**Git & Github : mode d’emploi.**

***Ce document est une prise de notes sur le MOOC OpenClassrooms « Gérer votre code avec Git et Github », disponible à l’adresse suivante :*** [**https://openclassrooms.com/fr/courses/2342361-gerez-votre-code-avec-git-et-github**](https://openclassrooms.com/fr/courses/2342361-gerez-votre-code-avec-git-et-github)**.**

* Logiciel de gestion de version de code : logiciel de qui gère les versions d’un code au fur et à mesure que celui-ci est modifié
* Objectif : apprendre les commit, branche, merge…

Les modifications du code sont enregistrées dans un **commit (=version d’un code à un instant t).**

Stack Overflow : forum pour entraide entre développeurs

Logiciel de gestion de version basé soit sur modèle centralisé (un serveur central contrôle toute la base de code du logiciel), soit sur modèle distribué (toutes les machines ont accès à la base de code)

Git est basé sur un modèle centralisé, ce qui a pour avantage : moins de risque de perdre code, qui est distribué sur toutes les machines, pas besoin d’internet car pas de connexion à un serveur central

Installation de la console (=terminal) de Git sur mon ordi

Une console permet de naviguer dans une arborescence de fichiers de la même façon qu’une interface graphique sauf qu’au lieu de clics on écrit des commandes

Après installation de Git, on peut ouvrir Git Bash depuis le menu Démarrer

Commandes de base d’une console :

* pwd : permet de connaître le répertoire courant, soit l’endroit où on se situe dans l’arborescence
* ls : permet de voir la liste des fichiers et répertoires dans le dossier courant
* cd répertoire : permet de se placer dans le répertoire demander
* touch permet de créer un fichier (ex : touch fichier.txt créer un fichier texte dans le répertoire courant)
* mkdir : permet de créer un dossier dans le répertoire courant (ex : mkdir dossier)
* cat : permet d’afficher le contenu d’un fichier

(directory = fichier)

Pour coller un texte copié dans la console Git : Fn + Inser : attention, on ne peut copier qu’une ligne par une ligne

Créer un repository (répertoire) grâce à la commande mkdir. Pour activer ce dossier comme repository Git, il faut lancer la commande git init dans le terminal Bash en se plaçant dans le dossier en question. Pour gérer ce repository, Git génère un index de tous les fichiers dont il doit faire le suivi. Donc quand on crée un fichier dans un repository, on l’ajoute à l’index Git avec la commande :

**git add nomDeVotreFichier.extension** (il faut d’abord que le fichier se trouve dans le répertoire avant de faire un git add) ou sinon « git add . » Pour rajouter tous les fichiers dans le répertoire courant

Quand le repository est modifié, on demande à Git d’enregistrer ces modifs en faisant un **git commit.** L’option -m permet d’ajouter du texte au commit pour décrire la modification.

Pour lire l’**historique** des commits, on peut taper **git log** (commit le plus récent en haut de la liste) : on a alors pour chaque commit son SHA, identifiant unique sous la forme d’une suite de lettres et de chiffres.

Si on ne fait que mettre à jour un fichier déjà ajouté à l’index Git, on peut faire un commit avec la commande (au lieu de faire add puis commit) : git commit -a -m "Ajouté itinéraire dans checklist-vacances.md" : la commande -a permet de mettre à jour les fichiers déjà existant dans l’index Git

Si on veut se positionner sur une version antérieure du code, on fait **git checkout SHADuCommit** et pour revenir à la branche principale, soit le commit le plus récent, on tape **git checkout master**.

Si on veut modifier le message de notre dernier commit, on fait **git commit –amend -m « mon message »**

On peut pas supprimer un commit.

**Remote** : autre machine sur laquelle notre code est sauvegardé, permet d’avoir un backup de notre code si jamais mon ordi tombe en panne. Remote intern : un autre ordi qu’on possède. Remote externe : Github. Un remote externe permet également de pouvoir travailler à plusieurs grâce à un remote partagé.Github héberge donc les repository de code.

Il faut se créer un compte sur Github. Sur ce site, on peut communiquer avec d’autres developpeurs, partager des morceaux de code en ligne (« gists », façon de partager son code sur GitHub), récupérer du code depuis un autre repository.

Pour copier un repository sur ma machine, je me place sur le repository sur Github puis en bas à droite de l’écran, cliquer sur le bouton vert **« clone or download ».** On copie le lien https, puis dans Bash, on exécute la commande **git clone LienURLduRepository**.

Pour créer un nouveau repository, cliquer sur le + en haut à gauche de l’écran sur Github, puis New Repository. La fonction **Initialize this repository with a README** permet de cloner le repository directement sur ma machine, à faire quand on n’a pas encore créé de repository sur notre machine : on clone ensuite le repo sur notre machine. Ignorer les autres options et mettre en public le repo (sinon payant en privé).

Pour copier du texte dans Git, sélectionner le texte avec la souris puis simplement appuyer sur **Entrée**.

Une fois les commits effectués sur le repository, on peut les envoyer sur Github à l’aide de la commande **git push origin master**: permet de demander à Git d’envoyer les modifs dans la branche master de mon remote.

La **branche master** contient le code courant de de mon repo. Par défaut, le remote sur lequel on envoie notre code est appelé **origin**. Ici, ce remote est Github. On peut accéder aux différents commits sur Github en cliquant sur l’onglet **commits**.

Si mes potes modifient le code que j’ai mis en ligne et le push dans le repo Github, mon code local n’est plus à jour : il faut le récupérer en local, grâce à la commande **git pull origin master**, qui envoie dans le répertoire local les dernières modifications de la branche master située sur mon remote origin (ici, Github). Attention : bien penser à souvent faire un pull pour que notre code soit à jour.

Sur Git, on peut créer des branches : pour travailler sur des versions de code qui divergent de la branche principale (qui contient le code courant) : utile quand on veut expérimenter sur notre projet, ou quand on veut se focaliser sur le développement d’une fonctionnalité spécifique.

A l’initialisation d’un repo Git, le code est placé dans la branche principale, appelée **master**. Pour voir les branches existantes dans le repo, faire la commande **git branch** : retourne les branches, avec une étoile \* devant la branche sur laquelle on est placé. Pour créer une nouvelle branche, on fait **git branch nouvelle-branche**. Pour se déplacer sur une nouvelle branche, on fait **git checkout ma-branche**. Astuce : avec **git checkout -b branche**, on peut créer une branch et s’y positionner directement.

Pour ajouter les mises à jour d’une autre branche B sur ma branche A, on peut fusionner les branches. On se place sur la branche A avec git checkout brancheA et on merge les branches avec **git merge brancheB**. Idée : toujours chercher à remettre les modifications faites sur mes différentes branches sur la branche master.

Souvent, il y a des conflits quand on veut fusionner des branches. Si un fichier texte hello.txt dans la branche master contient à la première ligne « hello » et qu’un autre fichier aussi nommé hello.txt dans la branche Autre-branche contient à la première ligne « bonjour », il va y avoir un conflit entre les deux branches. L’erreur suivante s’affichera :

Auto-merging hello.md

CONFLICT (content): Merge conflict in hello.md

Automatic merge failed; fix conflicts and then commit the result.

Avec l’éditeur de texte, on regarde les différences entre les deux fichiers et on regarde quel contenu on garde pour la branche master. Là, si on fait git status, Git va nous dire qu’il y a des branches non fusionnées (« unmerged paths »). Pour dire à Git qu’on a résolu le conflit, on envoit sans message sur la console un git commit. Git va alors proposer un message de commit par défaut. On peut sauvegarder ce message en tapant **:x** dans la console.

(Outils permettant de comparer les différences de version d’un fichier : vimdiff, meld, opendiff, kdiff3, tkdiff, xxdiff, tortoisemerge, gvimdiff, diffuse, ecmerge, p4merge, araxis, emerge)