Git, compléments

Franck Pérignon

Journée Gitlab, 29 juin 2023

La commande git

Git s'utilise en général via la ligne de commande (terminal, PowerShell). Le fonctionnement est identique sous Windows, mac ou Linux.

```
git <command> <arguments>
```

Pour un aperçu des différentes commandes, essayez

```
git help
```

ou, pour obtenir la documentation d'une commande particulière :

```
git help <command>
```

Configuration de git

Quelques paramétrages sont nécessaires, à la première utilisation de git :

```
git config --global user.name "Franck Perignon"
git config --global user.email "Franck.Perignon@univ-grenoble-alpes.fr"
```

Tout est enregistré dans un fichier *.gitconfig* à la racine de votre compte,

```
[user]
    email = franck.perignon@univ-grenoble-alpes.fr
    name = Franck Perignon
```

Configuration de git

Vous pouvez notamment choisir un éditeur par défaut (pour la gestion des conflits par exemple) :

```
git config --global core.editor emacs
```

ou définir des alias en éditant le fichier .gitconfig

```
[alias]
      df = diff origin/master
```

Git, première étape : création du répertoire de travail

Pour utiliser git, il faut avant tout un répertoire de travail, qui va contenir entre autres la base de donnée et la version courante de votre repository.
Création (**init**) d'un dépôt vide :

```
mkdir WorkingDir ; cd WorkingDir
git init
```

Ou copie (**clone**) un dépôt existant

```
git clone addresse_depot
# exemples :
# git clone /chemin/vers/depot_local WorkingDir
# git clone username@server:/chemin/vers/depot WorkingDir
```

adresse_depot représente un dépôt "distant" (**remote**) : un autre répertoire sur votre machine, un dépôt git sur un autre serveur accessible via le réseau ...

Git, première étape : création du répertoire de travail

Conséquences (du git init ou du git clone)

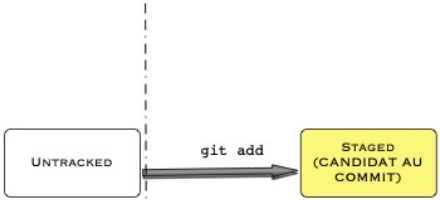
- Création dans WorkingDir d'un répertoire .git,
  qui contient toutes les infos sur votre projet : les méta-données et la base de données git avec notamment les différentes sauvegardes de l'ensemble des fichiers de votre projet.
- Extraction à partir de la base de données de la dernière version du repository (vide si git init).

Git, ajouter un fichier à la base

Comment prendre en compte un fichier inconnu (UNTRACKED) et le sauvegarder dans la base ?

1. Marquer le fichier comme suivi par git (“ STAGED ”)

```
git add nomFichier
```



Git, ajouter un fichier à la base

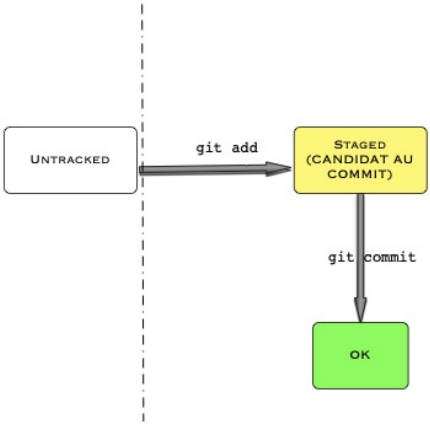
Comment prendre en compte un fichier inconnu (UNTRACKED) et le sauvegarder dans la base ?

1. Marquer le fichier comme suivi par git (“ STAGED ”)

```
git add nomFichier
```

2. Commit (STAGED → OK)

```
git commit -m "Some useful comme
```

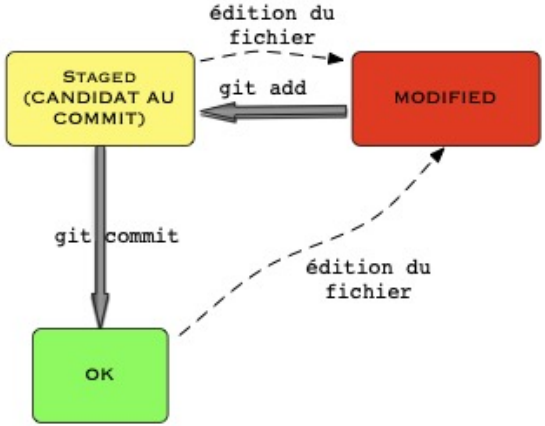


Git, ajouter un fichier à la base

Comment prendre en compte un fichier modifié et le sauvegarder dans la base ?

1. Ajout du fichier à l'index
   MODIFIED → STAGED

```
git add nomFichier
```



Git, ajouter un fichier à la base

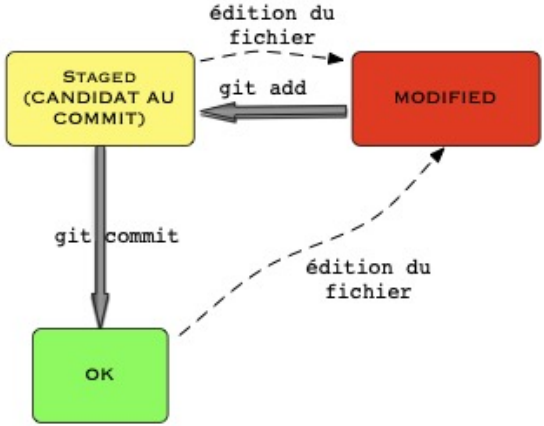
Comment prendre en compte un fichier modifié et le sauvegarder dans la base ?

1. Ajout du fichier à l'index
   MODIFIED → STAGED

```
git add nomFichier
```

2. Commit (STAGED → ok)

```
git commit -m "commentaire"
```



Git, ajouter un fichier à la base

Comment prendre en compte un fichier modifié et le sauvegarder dans la base ?

1. Ajout du fichier à l'index
   MODIFIED → STAGED

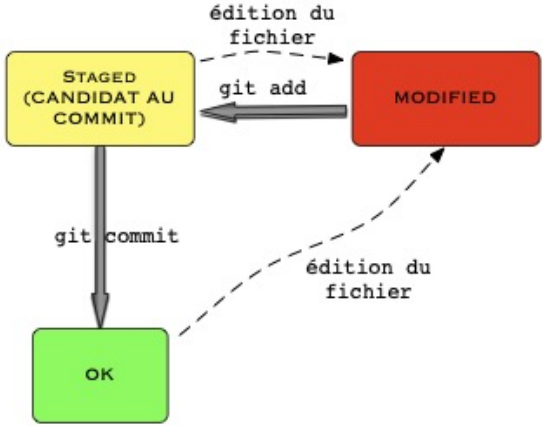
```
git add nomFichier
```

2. Commit (STAGED → ok)

```
git commit -m "commentaire"
```

3. 1 et 2 en une seule commande
   MODIFIED → ok

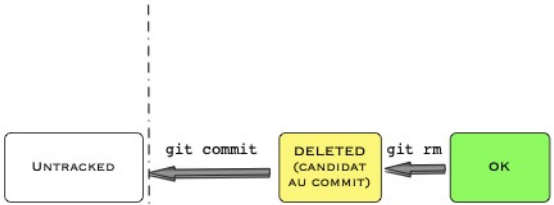
```
git commit -a -m "commentaire"
```



Git, supprimer un fichier

1. Marquer le fichier “à supprimer” dans l'index OK → STAGED

```
git rm nomFichier
```



Git, supprimer un fichier

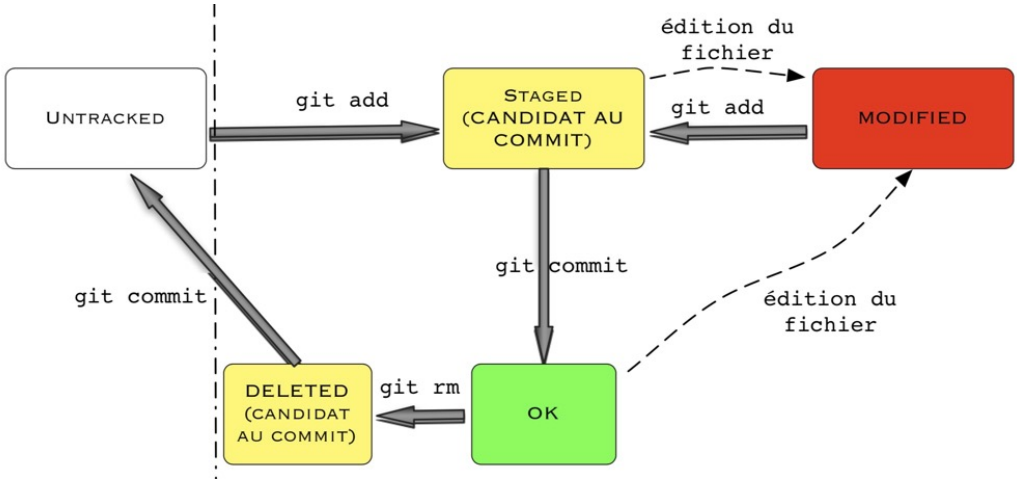
1. Marquer le fichier “à supprimer” dans l’index OK → STAGED

```
git rm nomFichier
```

2. Commit (STAGED → UNTRACKED

```
git commit -m "Nettoyage
```

Git, les différents états d'un fichier



git status, état du répertoire de travail

Permet d'obtenir des infos sur l'état de tous les fichiers du répertoire de travail.

- Modification d'un fichier

```
vim nomFichier
git status
```

```
└─o git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   nomFichier

no changes added to commit (use "git add" and/or "git commit -a")
```

git status, état du répertoire de travail

Permet d'obtenir des infos sur l'état de tous les fichiers du répertoire de travail.

- Modification d'un fichier

```
vim nomFichier
git status
```

- Staging

```
git add nomFichier
git status
```

```
└─o git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   nomFichier

no changes added to commit (use "git add" and/or "git commit -a")
```

```
└─o git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   nomFichier
```

git status, état du répertoire de travail

Permet d'obtenir des infos sur l'état de tous les fichiers du répertoire de travail.

- Modification d'un fichier

```
vim nomFichier
git status
```

```
└─o git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   nomFichier

no changes added to commit (use "git add" and/or "git commit -a")
```

- Staging

```
git add nomFichier
git status
```

```
└─o git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   nomFichier
```

- Commit

```
git commit -m "message"
git status
```

```
└─o git status
On branch master
nothing to commit, working tree clean
```

git log, historique des commits

liste de tous les commits (et donc de toutes les versions) du repository

- identifiant (hash) unique de chaque commit
- date, auteur, message

```
 git log
commit 871da133938c8e8a423d1010ab969625078b4b79 (HEAD -> master)
Author: Franck Pérignon <franck.perignon@univ-grenoble-alpes.fr>
Date:   Fri Jan 11 17:27:47 2019 +0100

    dernier commit

commit d18e05e80fe11c6206e2cb5a0624b477a6c31226
Author: Franck Pérignon <franck.perignon@univ-grenoble-alpes.fr>
Date:   Fri Jan 11 16:26:05 2019 +0100

    bonjour
```

git diff, comparaison de fichier

La commande git diff permet de comparer n'importe quelle version d'un fichier avec n'importe quelle autre, par exemple :

```
└─o git diff dec9833da2c5daced5d37cd4f82d0e4625fde553
diff --git a/fichier2 b/fichier2
new file mode 100644
index 0000000..5e0a511
--- /dev/null
+++ b/fichier2
@@ -0,0 +1,2 @@
+un nouveau fichier
+
diff --git a/nomFichier b/nomFichier
index 525960d..83baae6 100644
--- a/nomFichier
+++ b/nomFichier
@@ -1,2 +1 @@
-version 0
-
+version 1
```

Le même en version courte (option –stat)

```
└─o git diff dec9833da2c5daced5d37cd4f82d0e4625fde553 --stat
 fichier2   | 2 ++
 nomFichier | 3 +--
 2 files changed, 3 insertions(+), 2 deletions(-)
```

git diff, comparaison de fichier

- Diff entre la version staged et la version modified

```
git diff nomFichier
```

- Diff entre la version staged et la version référence

```
git diff --staged nomFichier
```

- Diff entre la version courante et celle d'un commit précédent

```
git diff d18e05e80fe11c6206e2cb5a0624b477a6c31226 nomFichier
```

Remarque : sans l'argument nomFichier, diff liste toutes les différences pour TOUS les fichiers.

Git, comment revenir à un état antérieur?

- Annulation des modifications sur un fichier (MODIFIED → OK)

```
git restore nomFichier # ou git checkout -- nomFichier pour les anc
```

- Annulation du passage à "staged" (staged → MODIFIED)

```
git restore -- staged nomFichier # ou git reset HEAD nomFichier
```

- Modification du dernier commit (message et ajout du contenu de l'index en cours)

```
git commit --amend
```

- Retour à une ancienne version du dépôt pour un fichier ou tout le répertoire (sans l'argument nomFichier)

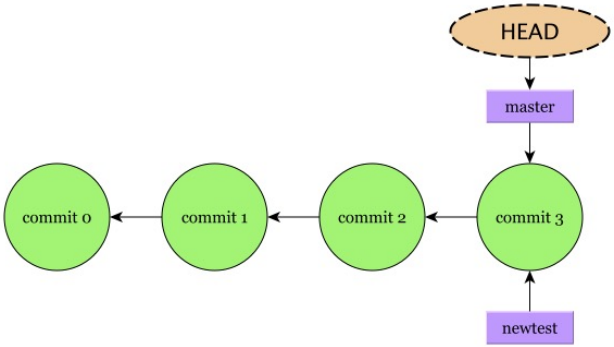
```
git checkout 1bbee103204c nomFichier
```

Git, créer et utiliser une branche

Création d'une nouvelle branche

```
> git branch newtest
```

newtest : un nouveau pointeur vers le dernier commit.



Git, créer et utiliser une branche

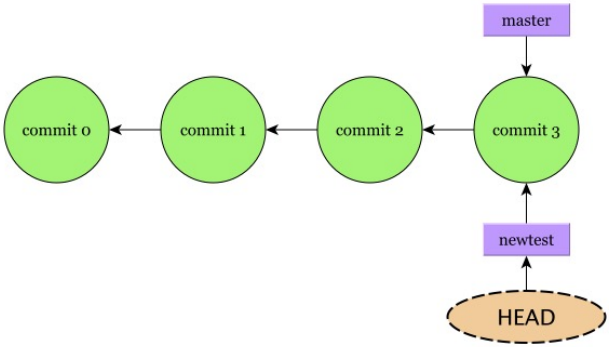
Création d'une nouvelle branche

```
> git branch newtest
```

newtest : un nouveau pointeur vers le dernier commit.

Basculer vers une branche :

```
> git switch newtest # ou git
Switched to branch 'newtest'
```

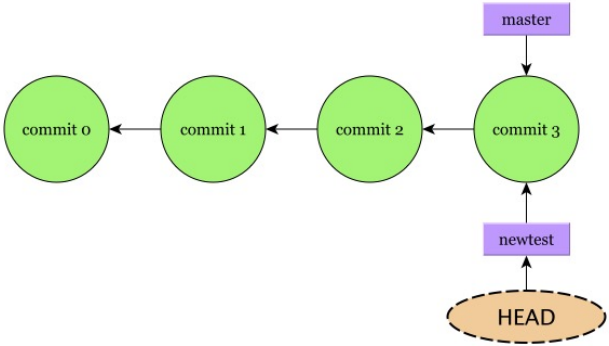


Identification de la branche en cours : le pointeur HEAD.

Git, créer et utiliser une branche

Création et switch en une seule opération

```
> git checkout -b newtest # ou g
Switched to branch 'newtest'
# Liste des branches existantes
> git branch
  master
* newtest
```



Liste des branches existantes

```
> git branch
  master
* newtest
```

Notez le '*' devant la branche en cours (HEAD) dans la liste des branches.

Git, créer et utiliser une branche

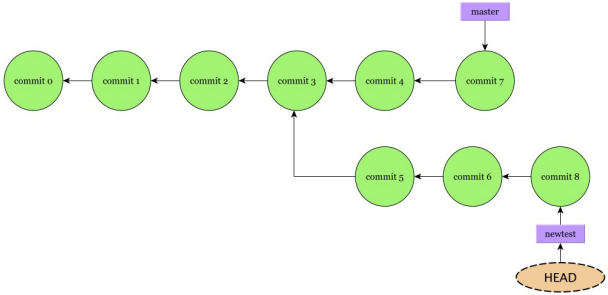
Lors d'une bascule d'une branche à une autre (git checkout), les fichiers présents dans le "working directory" vont être modifiés. L'état des fichiers des différentes branches est contenu dans le "git repository" (.git).

```
> git branch
* master
  branch_dev1
> ls
bin documentation examples include src
> git checkout branch_dev1
Switched to branch 'branch_dev1'
> ls
bin documentation examples include python_tools src visu_3D
```

Apparition dans cet exemple des répertoires python_tools et visu_3D présents uniquement sur la branche branch_dev1

Git, fusionner les branches

Après création d'une (ou plusieurs) branches, chacune peut évoluer indépendamment des autres, en attendant une éventuelle fusion.

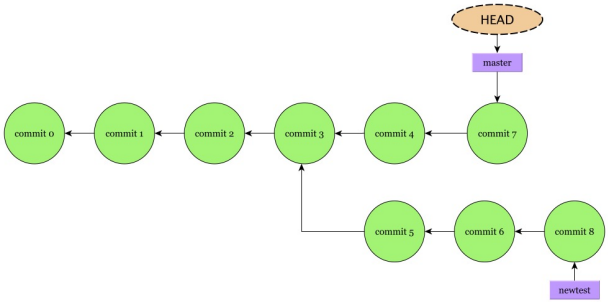


Git, fusionner les branches

Après création d'une (ou plusieurs) branches, chacune peut évoluer indépendamment des autres, en attendant une éventuelle fusion.
Pour intégrer une branche (source) à une autre (cible), il faut :

Basculer vers la branche cible (ici master)

```
> git checkout master
Switched to branch 'master'
```

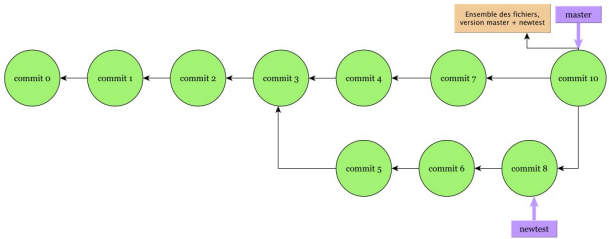


Git, fusionner les branches

Après création d'une (ou plusieurs) branches, chacune peut évoluer indépendamment des autres, en attendant une éventuelle fusion.
Pour intégrer une branche (source) à une autre (cible), il faut :

Basculer vers la branche cible (ici master)

```
> git checkout master
Switched to branch 'master'
```



puis fusionner

```
# Fusion de newtest dans master
> git merge newtest
```

Git, fusionner les branches

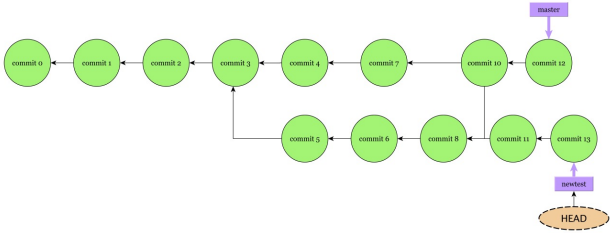
Après création d'une (ou plusieurs) branches, chacune peut évoluer indépendamment des autres, en attendant une éventuelle fusion.
Pour intégrer une branche (source) à une autre (cible), il faut :

Basculer vers la branche cible (ici master)

```
> git checkout master
Switched to branch 'master'
```

puis fusionner

```
# Fusion de newtest dans master
> git merge newtest
```



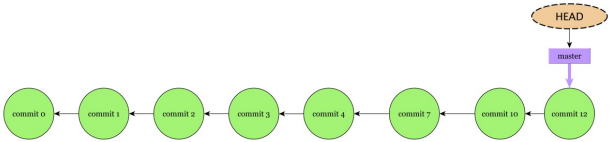
Chaque branche peut ensuite continuer à évoluer ...

Git, fusionner les branches

Après création d'une (ou plusieurs) branches, chacune peut évoluer indépendamment des autres, en attendant une éventuelle fusion.
Pour intégrer une branche (source) à une autre (cible), il faut :

Basculer vers la branche cible (ici master)

```
> git checkout master
Switched to branch 'master'
```



puis fusionner

```
# Fusion de newtest dans master
> git merge newtest
```

Ou être supprimée

```
> git branch -d newtest
Deleted branch newtest (was 87469b4
```

Gestion des conflits

Lors du merge, tout ne se passe pas nécessairement toujours très bien ...

```
> git merge newtest
...
CONFLICT
...
> git status

unmerged : fichier_probleme
```

git ajoute des chevrons autour des zones de conflit :

```
vim fichier_probleme

<<<<<<< HEAD:fichier_probleme
version de master
=======
version newtest en conflit
>>>>>>> newtest:fichier_probleme
```

Gestion des conflits

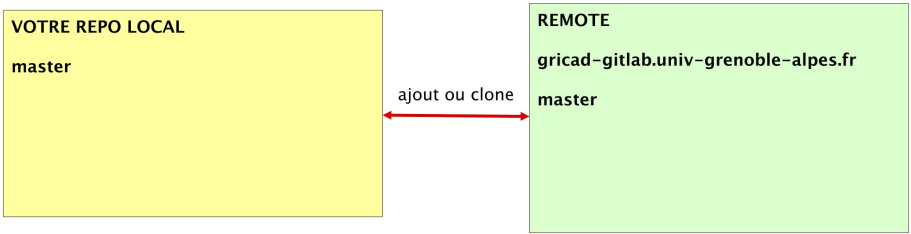
Pour corriger les problèmes, il faudra :

- résoudre les conflits à la main ou via un outil dédié, par exemple :

```
> git mergetool
```

  L'outil graphique ouvert par mergetool dépendra de votre choix d'éditeur dans .gitconfig.
- basculer les fichiers corrigés vers l'index (git add fichier_probleme),
- valider (commit).

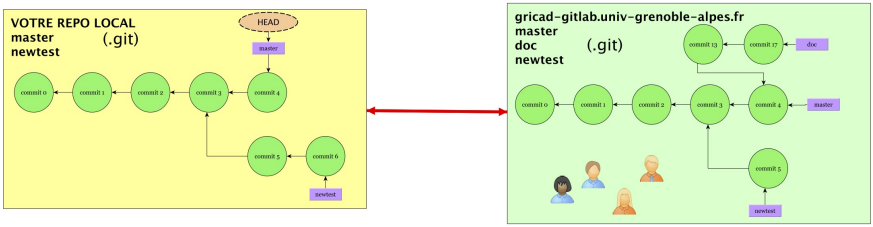
Dépôt remote, principe



Etapes principales :

- **connexion** entre dépôts : ajout ou copie (clone) d'un dépôt distant (en général une seule fois, au début du projet),

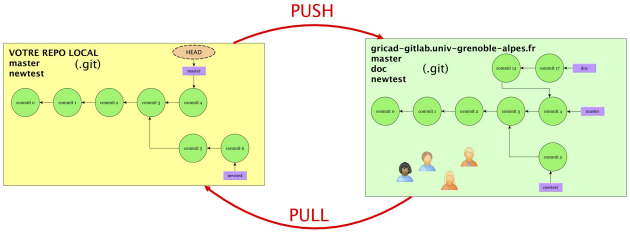
Dépôt remote, principe



Etapes principales :

- **connexion** entre dépôts : ajout ou copie (clone) d'un dépôt distant (en général une seule fois, au début du projet),
- développements (indépendants) dans chaque repository (avec éventuellement plusieurs branches),

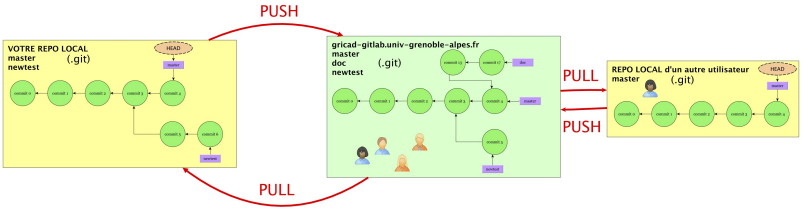
Dépôt remote, principe



Etapes principales :

- **connexion** entre dépôts : ajout ou copie (clone) d'un dépôt distant (en général une seule fois, au début du projet),
- développements (indépendants) dans chaque repository (avec éventuellement plusieurs branches),
- intégration des modifications du dépôt remote (pull),
- transfert de vos modifications vers le remote (push),

Dépôt remote, principe



Etapes principales :

- **connexion** entre dépôts : ajout ou copie (clone) d'un dépôt distant (en général une seule fois, au début du projet),
- développements (indépendants) dans chaque repository (avec éventuellement plusieurs branches),
- intégration des modifications du dépôt remote (pull),
- transfert de vos modifications vers le remote (push),
- push/pull éventuels d'autres utilisateurs.

Ajout d'un dépôt distant

Déclaration d'un nouveau dépôt distant :

```
git remote add NomDepot AdresseDuDepot # dans votre répertoire de travail
# AdresseDuDepot = adresse d'un serveur ou chemin vers
# un autre répertoire contenant un .git. Par exemple :
git remote add some_name git@gricad-gitlab.univ-grenoble-alpes.fr:vide/
```

L'adresse indiquée est maintenant référencé localement comme some_name ouvrant la
$\Rightarrow$ possibilité de synchroniser vos données locales avec some_name.
Liste des dépôts remote referencés :

```
> git remote
some_name
```

Ajout d'un dépôt distant

Dépôt local obtenu par copie (git clone) : serveur cloné automatiquement ajouté comme remote (Nom par défaut : **origin**)

```
> git clone git@gricad-gitlab.univ-grenoble-alpes.fr:vide/rien.git monprojet
> cd monprojet
> git remote -v # -v pour le mode verbeux
origin        git@gricad-gitlab.univ-grenoble-alpes.fr:vide/rien.git (fetch)
origin        git@gricad-gitlab.univ-grenoble-alpes.fr:vide/rien.git (push)
```

Compléments : renommer ou supprimer (localement) un dépôt distant :

```
git remote rename current_name new_name
git remote rm  local_name
```

⚠ il s'agit de suppression dans la liste des dépôts distants de votre dépôt local. Le dépôt sur le serveur distant ne sera pas physiquement supprimé mais simplement ignoré par votre repository local.

Ajout de plusieurs dépôts distants

```
> git clone git@gricad-gitlab.univ-grenoble-alpes.fr:vide/rien.git monprojet
> cd monprojet
# --> lien automatique avec le dépôt remote, nommé localement 'origin'
> git add remote git@github.com:vide/rien.git projet_github
> git add remote git@gitlab.com:vide/rien.git projet_gitlab
# La liste de tous les dépôts remotes :
> git remote -v
origin        git@gricad-gitlab.univ-grenoble-alpes.fr:vide/rien.git (fetch)
origin        git@gricad-gitlab.univ-grenoble-alpes.fr:vide/rien.git (push)
projet_github git@github.com.fr:vide/rien.git (fetch)
projet_github git@github.com:vide/rien.git (push)
projet_gitlab git@gitlab.com:vide/rien.git (fetch)
projet_gitlab git@gitlab.comvide/rien.git (push)
```

Il est préférable que ces dépôts aient un ancètre (une version) en commun ...

Ajout d'un dépôt distant - Résumé

cas 1 : from scratch

```
# Création d'un dépôt vide
> git init
# 'connexion' avec un dépôt remote
> git add remote some_name repository_address
```

cas 2 : copie

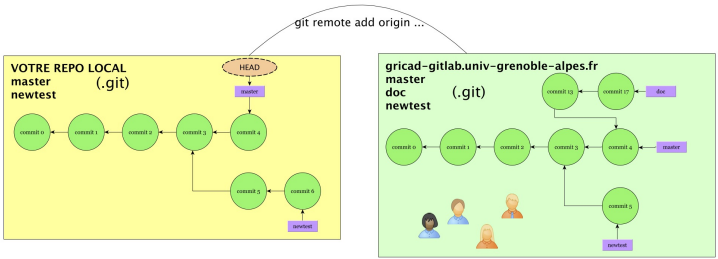
```
> git clone git@gricad-gitlab.univ-grenoble-alpes.fr:vide/rien.git monprojet
# --> lien automatique avec le dépôt remote, nommé localement 'origin'
```

Git, travailler avec des dépôts distants

1. Visualiser (git branch) la liste des branches remotes

Liste complète des branches des dépôts remotes :

```
> git branch -r
# r for remote
> git branch -a
# a for all, local + remote
* master
newtest
remotes/origin/master
remotes/origin/newtest
remotes/origin/doc
```



Comparer n'importe quelle branche de votre dépôt à celles du dépôt distant :

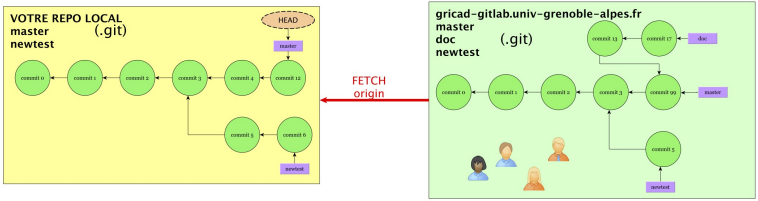
```
git diff some_name/branch_name
# avec les options habituelles de git diff
```

Git, travailler avec des dépôts distants

2. Récupérer (*git fetch*) le contenu d'un dépôt remote

Dès qu'un dépôt est référencé comme "remote", vous pouvez synchroniser votre repository avec celui-ci.
La première étape consiste à collecter toutes les infos (données, branches ...) du dépôt distant via la commande **fetch** :

```
git fetch nom_depot
# par exemple
git fetch origin
```



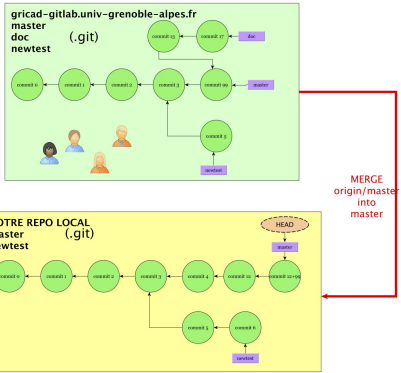
⚠ fetch importe les données remote mais ne modifie pas vos branches locales.

Git, travailler avec des dépôts distants

3. Fusionner (git merge/pull) votre branche locale avec une branche remote

Ensuite, vous pourrez fusionner une branche distante et une branche locale, avec la commande **merge** vu précédemment

```
git merge NomDepot/NomBranche
# par exemple
git merge origin/master
```



fetch et merge peuvent être combinés en une seule opération **pull**

```
git pull NomDepot NomBranche
# par exemple
git pull origin master
```

⚠ pour un fonctionnement correct, il faut un “ancètre” (commit) commun aux

Git, travailler avec des dépôts distants

Il est possible (et recommandé !) d'associer/connecter (**tracking**) une branche locale et une branche remote (nommmée "upstream")

```
# on indique que origin/newtest est l'upstream de newtest local
git branch --set-upstream-to=origin/newtest newtest
# et que origin/master est l'upstream de master local
git branch -u origin/master master
```

ce qui autorise :

```
git checkout newtest  # On se place localement sur la branche newtest
git pull  # fetch + merge de origin/newtest dans newtest
# Même chose dans master
git checkout master
git pull # fetch + merge de origin/master dans master
```

Git, travailler avec des dépôts distants

4. Transférer (git push) votre branche vers un dépôt remote

```
git push NomDepotRemote branche_locale:branche_distante
```

Ou, si vous avez explicitement indiqué une branche upstream pour la branche courante (voir transparent précédent)

```
git checkout master # si origin/master est l'upstream de master local
git push # push de master vers origin/master
```

Autre possibilité : indiquer une branche upstream au premier push

```
> git checkout -b newbranch
> git push # pas de branche upstream !
fatal: The current branch newbranch has no upstream branch.
> git push -u origin newbranch
# ok ! push de newbranch vers origin/newbranch
```

Git, travailler avec des dépôts distants, résumé

Cycle classique de travail dans un dépôt git.

1. Démarrage du suivi d'une branche distante (une seule fois en général)

```
git clone git@gricad-gitlab.univ-grenoble-alpes.fr:vide/rien.git
git branch -u origin/master master
```

2. Développements locaux, vie du repository local

```
git add, commit, status, diff, branch
```

3. Synchronisation (fetch/merge/pull) avec la branche distante

```
git pull
```

4. Transfert de votre travail vers le dépôt remote

```
git push
```

Git - Compléments, quelques autres commandes utiles

- Gestion des tags (étiquettes) : mise en valeur d'un état dans l'historique

```
git help tag
```

- Rebasing : une autre manière de gérer les fusions. e.g. dans votre branche locale:

```
git rebase master
```

Recherche d'un ancètre (commit) commun, ajout des commits de la branche master PUIS ré-execution de vos commits locaux (depuis l'ancètre commun). A manipuler avec précaution ….
Plus de détails
https://git-scm.com/book/fr/v2/Les-branches-avec-Git-Rebaser-Rebasing.

Git - Compléments, quelques autres commandes utiles

- git bisect : recherche de bugs par dichotomie

```
git bisect start
git bisect bad # bug dans la version courante
git bisect good 18f77f6a6d3d3de0 # id d'un commit ok
# good / bad par dichotomie jusqu'au commit coupable
git bisect reset
```

- Cacher les modifications en cours (modified ou staged)

```
git stash
git pull
git stash pop
```