

Introduction à Git :

Un logiciel de gestion de Versions

Décentralisé

(DVCS)



Isabelle BLASQUEZ
@iblasquez

2017



Isabelle BLASQUEZ



[@iblasquez](https://twitter.com/iblasquez)

Enseignement : Génie Logiciel

Recherche : Développement logiciel agile



ICSTUG #IUTAgile

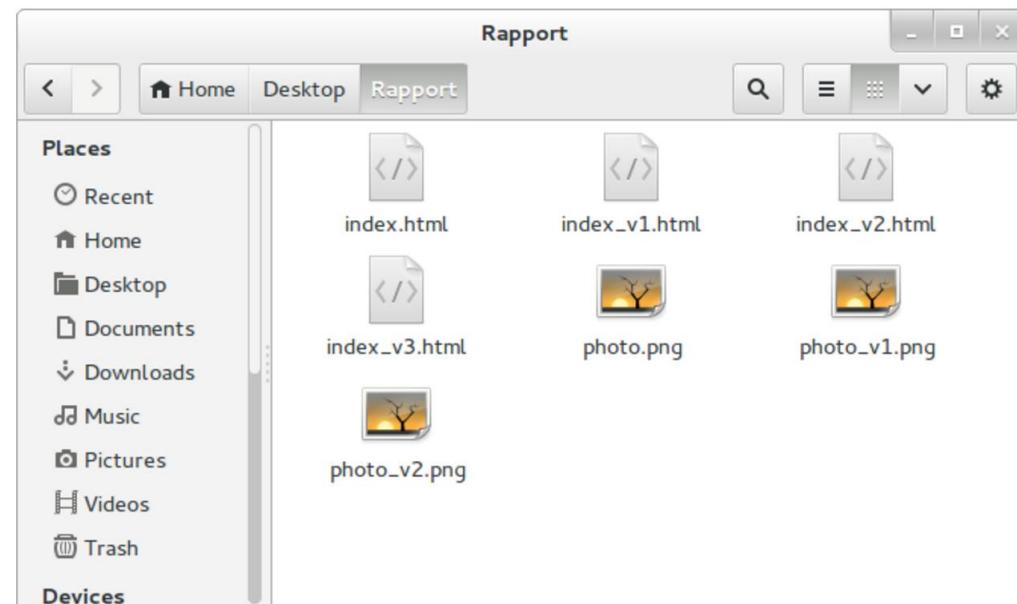
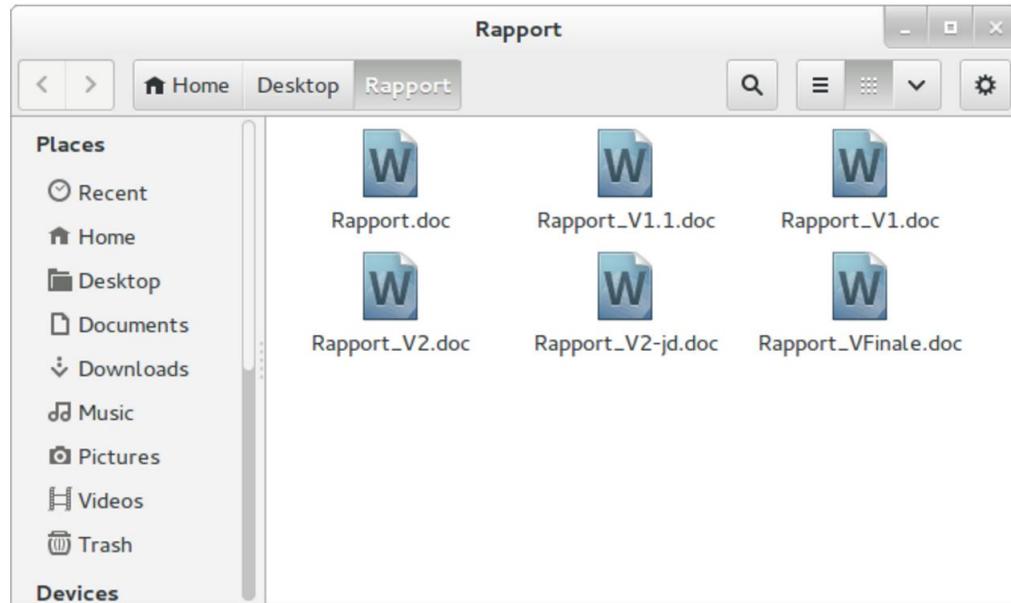


**#Software
Craftsmanship**



**Pourquoi un
gestionnaire de version ?
... et pourquoi décentralisé ?**

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

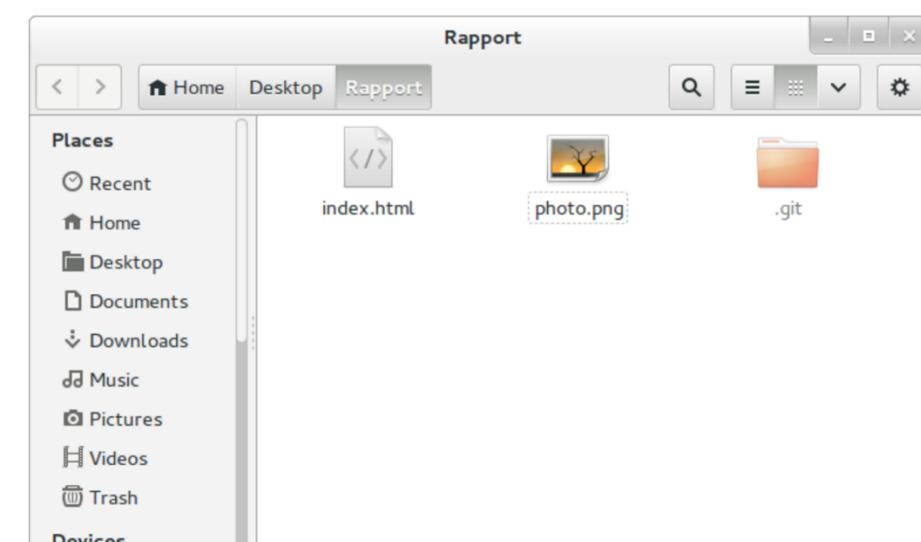


Gestionnaires de Versions à la rescousse !

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** ...)



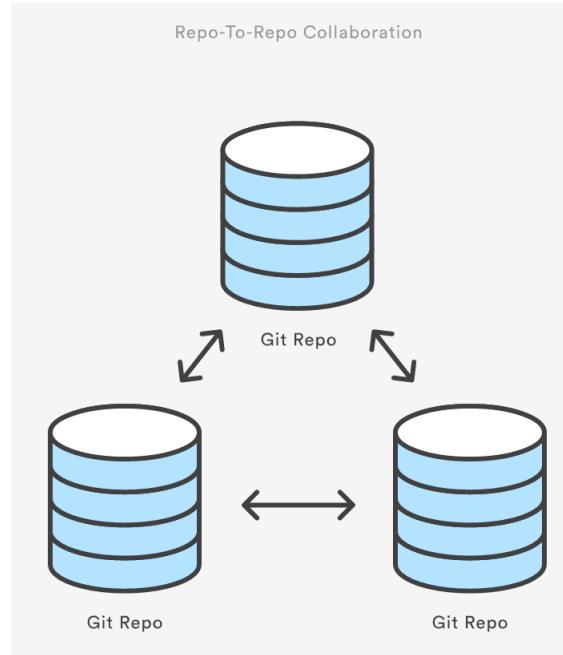
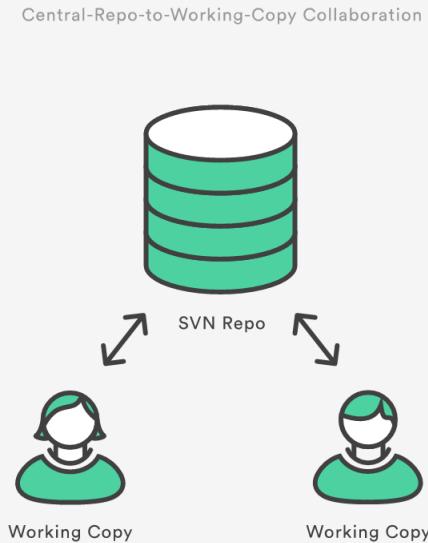
(2005)



(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 »

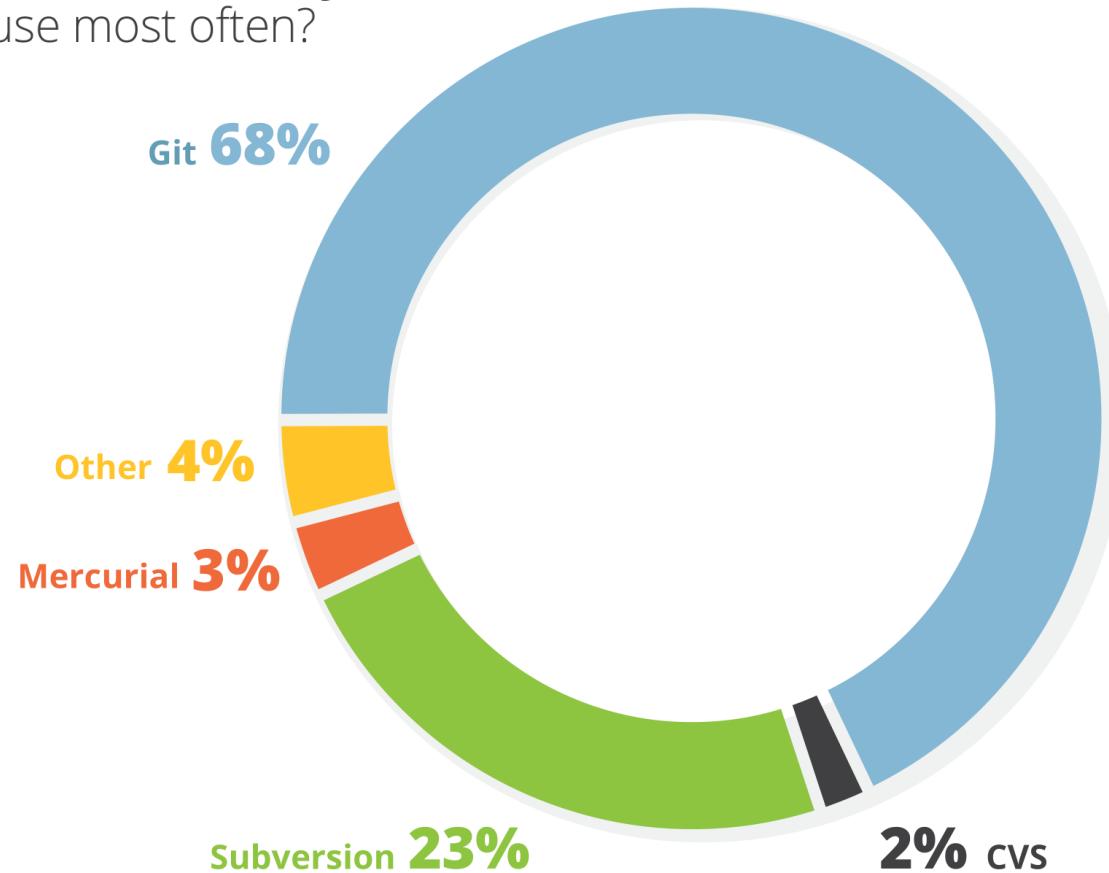
A lire : <https://www.joelonsoftware.com/2010/03/17/distributed-version-control-is-here-to-stay-baby>

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

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

Git : le plus populaire des gestionnaires de version

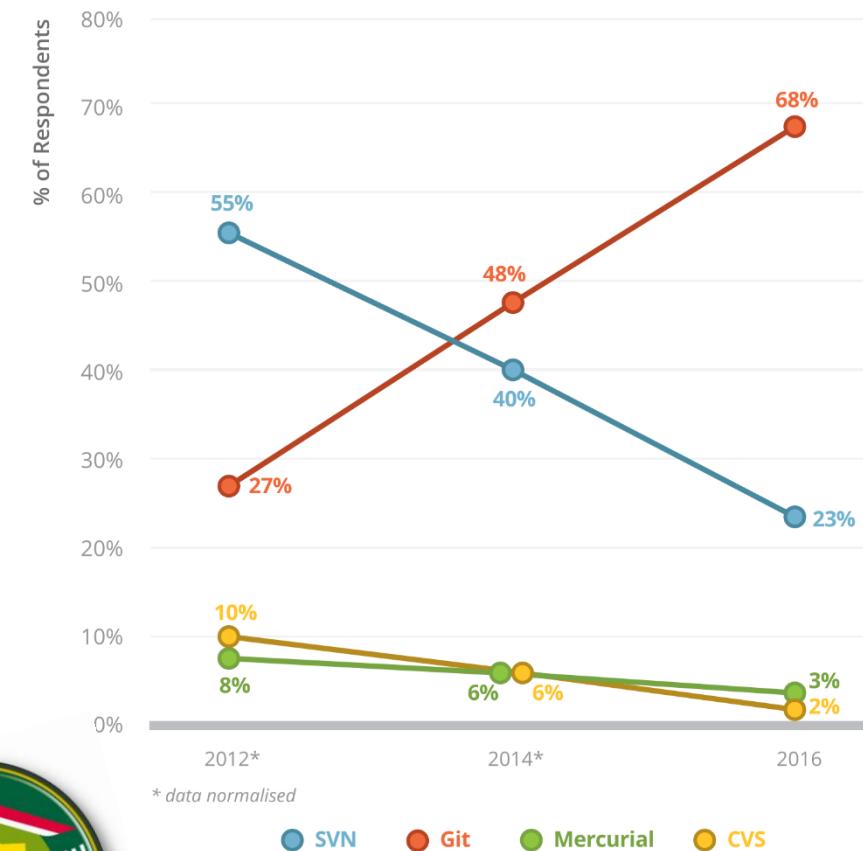
Which VCS do you
use most often?



All rights reserved. 2016 © ZeroTurnaround Inc.

Extrait : <http://pages.zeroturnaround.com/RebelLabs-Developer-Productivity-Report-2016.html>

Figure 3.6 VCS Usage Since 2012



Cyril Lacôte @clacote - 30 sept.
En soutien aux développeurs et à l'artisanat du code source français,

bientôt un label "git de France".

Isabelle BLASQUEZ

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).

Git : les premiers pas ...

Installation

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

Downloads



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

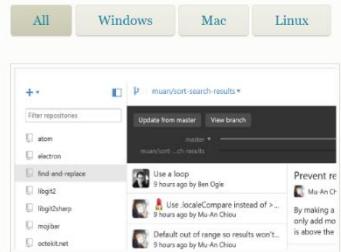
- ✓ **Utilisation de git :**

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

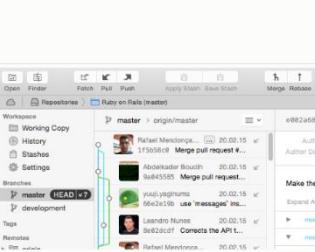
→ **via un outil graphique**

GUI Clients

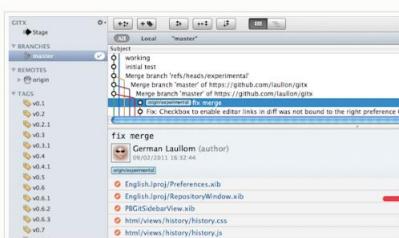
Git comes with built-in GUI tools for committing ([git-gui](#)) and browsing ([gitorious](#)), but there are several third-party tools for users looking for platform-specific experience.



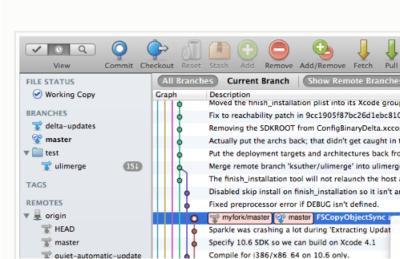
GitHub Desktop
Platforms: Windows, Mac
Price: Free



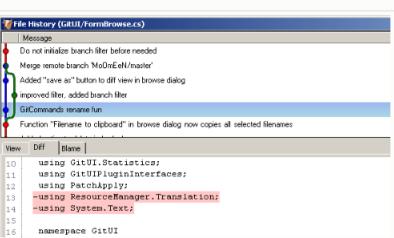
Tower
Platforms: Windows, Mac
Price: \$79/user (Free 30 day trial)



GitX-dev
Platforms: Mac
Price: Free



SourceTree
Platforms: Mac, Windows
Price: Free



Git Extensions
Platforms: Windows
Price: Free

```
$ 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.

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

Les commandes Git

Commandes de porcelaine

(porcelain commands **haut niveau**)

commandes utilisées au quotidien



git-grep
git-help
git-revert
git-citool
git-describe
git-diff
git-bundle
git-rebase
git-shortlog
git-add
git-pack-refs
git-show
git-archimport
git-whatchanged
git-worktree
git-verify-commit
git-rev-parse
git-filter-branch
git-get-tar-commit-id
git-cvsexportcommit
git-count-objects
git-cvsimport
git-difftool
git-format-patch
git-gc
gitweb
git-clone
git-reflog
git-tag
git-remote
git-fetch
git-checkout-index
git-check-mailmap
git-credential-cache
git-update-server-info
git-interpret-trailers
git-credential-store
git-check-ref-format
git-upload-archive
git-fmt-merge-msg
git-receive-pack
git-write-tree
git-ls-tree
git-http-push
git-rev-list
git-mailinfo
git-mktag
git-symbolic-ref
git-http-fetch
git-updated-ref
git-prune-packed
git-show-ref
git-read-tree
git-fetch-pack
git-var
git-daemon
git-show-index
git-send-pack
git-clean
git-commit
git-archive
git-cvsserver
git-rerere
git-branch
git-checkout
git-stash
git-submodule
git-cherry
git-fsck
git-reset
git-br
git-cherry-pick
git-rm
git-merge-tree
git-cvs
git-prune
git-init
git-clean

Commandes de plomberie

(plumbing commands **bas niveau**)

commandes qui manipulent les informations
au cœur de Git



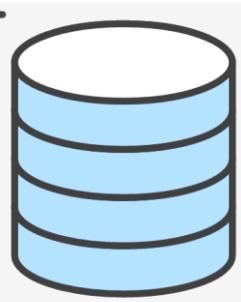
git-column
git-name-rev
git-merge-file
git-cat-file
git-merge-index
git-upload-pack
git-mktree
git-shell
git-ls-files
git-check-attr
git-verify-pack
git-pack-objects
git-checkout-index
git-check-mailmap
git-credential-cache
git-update-server-info
git-interpret-trailers
git-credential-store
git-check-ref-format
git-upload-archive
git-fmt-merge-msg
git-receive-pack
git-write-tree
git-ls-tree
git-http-push
git-rev-list
git-mailinfo
git-mktag
git-symbolic-ref
git-http-fetch
git-updated-ref
git-prune-packed
git-show-ref
git-read-tree
git-fetch-pack
git-var
git-daemon
git-show-index
git-send-pack

Configuration (à la première utilisation)

```
git config --global user.name "Prénom Nom"  
git config --global user.email email@domaine.extension
```

Pas de modification anonyme avec Git !!!

Le dépôt au cœur de Git (ou repository)



Créer un nouveau dépôt :

`git init`

(dans le répertoire courant)

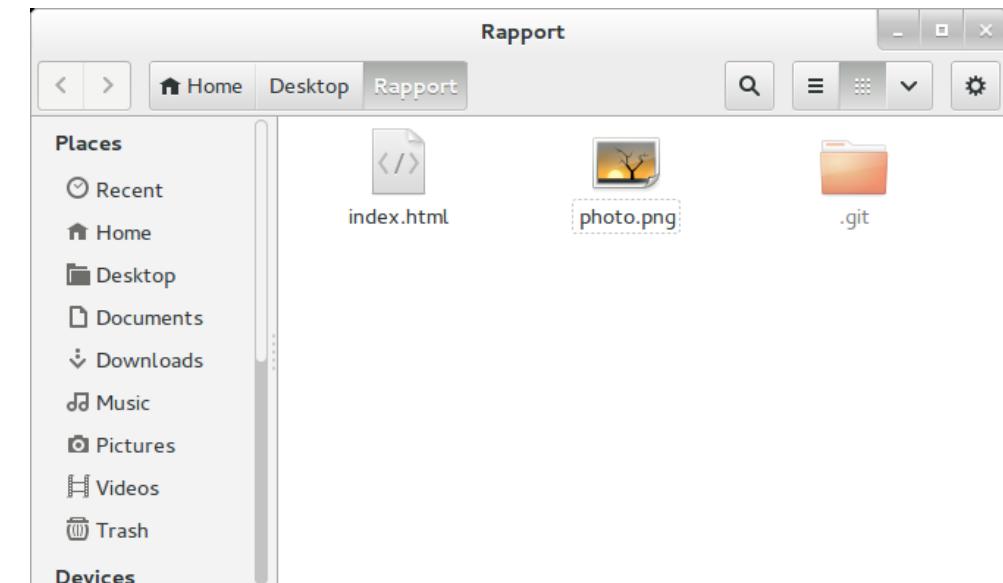
`git init <directory>`

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...

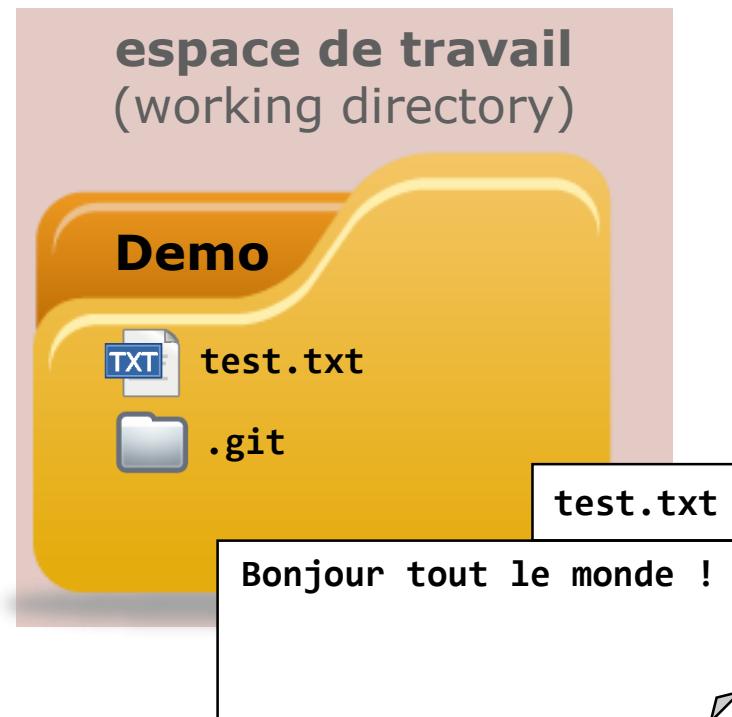
(En savoir plus sur le contenu réel de ce dépôt :

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

Enregistrer des changements

(git add, git commit, .gitignore)

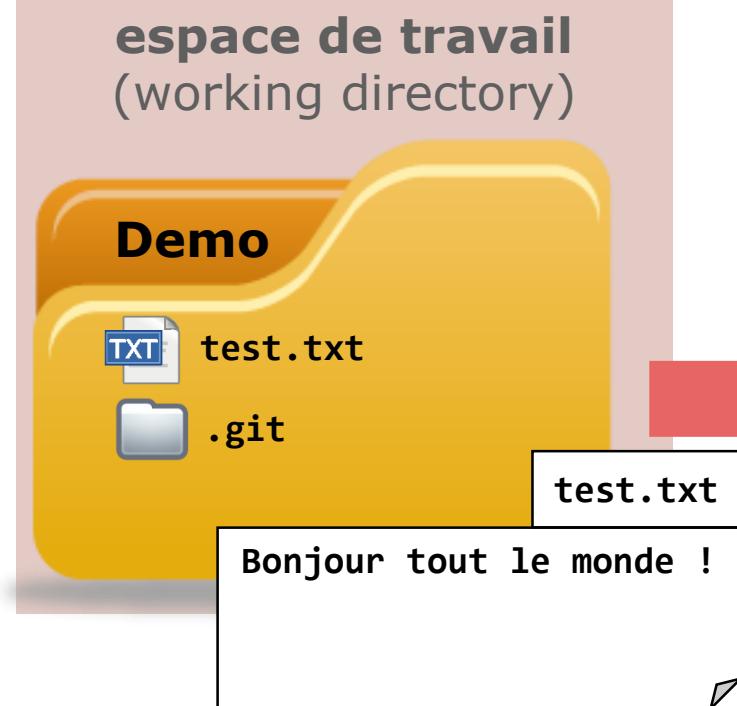
Un répertoire avec son fichier prêt à être versionné



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
```

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

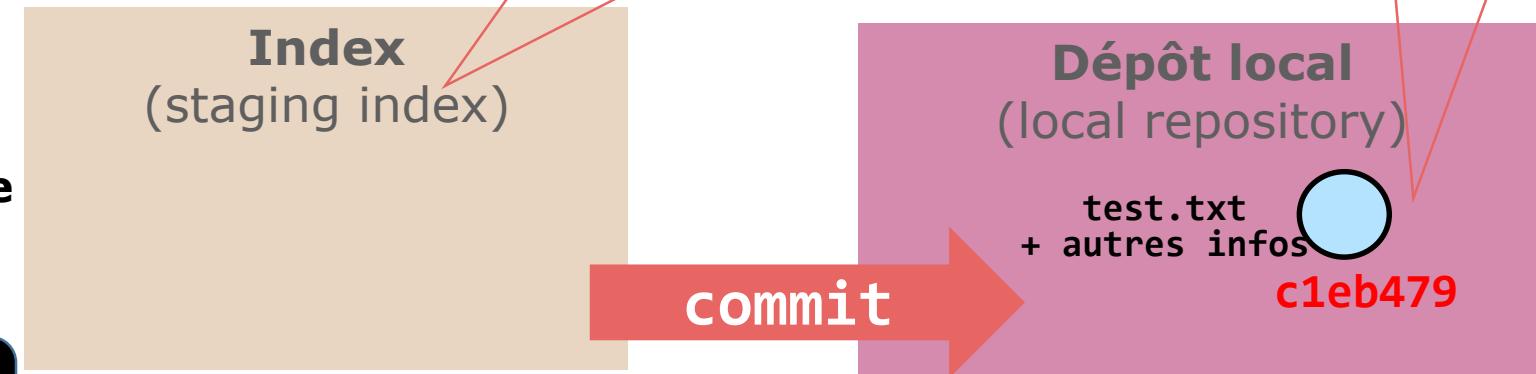
Liste des commits
(dans le dépôt local)

```
$ 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
```

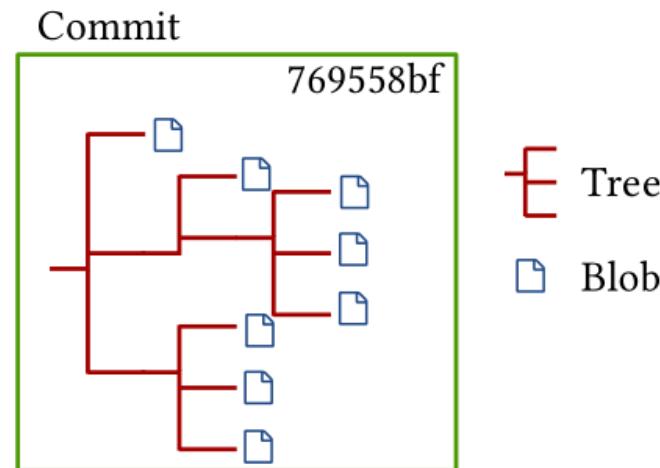


```
$ git commit -m "Premier Commit"  
[master (root-commit) c1eb479] Premier Commit  
1 file changed, 1 insertion(+)  
create mode 100644 test.txt
```

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**

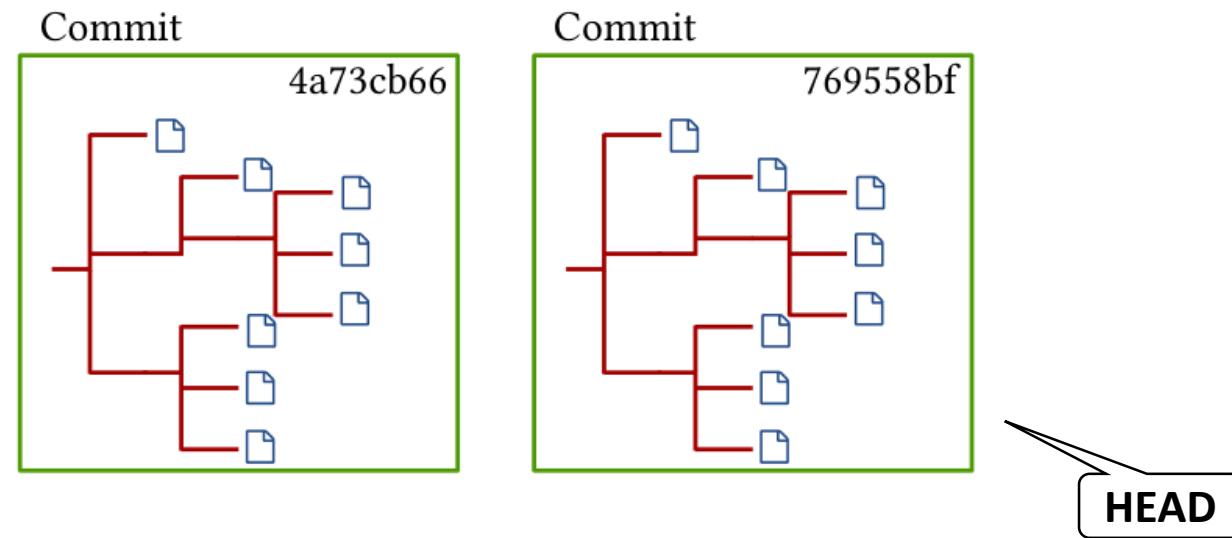


Chaque objet est identifié
40 caractères hexadécimaux
(**somme de contrôle SHA-1** de son contenu)

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)**.

A propos du dépôt Git

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

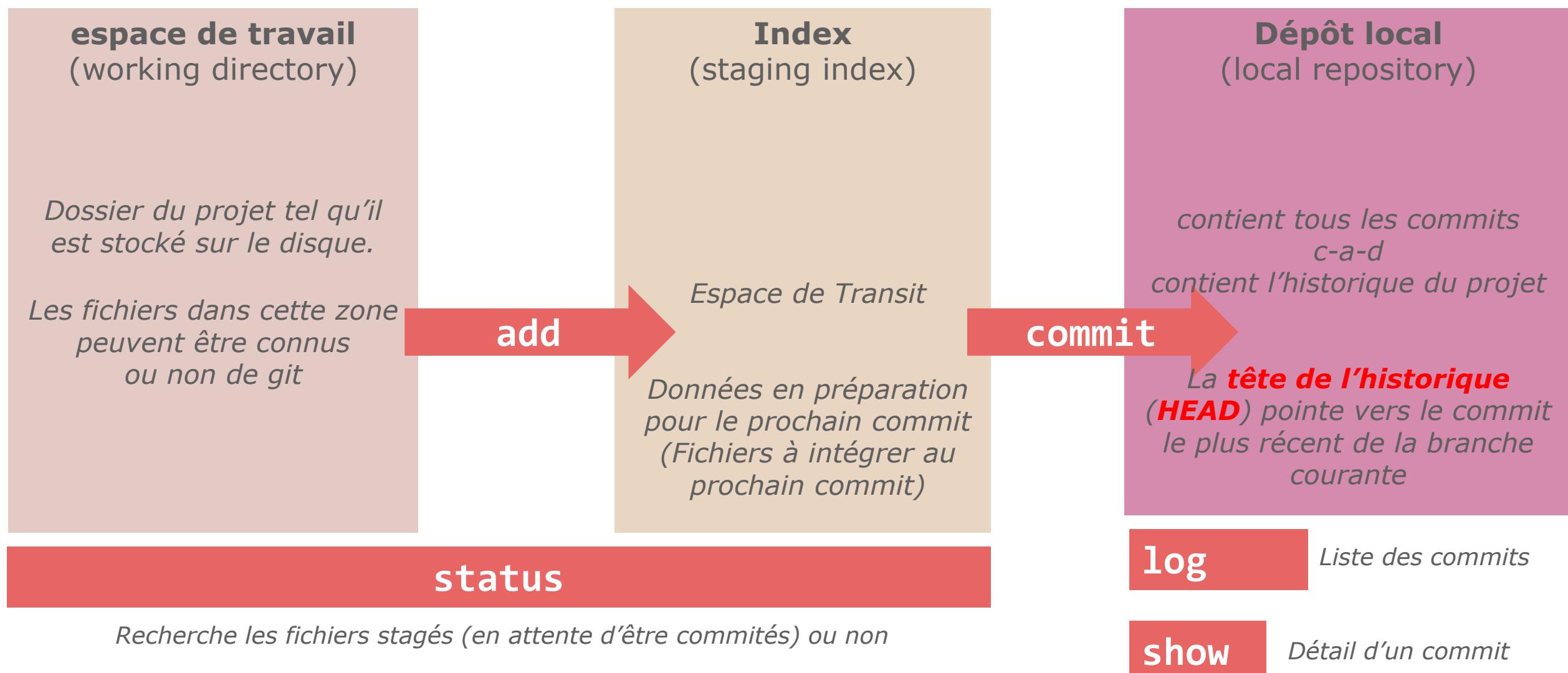


L'historique d'un dépôt Git est l'ensemble des versions du répertoire de travail (succession de commits)

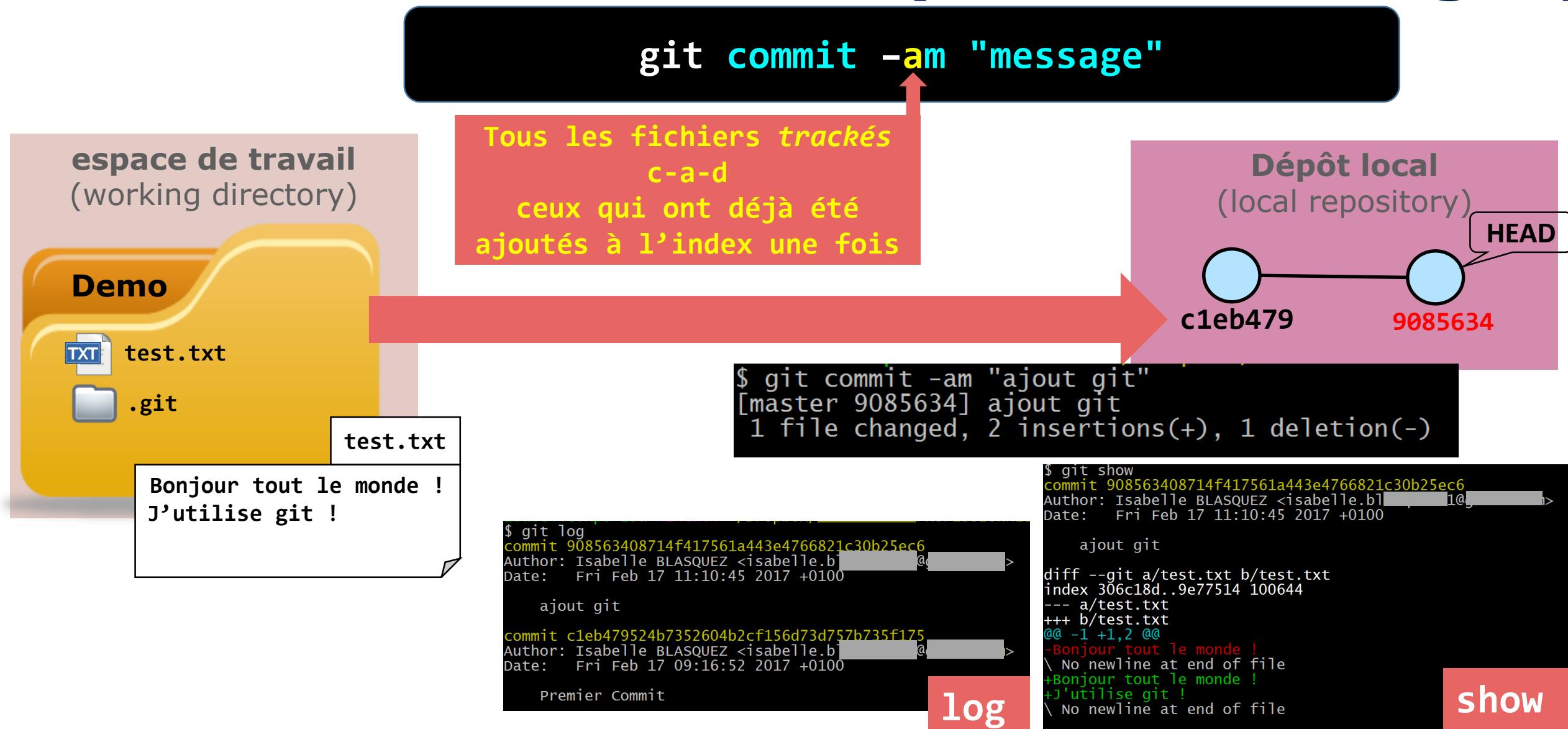
La tête (HEAD) de l'historique est un pointeur vers le dernier commit (utilisé comme prochain commit parent)

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)



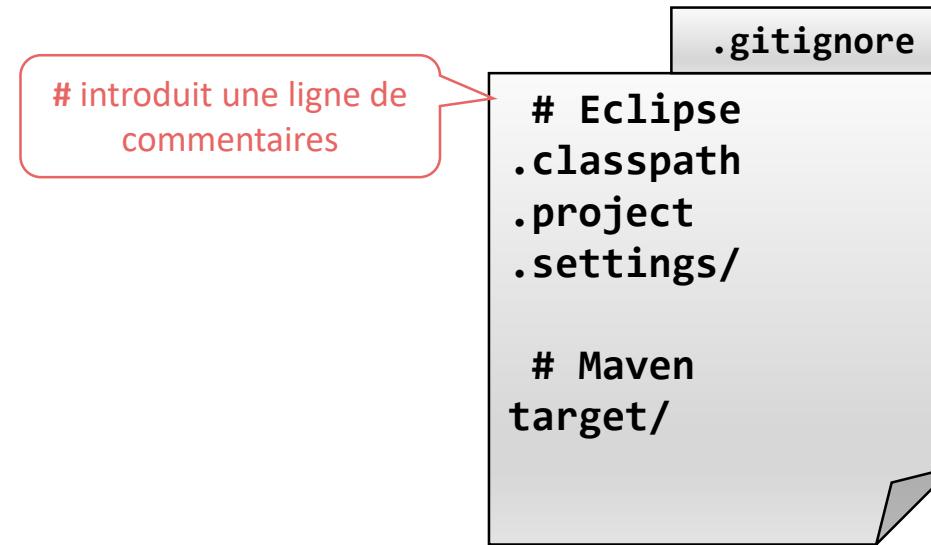
Ignorer des fichiers dans l'historique (.gitignore)

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

Consulter l'historique

(git status, git log, git diff)

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

...

**Consulter un changement
c-a-d se positionner sur
un ancien commit de l'historique
(git checkout)**

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>`

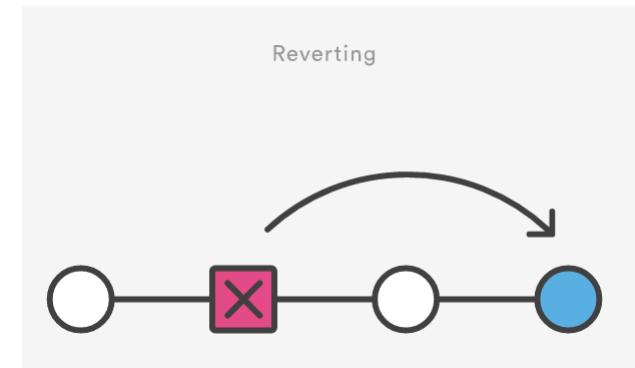
Annuler un changement

(git revert, git reset)

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

`git revert commitparSonSHA`

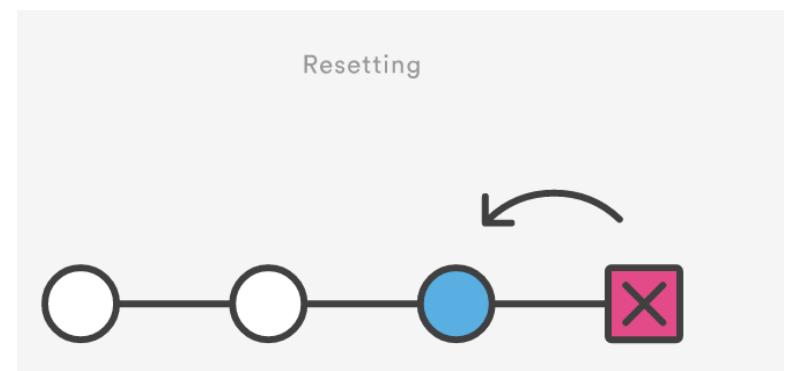
Revenir à un état antérieur en créant un nouveau nouveau commit (défait en créant un nouveau commit, mais ne revient pas en arrière à proprement parlé...)



`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

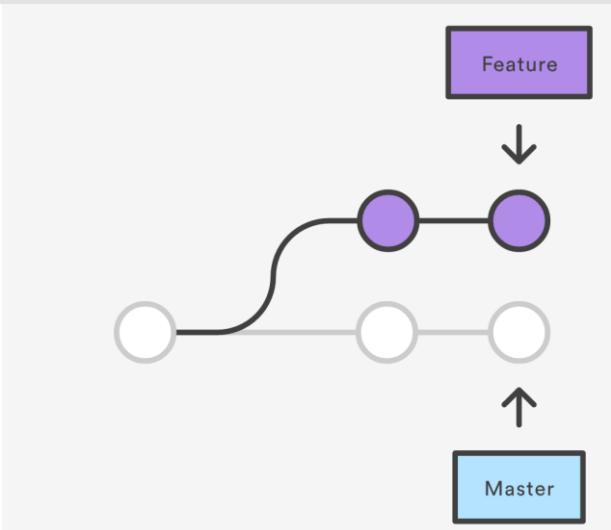
(Attention, **commande irréversible !!!**)

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

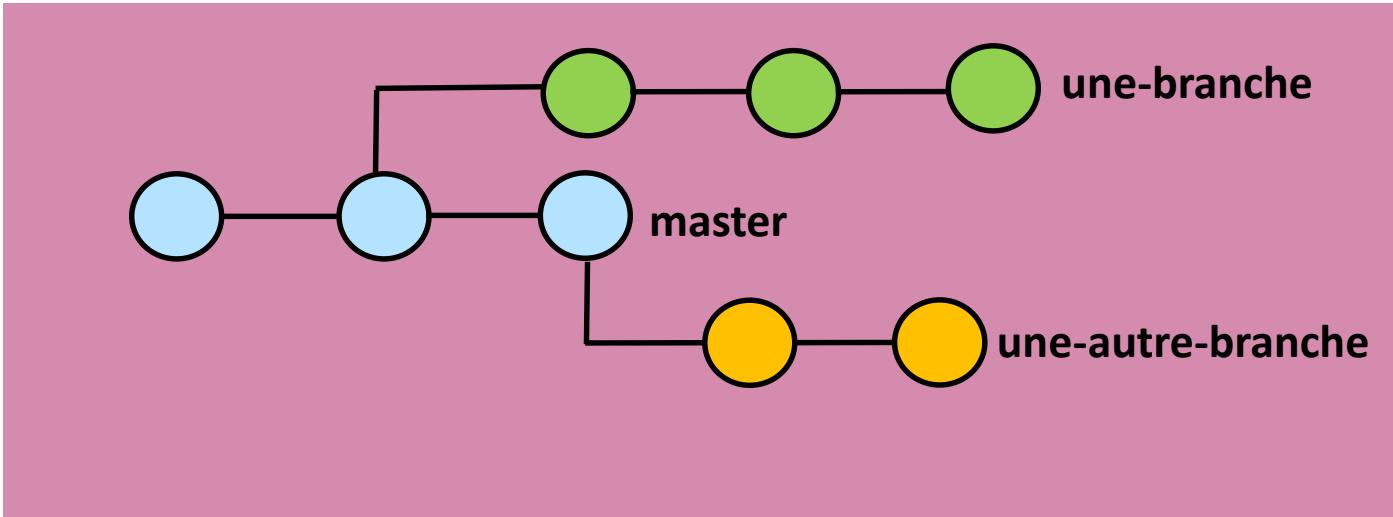
Modifier le message du commit (possible uniquement si commit pas non pushé sur dépôt distant (origin))

Créer des Branches

(git branch, git checkout)



A propos des branches

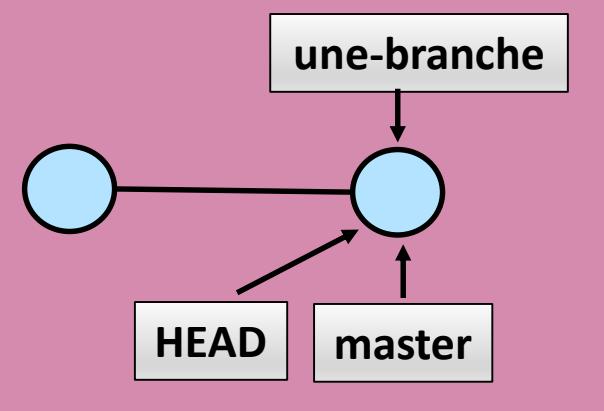


La branche par défaut est `master`

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

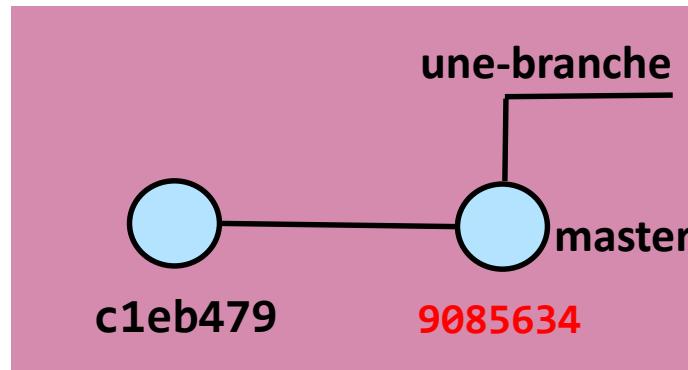
```
git checkout une-branche
```

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
```

En pratique ...



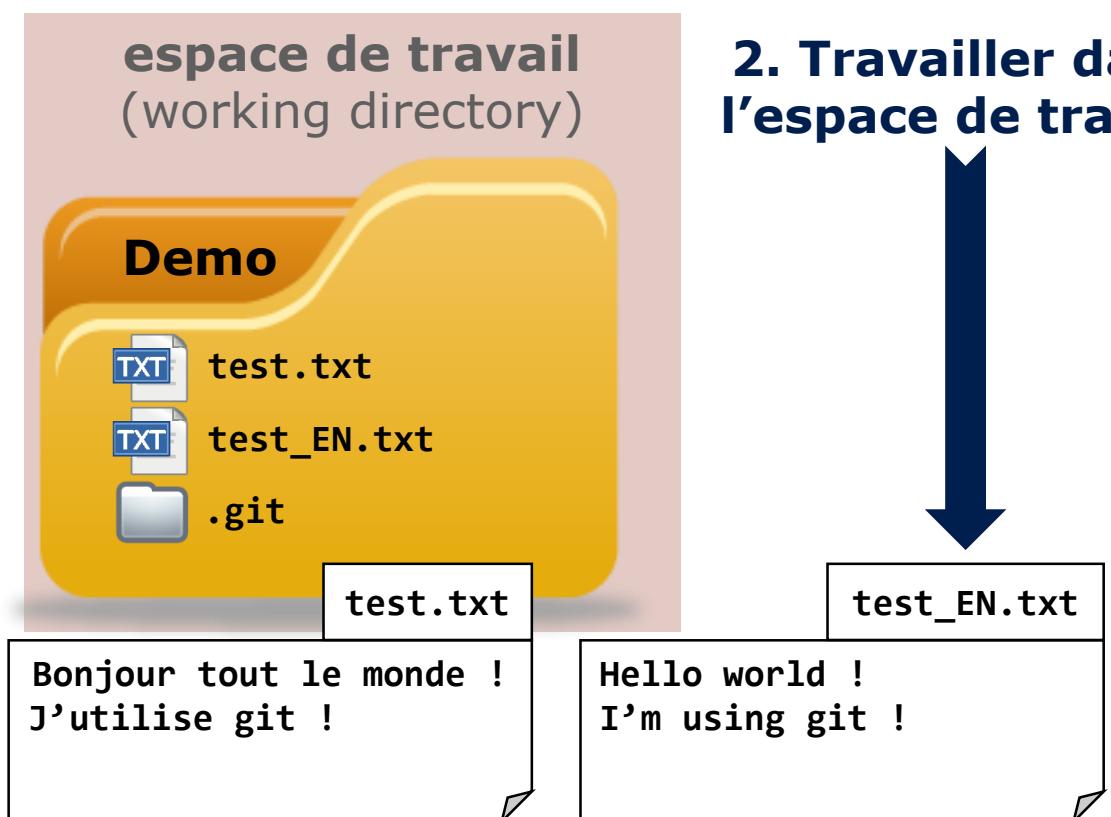
1. Créer une nouvelle branche *anglais* et se positionner dessus

```
git checkout -b anglais
```

3. Committer
(sur *anglais* puisqu'on est positionné sur cette branche !)

```
git add test_EN.txt
```

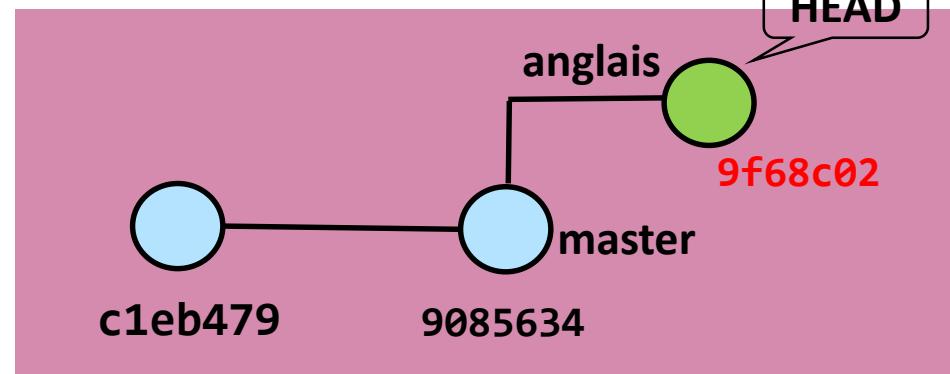
```
git commit -am "message en anglais"
```



2. Travailler dans l'espace de travail



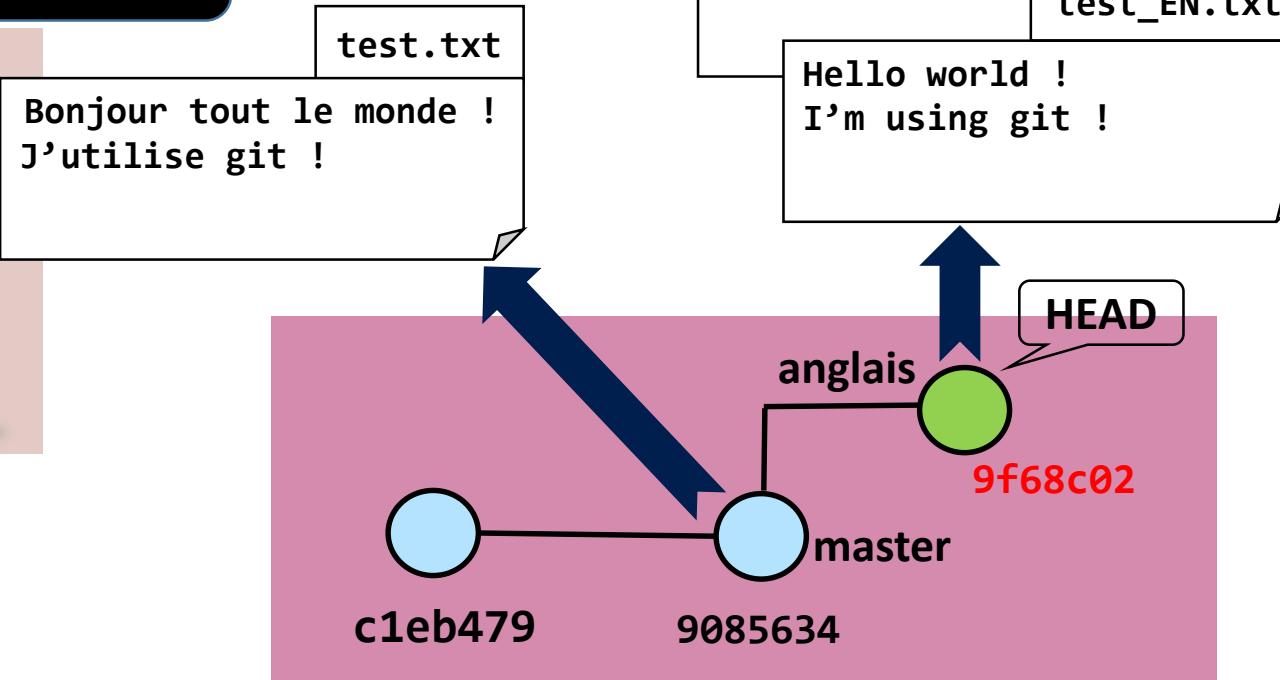
Si nouveau fichier,
il doit être ajouté à l'index !



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

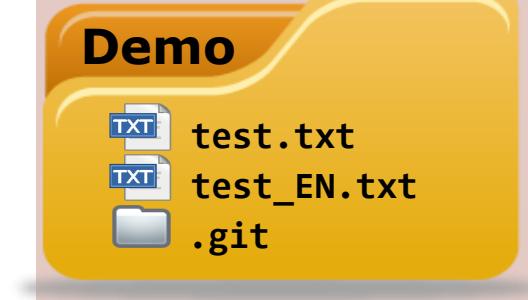
git checkout master

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)

Liste des branches existantes

git branch

```
$ git branch  
* anglais  
  master
```

* indique la branche courante
(c-a-d la branche qui contiendra le commit)

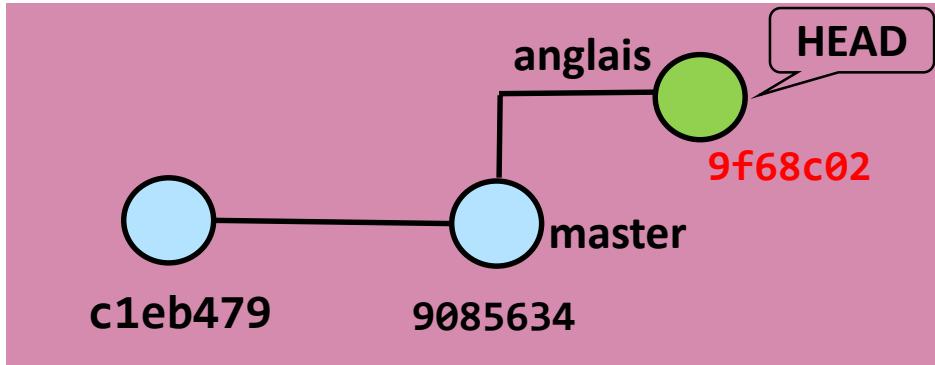
Remarque : Le prompt de la console (invite de commande) indique la branche courante :

ou

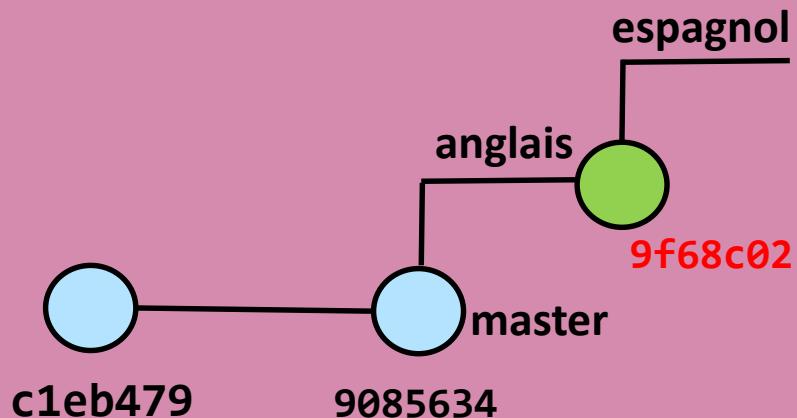
demo (anglais) : si anglais est la branche courante

demo (master) : si master est la branche courante

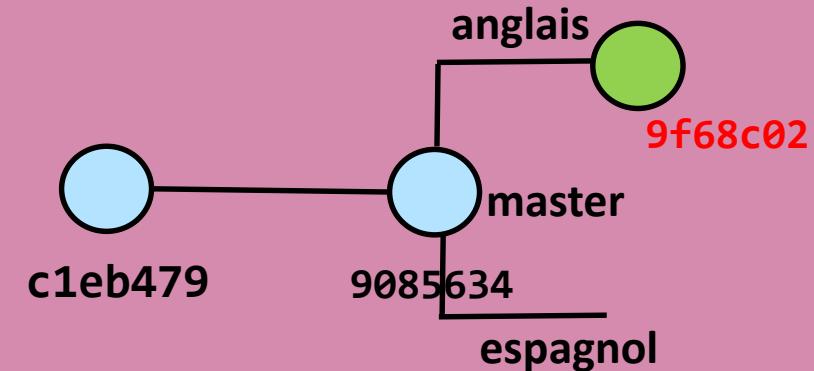
Créer (tirer) une autre branche ...



La branche est tirée depuis la branche courante ...



```
git branch espagnol
```



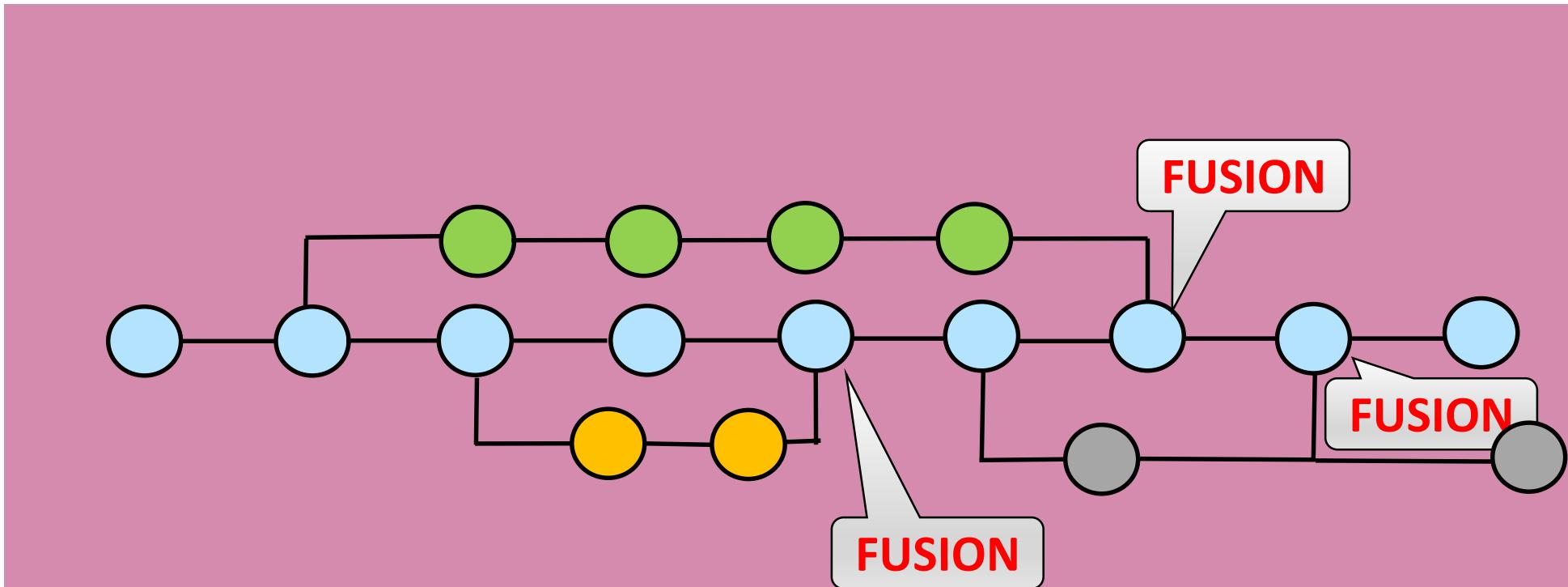
```
git checkout master
```

```
git branch espagnol
```

Fusionner des branches

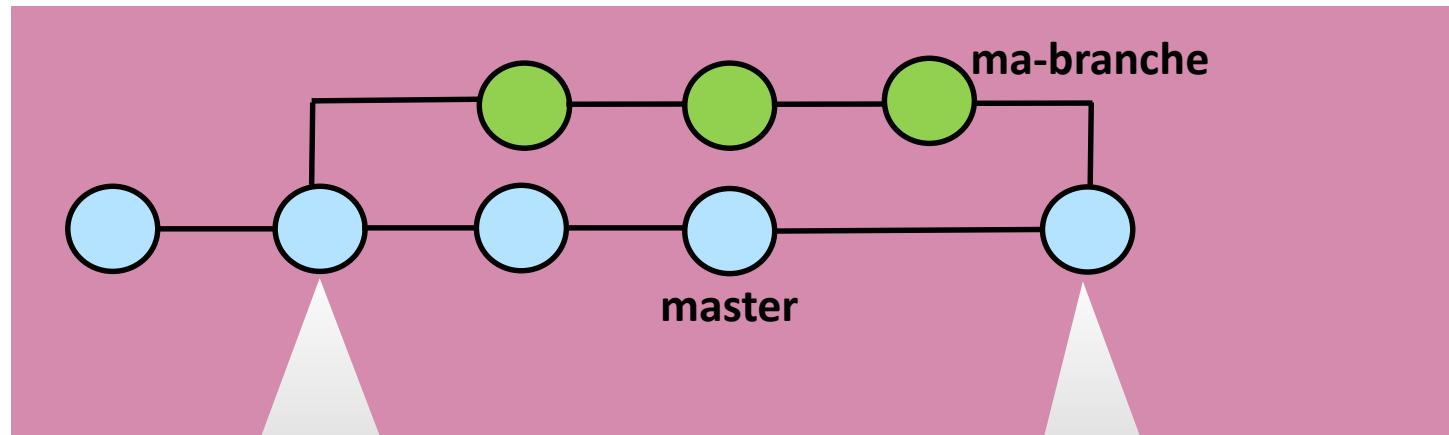
(git merge)

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 :

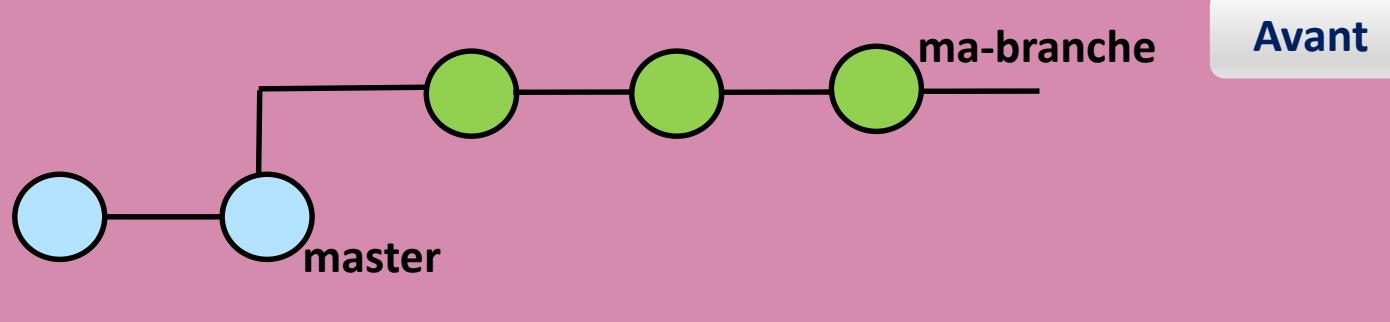
`git merge ma-branche`

→ 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

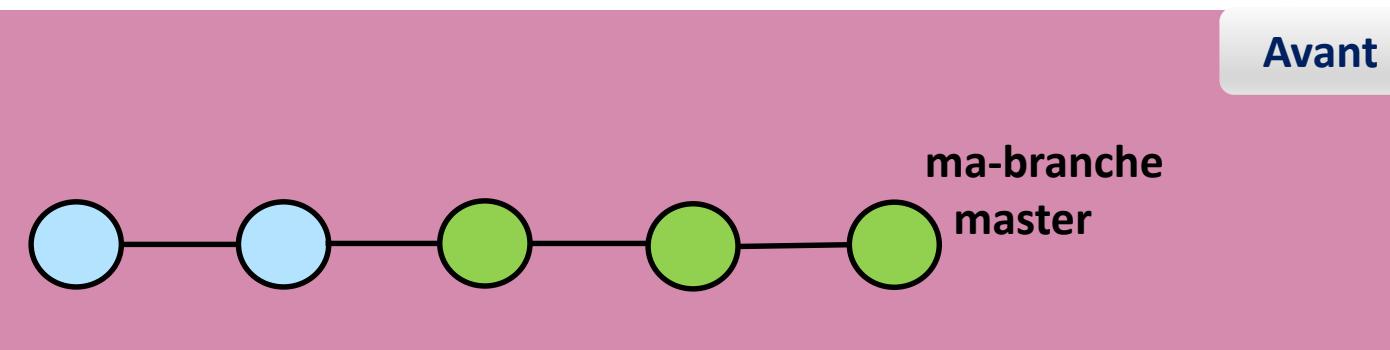
Fusionner une branche : Cas particulier de l'avance rapide (Fast-Forward)



Si pas de modification sur le « commit ancêtre » depuis la création de la branche...

```
git merge ma-branche
```

Avant



La fusion Fast-Forward a simplement pour effet de déplacer le sommet de la branche cible

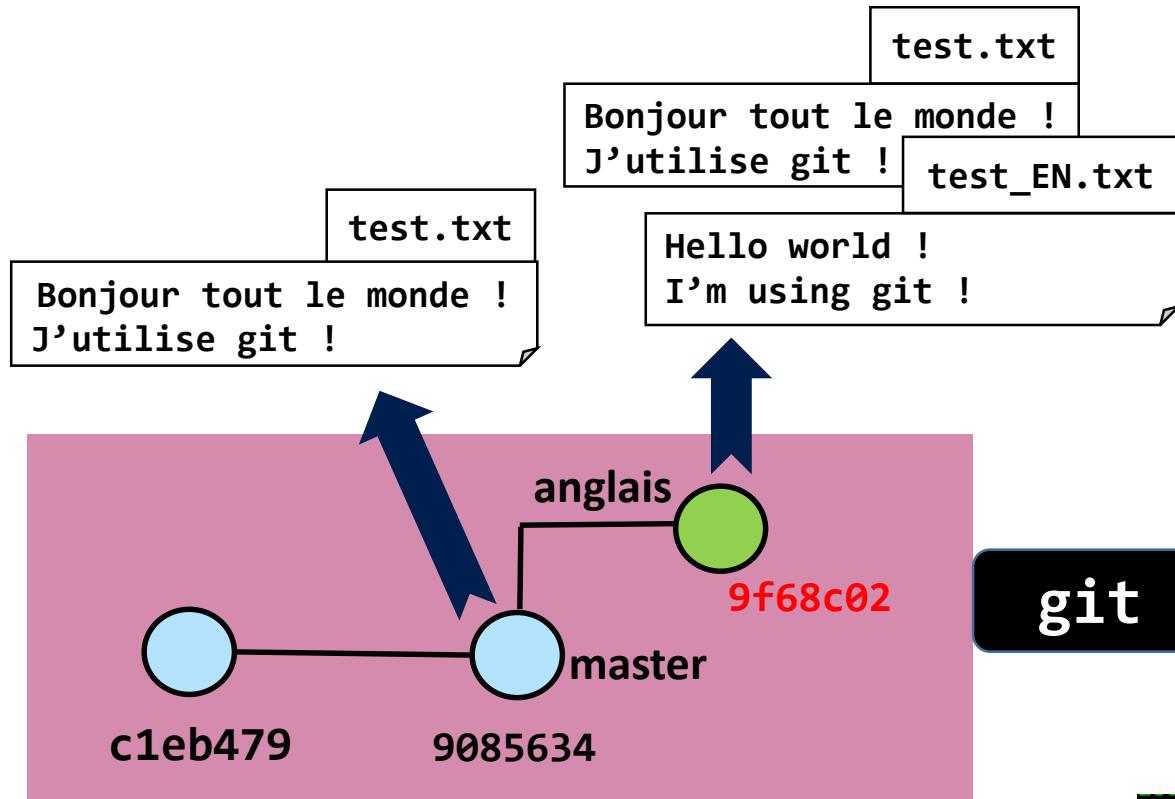
⇒ pas de commit de fusion,
simple avance de pointeur :
gain de temps ... mais ...



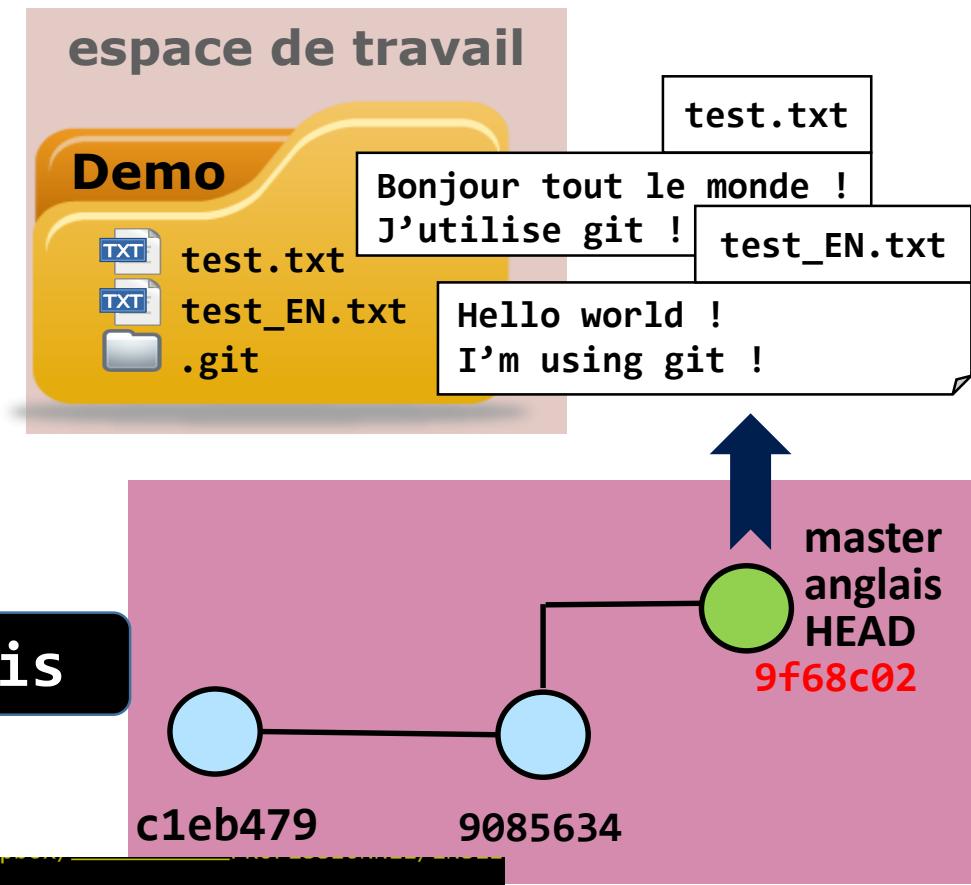
Avec une avance rapide,
l'historique restera linéaire
(perte d'indentité)

Remarque : possibilité de forcer la création d'un commit de fusion,
même dans cette situation, grâce à l'ajout de l'option **--no-ff** : ***no fast forward***)

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.████████@████.████>
Date: Tue Mar 7 16:18:47 2017 +0100

    message en anglais

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

    ajout git

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

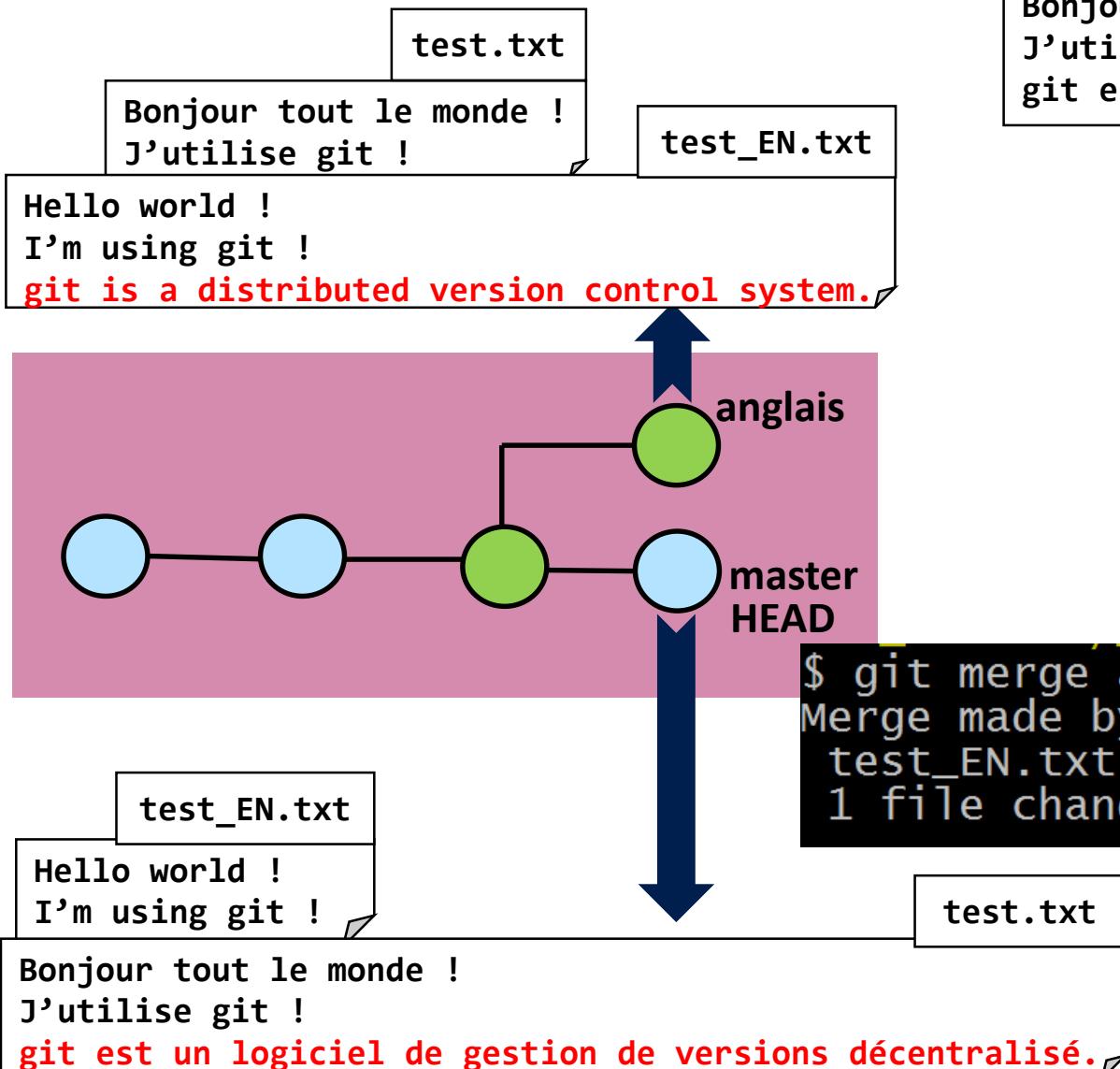
    Premier Commit
```

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

Fusion classique : Exemple



Bonjour tout le monde !

J'utilise git !

git est un logiciel de gestion de versions décentralisé.

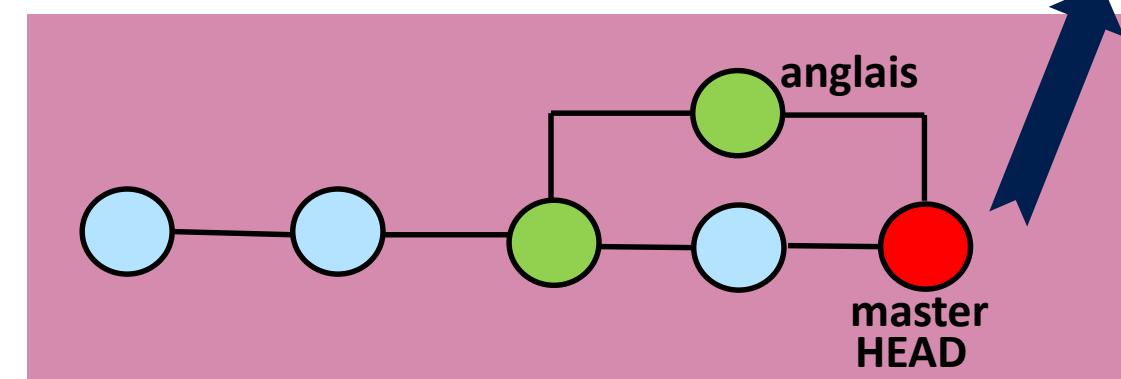
test.txt

Hello world !

I'm using git !

git is a distributed version control system.

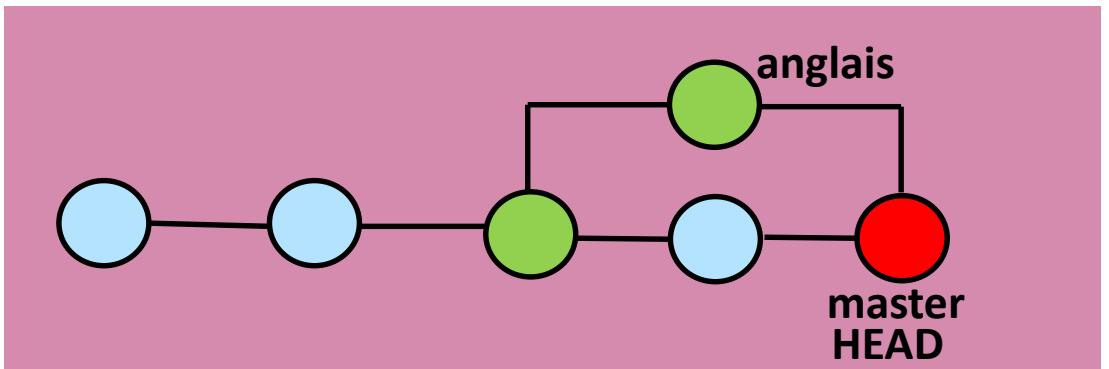
test_EN.txt



```
$ git merge anglais -m "fusion définition git en anglais"
Merge made by the 'recursive' strategy.
test_EN.txt | 3 +-+
1 file changed, 2 insertions(+), 1 deletion(-)
```

Comme la fusion crée un **nouveau commit**,
Si vous ne spécifiez pas de **message**,
il vous sera automatiquement demandé !

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
```

Tagger un commit

(git tag)

A propos des tags

**Un tag est un alias (nom)
dont le rôle est de pointer vers un commit...**

**... pour éviter l'utilisation des hash SHA-1,
peu explicites et difficiles à retenir**

**Les tags sont utilisés pour marquer des numéros de version
sur les commits**

A propos des numéros de version **x.y.z :**

x : version majeure (modifications dans le fonctionnement de l'application :
incompatibilité avec une version précédente)

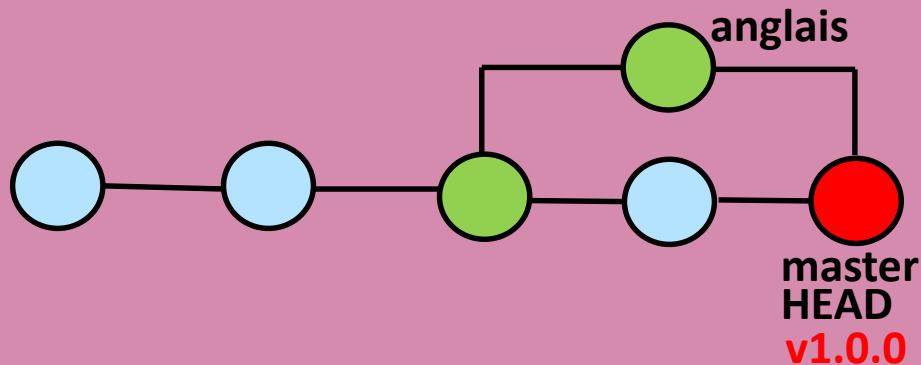
y : version mineure (ajout des fonctionnalités:
compatibilité avec l'ancien système)

z : patch (correction de bugs sans ajout de fonctionnalité)

Les commandes git autour des tags ...

Créer un tag sur le commit courant :

`git tag nom-du-tag`



Exemple :

`git tag v1.0.0`

Par la suite, il sera possible
de revenir sur le commit via

`git checkout v1.0.0`

Liste des tags

`git tag --list`

Détail d'un tag

`git show nom-du-tag`

Supprimer un tag

`git tag -d nom-du-tag`

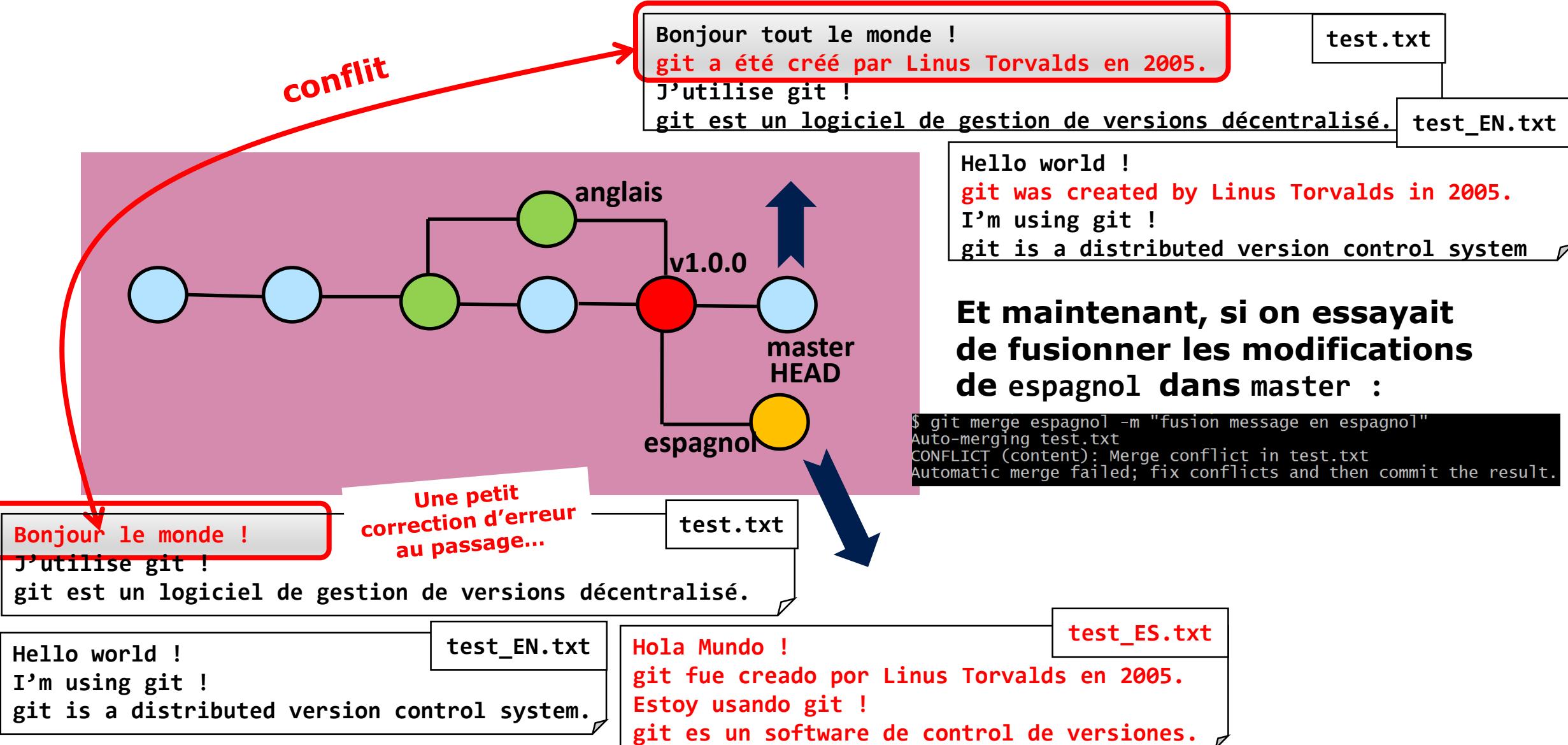
Créer un tag à partir de son hash : `git tag nom-du-tag hash`

Créer un tag annoté `git tag -a nom-du-tag` + message

`git tag -am nom-du-tag " message "`

Gérer un conflit

...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)
```

Supprimer une branche

Supprimer une branche

```
git branch -d branch-a-supprimer
```



Cette commande supprime réellement le pointeur de la branche.

A utiliser avec précaution, uniquement si :

- la branche a été fusionnée dans master et
aucune autre modification n'est prévue dans cette branche

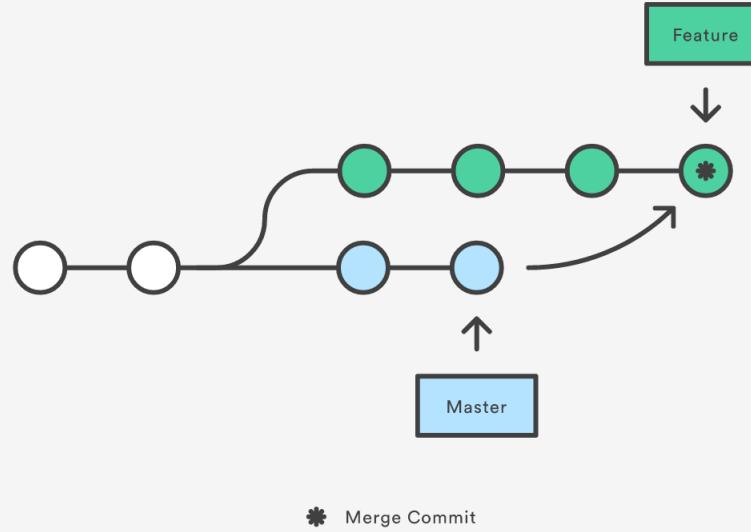
- les modifications de la branche ne sont plus à l'ordre du jour et elles
ne seront jamais intégrées.

Rebaser une branche dans une autre

Intégrer les modifications d'une branche dans une autre : Merge vs Rebase

`git checkout feature
git merge master`

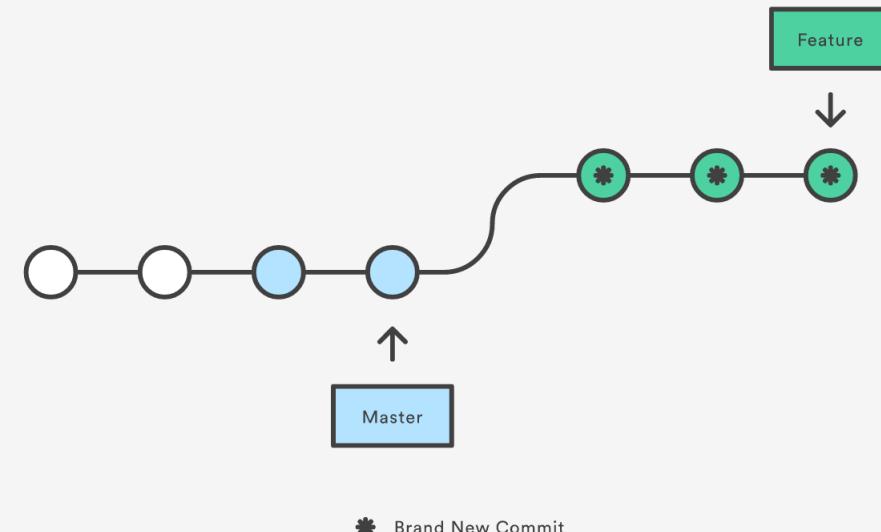
Merging master into the feature branch



merge : intégrer plusieurs commits cohérents entre eux dans un **commit unique** représentant tous les commits mergés

`git checkout feature
git rebase master`

Rebasing the feature branch onto master



rebase : réécriture de l'historique pour intégrer **tous les commits** d'une branche dans une autre (rendre l'historique linéaire)
⇒ permet de rejouer une suite de commits à partir d'un autre point de l'histoire

A propos de merge et rebase



Il ne faut pas rebaser des commits envoyés sur un dépôt distant c-a-d

Ne pas rebaser un historique public

(un rebase revient à transformer des commits comme s'ils étaient supprimés pour être recréés)

En savoir plus sur merge et rebase :

→ **Bien utiliser git merge et rebase**

<https://delicious-insights.com/fr/articles/bien-utiliser-git-merge-et-rebase/>

→ **Comparaison entre un merge et un rebase**

<https://fr.atlassian.com/git/tutorials/merging-vs-rebasing>

→ **Git++ : Passez au niveau supérieur de la gestion de version**

https://www.youtube.com/watch?v=m0_C2cfM9IM

<https://www.youtube.com/watch?v=rt-9mPaYtKo>

Transparents : <http://webadeo.github.io/git-simpler-better-stronger/#1.0>

→ **Exercices interactifs pour apprendre le branching sous Git**

<http://learngitbranching.js.org/>

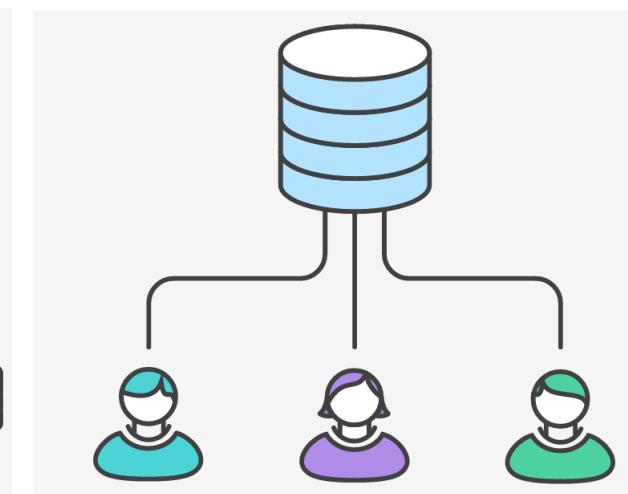
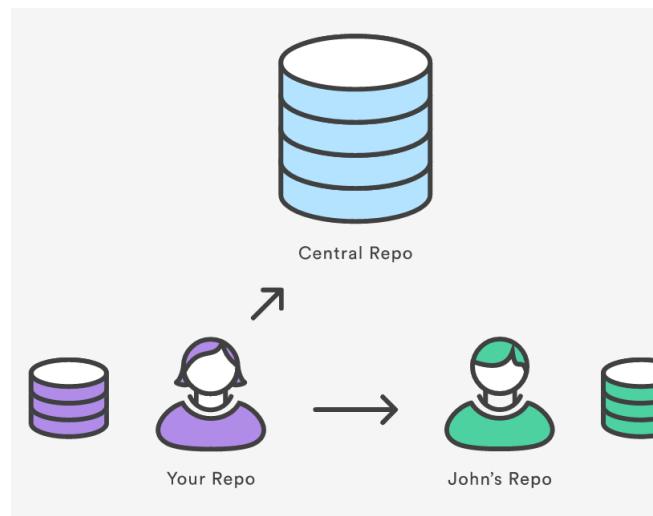
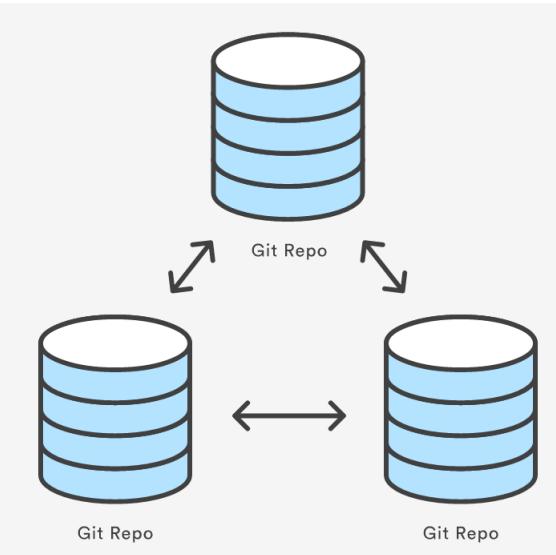
Dépôt distant (ou remote)

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 terminal output to the 'Clone or download' button, which is highlighted with a red box. The terminal output shows the command \$ git clone https://github.com/iblasquez/geekgit.git and the progress of the cloning process.

```
$ 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.
```

Branch: master ▾ New pull request

Clone with HTTPS ? Use SSH

Use Git or checkout with SVN using the web URL.

<https://github.com/iblasquez/geekgit.git>

Open in Desktop Download ZIP

iblasquez committed on GitHub Ajout lien poesie.md

README.md Ajout lien poesie.md

poesie.md Ajout poésie : Source Code Poetry Challenge

README.md

Isabelle BLASQUEZ

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

```
geekgit
poesie.md
README.md
.git

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

    Ajout lien poesie.md

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

    Ajout poésie : Source Code Poetry Challenge

commit fc8cd200702f6052bd81a474e234910171e94f86
Author: Isabelle BLASQUEZ <isabelle@[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

Travailler en local ...

espace de travail

geekgit

TXT poesie.md
TXT README.md
.git

espace de travail

geekgit

TXT peinture.md
TXT poesie.md
TXT README.md
.git

un commit ...

README.md

Hello les geeks !

Pour commencer un peu de poésie par [ici](poesie.md)

Et quelques peintures par [là](peinture.md)

Modifier
un fichier

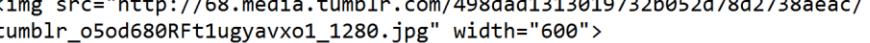
```
peinture.md ✘
# Peinture & Développement logiciel

## Classic Programmer Paintings

Connaissez-vous le site **Classic Programmer Paintings** ?  

[Classic Programmer Paintings is a hilarious resource with classic paintings featured with modern captions from the programming world.](http://mamchenkov.net/wordpress/2016/08/08/classic-programmer-paintings)  

Quelques exemples :  

[*Programmer finds 1395 conflicts after `git merge develop master`, three days before deadline*]  

Gustav Courbet  

1844-1845  

Oil paint  

(Collaboration form Mario)](http://classicprogrammerpaintings.com/post/143064709101/programmer-finds-1395-conflicts-after-git-merge)
```

Ajouter
un fichier

```
Isabelle@xps-Isa MINGW64 /c/demoRemote/geekgit (master)
$ git add peinture.md
```

```
Isabelle@xps-Isa MINGW64 /c/demoRemote/geekgit (master)
$ git commit -am "ajout page peinture avec classic programmer paintings"
[master 51b755d] ajout page peinture avec classic programmer paintings
1 file changed, 71 insertions(+)
create mode 100644 peinture.md
```

un autre commit ...

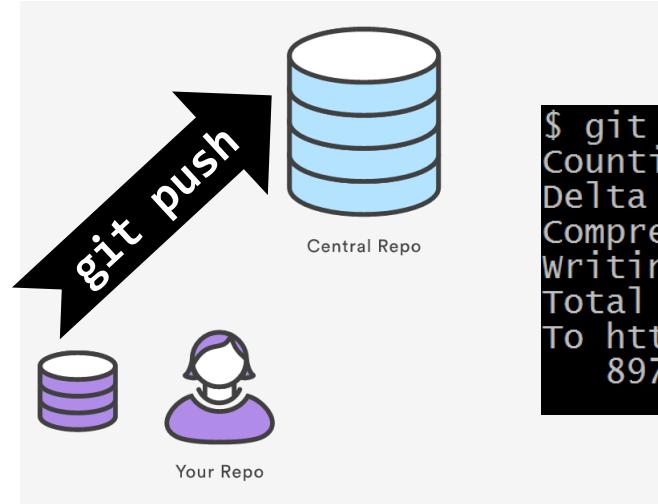
```
Isabelle@xps-Isa MINGW64 /c/demoRemote/geekgit (master)
$ git commit -am "maj README : lien page peinture"
[master 501f13b] maj README : lien page peinture
1 file changed, 2 insertions(+)
```

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

Chez mon client du jour 😊 #BlagueDeDev ?



Il existe même un plugin **git-fire** à découvrir sur <https://github.com/qw3rtman/git-fire> (Save Your Code in an Emergency)

Extraits de <https://twitter.com/agilex/status/836123942683308032>

<https://openclipart.org/detail/130795/trollface>

Voir aussi : https://www.reddit.com/r/ProgrammerHumor/comments/3nc531/in_case_of_fire

Affiche à imprimer depuis : http://abload.de/img/in_case_of_fireirrtb.jpg ou

<https://hikaruzone.wordpress.com/2015/10/06/in-case-of-fire-1-git-commit-2-git-push-3-leave-building>

Isabelle BLASQUEZ

Le dépôt distant a subi/reçu de nouvelles modifications ...

iblasquez / geekgit

Code Issues Pull requests Projects Wiki

dépôt utilisé pour le cours git

Add topics

5 commits 1 branch

Branch: master New pull request

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



iblasquez / geekgit

Code Issues Pull requests Projects Wiki Pulse

dépôt utilisé pour le cours git

Add topics

7 commits 1 branch 0

Branch: master New pull request

iblasquez committed on GitHub Maj README : lien page BD

README.md Maj README : lien page BD

bandeDessinee.md ajout bd : Commit Strip

peinture.md ajout page peinture avec classic programmer paintings

poesie.md Ajout poésie : Source Code Poetry Challenge

Synchroniser les dépôts : récupérer des commits distants (pull)

TIRER

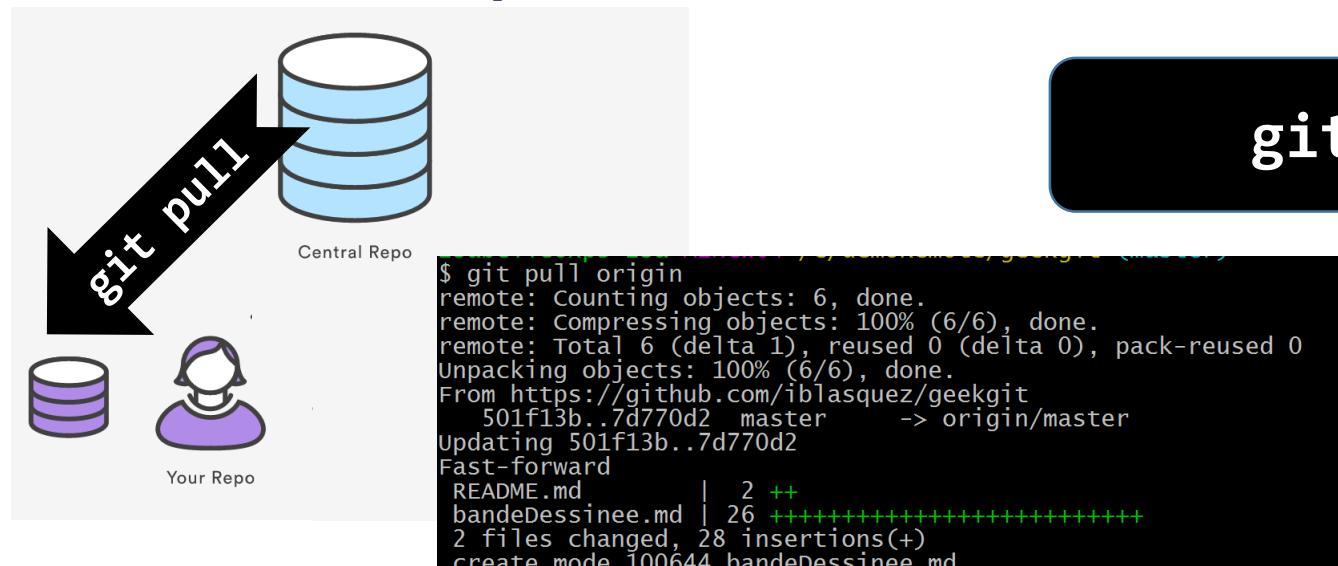
Télécharger les commits du dépôt distant de la branche courante...

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>

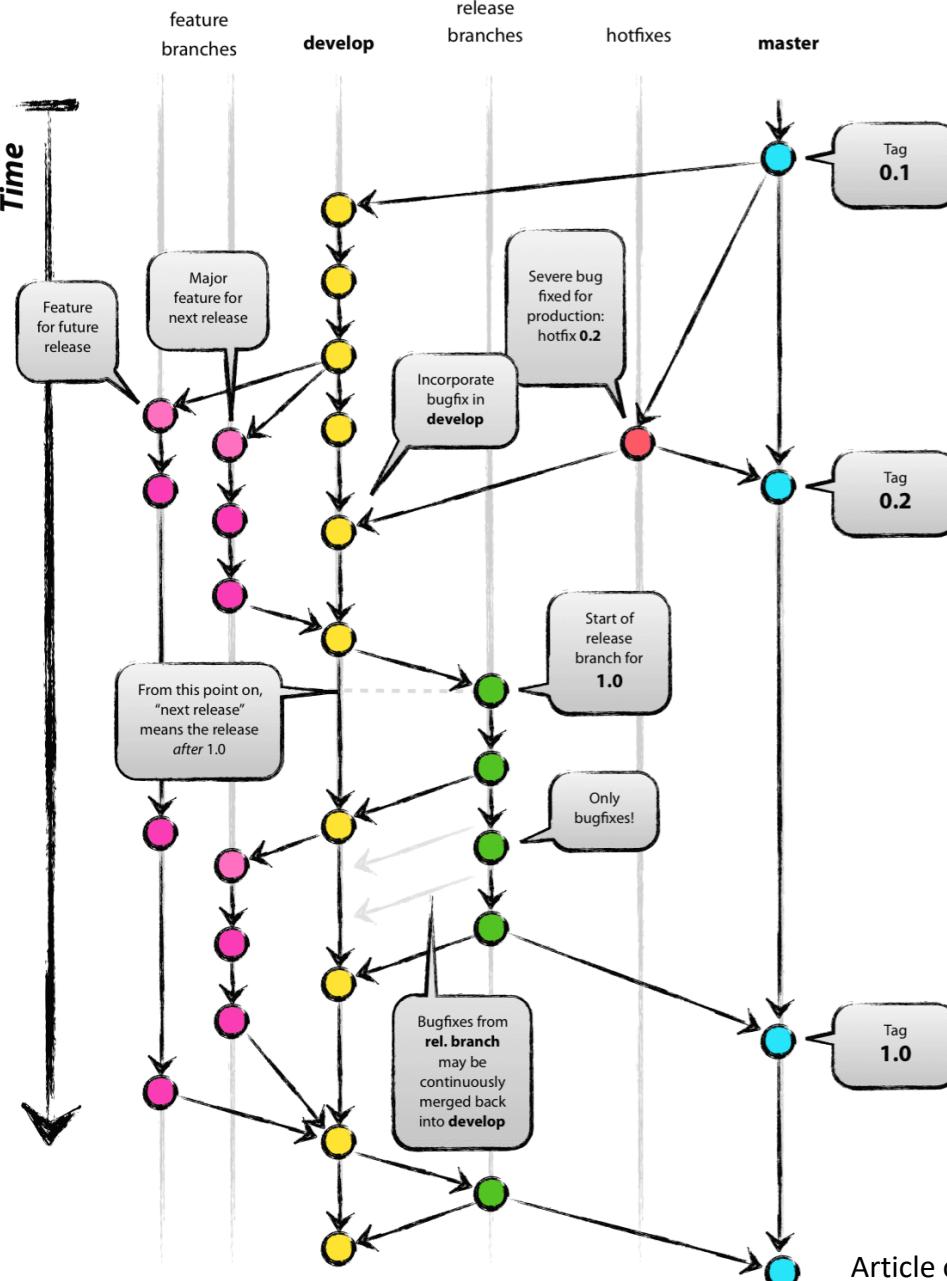
Remarque : Si un conflit se présente git merge --abort revient à l'état avant le pull.

Isabelle BLASQUEZ

Les workflows

(modèles de développement)

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



Les branches principale

→ **master sur origin**

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

→ **develop** (appelée aussi branche d'*intégration*)

(reflète 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.)

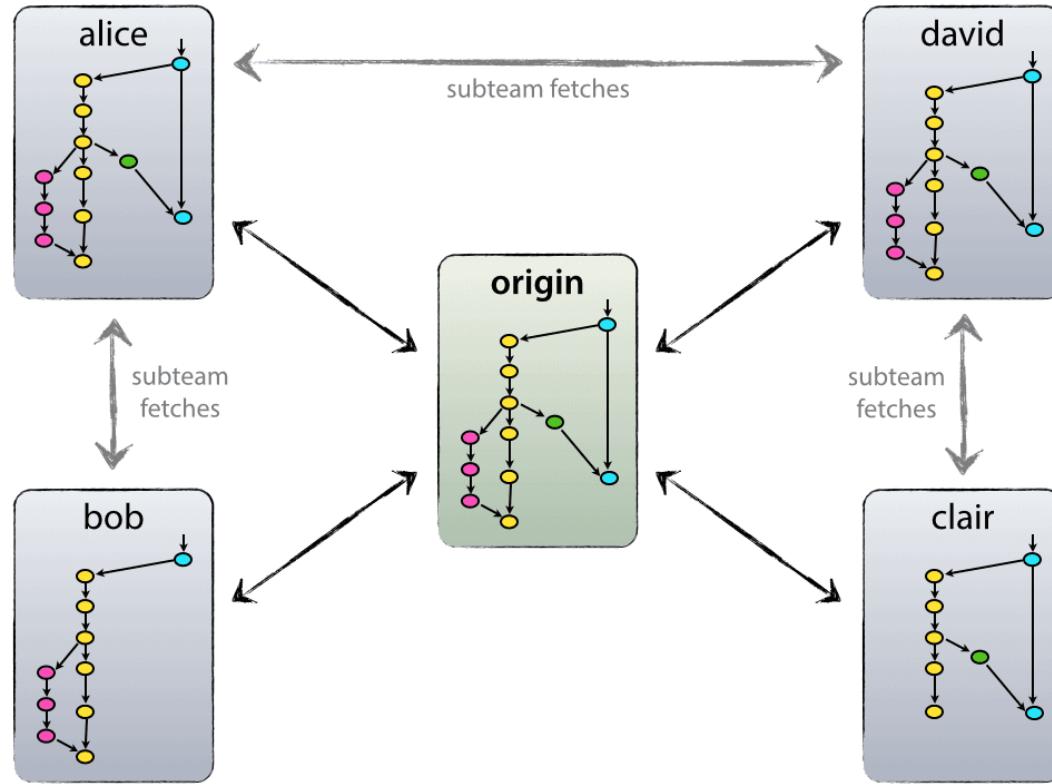
→ Les branches de **correctifs** (hotfixes)

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/>

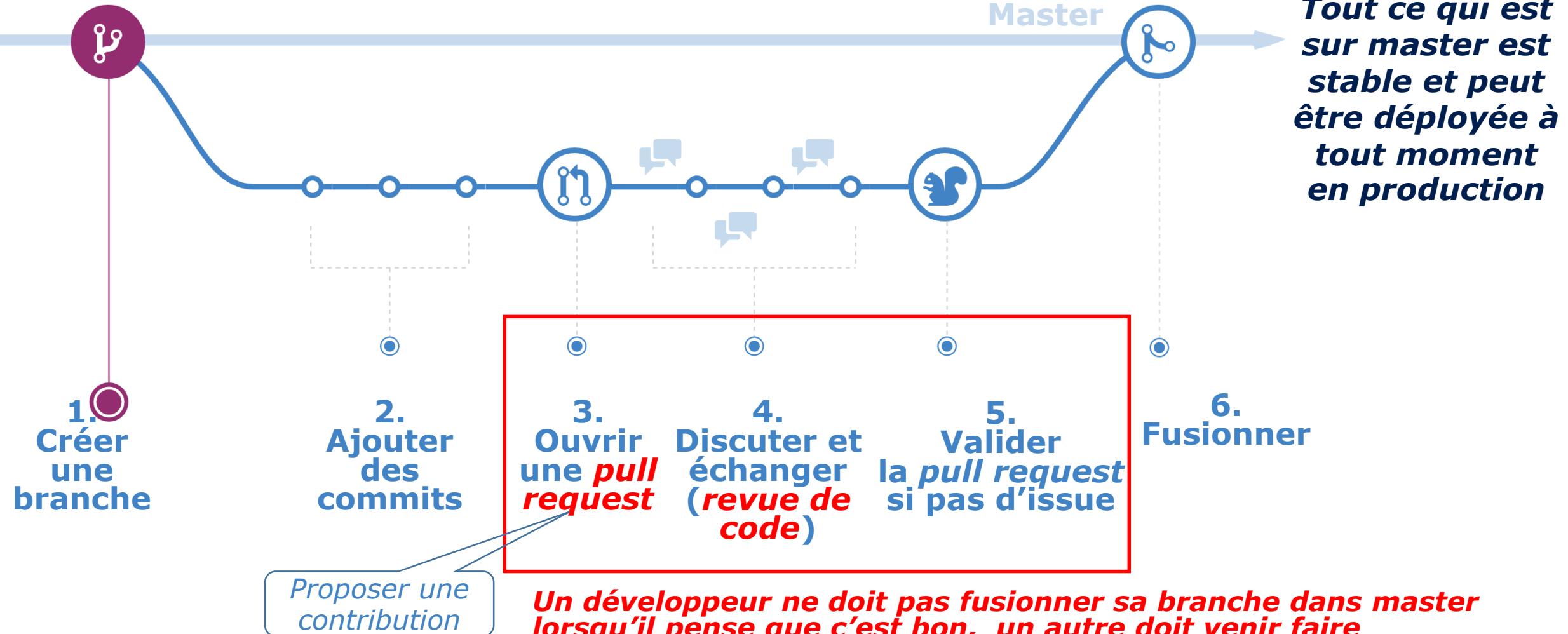
Git Flow : un workflow à base de branches (2/2)



- 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.

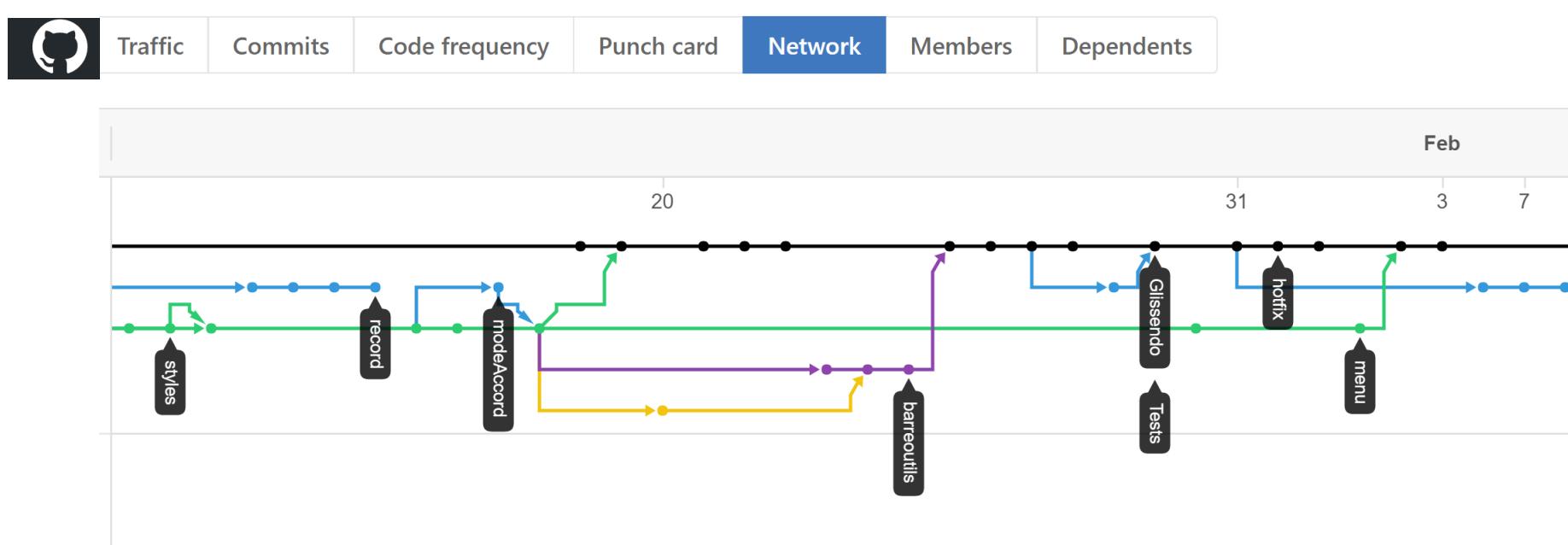
GitHub Flow : un workflow branché simplifié ...

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



Un développeur ne doit pas fusionner sa branche dans master lorsqu'il pense que c'est bon, un autre doit venir faire une revue du code et confirmer la stabilité de la branche.

Un exemple de workflow mis en place lors d'un projet tuteuré



1 fonctionnalité => 1 branche

Periodic table of devops tools

Autres outils autour de la gestion de version

XebiaLabs
Deliver Faster

Deliver Faster

 Follow @xebialabs

Carte interactive disponible sur : <https://xebialabs.com/periodic-table-of-devops-tools/>

The Complete DevOps Glossary

DevOps and Continuous Delivery Terms Defined!

A compléter par :

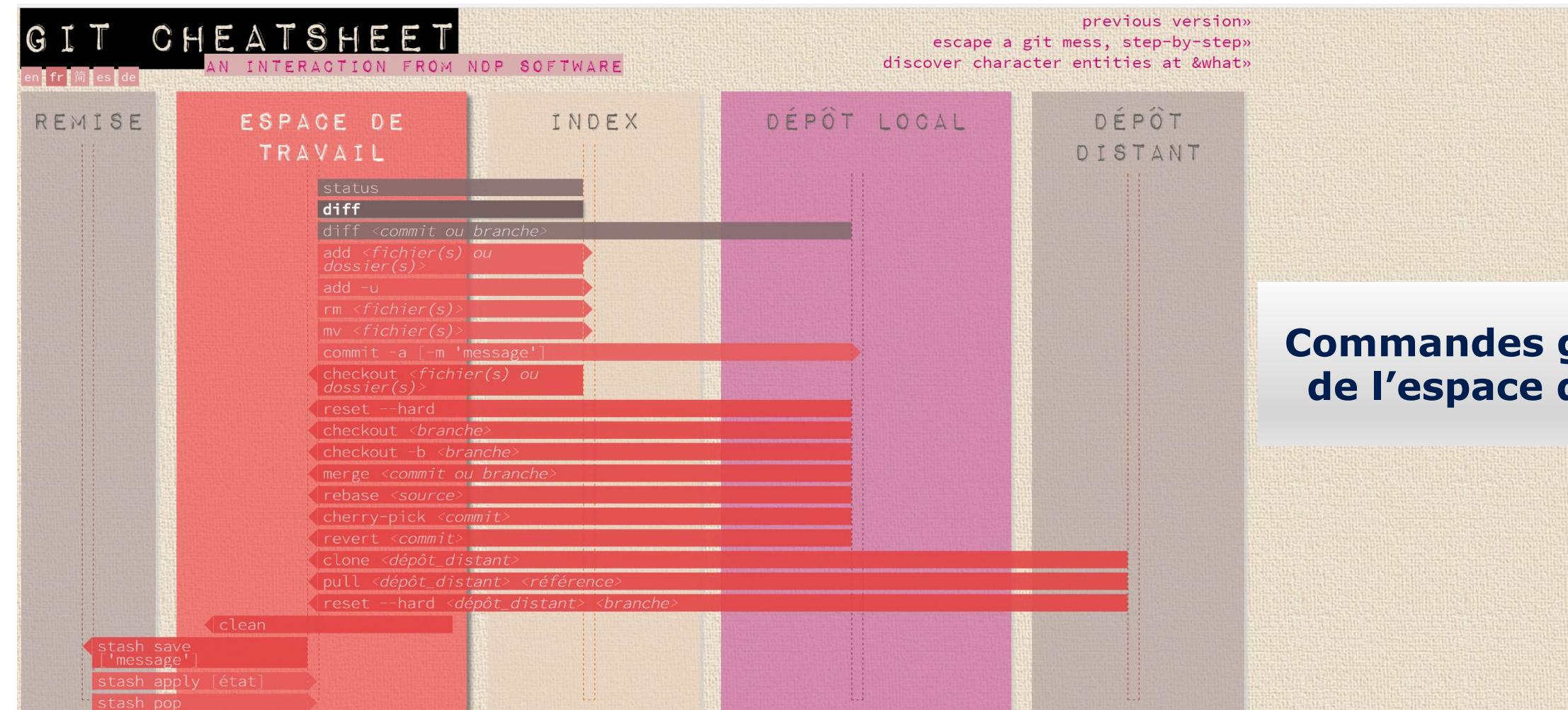
<https://xebialabs.com/glossary/>

Isabelle BLASQUEZ

Annexes

Un mémo interactif (1/2)

<http://ndpsoftware.com/git-cheatsheet.html>



Commandes git autour de l'espace de travail

(c) Andrew Peterson 2009–2016 All Rights Reserved.

Isabelle BLASQUEZ

Un mémo interactif (2/2)

<http://ndpsoftware.com/git-cheatsheet.html>



Commandes git autour du dépôt distant

Reste à découvrir en ligne...
commandes git autour :
→ de la remise
→ de l'index
→ du dépôt local

(c) Andrew Peterson 2009–2016 All Rights Reserved.

Aide-mémoire de commandes git

<https://services.github.com/on-demand/downloads/github-git-cheat-sheet.pdf>

<https://education.github.com/git-cheat-sheet-education.pdf>

<https://zeroturnaround.com/rebellabs/git-commands-and-best-practices-cheat-sheet>

<http://rogerdudler.github.io/git-guide/index.fr.html>

Bien nommer sous git :

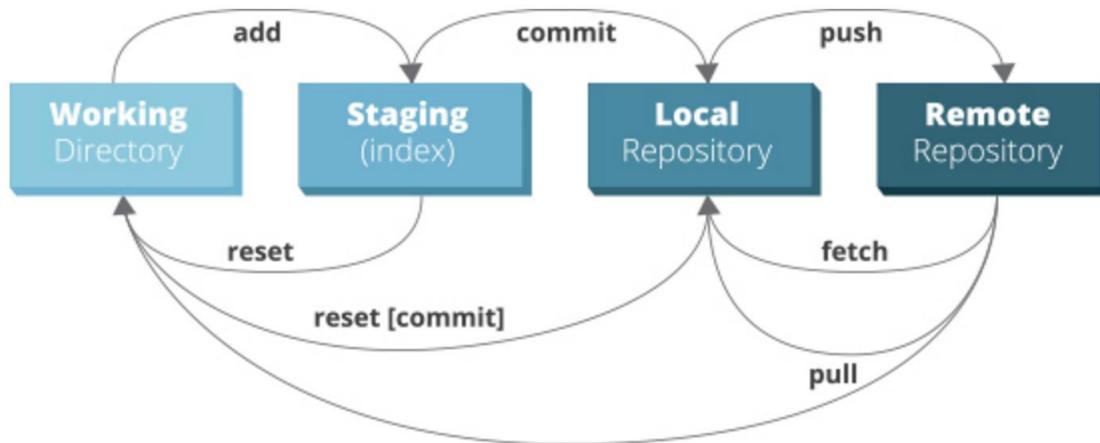
→ **Guide de style git :** <https://github.com/piererroth64/git-style-guide>

→ **How to use GitHub like a proper human being :**

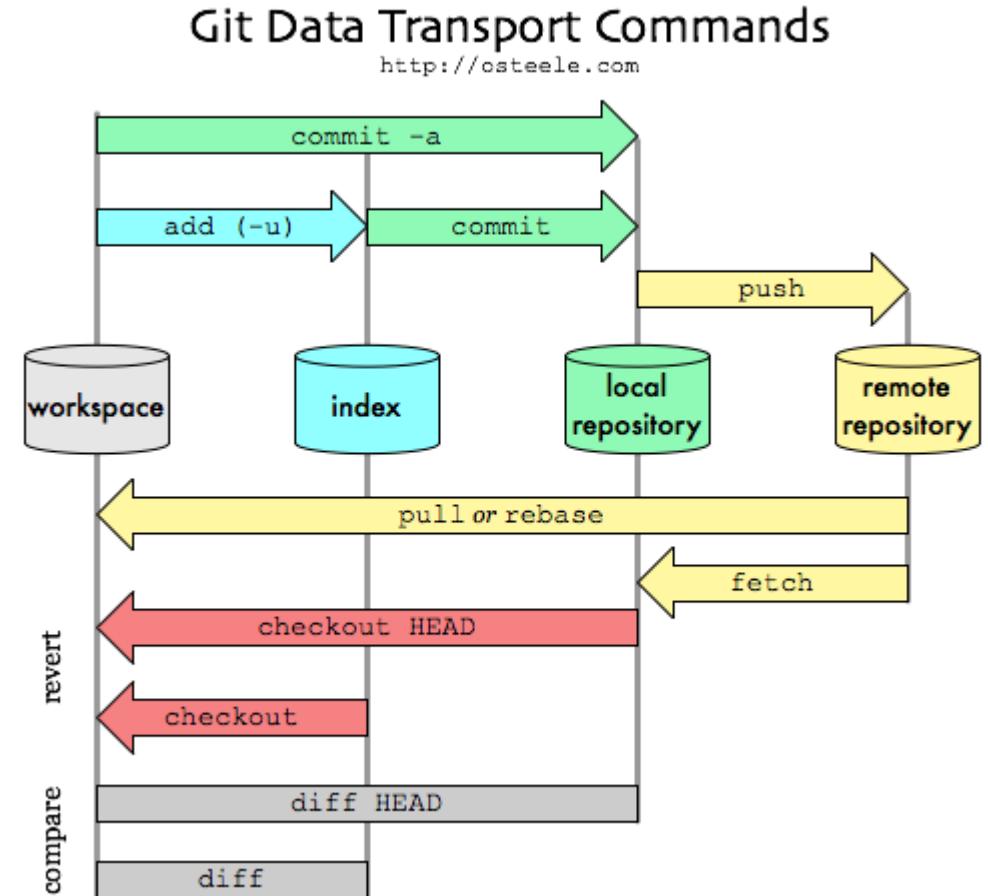
<https://stories.devacademy.la/how-to-use-github-like-a-proper-human-being-1a9c895c4e13#.y5owzljfh>

Time for a Change

Let's make a change to our code — first we'll... whooaooaaaa there! We're going to use terminology here which we'll need to explain and point at a state diagram to understand fully. Check out this image:



Extrait : <https://zeroturnaround.com/rebellabs/git-commands-and-best-practices-cheat-sheet/>



Extrait : <http://stackoverflow.com/questions/2745076/what-is-the-difference-between-git-commit-and-git-push>

commit: adding changes to the local repository
push: to transfer the last commit(s) to a remote server

Git : Quelques liens ...

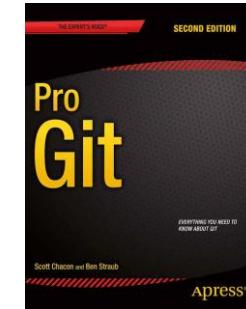


✓ **Git : le site de référence :** <https://git-scm.com>

✓ **Documentation**

→ **Livre Pro Git Book**

version française en ligne : <https://git-scm.com/book/fr/v2>



→ **Git Wiki Homepage :** <https://git.wiki.kernel.org>

(à voir aussi sur ce wiki un sondage sur l'utilisation de Git en 2016: <https://git.wiki.kernel.org/index.php/GitSurvey2016>)

✓ **Liste de projets sous Git :**

GitProjects

Projects that use Git for their source code management.

If your project is not listed, please add it here.

List does not currently include [Git](#) or git-related tools; see [InterfacesFrontendsAndTools](#) page for those.

Kernel-related projects that use Git for their source code management.

- Linux Kernel
 - <http://kernel.org> (gitweb)
 - <http://git.kernel.org/?p=linux/kernel/git/torvalds/linux-2.6.git> (gitweb)
- Linux/MIPS (port of Linux to the MIPS architecture)
 - <http://www.linux-mips.org/wiki/Git> (description, includes repository URLs)
 - <http://www.linux-mips.org/git> (gitweb)
- NFS client patches for Linux
 - <http://linux-nfs.org/> (homepage)
 - <http://linux-nfs.org/cgi-bin/gitweb.cgi> (gitweb)
- OpenVZ (Container virtualization for Linux)

... toute la liste à découvrir sur <https://git.wiki.kernel.org/index.php/GitProjects>

Références sur git

Git

- Site de référence : <https://git-scm.com>
Documentation sur le site de référence : <https://git-scm.com/doc>
- Pro Git book (guide officiel) en version française
- Listes d'outils permettant d'utiliser git en mode graphique : [ici](#) et [là](#)
- A propos de .gitignore
 - La rubrique gitignore dans la documentation de git
 - A collection of useful .gitignore templates
 - A .gitignore file for IntelliJ and Eclipse with Maven
- Bien nommer sous git
 - Guide de style git
 - How to use GitHub like a proper human being
- Aide mémoire de commandes git
 - <https://services.github.com/on-demand/downloads/github-git-cheat-sheet.pdf>
 - <https://education.github.com/git-cheat-sheet-education.pdf> <https://zeroturnaround.com/rebellabs/git-commands-and-best-practices-cheat-sheet/> <https://www.git-tower.com/blog/git-cheat-sheet/>
- A propos des branches
 - Utiliser des branches Rubrique Branching in Eclipse du tutorial de vogella.com
- Merge et Rebase
 - Bien utiliser Git merge et rebase
 - Comparaison entre un merge et un rebase
 - Resolving a merge conflict : wiki Eclipse
 - Git++ : Passez au niveau supérieur de la gestion de version et une vidéo [ici](#) et [là](#)
 - Exercices interactifs pour apprendre le branching sous Git
- Autres
 - Reset, checkout et revert
 - Tagging
- Comment créer des dépôts GIT distants sans accès réseau

Egit (Tutorial complet)

EGit/User Guide sur le wiki d'eclipse.org EGit Tutorial sur eclipsesource.com
Git version control with Eclipse (EGit) - Tutorial sur vogella.com

Github

- Guides Github
- Aide en ligne Github : [ici](#) ou [là](#)
- Mastering Markdown

Autres

- The Designers Guide to Git
- Présentation de Git sous Wikipedia
- Git et Github

Fork & Co (contribuer à un projet existant)

- Forking Projects
- Ma méthode de travail avec Git et GitHub qui est une traduction de Documenting my git/GitHub workflow
- Chapitre 6.2 : GitHub - Contribution à un projet du Pro Git book

Exercices interactifs pour apprendre à manipuler les commandes git

- Got 15 minutes and want to learn Git? (les commandes de bases : lisez, cliquez, comprenez !)
- Learn Git Branching (apprendre le branching sous git)

Tutoriels git avec Eclipse

- Premier projet dans Eclipse avec Bitbucket (José Paumard)
- Egit : quand git s'invite dans Eclipse (o penclassrooms)
- GitHub avec Eclipse et "eGit" en 15 minutes (Laurent Guérin)
- Utiliser Git avec Eclipse (Yannick Le Goff)
- Introduction à Git (Pierre-Antoine Champin)

Des vidéos à ne pas manquer ...

- Git and GitHub for Poets (une série de courtes vidéos par Coding Train) : [ici](#)
- Git++ : Passez au niveau supérieur de la gestion de version : [ici](#) et [là](#) avec les transparents [ici](#)

Retrouvez cette page de références disponible dans le dépôt :

https://github.com/iblasquez/tuto_git avec un tutoriel de découverte de git au travers d'EGit

Isabelle BLASQUEZ

Références sur les workflows ...

Le modèle GitFlow (workflow de base):

→ 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>

Le modèle GithubFlow (workflow simplifié)

<https://guides.github.com/introduction/flow/>

Quel git workflow pour mon projet ?

<http://www.nicoespeon.com/fr/2013/08/quel-git-workflow-pour-mon-projet>

Ma méthode de travail avec Git et GitHub

<https://tech.mozilla.org/post/2016/04/16/Ma-methode-de-travail-avec-Git-et-GitHub>

Intégration et déploiement en continu avec Github (ou comment github utilise git...)

<https://www.youtube.com/watch?v=jCwzf9adAtE>

La puissance des workflows git

<https://medium.com/@OVHUXLabs/la-puissance-des-workflows-git-12e195cafe44#.rbkmhjcc2>



Git and GitHub for Poets

The Coding Train • 9 vidéos • 18 946 vues • Dernière modification le 1 août 2016

Tout regarder

Partager

Enregistrer

1.1: Introduction - Git and GitHub for Poets

de The Coding Train

1.2: Branches - Git and GitHub for Poets

de The Coding Train

1.3: Forks and Pull Requests - Git and GitHub for Poets

de The Coding Train

1.4: GitHub Issues - Git and GitHub for Poets

de The Coding Train

1.5: Intro to the Command Line - Git and GitHub for Poets

de The Coding Train

1.6: Cloning Repo and Push/Pull - Git and GitHub for Poets

de The Coding Train

1.7: git init and git add - Git and GitHub for Poets

de The Coding Train

1.8: GitHub Pages - Git and GitHub for Poets

de The Coding Train

9.10: GitHub Pages for Hosting p5.js Sketches - p5.js Tutorial

de The Coding Train

Vidéos pour comprendre facilement git et Github

Vidéos à regarder sur :

<https://www.youtube.com/playlist?list=PLRqwX-V7Uu6ZF9C0YMKuns9sLDzK6zoiV>

sur la chaîne de <https://www.youtube.com/user/shiffman>



Détail du dernier commit

auteur du commit

date de commit

message de commit

commit parent

diff par rapport à la dernière sauvegarde

hash du commit

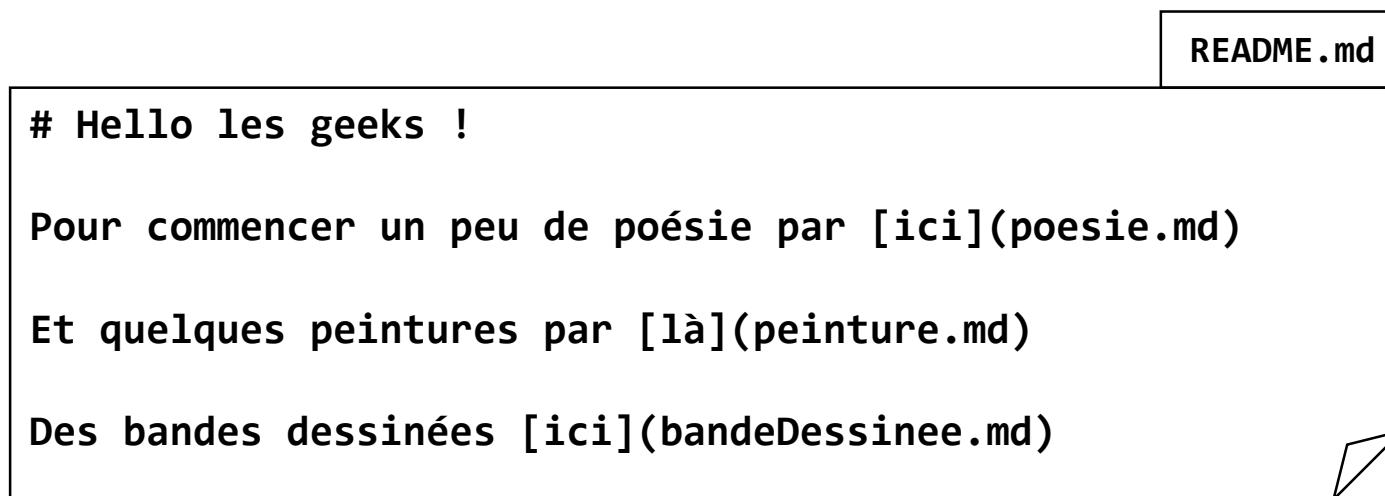
```
$ git show  
commit 908563408714f417561a443e4766821c30b25ec6  
Author: Isabelle BLASQUEZ <isabelle.blasquez@...>  
Date:   Fri Feb 17 11:10:45 2017 +0100  
  
ajout git  
  
diff --git a/test.txt b/test.txt  
index 306c18d..9e77514 100644  
--- a/test.txt  
+++ b/test.txt  
@@ -1 +1,2 @@  
-Bonjour tout le monde !  
\ No newline at end of file  
+Bonjour tout le monde !  
+J'utilise git !  
\ No newline at end of file
```

show

Fichiers utilisés dans ce cours pour illustrer l'interaction avec un dépôt distant

(si vous voulez rejouer les exemples pour vous entraîner ...)

<https://github.com/iblasquez/geekgit>



The screenshot shows the same GitHub repository page for 'geekgit'. The commit history is displayed in a table format:

Commit	Description
iblasquez committed on GitHub Maj README : lien page BD	Maj README : lien page BD
README.md	Maj README : lien page BD
bandeDessinee.md	ajout bd : Commit Strip
peinture.md	ajout page peinture avec classic programmer paintings
poesie.md	Ajout poésie : Source Code Poetry Challenge
README.md	

The commit for 'README.md' has been highlighted with a callout box containing the text 'Hello les geeks !'.

Hello les geeks !

Pour commencer un peu de poésie par [ici](#)
Et quelques peintures par [là](#)
Des bandes dessinées [ici](#)

Amis de la poésie !

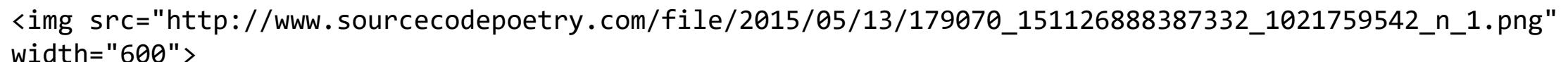
> [**Code poetry** is literature that intermixes notions of classical poetry and computer code.(extrait Wikipedia)](https://en.wikipedia.org/wiki/Code_poetry)

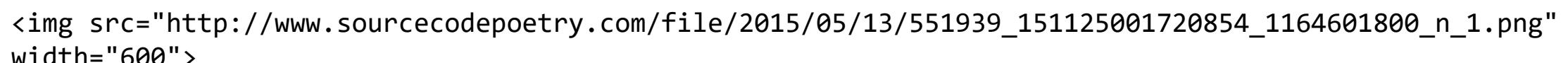
Source Code Poetry Challenge

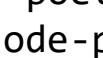
Connaissez-vous le **source code poetry challenge** qui consiste à écrire un poème de manière à ce que :

➤ The poetry must properly compile in any programming language.

Quelques exemples de soumissions des années précédentes :

A screenshot of a web page showing a complex program structure that forms a poem. The code is written in a programming language like Python or C, where the syntax itself creates the words and structure of a poem.

A second screenshot of a source code poetry submission, showing a different programmatic poem structure. The code is again designed to look like a poem when run or viewed.

Pour en savoir plus, rendez-vous sur le site : <http://www.sourcecodepoetry.com/> et jetez un petit coup d'oeil sur le livre : [./code --poetry](<https://leanpub.com/code-poetry>) 

--

Autre pratique, autre challenge : autre lien à proposer ?

Peinture & Développement logiciel

Classic Programmer PaintingsConnaissez-vous le site **Classic Programmer Paintings** ?

[Classic Programmer Paintings is a hilarious resource with classic paintings featured with modern captions from the programming world.](<http://mamchenkov.net/wordpress/2016/08/08/classic-programmer-paintings>)

Quelques exemples :

[*Programmer finds 1395 conflicts after `git merge develop master` , three days before deadline*
Gustav Courbet

1844-1845

Oil paint

(Collaboration form Mario)](<http://classicprogrammerpaintings.com/post/143064709101/programmer-finds-1395-conflicts-after-git-merge>)

[*The new project manager*

Francisco Goya

1787-88

Oil on canvas] (<http://classicprogrammerpaintings.com/post/14550665554/the-new-project-manager-francisco-goya-178788>)

[*Haskell meetup*
Edward Hopper
Oil on canvas
1942](<http://classicprogrammerpaintings.com/post/143847262458/haskell-meetup-edward-hopper-oil-on-canvas>)

[*Resolving technical debt*
Accumulated technical debt is visible in the middle image. Right image after refactoring.
About 1930 Elías García Martínez, refactored by Cecilia Giménez Fresco
Composition image CC-BY 2.0 by cea+ in Flickr <https://www.flickr.com/photos/33255628@N00/7923536516> (collaboration Tero Kinnunen)](<http://classicprogrammerpaintings.com/post/143537166294/resolving-technical-debt-accumulated-technical>)

[*Multiple inheritance in C++*
ca 1590 - Jacopo Ligozzi
Pen and brush and brown ink over pencil, brown wash, white lead
(collaboration from Jakub Konecki)](<http://classicprogrammerpaintings.com/post/143486406473/multiple-inheritance-in-c-ca-1590-jacopo>)

**La suite sur le site : <http://classicprogrammerpaintings.com/> et/ou sur le compte twitter
[@progpaintings](<https://twitter.com/progpaintings>) **

--

Autre lien à proposer ?

Bande dessinée & Développement logiciel

Commit Strip

Connaissez-vous ****Commit Strip**** ? Des webcomics sur la vie des codeurs ...

Quelques exemples :

Extrait de : <http://www.commitstrip.com/fr/2013/11/05/git-svn-ou/>

Extrait de : http://www.commitstrip.com/fr/2017/03/10/anyone-can-be-wrong-sometimes/

Extrait de : <http://www.commitstrip.com/fr/2017/02/08/where-are-the-tests/>

**La suite sur le site : <http://www.commitstrip.com> et/ou sur les comptes twitter
[@CommitStrip](https://twitter.com/CommitStrip) et [@CommitStrip_fr](https://twitter.com/CommitStrip_fr)**

--

Autre lien à proposer ?