ESPRIT WEB

Mémo Git

Vocabulaire

Ligne de commande : Git s'exécute en ligne de commande. C'est-à-dire qu'il faut exécuter un enchainement de commande pour pouvoir l'utiliser. Tu peux retrouver ces commandes ci-dessous (« Les commandes »). Sur Windows, Git n'est pas natif, il n'est pas intégré par défaut dans le système d'exploitation. Il faut donc installer un programme téléchargeable. Ainsi tu pourras exécuter toute les commandes de Git via ton invité de commande. Pour Windows, je recommande d'utiliser Cmder, un invité de commande qui intègre directement Git et qui est beaucoup plus ergonomique.

Repository / dépôt : Les dépôts, aussi appelé repository, ou même « repo », sont les espaces sur lesquels tu déposes ton code, où tes projets vivent. C'est là que tout ton code est stocké. Ces repos peuvent être local (sur votre machine) ou distant (hébergé sur GitHub par exemple).

Contrôle de version : C'est le fonctionnement fondamental de Git. C'est le système qui permet de gérer les différentes versions de tes fichiers. Chaque fois que tu modifie un fichier, tu écrases le précédent. Git sauvegarde toutes ces modifications et créé des versions de chaque modification.

Commit : c'est la commande qui permet de dire à Git de générer une version. Lorsque que tu fais un « git commit », tu dis à Git : prends-moi une photo de l'état actuel de mes fichiers. Tu créer alors une version.

Branche : Lorsque que plusieurs personnes travaillent sur des fonctionnalités différentes, imagine le bazar que ça serait si tout le monde commité ces fichiers en même temps. Pour régler ce problème, git propose un système de branche. Imagine un arbre où le tronc est la partie commune du projet et à chaque nouvelle fonctionnalité, le développeur, part de ce tronc, et créer une branche indépendante pour créer sa fonctionnalité. Ainsi, chacun de ces commit ne vient pas gêner la partie commune. Une fois la fonctionnalité terminée, la branche peut être fusionner (merge) avec la partie commune.

Merger : le « merge », permet de fusionner les branches entre elles. C'est là tout la puissance de Git. Il compare les différentes modifications apportées aux différents fichiers et les fusionne ensemble.

Cloner : lorsque que tu souhaites récupérer le repository, d'un autre développeur sur ton ordinateur, tu effectues un clone. C'est-à-dire que « clone » son repository sur ta machine. Ainsi tu peux travailler sur ce code depuis ton poste.

.gitignore : c'est un fichier très pratique. Dans ce fichier, tu as la possibilité d'indiquer à Git, quels sont les fichiers que tu ne souhaites pas versionner. Par exemple, si tu as un fichier de configuration avec des mots de passes dans ton projet et que tu ne souhaites par le versionner, tu l'indique dans le fichier .gitignore

Les commandes

git init : initialise un dépôt git. Lorsque que tu viens de créer un dossier avec ton projet dedans, il n'est pas encore lié à Git. La commande git init, permet donc d'initialiser Git dans ce dossier. Ainsi toute les modifications à l'intérieur de ce dossier seront enregistrées via Git. Lorsque que tu initialise Git, un dossier masqué «.git » est créé avec, à l'intérieur, la configuration.

git config : Commande pour configurer Git. Elle permet notamment d'indiquer à Git les préférences de l'utilisateur. Tu peux y renseigner, ton email, ton nom d'utilisateur, l'algorithme utilisé pour le « diff ». Ex : *git config –global user.email paul@mail.com*. –global indique à Git que tu souhaites que cette configuration soit appliquée à chaque fois que tu utiliseras Git, peu importe le projet.

git clone : commande qui te permets de récupérer un dépôt existant. Exemple : *git clone https://github.com/WordPress/WordPress.git wordpress*

git help : très pratique lorsque tu débutes avec git. Cette commande te permet de lister l'ensemble des commandes disponible sous Git avec les paramètres qu'elles attendent.

git status : te permets de lister les fichiers qui ont été modifiés et qui doivent être ajoutés

git diff : te permets voir les modifications apporter à un fichier ou de comparer plusieurs branches

git add : permet d'ajouter un fichier à l'index de Git. Lorsque tu modifie tes fichiers, il faut que tu indiques à Git que tu souhaites, ou non, les sauvegarder. Ainsi, si tu souhaites sauvegarder un fichier, tu peux l'ajouter à l'index : *git add nomDuFichier*

git commit : une fois que tu as ajouté tous tes fichiers que tu souhaité versionné à l'index Git, le commit te permet de créer une version. Chaque commit possède un hash (« 85zd59z »), c'est le numéro de version.

git push : lorsque tu as créé ta version en « commitant », tu dois ensuite envoyer cette version sur un repository distant, par exemple sur GitHub. Le commit se fait en spécifiant sur quelle branche tu souhaites « pousser » tes modifications

git pull : lorsque vous êtes plusieurs à travailler sur un même projet, il faut parfois récupérer, sur ton poste, le travail que les autres développeurs ont poussés sur le repository. **git pull** te permets de faire ça

git checkout : permet de changer de branche ou d'en créer une nouvelle : *git checkout –b nomDeLaNouvelleBranche*

git merge : te permets de fusionner deux branches entre elles. Très pratique lorsque tu as fini de développer une nouvelle fonctionnalité et que tu souhaites l'intégrer à la branche principale « master »

git stash / git stash apply : c'est une sorte de commit temporaire. **git stash** te permets d'enregistrer temporairement des fichiers qui ne doivent pas être commité directement. C'est très pratique lorsque tu t'aperçois que tu viens de faire des modifications sur la mauvaise branche. Tu peux ainsi créer un commit temporaire avec **git stash**, changer de branche, et appliquer ces changements à la branche actuelle avec « **git stash apply** »