

Introduction à git

Comment fusionner le travail de plusieurs membres d'une équipe de dev, travaillant sur la même base de code ? A l'aide d'un **gestionnaire de versions**. Git est le plus populaire en entreprise : il sauvegarde l'historique des modifications du projet et permet de fusionner le travail proprement.

Git est installé **en local** et se synchronise avec un **dépôt distant** (ou repository - repo) hébergé sur une plateforme comme gitlab, github, gitea, bitbucket, etc.

Préparation

1. Installez Git : <https://git-scm.com/downloads> ;
2. Lancez un terminal ou `git bash` et tapez `git --version` pour afficher la version installée.

Utilisation

3. Créez un projet sous Visual / Code blocks / vscode / whatever ;
4. Ajoutez un `main.cpp` auquel vous intégrez un `cout << helloworld` ;
5. Compilez et lancez le projet ;
6. A la racine du projet, tapez `git init` : les fichiers vont être indexés ;
7. Tapez `git status` : vous voyez apparaître les modifications à prendre en compte ;
8. On remarque beaucoup de fichiers qui ne sont **pas du code source** : les fichiers compilés, les binaires, les fichiers de l'IDE, etc. Il faut les **exclude** de git. Créez un fichier `.gitignore` à la racine et listez les fichiers à ignorer (un par ligne) : `.exe`, `Debug`, `.vs`, `.vcxproj`, `.sln` ;
9. Tapez au fur et à mesure `git status` : les fichiers listés ont disparu des fichiers indexés.

Premier commit

Un commit est une sorte de photo de votre code, que vous souhaitez "enregistrer".

10. Tapez `git add main.cpp .gitignore` pour intégrer ces fichiers au prochain commit ;
11. Tapez `git status` : `main.cpp` et `.gitignore` sont en vert, prévus au prochain commit ;
12. Tapez `git commit -m "Initialisation"` pour créer un commit ;
13. Tapez à nouveau `git status` : il n'y a plus de changements depuis le dernier commit ;
14. Tapez `git log` : votre dernier commit apparaît.

Bonne pratique : faire des commits **très petits, unitaires, explicitement nommés**.

Synchronisation avec un repo distant

15. Créez un compte sur gitlab ou github ;
16. Créez un projet : vide, avec pour nom : "Git lo21 tuto", privé, non initialisé, sans readme ;
17. Configurez git en tapant dans votre terminal `git config user.name "votre_nom"` et `git config user.email "votre_email"` ;
18. Liez votre travail local à votre projet gitlab en tapant dans votre terminal `git remote add origin https://gitlab.com/user/projet` (l'URL dans sur votre navigateur) ;
19. Tapez `git push -u origin master` (ou `main`) : votre travail local va être envoyé sur gitlab ;
20. Rafraîchissez la page de votre projet sur gitlab, vous verrez vos fichiers..

Modifications supplémentaires

21. Ajoutez un fichier `fonction.h` contenant `void hello();`
22. Définissez librement la fonction `void hello()` dans `main.cpp` ;
23. Envoyez vos modifications sur gitlab :
 - a. `git status` : `main.cpp` et `fonction.h` doivent être écrits en **rouge** ;
 - b. `git add main.cpp fonction.h`
 - c. `git status` : `main.cpp` et `fonction.h` doivent être écrits en **vert** ;
 - d. `git commit -m "Added fonction.h and hello()"`
 - e. `git status` : la liste doit être maintenant vide ;
 - f. `git log` : vos deux commits doivent être visibles
 - g. `git push`
24. Vérifiez l'état du projet sur gitlab.

Partage d'un projet

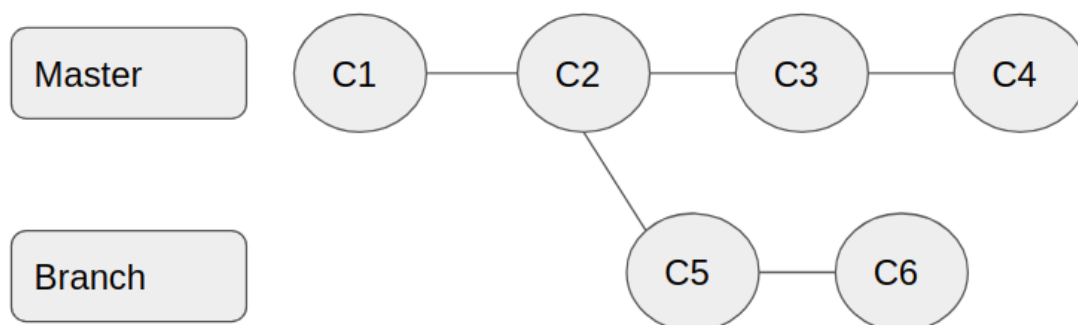
Lorsque vous travaillez à plusieurs, utilisez `git clone` <https://gitlab.com/user/projet.git> pour importer localement le projet dans un répertoire.

Branches

Une branche est une dérivation du code pouvant contenir plusieurs commits. En général, chacun·e travaille sur sa branche pour traiter une fonctionnalité particulière. Les branches sont fusionnées lorsqu'elles sont prêtes. La branche principale est nommée `Master` ou `Main`.

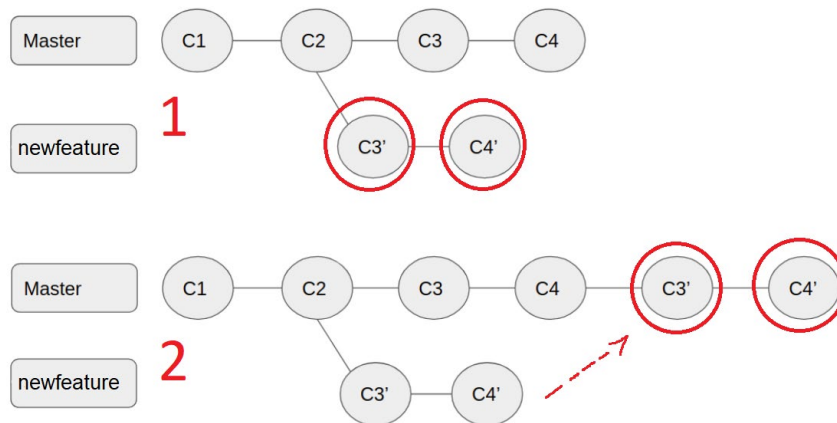
25. Tapez `git checkout -b newfeature` pour créer une branche `newfeature` ;
26. Ajoutez à `fonction.h` le prototype d'une fonction `void newfeature();`
27. Créez un commit avec le message `Added prototype newfeature();`
28. Définissez librement la fonction `void newfeature()` dans `main.cpp` ;
29. Créez un commit avec le message `Defined newfeature();`
30. Poussez cette branche sur gitlab ;
31. Note : vous pouvez lister les branches locales avec `git branch`.

Bonne pratique : une branche a un objectif unique et précis, créez **une branche par fonctionnalité**.



Fusion de branches

Lorsque la fonctionnalité d'une branche est validée, il faut la fusionner dans `master`.



Sur gitlab (ou github) :

32. Ouvrez une `merge request - MR` (ou `pull request - PR`) ;
33. Choisissez de `merge` la branche `newfeature` → `master` ;
34. Observez les changements dans l'onglet `Changes` ou `Files changed` ;
35. Validez.

Sur votre terminal :

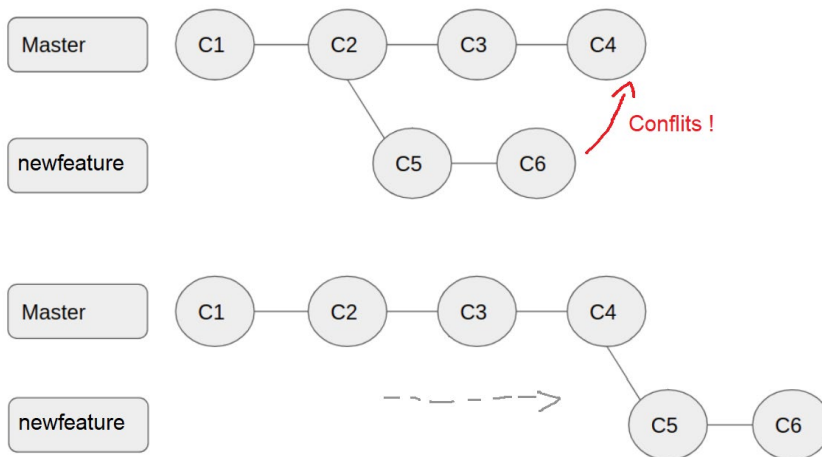
36. Tapez `git checkout master` pour revenir sur la branche `master` ;
37. Tapez `git pull` pour récupérer la dernière version de `master` (modifiée avec la MR) ;
38. Tapez `git log` pour vérifier que les commits de `newfeature` sont bien présents.

Conflits et Rebase

Lorsque vous tentez de fusionner des branches, il peut apparaître des conflits. Cela signifie que vous avez modifié les mêmes lignes de code dans les 2 branches : par exemple, l'une a modifié le nom d'une variable quand l'autre a modifié son type. Git ne peut alors réconcilier automatiquement.

Pour corriger les conflits, vous pouvez agir directement sur gitlab et github avec le `mode interactif` ou le `inline editor`.

Cependant, pour être plus rigoureux, utilisez `git rebase`. Cette commande permet d'ajouter les commits manquants d'une branche vers l'autre. Elle vous prévient d'éventuels conflits que vous aurez alors à traiter en local, sur votre branche, avant d'aller plus loin.



39. Créez une branche `hello-v2` ;
40. Modifiez la fonction `hello()`, affichez "salut" ;
41. Créez un commit et poussez sur gitlab ;


Nous allons simuler une modification simultanée de `hello()` par un autre membre du groupe :

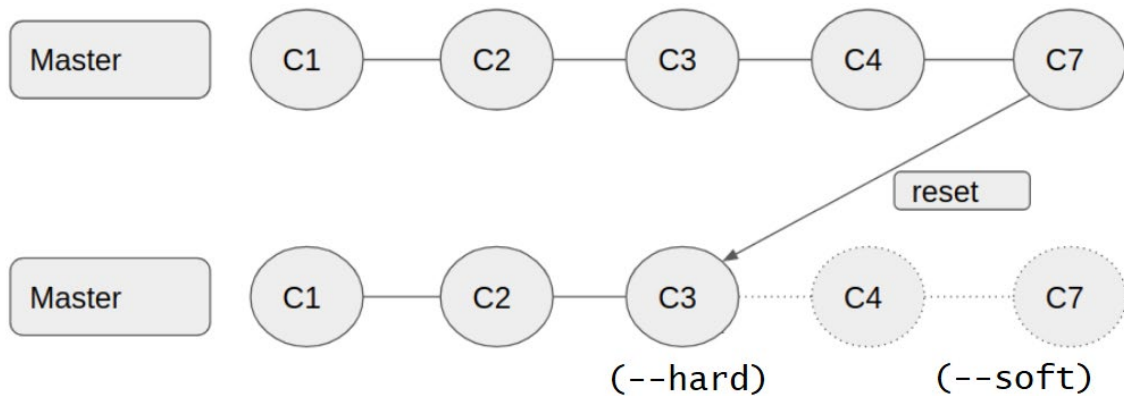
42. Tapez `git checkout master` pour revenir sur la branche `master` ;
43. Modifiez la fonction `hello()`, affichez "BONJOUR A VOUS" ;
44. Créez un commit et poussez sur gitlab ;

Nous tentons maintenant de fusionner `hello-v2` dans `master` :

45. Créez sur gitlab une MR de `hello-v2` vers `master` ;
46. La MR est refusée car elle génère un conflit sur la `hello()` ;
47. En local, revenez sur `hello-v2` avec `git checkout hello-v2` ;
48. Lancez le rebase avec `git rebase master` : le commit de `master` manquant est ajouté à `hello-v2`, mais des conflits sont remontés ;
49. Traitez les conflits manuellement dans le code : ils apparaissent généralement en bleu ;
50. Ajoutez les modifications créées pour le traitement des conflits avec `git add xxx` ;
51. Finalisez le rebase avec `git rebase --continue` ;
52. Vérifiez que vous voyez bien le commit de `master` dont vous manquez avec `git log` ;
53. A la fin du rebase, poussez votre branche sur gitlab avec `git push` : cela ne fonctionne pas, en effet, vous avez modifié l'historique de votre branche `hello-v2`, git est perdu ;
54. Utilisez alors `git push --force-with-lease`, qui ignore l'avertissement de git ;
55. Actualisez la MR sur gitlab : elle est maintenant fusionnable.

Autre commande utile : `git reset`

- `git reset 28d41e7` permet de revenir en arrière au commit `28d41e7` ;
- `git reset HEAD~2` permet de revenir en arrière de 2 commits ;
- `git reset origin/newfeature` réinitialise la branche à l'état où elle est sur gitlab ;
- l'option `--hard` oublie toutes les modifications que vous auriez effectuées depuis ;
- l'option `--soft`  conserve vos modifications en "non commitées" (option par défaut).



aller plus loin

Interfaces graphiques pour visualiser les branches (déconseillé) :

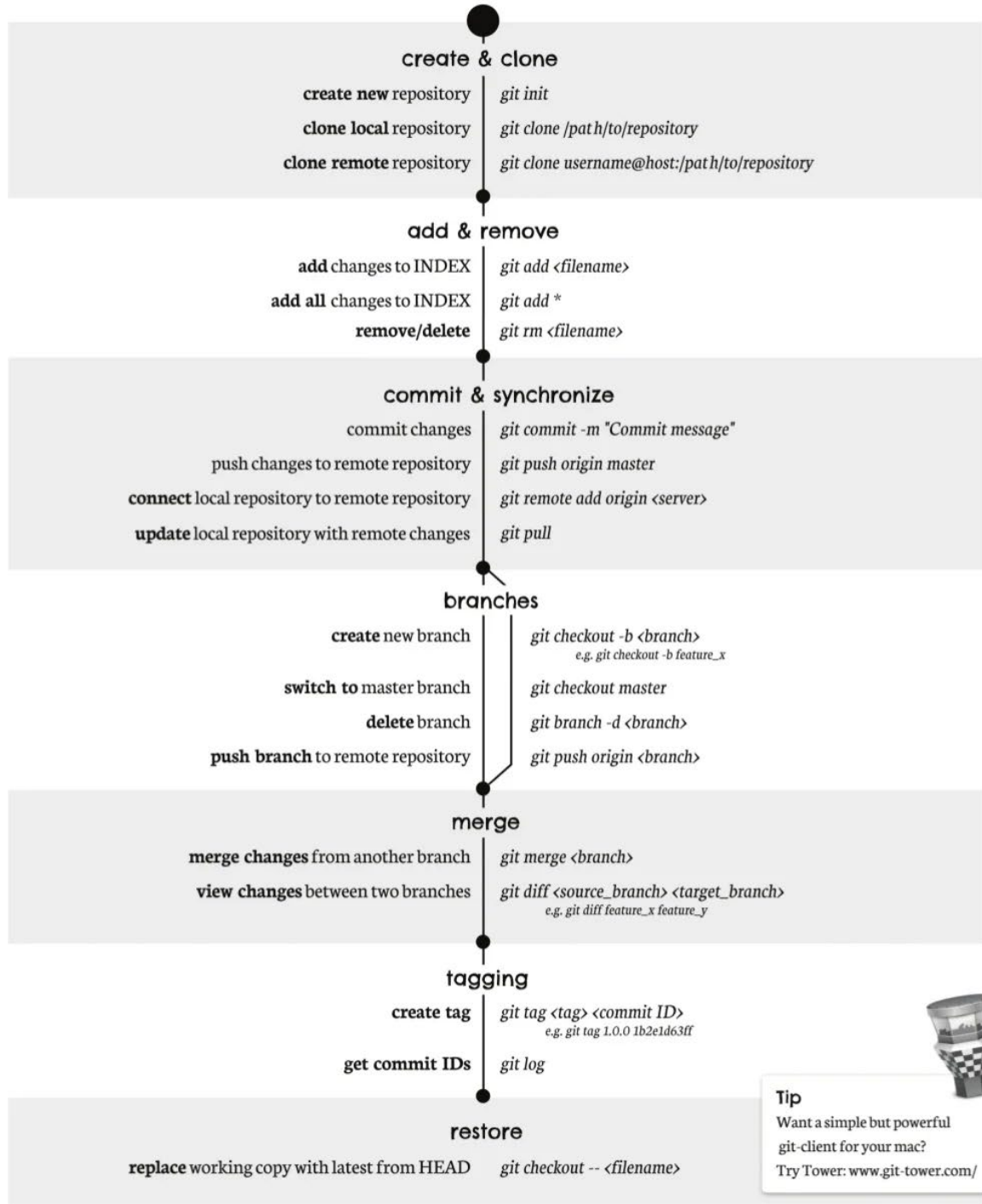
- <https://www.gitkraken.com/> ;
- <https://gitextensions.github.io/> ;
- **Continuez à taper les commandes manuellement**, sinon vous ne maîtriserez rien.

Quelques autres ressources :

- <https://openclassrooms.com/fr/courses/1233741-gerez-vos-codes-source-avec-git>
- <http://www-igm.univ-mlv.fr/~dr/XPOSE2011/foucault/concepts.php>
- <https://blog.lesieur.name/comprendre-et-utiliser-git-avec-vos-projets/>
- <https://www.hostinger.fr/tutoriels/tuto-git/>

git cheat sheet

learn more about git the simple way at rogerdudler.github.com/git-guide/
cheat sheet created by Nina Jaeschke of ninagrafik.com



Tip

Want a simple but powerful
git-client for your mac?
Try Tower: www.git-tower.com/

GIT CHEAT SHEET

presented by Tower - the best Git client for Mac and Windows



CREATE

Clone an existing repository

```
$ git clone ssh://user@domain.com/repogit
```

Create a new local repository

```
$ git init
```

LOCAL CHANGES

Changed files in your working directory

```
$ git status
```

Changes to tracked files

```
$ git diff
```

Add all current changes to the next commit

```
$ git add .
```

Add some changes in <file> to the next commit

```
$ git add -p <file>
```

Commit all local changes in tracked files

```
$ git commit -a
```

Commit previously staged changes

```
$ git commit
```

Change the last commit

Don't amend published commits!

```
$ git commit --amend
```

COMMIT HISTORY

Show all commits, starting with newest

```
$ git log
```

Show changes over time for a specific file

```
$ git log -p <file>
```

Who changed what and when in <file>

```
$ git blame <file>
```

BRANCHES & TAGS

List all existing branches

```
$ git branch -av
```

Switch HEAD branch

```
$ git checkout <branch>
```

Create a new branch based on your current HEAD

```
$ git branch <new-branch>
```

Create a new tracking branch based on a remote branch

```
$ git checkout --track <remote/branch>
```

Delete a local branch

```
$ git branch -d <branch>
```

Mark the current commit with a tag

```
$ git tag <tag-name>
```

UPDATE & PUBLISH

List all currently configured remotes

```
$ git remote -v
```

Show information about a remote

```
$ git remote show <remote>
```

Add new remote repository, named <remote>

```
$ git remote add <shortname> <url>
```

Download all changes from <remote>, but don't integrate into HEAD

```
$ git fetch <remote>
```

Download changes and directly merge/integrate into HEAD

```
$ git pull <remote> <branch>
```

Publish local changes on a remote

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

Delete a branch on the remote

```
$ git branch -dr <remote/branch>
```

Publish your tags

```
$ git push --tags
```

MERGE & REBASE

Merge <branch> into your current HEAD

```
$ git merge <branch>
```

Rebase your current HEAD onto <branch>

Don't rebase published commits!

```
$ git rebase <branch>
```

Abort a rebase

```
$ git rebase --abort
```

Continue a rebase after resolving conflicts

```
$ git rebase --continue
```

Use your configured merge tool to solve conflicts

```
$ git mergetool
```

Use your editor to manually solve conflicts and (after resolving) mark file as resolved

```
$ git add <resolved-file>
```

```
$ git rm <resolved-file>
```

UNDO

Discard all local changes in your working directory

```
$ git reset --hard HEAD
```

Discard local changes in a specific file

```
$ git checkout HEAD <file>
```

Revert a commit (by producing a new commit with contrary changes)

```
$ git revert <commit>
```

Reset your HEAD pointer to a previous commit ...and discard all changes since then

```
$ git reset --hard <commit>
```

...and preserve all changes as unstaged changes

```
$ git reset <commit>
```

...and preserve uncommitted local changes

```
$ git reset --keep <commit>
```

Git Flow

