Rechercher des étiquettes :

git tag -l 'v1.4.2.*'

```
$ git tag -l 'v1.4.2.*'
v1.4.2.1
v1.4.2.2
v1.4.2.3
v1.4.2.4
```

Créer une étiquette :

git tag -a v1.4 -m 'my version 1.4'

```
$ git tag -a v1.4 -m 'my version 1.4'
$ git tag
v0.1
v1.3
v1.4
```

Visualiser les données d'une étiquette :

git show v1.4

```
$ git show v1.4
tag v1.4
Tagger: Scott Chacon <schacon@gee-mail.com>
Date:   Mon Feb 9 14:45:11 2009 -0800

my version 1.4
commit 15027957951b64cf874c3557a0f3547bd83b3ff6
Merge: 4a447f7... a6b4c97...
Author: Scott Chacon <schacon@gee-mail.com>
Date:   Sun Feb 8 19:02:46 2009 -0800

    Merge branch 'experiment'
```

Etiqueter après coup :

Il est possible d'étiqueter un commit ancien en récupérant son hash raccourci :

```
$ git tag -a v1.2 -m 'version 1.2' 9fceb02
```

Auto-complétion :

Pressez la touche Tab lorsque vous écrivez une commande Git, et le shell indiquera une liste de suggestions pour continuer la commande :

```
$ git co<tab><tab>
commit config
```

Les alias Git :

Il est possible de définir des alias pour chaque commande en utilisant **git config**. Quelques exemples :

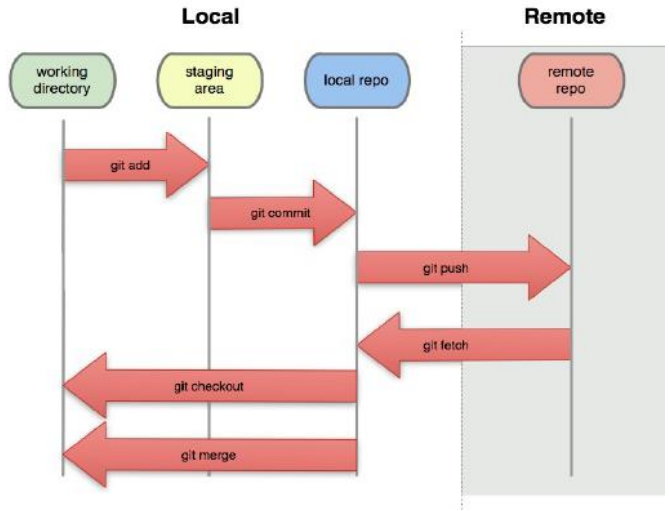
```
$ git config --global alias.co checkout
$ git config --global alias.br branch
$ git config --global alias.ci commit
$ git config --global alias.st status
```

Voir git-scm.com pour plus d'infos.

GIT : fiche pense-bête

GIT est un logiciel de version décentralisé.

Schéma de présentation :

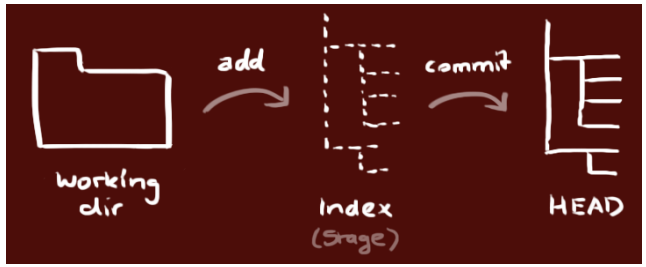


Créer un dépôt local :

Nouveau dépôt local : **git init**

Clônage depuis un dépôt distant : **git clone url**

Les arbres :



Le dépôt local est composé de 3 arbres :

- L'espace de travail qui contient réellement nos fichiers
- L'Index qui joue un rôle de transit
- HEAD qui pointe vers la dernière validation faite (dernier commit)

Ajouter un ou des fichiers à l'index : git add

Pour ajouter manuellement un fichier à l'index, fichiers qui auraient subis des changements :

```
git add <filename>
git add --all //ajoute tous les fichiers du dossier en cours et de ses sous-dossiers.
```

Validation des changements : Commit

```
git commit -m « Message de validation »
```

Il est primordial de saisir un message clair et explicite afin que d'autres développeurs puissent comprendre les modifications apportées.

Le ou les fichiers sont désormais ajoutés au HEAD, à ce stade là les fichiers sont toujours situés sur le dépôt local.

Vérifier l'état des fichiers :

```
git status
```

```
$ git status
On branch master
Your branch is up to date with 'origin/master'.

nothing to commit, working tree clean
```

Annuler des actions :

Lorsqu'on a validé une modification trop tôt (oubli d'ajout de fichiers, erreur sur le message de commit) et qu'aucun autre commit n'a été effectué depuis :

```
git commit -amend
```

Définition du dépôt distant :

```
git remote add origin <url du dépôt distant>
```

Désormais il est possible d'envoyer des changements vers le serveur distant désigné par **origin**

Pousser ses validations sur un dépôt distant

```
git push origin master
```

Récupérer les validations depuis un dépôt distant

Il est possible de récupérer les validations depuis un dépôt distant avec la commande **git fetch** (branche courante uniquement).

Ces données seront stockées dans le répertoire de travail mais ne seront pas fusionnées avec votre branche locale (voir branches).

```
git fetch
```

La commande suivante équivaut à récupérer les commit distants puis à les fusionner avec la branche locale master :

```
git pull origin master
```

Inspecter un dépôt distant :

```
git remote show origin
```

Afficher les dépôts distants :

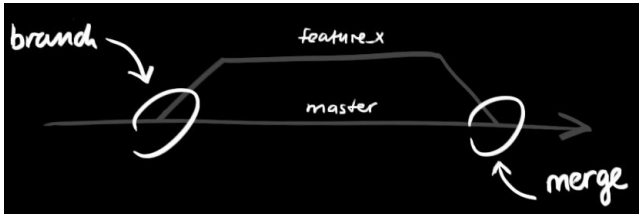
```
git remote
git remote -v //affichage synthétique
```

Clôner un dossier local depuis un dépôt distant :

```
git clone urldudepotdistant
```

Les branches :

Les branches sont utilisées pour développer des fonctionnalités isolées des autres. La branche master est la branche par défaut quand on crée un dépôt. Il est recommandé de travailler sur d'autres branches que celle-ci et de les fusionner (merge) à la branche principale quand on a fini.



Créer une nouvelle branche :

```
git branch nombranche
```

Utiliser cette nouvelle branche :

```
git checkout nombranche
```

Créer & Utiliser une nouvelle branche :

```
git checkout -b nombranche //plus court !
```

Envoyer une branche sur le dépôt distant :

```
git push origin nombranche
```

Fusionner une branche avec la branche active :

```
git merge nombranche
```

La fusion peut être impossible si plusieurs personnes ont travaillé sur les mêmes fichiers. Il conviendra de résoudre ces conflits en modifiant manuellement les fichiers indiqués par git. Ensuite, il faudra ajouter les fichiers modifiés à l'index (git add)

Pour connaître les différences entre 2 branches avant une fusion (merge) :

```
git diff branche1 branche2
```

Historique des validations :

Pour visualiser les validations (commits) on utilise la commande :

```
git log
```

:q pour sortir

```
commit f3728a8e08c8e7f78dcac5e70aa75af984c604e8
Merge: 4f37709 7c3f02c
Author: Fred VAILLANT <fred@frioclemy.net>
Date:   Fri Aug 3 15:12:32 2018 +0200
```

Un commit est identifié par un code unique appelé hash

Sur un historique important le paramètre **–-oneline** permet de raccourcir chaque commit sur une seule ligne :

```
git log --oneline
```

```
* cf1d833 Controle de saisie eleve sur nom prenom
* 965ea07 Ajout TextField (champs obligatoires)
* d15b704 DialogEcole : ajout controle si nom est vide à l'enregistrement
* 3caa4bd DAONote ajout méthode de classe ReadNoteEleve(eleveid) Tests de cette méthode dans le main.
* 4aaad50 Modif DAOClasse ajout enum AlertMsgType pour le type d'AlertMsg
* fb1e977 Premier commit incluant AlertMsg
```

Ici chaque commit voit son hash écrit de manière raccourcie. C'est cette forme raccourcie qui sera utilisée.

Etiquetage :

L'étiquetage permet d'étiqueter un certain état dans l'historique comme important. Généralement, les gens utilisent cette fonctionnalité pour marquer les états de publication (**v1.0** et ainsi de suite).

Lister ses étiquettes :

```
git tag
```

```
$ git tag
v0.1
v1.3
```