# Fiche récapitulative des commandes pour git
# UE 2I013 (2015-2016)

Infos utiles, vocabulaire et commandes, en français : `http://christopheducamp.com/2013/12/15/github-pour-nuls-partie-1/`
Et une version plus détaillée ici `http://tinyurl.com/jtcj3bo`

## 1 Commandes de base

**Attention :** il faut configurer le proxy sur les machines de la PPTI pour pouvoir accéder à github.com. Le protocole est en https qui n'est pas configurer par défaut. Avant toute utilisation, exécuter dans la console (ou rajouter dans le `.bash` ou le `.profile`) :
`export https_proxy=proxy:3128`

- `>git help` : aide et recapitulatif des commandes de bases;
- `git init` : transformer un repertoire (dossier,*repository*) classique en repo git.
- `>git add mon_fichier` : "Linker" un fichier à votre repo git.
  Cette commande signale à git que le fichier doit être "suivi" (indexé). Elle indexe la version du fichier au moment où elle est appelée, si vous remodifiez "mon_fichier" ultérieurement sans rappeler "add", cela ne sera pas pris en compte (voir exemple plus bas).
- `>git add` : pour indexer tous les élements du repo courrant.
- `>git clone adresse_du_repository/nom_du_repo` : copier un repository existant dans un nouveau repertoire (e.g une url github). Cela crée un dossier "nom_du_repo" qui contient la dernière version du repository.
- `>git status` : obtenir l'état du repository et des indexations en cours.
- `>git commit -m "Message"` : soumettre les modifications de fichiers - i.e enregistrer un instantanné des versions des fichiers suivis. Attention: dans ce cas là, la soumission n'envoie que les modifications enregistrées au moment des appel `add`.
  Pour forcer la prise en compte des modifications effectuées (sur des fichiers déjà indexés) *après* un add : `>git commit -am "Message"`
- `>git checkout -- nom_fichier` : Réinitialiser un fichier à sa version commitée. Attention !!! Ceci ecrase toutes les modifications potentielles en cours sur votre fichier "local" !

- `fichier .gitignore` : contient les expressions régulières des fichiers qu'il ne faut pas indéxer (par exemple `*.pyc`)

## 2 Dépots distants

Les commandes précédentes n'agissaient que en local (création et utilisation d'un dépot local). L'intérêt des logiciels de gestion de version est de pouvoir être décentralisé et accessible par réseau. Les commandes suivantes permettent d'interagir avec un dépot distant.

Pour plus d'infos sur le fonctionnement et l'interet des dépots distans : `http://tinyurl.com/gopz8bf`

- `>git remote -v` : visualiser les dépots distants auxquels vous êtes liés.
- `>git remote add le_depot adresse_du_depot` : ajouter un dépot distant.
- `>git pull origin master` (ou `depot branche`) : récupérer et fusionner les données du dépot distant.
- `> git fetch le depot` : récupérer sans fusionner les données du dépot distant. Cela récupère aussi les branches du dépot.
- `>git push [branche_machin le_depot]` : pousser votre travail (par défaut sur le dépot *origin*, sinon sur la branche *branche_machin*) sur le dépot Il faut que vous ayez les droits en écriture pour push.

## 3 Branches

Vous pouvez créer avec git des *branches* dans votre code, ce qui permet de travailler sur différents aspects en parallèle en gardant un code commun (branche *master*) stables. Une branche peut ensuite être mergée à la branche *master*.

- `>git branch -v` : obtenir la liste des branches existantes, celle précédée d'une étoile étant celle où vous vous trouvez.
- `>git branch branche_machin` : pour créer une branche.
- `>git checkout branche_machin` : pour basculer dans cette branche.
- Pour merger une branche avec la branche master (principale):

  ```
  #Se déplacer dans la branche master
  >git checkout master
  #merger
  >git merged branche_machin
  ```

  Après merging, vous pouvez effacer la branche : `>git branch -d branche_machin`

# 4 Un exemple pour initialiser le projet et votre dépot:

Enregistrez-vous d'abord sur github.com et créér un projet.

```
### configurer le proxy
> export https_proxy = proxy:3128

### se placer dans un repertoire pour cloner le dépot du projet, puis
> git clone http://github.com/baskiotisn/soccersimulator

### Ceci est un clone du projet initial, pas votre espace de travail.
### Il permet de récupérer les nouveaux cours, les versions de la plateforme etc.
### Vous pouvez installer le paquet python  (il faudra le reinstaller a chaque changement
> pip install -e soccersimulator --user

### Vous allez créer maintenant votre dossier de travail.
>mkdir mon_repo_git
>cd mon_repo_git
>git init
### Git vide initialisé dans mon_repo_git/.git/
### Creation d'un fichier et ajout au depot
>echo "import soccersimulator" > exemple.py
>git add exemple.py
>git status
 Sur la branche master
 Validation initiale
Modifications qui seront validées :
  (utilisez "git rm --cached <fichier>..." pour désindexer)
        nouveau fichier: exemple.py
>echo "print 1" >> exemple.py

>git status
 Sur la branche master
 Validation initiale
 Modifications qui seront validées :
  (utilisez "git rm --cached <fichier>..." pour désindexer)

        nouveau fichier: exemple.py

Modifications qui ne seront pas validées :
  (utilisez "git add <fichier>..." pour mettre à jour ce qui sera validé)
  (utilisez "git checkout -- <fichier>..." pour annuler les modifications dans
  la copie de travail)

        modifié:         exemple.py

>git commit -m "blabla"
[...]
 1 files changed, 1 insertion(+)
```

```
 create mode 100644 exemple.py
#### Vous avez enfin un fichier dans votre git (apres le commit).
#### Pour ajouter un dépot distant
> git remote add origin https://github.com/username/projet.git
#### Pour envoyer les modifications au dépot distant
> git push origin master
#### Allez voir sur github, il devrait y avoir un fichier maintenant.
#### Pour récupérer les modifications
> git pull origin master
#### Observer le comportement de git sur ces commandes :
> touch ex2.py
> git status
> echo "modification_b" > ex2.py
> git add ex2.py
> echo "print 2" >> exemple.py
> git status
> git checkout -- exemple.py
> git status
```