**Mes fonctions Git**

**git statut** = voir l'état du dossier, si il est sous git ou non,

**git init** = initialise le dossier pour git.

**git log** = affiche la liste de tous les commits réalisés

**git checkout** numéro du commit = le numéro du commit (cat) pour revenir sur des versions de commit plus ancienne. utile pour trouvé certaines erreurs et ne pas les mettres sur la branche master

**git checkout master** = retour sur la branche principale.

**cat** nom du fichier = cat est le numero du commit, affiche le code de la periode voulu.

**cat AUTHORS** = personne(s) ayant contribué au projet.

**git reset --hard** = Je n'ai pas encore fait mon nouveau commit, mais je veux annuler tous les changements que je n'ai pas encore commités.

**cd** dossier= entre dans le dossier

**cd ..** = sort du dossier

**ls** = agit comme **dir** mais différencie les fichiers des dossiers avec de la coloration.

**dir** = affiches tous les documents et fichiers.

**mkdir** non de dossier= création d'un dossier

**touch** fichier = création d'un fichier

**git add** nom de fichier = ajoute le ficher au répertoire de git

**git commit -m** = suite de modifications avec option -m (pour message) qui designe l'action, modification faites.

**-a** = l'option -a, demande a git de mettre à jour un fichier déja existant dans son index (répertoire)

Lorsque vous mettez à jour un fichier dans votre repository, vous devez procéder en deux étapes :

Ajouter votre fichier à l'index avec la commande **git add**,

**git add** checklist-vacances.md

Faire un commit qui décrit la mise à jour de votre fichier avec la commande **git commit**.

**git commit** -m "Ajouté itinéraire dans checklist-vacances.md"

Si vous ne faites que mettre à jour un fichier que vous aviez déjà ajouté à l'index, vous pouvez condenser ces deux étapes de la façon suivante :

**git commit -a -m** "Ajouté itinéraire dans checklist-vacances.md"

ou **git commit -am** "Ajouté itinéraire dans checklist-vacances.md"

**git remote** = c'est juste après l'avoir commit. **Sauvegarde**r son code sur un pc autre que celui où l'on travail ou sur un site internet ex: github, bitbucket.

**git push origin master** = envoye du PC à Github, origine (destination ) et master (nom de la branche).

**git pull origin master** = inverse de **git push**. Récupère seulement la dernière version du fichier modifier.

**Github**

Github permet de :

Communiquer avec d'autres développeurs et signaler des problèmes de code en déclarant des "issues" ;

Partager des morceaux de code en ligne à l'aide de "gists" ;

Proposer des modifications de code à d'autres repos en faisant des "pull requests" ;

Et même récupérer du code depuis un autre repository.

**le clonage d'un repository:**

À partir de GitHub, vous pouvez copier un repository sur votre machine. Pour cela, il vous suffit de rechercher le repository qui vous intéresse sur GitHub, de vous y placer, puis d'utiliser l'option "clone URL" en bas à droite de l'écran.

Cette option vous propose un lien SSH, HTTPS ou Subversion. Ici, nous allons choisir un lien HTTPS, le copier, puis coller ce lien en utilisant la commande git clone dans le dossier que vous aurez choisi sur votre machine :

**git clone** lienFourniParGitHub

Dans cette vidéo, nous avons cloné le repo de la librairie React.js, une librairie créée par Facebook et qui permet de mieux gérer ses interfaces graphiques grâce à JavaScript. Nous avons donc effectué sur notre machine la commande :

**git clone** <https://github.com/facebook/react.git>

**Mettre son repository sur Github :**

C'est parti pour créer votre premier repository open-source :) . Tout d'abord, si ce n'est pas déjà fait, connectez-vous à votre compte GitHub. Cliquez sur le bouton "Create new" symbolisé par un signe "+" en haut à droite de votre écran, puis "New repository".

GitHub vous demandera alors de préciser quelques détails sur votre repository:

son nom

sa description

son statut public ou privé. Ici, nous partageons du code open-source, c'est donc public et gratuit :).

Vous avez également une option "Initialise with a README", qui vous permet de cloner votre repository sur votre machine. Cette option est à cocher uniquement dans le cas où vous n'avez pas encore créé le repository en question sur votre machine (ce qui est bien notre cas ici, cochez donc !)

Et voilà, vous avez créé votre premier repository sur GitHub ! Vous pouvez maintenant :

voir les commits effectués sur votre repository (par vous et vos éventuels contributeurs)

cloner votre projet et le ramener sur votre machine avec la commande git clone que l'on a vue dans le chapitre précédent.

... et en parlant de la commande git clone, notez que vous pouvez cloner votre repo avec deux options :

L'option HTTPS que je vous ai invité à utiliser dans ce cours : c'est l'option la plus simple, qui demande de fournir vos identifiants GitHub (nom d'utilisateur et mot de passe) à chaque fois que vous faites un git clone.

L'option SSH qui est plus pratique car elle ne vous demande pas vos identifiants à chaque fois. Par contre, pour l'utiliser, il faut générer une clé SSH, ce qui est un peu plus compliqué pour ce cours d'initiation. Mais une fois que vous vous sentez plus à l'aise, je vous invite à consulter la marche à suivre dans la documentation de GitHub et à utiliser cette option SSH.

**Les branches :**

À quoi ça sert de créer des variations de la branche principale ?

Travailler sur plusieurs branches est très utile lorsque vous souhaitez tester une expérimentation sur votre projet, ou encore pour vous concentrer sur le développement d'une fonctionnalité spécifique.

Voyons les commandes Git qui vous permettent de manipuler les branches.

Lorsque vous initialisez un repo Git, votre code est placé dans la branche principale appelée master par défaut.

Pour voir les branches présentes dans votre repo, utilisez la commande git branch. Elle vous retournera les branches présentes, et ajoutera une étoile devant la branche dans laquelle vous êtes placés. Par exemple, dans le 1er repos que vous avez créé dans la partie précédente, la commande git branch n'affichera qu'une seule branche, la branche principale dans laquelle vous vous situez : \* master.

Pour créer une nouvelle branche, il vous suffit d'ajouter le nom de la branche à créer à la suite de la commande précédente :

**git branch** nouvelle-branche

Pour supprimer une branche :

git branch -D ma-branche (seulement si elle n'est pas "merge" avec une autre)

Pour vous placer dans une autre branche à l'intérieur de votre repo, vous allez avoir besoin d'un nouveau mot-clé : checkout :

**git checkout** nouvelle-branche

Petite astuce pour manipuler vos branches : vous pouvez utiliser la commande 'git checkout -b' pour créer une branche et vous y positionner. Ainsi, au lieu de taper la commande suivante pour créer votre branche :

**git branch** ma-branche

, puis une deuxième commande pour vous y positionner :

**git checkout** ma-branche

, vous pouvez regrouper ces deux opérations en une seule commande :

**git checkout -b** ma-branche

, pour voir sur quel branch on est :

**git branch**

**,** fusionner des branches , il faut se positionner une branche autre que celle que l'on va fusionner :

**git merge** ma-branche