Mémento GIT – STS SIO

Configurer GIT

```
$ git config --global user.name " Votre nom ou votre pseudo "
$ git config --global user.email "votre@email.com"
$ git config --global http.proxy http://10.1.2.5:8080
$ git config --global http.sslVerify false
```

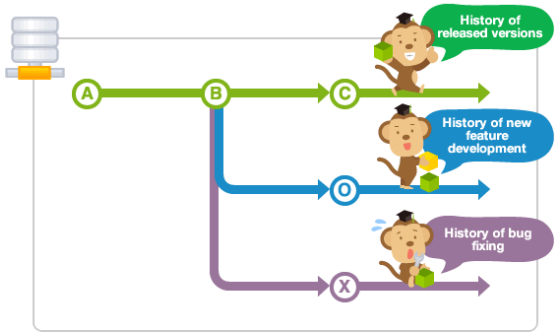
(avec nom et email utilisé dans Github) -- les deux denières lignes, uniquement au lycée !
L'option --global permet de définir les paramètres pour votre compte utilisateur. Sinon, elles sont définies uniquement pour le projet.

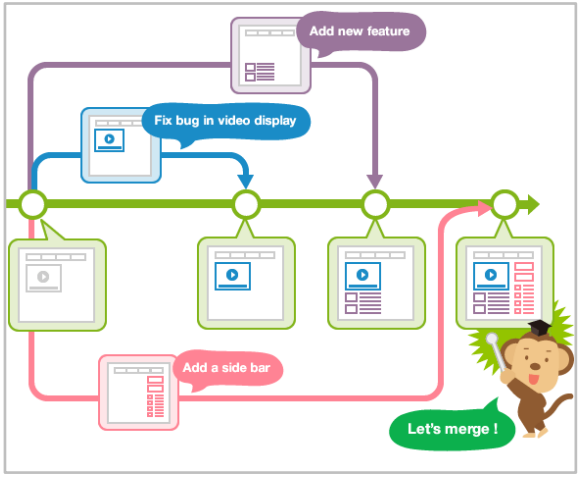
Ignorer des fichiers ou répertoires

Si, à l'intérieur d'un dépôt, on souhaite ignorer certains fichiers ou ou répertoires, il faut créer un fichier .gitignore et y rajouter la liste de ce que l'on ne souhaite pas voir :

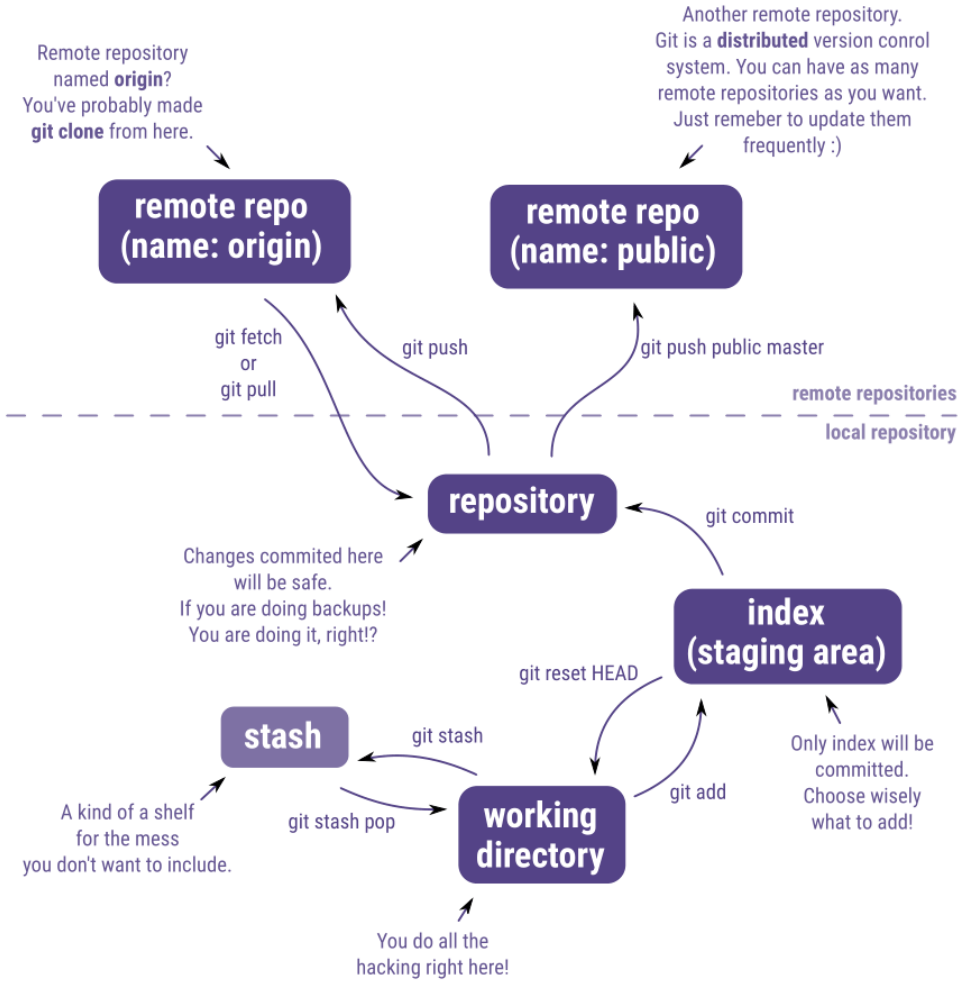
```
!.gitignore          (ne pas ignorer le fichier .gitignore)
.~lock.*             (ignorer les fichiers temporaires libreoffice)
/logs/*              (ignorer tous les fichiers du répertoire logs)
/tmp                 (ignorer le dossier tmp et son contenu)
*.docx               (ignorer tous les fichiers ayant une extension docx)
monfic               (ignorer le fichier « monfic » à la racine du projet)
```

Les branches en image





Les différentes zones de travail de Git (pour une unique branche)



Mémento des commandes GIT usuelles – STS SIO

Démarrer un projet

`$ git init`

Crée un dépôt vide dans le dossier courant, ne contenant rien, quelque soit l'état du répertoire. Le dépôt est géré par l'entremise du répertoire .git que la commande vient de créer.

Travailler au quotidien sur son dépôt local

`$ git config --list`

Affiche les paramètres de configurations. Permet notamment de voir l'url du dépôt distant associé.

`$ git status`

Vérifie le statut du répertoire de travail (fichiers à l'état new, staged, modified et fichiers ignorés) de la branche courante.

`$ git log`

Affiche l'historique des commits

`$ git log --graph`

Affiche l'historique des commits avec une représentation graphique des branches

`$ git diff file`

Montre les différences entre répertoire de travail et index (staging area)

`$ git diff --staged`

Montre les changements effectués dans l'index mais non encore committés pour le fichier `unFichier`.

`$ git add .`

Ajoute tous les fichiers mis à jour (crées, modifiés, supprimés) dans l'index (➔ mettre en attente)

`$ git add fichier1`

Place le fichier `fichier1` *dans l'index.*

`$ git rm unFichier`

Supprime le fichier `unFichier` *et ajoute la suppression à l'index.*

`$ git rm –r unRepertoire`

Supprime le répertoire `unRepertoire` *et tout ce qu'il contient et ajoute la suppression à l'index.*

`$ git commit –m "message de commit"`

Commit des fichiers ajoutés à l'index (➔ valider)

`$ git commit –am "message de commit"`

Commit des fichiers modifiés/supprimés de la zone de travail et des nouveaux fichiers de la zone index.

`$ git stash`

Remisage des modifications non encore validées (commit) pour pouvoir quitter la branche courante

`$ git stash apply --index`

Applique les modifications remisées précédemment

`$ git stash drop`

Supprime les modifications remisées

`$ git stash pop`

Applique les modifications remisées précédemment et les supprime s'il n'y a pas de conflit (équivalent de git stash apply suivi de git stash drop)

Travailler avec les branches en local

$ git branch

Affiche toutes les branches locales.

$ git branch *uneBranche*

Crée une nouvelle branche `uneBranche` *à partir de la branche courante. On reste positionné dans la branche courante.*

$ git checkout -b maBranche

Crée en local une nouvelle branche `uneBranche` *à partir de la branche courante . On est positionné automatiquement dans la nouvelle branche*

$ git checkout -b maBranche autreBranche

Crée en local une nouvelle branche `uneBranche` *à partir de la branche* `autreBranche` *. On est positionné automatiquement dans la nouvelle branche*

$ git checkout *uneBranche*

Bascule vers la zone de travail de la branche `uneBranche`

$ git switch *uneBranche*

Bascule vers la zone de travail de la branche `uneBranche`

$ git branch –D *uneBranche*

Supprime la branche locale `uneBranche`

$ git merge *uneBranche*

Fusionne le contenu de `uneBranche` *avec la branche courante*

Travailler avec un dépôt distant

$ git clone url_projet

Télécharge le dépôt distant situé à l'adresse url_projet dans le répertoire courant

$ git remote add origin https://url_depot_distant

Associe le dépôt local au dépôt distant existant et vide

$ git remote remove origin

Supprime l'association entre le dépôt local et le dépôt distant (les dépôts ne sont pas supprimés)

$ git remote -v

Donne la liste de tous les référentiels distants actuellement connectés à votre référentiel local

$ git pull

Télécharger les nouveaux commits, branches et tags d'un dépôt distant sur votre machine, sans modifier vos fichiers de travail ni les branches locales actuelles.

$ git pull

Met à jour la branche locale active en y fusionnant les modifications de sa branche distante associée, sans toucher aux autres branches.

$ git push

Envoie les modifications de la branche locale active vers sa branche distante associée.

$ git branch

Affiche toutes les branches locales.

$ git branch -a

Affiche toutes les branches locales et distantes.

$ git checkout --track origin/maBranche

Récupère en local la branche `uneBranche` *se trouvant sur le serveur distant (mais pas en local)*

$ git push origin maBranche

Envoie la branche `maBranche` *sur le dépôt distant*

$ git pull origin maBranche

Récupère les nouveautés de la branche distante `maBranche` *les fusionne dans la branche locale active. ATTENTION à la branche locale dans laquelle vous vous trouvez. Equivaut à* `git fetch` *suivi de* `git merge`

$ git push --all

Envoie toutes les branches sur le dépôt distant (origin)

$ git push origin -d maBranche

Supprime la branche `maBranche` *sur le dépôt distant (origin)*