ENSI CAEN
ÉCOLE PUBLIQUE D'INGÉNIEURS
CENTRE DE RECHERCHE

Manuel du développeur

Travailler avec Git au quotidien

Ce vade-mecum liste les commandes Git les plus utilisées lors du travail de développement, ainsi que quelques commandes qui permettent de sortir de situations particulières.

1. Création d'un projet

1.1 Créer un nouveau dépôt

- Créer un dépôt local :

```
mkdir depot
cd depot
git init
```

Remarque

La commande `git init` crée un sous-dossier `.git` qui contient le dépôt local.

- Lier le dépôt local à un dépôt distant existant, par exemple sur le Gitlab de l'école avec « *toto* » l'identifiant du compte :

```
git remote add origin https://toto@gitlab.ecole.ensicaen.fr/toto/depot.git
```

Remarque

Le nom `origin` du lien vers le dépôt distant est purement conventionnel. C'est le nom par défaut donné par Git pour ce lien. Il peut donc être changé.

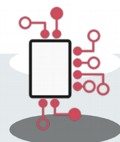
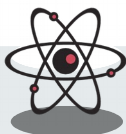
- Construire le dépôt distant à partir du dépôt local :

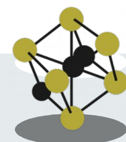
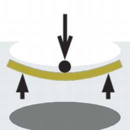
```
touch README.md
git add -A
  ☑git commit -m 'Création du dépôt'
git push origin master
```

1.2 Créer un dépôt local à partir d'un dépôt existant

- À l'inverse, le dépôt local peut être construit à partir d'un dépôt distant existant, par exemple hébergé sur le Gitlab de l'école :

```
git clone https://toto@gitlab.ecole.ensicaen.fr/toto/depot.git
cd depot
```

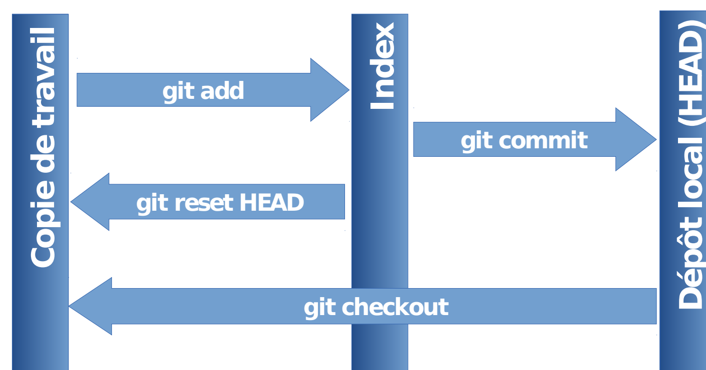



Remarques

Le lien `origin` vers le dépôt distant est automatiquement créé. On retrouve dans le dossier créé, le sous-dossier `.git`.

2. Cycle de contrôle d'un fichier sur le dépôt local

Le dossier local distingue trois espaces : la copie de travail, l'index et le dépôt local. La copie de travail contient les fichiers créés et modifiés par le développeur. L'index est une zone de transit stockant les fichiers qui seront poussés vers le dépôt. Le dépôt garde un historique des versions des fichiers, repéré par sa tête, nommée HEAD. Les fichiers peuvent passer d'un espace à l'autre.



2.1 Connaître l'état du dépôt

- La commande `status` est la commande de base quid onne l'atat courant du dépôt local :

```
git status
```

2.2 Ajouter des fichiers dans l'index

La commande `git add` permet d'ajouter les fichiers dans la zone d'index.

- Ajouter un fichier en particulier :

```
git add /chemin/vers/fichier.ext
```

- Ajouter un dossier :

```
☑git add /chemin/vers/dossier
```

- En particulier, ajouter le dossier courant :

```
git add .
```

Les options permettent de sélectionner les fichiers à ajouter en fonction de leur état :

	Fichiers nouveaux	Fichiers modifiés	Fichiers détruits
`git add -A`	✓	✓	✓

ENSICAEN - Spécialité Informatique

`git add .`	✓	✓	✓
`git add -i`	✓	✓	✗
`git add -u`	✗	✓	✓

Remarque

La commande `git add` ajoute les fichiers tels qu'ils sont au moment de la commande. Si des modifications interviennent ultérieurement sur ces fichiers, les modifications ne seront pas prises en compte dans l'index et lors du prochain commit. Il est nécessaire de refaire un `git add` pour les prendre en compte. En tout état de cause, un `git status` fait un état des lieux pour vous aider à y voir clair.

2.3 Exclure des fichiers de l'indexation

Le fichier `.gitignore` contient la liste des fichiers ou des dossiers à ne pas inclure lors des `git add`. La syntaxe utilise des motifs standards :

```
fichier   # exclure un fichier
dossier/  # exclure un dossier
*~        # exclure tous les fichiers se terminant par ~
```

Remarques

Plusieurs fichiers `.gitignore` répartis dans les sous-dossiers du projet se cumulent. Mais cette pratique est fortement déconseillée parce que les motifs peuvent s'annuler les uns les autres.

Si un fichier est déjà pris en compte mais que l'on souhaite ne plus prendre en compte, il ne suffit pas de l'ajouter dans le fichier `.gitignore`. En effet, une fois un fichier pris en compte, il ne peut pas être retiré en l'ajoutant dans le `.gitignore`. La solution est d'utiliser `git rm` avec l'option `--cached` :

- pour un fichier :

```
git rm --cached /chemin/vers/fichier.ext
```

- pour un dossier :

```
git rm --cached -r /chemin/vers/dossier
```

La commande `rm` avec l'option `--cached` ne détruit pas physiquement le fichier, mais le supprime de l'index et de la liste des fichiers suivis par Git. La commande `rm` permet de :

- supprimer un fichier de la copie de travail et de l'index :

```
☑ git rm /chemin/vers/fichier.ext
```

- supprimer un fichier de l'index uniquement.

```
☑ git rm --cached /chemin/vers/fichier.ext
```

- Pour supprimer un fichier de la copie de travail uniquement, il faut utiliser la suppression physique du fichier par `/bin/rm`.

2.4 Retirer un fichier de l'index

La commande `reset` permet de supprimer les fichiers indexés. C'est la commande inverse de `git add` :

```
git reset
```

- Pour un seul fichier :

```
git reset /chemin/vers/fichier.ext
```

- Ainsi, pour ajouter tout un dossier sauf un fichier dans l'index :

```
git add /chemin/vers/dossier
  ☑ git reset /chemin/vers/dossier/fichier.ext
```

Remarque

La commande `git rm --cached` permet aussi de supprimer un fichier de l'index. Mais, la différence réside dans le fait qu'avec `rm` le fichier n'est ensuite plus suivi par Git.

2.5 Archiver les fichiers de l'index dans le dépôt git local

- Déplacer les fichiers de l'index vers le dépôt Git :

```
git commit -m 'Mon message expliquant le commit'
```

La commande crée une nouvelle ligne dans l'historique du dépôt Git, indexée par un SHA-1 et décrit par le message spécifié.

2.6 Voir l'historique des archivages

```
git log
```

Cette commande possède beaucoup d'options d'affichage.

2.7 Modifier le dernier archivage

Il y a au moins deux raisons de revenir sur un commit : modifier le message de commit ou ajouter ou supprimer des fichiers. Dans les deux cas, la commande `commit` avec l'option `--amend` fait le travail :

```
git commit --amend -m 'Mon nouveau message de commit'
```

Le message est changé avec le nouveau donné et un nouveau commit est créé avec le contenu courant de l'index, qui aura été mis à jour avec les fichiers oubliés ou débarrassé des fichiers non souhaités.

Remarque

Cette commande ne doit être effectuée que si vous n'avez pas encore poussé le commit sur dépôt distant sinon les autres utilisateurs du commit seront affectés puisque le SHA-1 est changé.

2.8 Annuler les modifications sur la copie de travail

- Annuler les modifications d'un fichier de la copie de travail qui sera comme au dernier commit :

```
▶ git checkout /chemin/vers/fichier.ext
```

- Revenir à une autre version que la dernière, identifiée par rapport à la tête du dépôt (HEAD) ou par un SHA-1 :

```
☑ git checkout HEAD~6
☑ git checkout SHA-1
```

Remarque

Ces dernières commandes créent une version détachée. Puisque cette version n'est plus référencée par rapport à la tête de l'historique, toutes les modifications faites sur cette version seront perdues. Il est alors nécessaire de construire une nouvelle branche, puis faire des commits pour insérer les modifications :

```
git checkout -b ma_branche
git commit -m 'message'
```


2.9 Annuler un commit sans trace dans l'historique

Nous l'avons précisé à la section 2.4, la commande `reset` est l'inverse de la commande `git add`. Elle correspond à l'appel avec l'option `--mixed`, qui est l'option par défaut. Mais avec d'autres options, il est possible d'annuler un commit. L'option `--soft` agit uniquement sur le dépôt local (HEAD). L'option `--hard` est plus brutale et supprime les modifications intervenues sur la copie de travail. Toutes ces modifications sont donc perdues ; la copie de travail, l'index et HEAD ont tous le même état.

	Copie de travail	Index	Dépôt local
État initial	V4	V3	V2
$ git reset --soft v1	V4	V3	V1
$ git reset --mixed v1	V4	V1	V1
$ git reset --hard v1	V1	V1	V1

En utilisant une autre valeur que HEAD, la commande permet de se déplacer dans l'historique des commits.

- Revenir à l'avant-dernier commit :

```
☑ git reset --soft HEAD^
```

- Revenir à 3 commits en arrière.

```
☑ git reset --soft HEAD~3
```

La figure suivante résume les effets de la commande `reset` selon les options, où *v1* est un SHA-1 ou une référence sur HEAD.

2.10 Annuler un commit avec trace dans l'historique

Comme `reset`, `revert` permet de revenir en arrière. La différence essentielle est que `revert` ajoute un commit à l'historique pour signaler cette réversion.

- Revenir 2 commits en arrière :

```
git revert HEAD^
```

2.11 Appliquer un commit sur le dépôt local

La commande `cherry-pick` permet d'appliquer à la copie de travail des changements effectués au sein d'un autre commit.

- L'exemple est celui d'un bug corrigé sur `master` dans le commit n°`d42f45a` que l'on souhaite appliquer à la branche `production` :

```
git checkout production
git cherry-pick d42f45a
```

2.12 Changer son historique localement

La commande `git rebase` avec l'option `-i` permet de reprendre l'historique du dépôt local pour changer l'ordre des commits, fusionner des commits successifs ou simplement changer le message du commit.

- Pour revenir sur les 6 derniers commits :

```
git rebase -i HEAD~6
```

Les commandes principales de modification d'un commit :

- edit : retrouver le commit tel qu'il était au moment du commit. Cela permet de le retravailler pour modifier son contenu ou encore le découper en plusieurs.
- reword : changer le message de commit.
- squash : fusionner le commit avec le précédent pour n'en créer qu'un.
- fixup : comme squash, mais supprime le message du second commit.

Git tentera de rejouer les commits selon le scénario que vous venez de définir. Des conflits peuvent alors apparaître plusieurs fois sur un même fichier. La commande s'arrête donc pour régler le temps de régler ces conflits. Pour poursuivre le `rebase` ou annuler le `rebase` en cours :

```
git rebase --continue
git rebase --abort
```

2.13 Comparer des commits

- Différences entre la copie de travail et l'index :

```
git diff
```

- Différences entre l'index et le dépôt local :

```
git diff --cached
```

- Différences entre les fichiers indexés et ceux du dernier commit :

```
git diff --staged
```

- Différences sur un fichier entre la copie de travail et l'index :

```
git diff /chemin/vers/fichier.ext
```

- Différences entre la version d'un fichier sur la copie de travail et celle d'un commit identifié par un SHA-1 :

```
diff 126df8c chemin/vers/fichier.ext
```

- Différences entre la copie de travail et le commit précédent du dépôt local :

```
git diff HEAD^
```

- Différences entre deux banches, par exemple `master` et `feature1` :

```
git diff master..feature1
```

En guise d'exemple, les deux commandes suivantes listent les changements qui seront opérés lors d'un `git push` ou d'un `git pull`, après avoir mis à jour les liens vers le dépôt distant `git fetch`.

- Liste les changements qui seront poussés sur le dépôt distant au prochain `git push` :

```
git diff origin/$(git currentbranch)...HEAD
```

- Liste les changements qui seront récupérés sur le dépôt local au prochain `git pull` :

```
git diff HEAD...origin:$(git currentbranch)
```

2.14 Nettoyer sa copie de travail

La commande `git clean` nettoie la copie de travail en supprimant tous les fichiers non-suivis.

- La commande suivante effacera en plus tous les dossiers vides ; l'option `-f` signifie « force » :

```
git clean -f -d
```

Remarque

L'option `-n` permet de visualiser ce qui serait fait si la commande était réellement

faite :

```
☑ $ git clean -d -n
Supprimerait README.md~
Supprimerait build
```

2.15 Remiser temporairement son dossier de travail

Si la copie de travail est modifiée, Git empêche toute opération qui peut changer son contenu, comme changer de branche ou rapatrier un commit passé. Il y a alors deux solutions, faire un commit pour ne pas perdre ces fichiers mais cela laisse une trace dans l'historique ou remiser temporairement le contenu de la copie de travail.

- Le remisage se fait par :

```
git stash
```

On se retrouve dans le même état qu'après un `git --hard HEAD` mais cette fois, les fichiers en cours de modification ne seront pas perdus.

La récupération des fichiers se fait par :

```
git stash pop
```

3. Travailler sur un branche

3.1 Créer un branche

- Créer une nouvelle branche, puis se déplacer dessus :

```
git branch 15_query_database
☑ git checkout 15_query_database
```

- Créer une branche et se déplacer dessus en une seule commande :

```
git checkout -b 15_query_database
```

3.2 Lister les branches

- Liste des branches locales :

```
git branch -v
```

- Liste des branches distantes :

```
git branch -r
```

3.3 Changer la branche courante

```
git checkout 15_query_database
```

Remarque

Cette commande n'est possible que s'il n'y a pas de modification dans la copie de travail (fichier ajouté, modifié ou supprimé). Pour forcer ce changement de branche,

deux solutions.

1. Utiliser l'option `-f` pour forcer le checkout. Dans ce cas, les fichiers modifiés dans la copie de travail sont définitivement perdus.

```
git checkout -f 15_query_database
```

2. Remiser temporairement les fichiers modifiés (voir la section 2.15).

```
git stash
```

3.4 Supprimer une branche

```
git branch -d 15_query_database
```

3.5 Fusionner une branche avec une autre

- Par exemple, pour fusionner la branche `15_query_database` dans la branche `master` :

```
git checkout master
git merge 15_query_database
```

- Pour annuler un merge en cours :

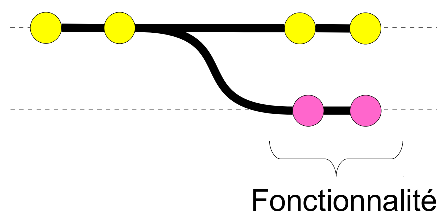
```
git merge --abort
```

Remarque

La commande `merge` utilise l'option `-ff` *(fast-forward)*, par défaut. En *fast-forward*, la branche est ajoutée dans `master` comme s'il s'agissait de commits classiques. Il n'y a plus trace de la branche. Il est impossible de voir à partir de l'historique quels commits font partie de la fonctionnalité ajoutée.

master

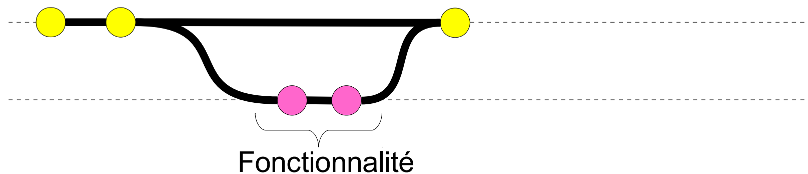
15_query-database



L'option `--no-ff` (*no fast-foward*) force la fusion à toujours créer une nouvelle dérivation, même quand le merge peut se faire avec fast-forward. Cela permet de ne pas perdre l'information sur l'existence d'une branche de fonctionnalité.

master

15_query-database



Lors de la fusion, tous les conflits à l'intérieur des fichiers de même nom doivent être réglés à la main.

Pour les fichiers binaires, il faut choisir entre l'un ou l'autre.

- Choisir de conserver l'ancienne version :

```
git checkout --ours /chemin/vers/file.ext
```

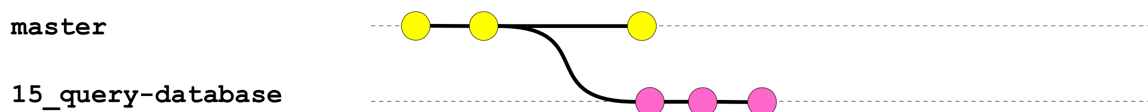
- Choisir de conserver la nouvelle version :

```
git checkout --theirs chemin/vers/file.ext
```

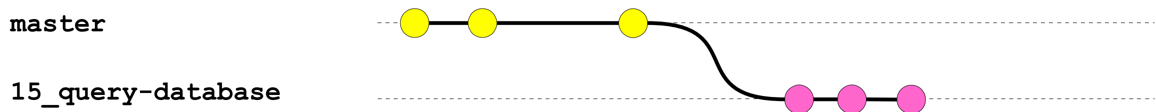
3.6 Changer la base d'une branche

La commande `rebase` a pour effet de modifier le départ de la branche. La différence avec un `merge` est illustrée sur les schémas suivants.

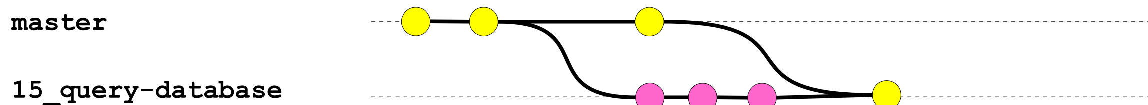
- Soit l'état courant du dépôt local :



- Après un `git rebase master` dans la branche `15_query_database` :

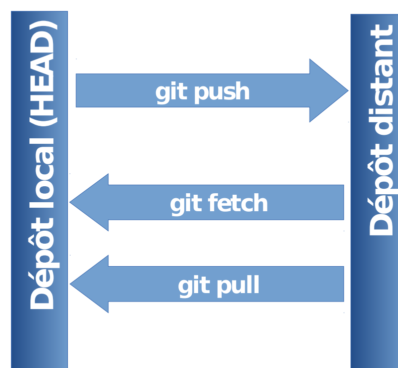


- Après un `git merge master` dans la branche `15_query_database` :



Un `git rebase` agit par application successive de chaque commit de la branche de base sur la branche récipiendaire. Cela peut conduire à résoudre de nombreux conflits, voire plusieurs fois le même. Il faut donc faire un rebase régulièrement pour éviter d'avoir à mettre à jour des conflits trop importants.

4. Échange entre le dépôt local et le dépôt distant



4.1 Pousser ses commits sur le dépôt distant

- Pousser les modifications du dépôt local vers le dépôt distant :

```
git push [-f]
```

L'option `-f` permet de forcer le push dans les cas d'incompatibilité entre les deux versions.

4.2 Mettre à jour les références du dépôt distant sur le dépôt local

- La commande `fetch` met à jour les références à toutes les branches contenues dans le dépôt local avec le dépôt distant.

```
git fetch orgin
```

La commande `fetch` ne fusionne pas les données distantes avec les données locales et ne modifie pas la copie de travail. Simplement, les données distantes sont actualisées sur le dépôt local et sont prêtes à être utilisées en local.

4.3 Récupérer les sources du dépôt distant

- Pour récupérer la version de master qui se trouve sur le dépôt distant :

```
git pull master
```

- Un git pull correspond à :

```
git fetch origin
git merge origin/master
```

Si la branche considérée a été modifiée, la commande `git pull` fait un merge et ajoute une banche visible dans l'historique.

Pour éviter d'ajouter une dérivation sur l'historique, on peut utiliser un `rebase` :

```
git fetch
git rebase -p origin origin/15_query_database
```

ou directement :

```
git pull -rebase=preserve origin 15_query_database
```

La valeur `preserve` (ou `-p`) permet de garder les dérivations qui existent dans l'historique du dépôt distant.

5. Créer une distribution des sources avec le dépôt

Pour avoir une archive de la version de votre dépôt sans les fichiers inutiles tel ceux du dossier `.git` :

```
cd depot
git archive --format=tar --prefix=dossier/ HEAD | gzip > v1.4.0.tar.gz
```

Vous obtenez alors une archive `v1.4.0.tar.gz` qui contient la dernière version de la branche courante du dépôt local. L'archive sera décompressée dans un dossier

nommé `dossier` avec tous les fichiers du dépôt.

6. Comment réduire la taille d'un dépôt Git

Quand le dépôt a atteint une taille conséquente, la première idée est de réduire l'historique. Mais, cette tâche est très complexe avec Git parce que les commit dépendent les uns des autres. L'autre solution, qui est proposée ici, est de supprimer les plus gros fichiers dans l'historique, par exemple les différentes versions d'un fichier binaire ou d'une archive.

6.1 Étape 1 : Prévenir les contributeurs

Les commandes suivantes vont modifier profondément le dépôt Git. Il faut donc que vos développeurs fassent une copie de leur dépôt courant pour ne rien perdre.

6.2 Étape 2 : Mise à jour profonde du dépôt local

Le fichier de commande suivant met à jour toutes les références au dépôt distant sur le dépôt local :

```
#!/bin/bash
for branch in `git branch -a | grep remotes | grep -v HEAD | grep -v
  master`; do
    git branch --track ${branch##*/} $branch
done
```

6.3 Étape 3 : Repérer les gros fichiers

Le fichier de commande suivant liste les plus gros fichiers[1] :

```
#!/bin/bash
#set -x

# Shows you the largest objects in your repo's pack file.
# Written for osx.
#
# @see http://stubbisms.wordpress.com/2009/07/10/git-script-to-show-
  largest-pack-objects-and-trim-your-waist-line/
# @author Antony Stubbs

# set the internal field seperator to line break, so that we can iterate
  easily over the verify-pack output
IFS=$'\n';

# list all objects including their size, sort by size, take top 10
objects=`git verify-pack -v .git/objects/pack/pack-*.idx | grep -v chain
  | sort -k3nr | head`

echo "All sizes are in kB. The pack column is the size of the object,
```

1 Antony Stubbs, https://gist.github.com/runemadsen/4236839.

ENSICAEN - Spécialité Informatique

```
  compressed, inside the pack file."

output="size,pack,SHA,location"
for y in $objects
do
        # extract the size in bytes
        size=$((`echo $y | cut -f 5 -d ' '`/1024))
        # extract the compressed size in bytes
        compressedSize=$((`echo $y | cut -f 6 -d ' '`/1024))
        # extract the SHA
        sha=`echo $y | cut -f 1 -d ' '`
        # find the objects location in the repository tree
        other=`git rev-list --all --objects | grep $sha`
        #lineBreak=`echo -e "\n"`
        output="${output}\n${size},${compressedSize},${other}"
done

echo -e $output | column -t -s ', '
```

Résultat :

```
size     pack    SHA         location
1111686  132987  a561d2510   tar.tgz
5002     392     e501b7944   version1.2.3.zip
```

6.4 Étape 4 : Supprimer les fichiers des différents commits

Il suffit maintenant de supprimer chaque « gros » fichier en répétant la commande suivante, ou `filename` doit être remplacé par le chemin vers le fichier à supprimer identifié par la commande précédente :

```
git filter-branch --tag-name-filter cat --index-filter 'git rm -r --
  cached --ignore-unmatch filename' --prune-empty -f -- --all
```

6.5 Étape 5 : Nettoyer l’espace

Il s’agit maintenant de supprimer l’espace qu’occupaient les références aux objets :

```
rm -rf .git/refs/original/
git reflog expire --expire=now --all
git gc --prune=now
git gc --aggressive --prune=now
```

Maintenant la taille du dépôt doit avoir considérablement diminuée.

6.6 Étape 6 : Pousser le nouveau dépôt

Ne reste plus qu’à pousser le dépôt local vers le dépôt commun.

```
git push origin --force --all
```

6.7 Étape 7 : Informer les contributeurs

Chaque développeur doit maintenant soit faire un `git rebase` sur le dépôt local ou créer un nouveau dépôt local.

ENSICAEN - Spécialité Informatique