uniquement
3 commits

Remarque : La branche anglais existe toujours, possibilité de s'y positionner dessus avec `checkout`

Isabelle BLASQUEZ

Après la fusion : git log (tous les commits)



```
$ git log
commit 8186d8f3c68841eef8e63dcc7a04e195aa09346c
Merge: a51cb7e fa9ed92
Author: Isabelle BLASQUEZ <isabelle.blasquez@...>
Date:   Thu Mar 9 11:45:00 2017 +0100

    fusion définition git en anglais

commit a51cb7ed0335be223a252e4d21630701b8e25c2a
Author: Isabelle BLASQUEZ <isabelle.blasquez@...>
Date:   Thu Mar 9 11:43:20 2017 +0100

    ajout définition git en français

commit fa9ed9295e405f55ccb815051708e06d9ca7bdb7
Author: Isabelle BLASQUEZ <isabelle.blasquez@...>
Date:   Thu Mar 9 11:42:00 2017 +0100

    ajout définition git en anglais

commit 9f68c02c75df9b88d138ba7559f34991f0b1b264
Author: Isabelle BLASQUEZ <isabelle.blasquez@...>
Date:   Tue Mar 7 16:18:47 2017 +0100

    message en anglais

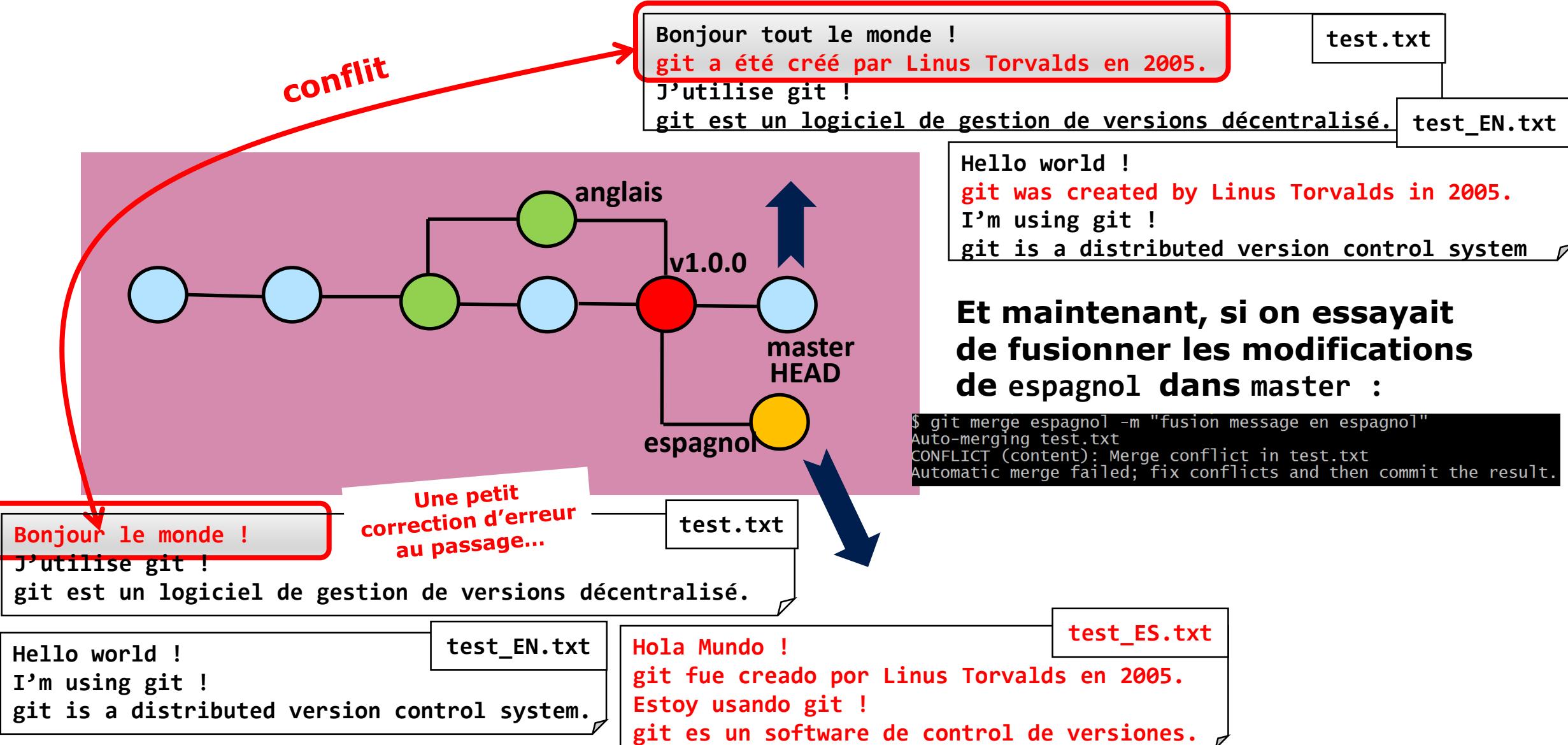
commit 908563408714f417561a443e4766821c30b25ec6
Author: Isabelle BLASQUEZ <isabelle.blasquez@...>
Date:   Fri Feb 17 11:10:45 2017 +0100

    ajout git

commit c1eb479524b7352604b2cf156d73d757b735f175
Author: Isabelle BLASQUEZ <isabelle.blasquez@...>
Date:   Fri Feb 17 09:16:52 2017 +0100

    Premier Commit
```

...Mais parfois, git peut ne pas savoir pas comment réaliser la fusion



... Et il y a un **Conflit** à résoudre !

```
$ git merge espagnol -m "fusion message en espagnol"
Auto-merging test.txt
CONFLICT (content): Merge conflict in test.txt
Automatic merge failed; fix conflicts and then commit the result.
```

Git met la fusion en pause (le prompt indique `(master|MERGING)`)
Et demande à l'utilisateur de résoudre le conflit manuellement

Pour voir les fichiers en conflit : **git status**

```
$ git status
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")

Changes to be committed:

  new file:  test_ES.txt

Unmerged paths:
  (use "git add <file>..." to mark resolution)

    both modified:  test.txt
```

Fichier qui présente un conflit :
test.txt

2 solutions :

- abandonner la fusion **git merge --abort**
- ou **résoudre manuellement le conflit**

Pour résoudre manuellement le conflit, ouvrir le fichier qui présente le conflit ...

<<<<< branche-de-base
début du conflit avec le contenu
de la branche de base
(**HEAD** qui correspond à master)

=====
séparation du contenu des deux branches

>>>>> branche-a-fusionner
fin du conflit avec ci-dessus
le contenu en conflit de la branche à fusionner
(ici branche **espagnol**)

test.txt

```
<<<<< HEAD
Bonjour tout le monde !
git a été créé par Linus Torvalds en 2005.
=====
Bonjour le monde !
>>>>> espagnol
J'utilise git !
git est un logiciel de gestion de versions décentralisé.
```

Résoudre le conflit et finaliser la fusion ...

test.txt

1. Corriger manuellement le conflit de manière à obtenir le fichier souhaité

```
Bonjour le monde !
git a été créé par Linus Torvalds en 2005.
J'utilise git !
git est un logiciel de gestion de versions décentralisé.
```

2. Avertir git que le conflit a été résolu (git add)

```
$ git add test.txt
```

3. Vérifier que le conflit a bien été résolu (git status)

```
$ git status
On branch master
All conflicts fixed but you are still merging.
(use "git commit" to conclude merge)

Changes to be committed:

modified:   test.txt
new file:   test_ES.txt
```

4. Valider le commit de la fusion (git commit avec un message par défaut prédéfini ou préciser le message)

```
$ git commit -m "fusion message en espagnol"
[master f304b2d] fusion message en espagnol
... et le prompt redevient (master)
```

Plein d'autres possibilités

- Supprimer des branches,
- Rebaser des branches,
- ... Et surtout faire le lien entre des repository locaux et distants !

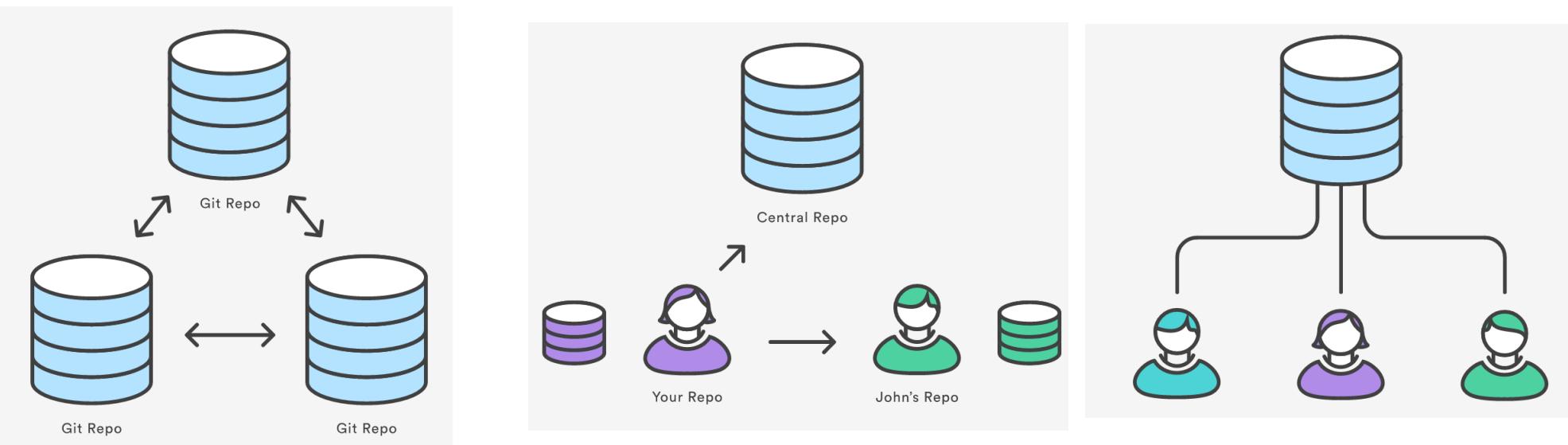
Dépôt distant

Dépôt distant

Un dépôt git peut être lié à d'autres dépôts git dits ***distant*** (ou ***remote repository***) avec lesquels il pourra partager des commits.

Un dépôt distant a un emplacement qui peut être :

- un répertoire (sur un disque local ou partagé),
- ou une URL (par exemple https://github.com/iblasquez/tuto_git).



Un *remote* peut être utilisé comme ***répertoire central***, auquel tous les développeurs se synchronisent.

git fournit à chaque développeur sa propre copie du dépôt, avec son propre historique local et sa structure de branche.

Cloner un dépôt distant

```
git clone emplacement-du-depot-à-cloner
```

Le clone peut se faire selon plusieurs protocoles : directement en local, via HTTPS, SSH ou git

Exemple : cloner un dépôt distant sur Github (https)

The screenshot shows a GitHub repository page for 'iblasquez / geekgit'. The repository has 3 commits, 1 branch, 0 releases, and 1 contributor. A red arrow points from the 'Clone or download' button in the top right to a terminal window displaying the cloning command and its execution.

Terminal output:

```
$ git clone https://github.com/iblasquez/geekgit.git
Cloning into 'geekgit'...
remote: Counting objects: 9, done.
remote: Compressing objects: 100% (7/7), done.
remote: Total 9 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (9/9), done.
Checking connectivity... done.
```

The 'Clone or download' button is highlighted with a red box. Below it, the 'Clone with HTTPS' field also contains a red box and displays the URL <https://github.com/iblasquez/geekgit.git>.

Que se passe-t-il lors d'un clone ?

```
$ git clone https://github.com/iblasquez/geekgit.git
Cloning into 'geekgit'...
remote: Counting objects: 9, done.
remote: Compressing objects: 100% (7/7), done.
remote: Total 9 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (9/9), done.
Checking connectivity... done.
```

espace de travail



```
$ git log
commit 897bb18c454879da7215ea8d7f905e4d067bae1a
Author: Isabelle BLASQUEZ <isabelle[REDACTED]@[REDACTED]>
Date:   Fri Mar 10 14:19:04 2017 +0100

    Ajout lien poesie.md

commit 0425e967093beb97763acfec269525bdb08778d5
Author: Isabelle BLASQUEZ <isabelle[REDACTED]@[REDACTED]>
Date:   Fri Mar 10 14:17:08 2017 +0100

    Ajout poésie : Source Code Poetry Challenge

commit fc8cd200702f6052bd81a474e234910171e94f86
Author: Isabelle BLASQUEZ <isabelle[REDACTED]@[REDACTED]>
Date:   Fri Mar 10 13:36:29 2017 +0100

    Initial commit
```

→ un nouveau **dépôt local** **geekgit** est créé.

→ ce dépôt est lié au dépôt distant connu
sous le nom **origin**

(c-a-d **origin** est un raccourci qui pointe vers l'URL du dépôt distant : <https://github.com/iblasquez/geekgit.git>)

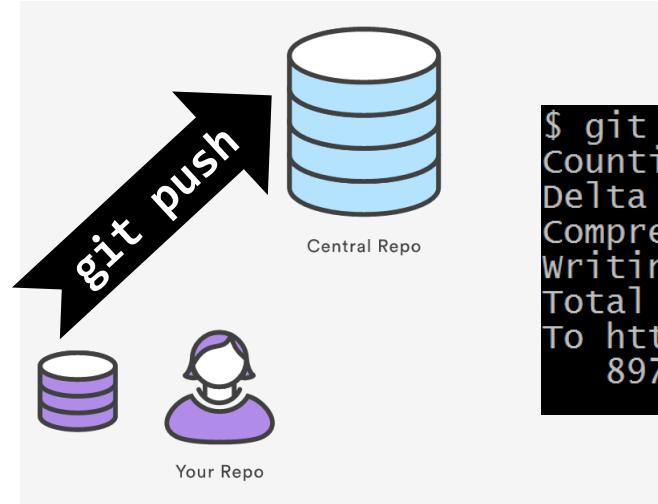
→ les commits de l'origine sont récupérés
dans ce dépôt

Synchroniser les dépôts : publier ses commits (push)

POUSSER

`git push <depot-distant> <branche-locale>`

**Envoyer les modifications
de la branche master du dépôt local
sur le dépôt distant (origin)**



`git push origin master`

```
$ git push origin master
Counting objects: 6, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (6/6), done.
Writing objects: 100% (6/6), 2.01 KiB | 0 bytes/s, done.
Total 6 (delta 0), reused 0 (delta 0)
To https://github.com/iblasquez/geekgit.git
  897bb18..501f13b  master -> master
```

Les identifiants Github
peuvent être demandés
avant de pousser ;-)

iblasquez	maj README : lien page peinture
README.md	maj README : lien page peinture
peinture.md	ajout page peinture avec classic programmer paintings
poesie.md	Ajout poésie : Source Code Poetry Challenge

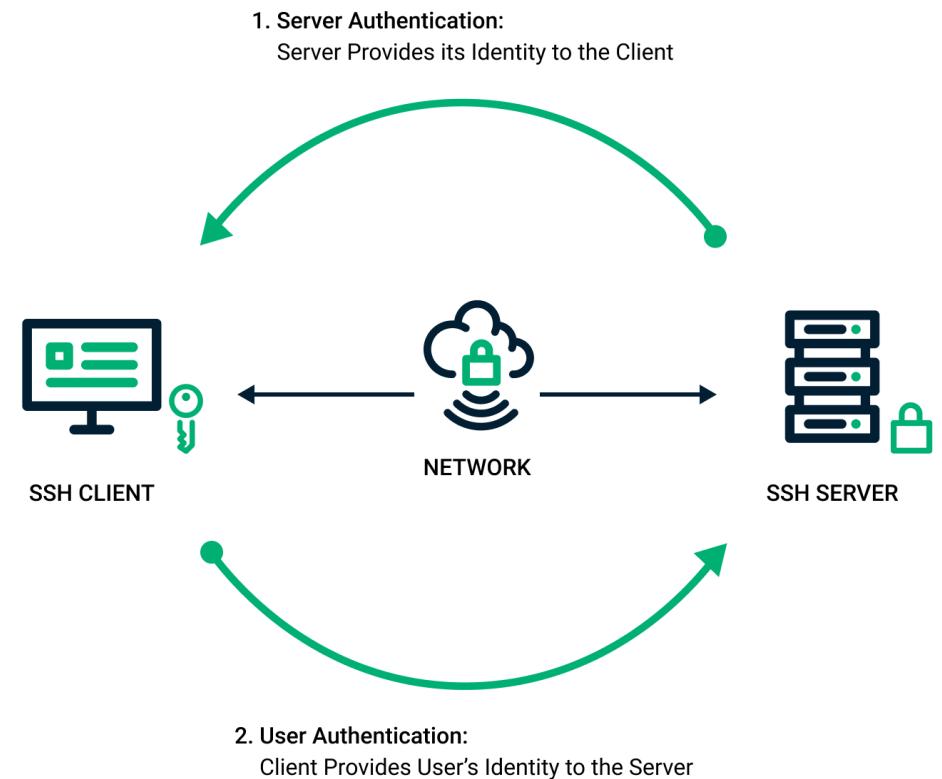
Remarques : Pousser toutes les branches `git push --all`

Intéressant à lire : <http://blog.xebia.fr/2017/01/17/git-depots-distants-sans-acces-reseau/>

Isabelle BLASQUEZ

Problème par rapport aux tutos

- Les procédures de push ont changé sur github en 2021
 - Il n'est plus possible de pousser un dépôt avec une authentification basique
- Il va falloir jouer avec des clés SSH !



Github

- Pourquoi github?
- Créer un compte
- Settings -> SSH -> ajouter une clé publique
- En parallèle : créer une clé sur votre ordinateur

Synchroniser les dépôts : récupérer des commits distants (pull)

TIRER

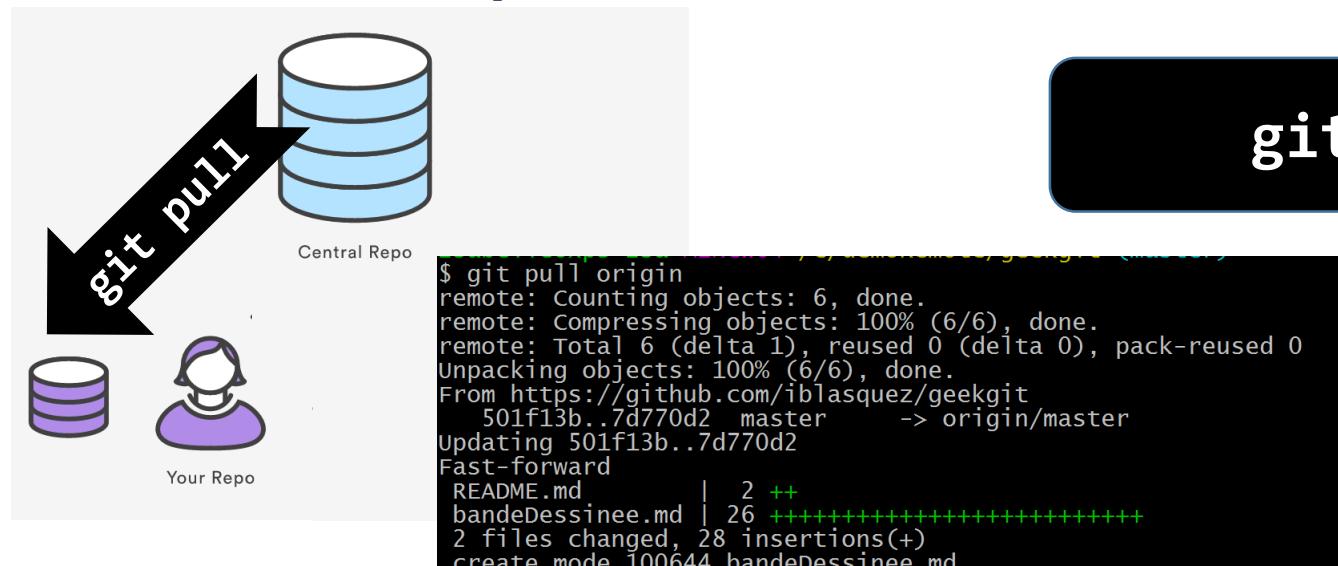
Télécharger les commits du dépôt distant de la branche courante...

git fetch <depot-distant>

... puis Fusionner ces commits à la branche locale
(git merge origin/master)

git merge origin/<branche-locale>

Ou directement : Recevoir sur la branche courante les modifications en provenance du dépôt distant



git pull <depot-distant>

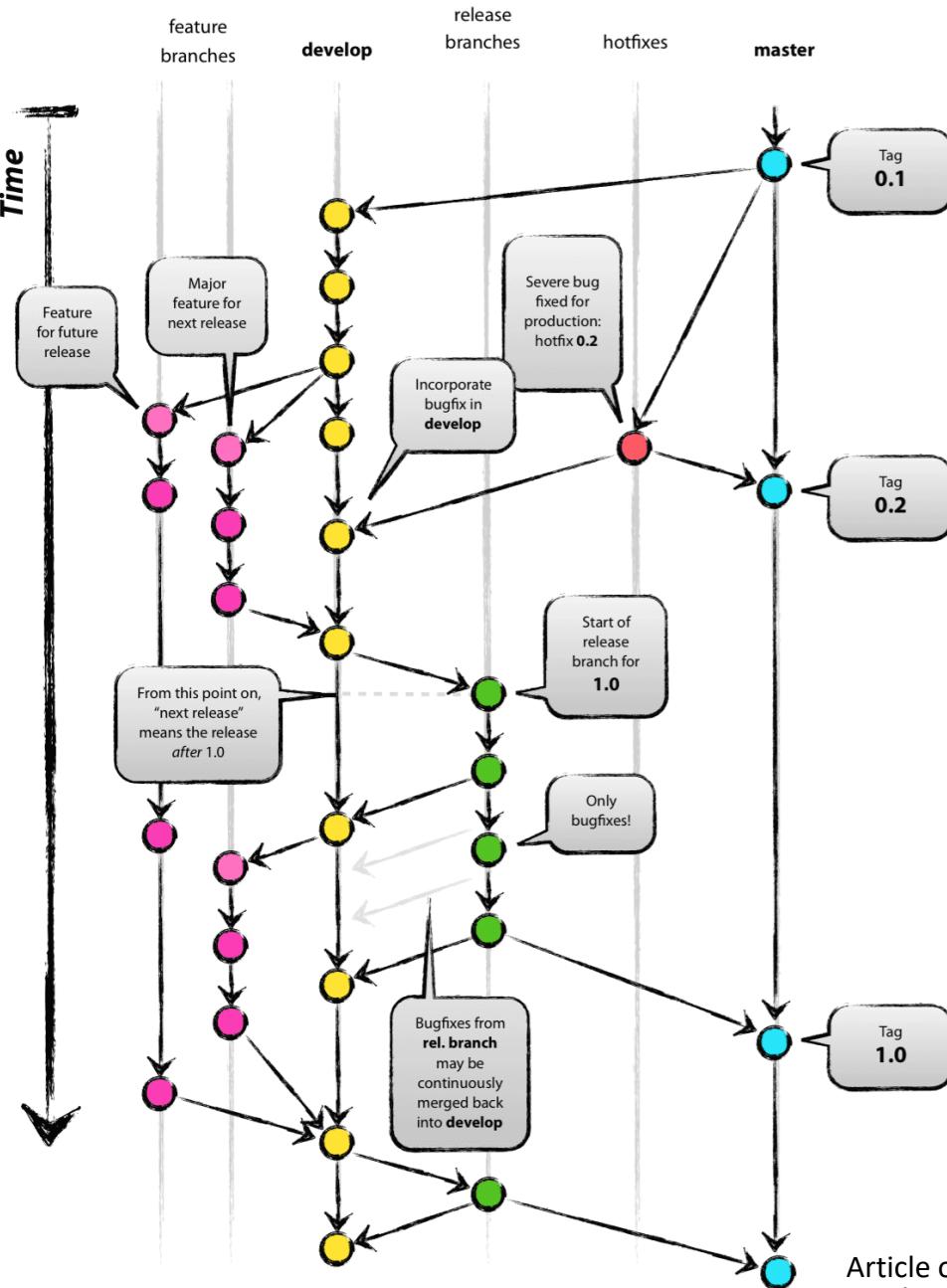
Possibilité de demander un rebase :
git pull --rebase <depot-distant>

Remarque : Si un conflit se présente git merge --abort revient à l'état avant le pull.

Isabelle BLASQUEZ

Usages avancés

Git Flow : un workflow à base de branches (1/2)



Les branches principales

→ master sur origin

(doit rester stable : prêt pour être déployé en production)

→ develop (appelée aussi branche d'intégration)

(réflète les derniers changements livrés pour la prochaine version)

Les branches de support (à durée de vie limitée)

→ les branches pour les **fonctionnalités** (features)

→ les branches pour les **versions** (releases)

(préparer les nouvelles versions de production, correction d'anomalies mineures, préparation des métadonnées pour une version (numéro, date de compilation, etc.)

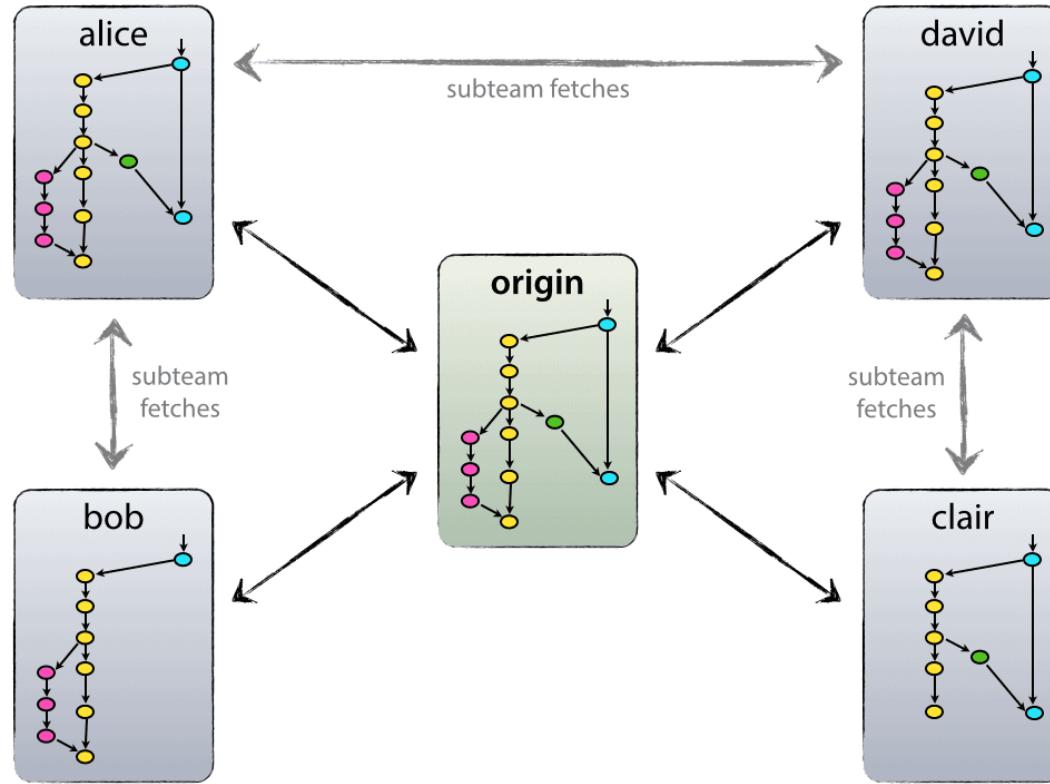
→ Les branches de **correctifs** (hotfixes)

Article original : <http://nvie.com/posts/a-successful-git-branching-model/>

Version française : <https://www.occitech.fr/blog/2014/12/un-modele-de-branches-git-efficace/>

Pour faciliter sa mise en place voir Git flow : <http://danielkummer.github.io/git-flow-cheatsheet/>

Git Flow : un workflow à base de branches (2/2)



- Chaque développeur **pull** et **push** sur **origin**
- Au delà de la relation centralisée push-pull, chaque développeur peut aussi **récupérer des changements d'autres équipiers** et ainsi former des **sous-équipes**...
- ...utile quand deux ou plusieurs développeurs travaillent ensemble sur une fonctionnalité, plutôt que de pousser prématûrement le travail en cours sur origin.

GitHub Flow : un workflow branché simplifié ...

1 master et des branches **features** !!!

