**GIT**

Un logiciel de gestion de versioning créé par Linus Torvald

**Qu’est-ce que versionner son code**

V**ersionner votre code signifie** gérer les versions de votre code au fur et à mesure que vous le modifiez.

Lorsque vous travaillez sur un projet de code, vous allez régulièrement y apporter des modifications, et par moments ces modifications vont provoquer des bugs. Lorsque vous revenez sur votre projet après quelques jours ou même quelques heures, il peut être difficile de vous souvenir des dernières modifications que vous avez effectuées et de retrouver vos repères dans votre code.

Avec un logiciel de versioning comme Git, vous pouvez garder la trace de toutes les modifications faites sur votre code pour pouvoir vous y retrouver à tout moment. À chaque fois que vous faites une série de modifications (créer un fichier, supprimer un fichier, modifier un texte dans un fichier, etc.),  vous allez pouvoir enregistrer ces modifs dans un **commit**.

Git gère les modifications apportées et non les fichiers eux mêmes.

Tu n’es pas obligé de faire un commit à chaque modification.

Par exemple, si vous travaillez sur un formulaire de newsletter en ligne :

• Vous allez d'abord faire une 1re série de modifications pour créer la base HTML/CSS de votre formulaire.    
=> Une fois ces modifications faites, vous pourrez faire un premier commit dans Git et le nommer "1re passe sur le formulaire de newsletter".

• Ensuite, vous rendrez peut-être votre formulaire dynamique avec du PH test .    
=> 2e commit : "PHP ajouté".

• Enfin, un collègue vous demandera de changer le wording sur le champs du formulaire.    
=> 3e commit : "Texte reformulé".

Un **commit** correspond donc à une **version** de votre code à un instant t.

La somme de tous les commits constitue l'historique de votre projet.

**Les différentes solutions de versionning**

Il y a deux types de logiciels de versioning:

**• Modèle centralisé** : un serveur central contrôle toute la base de code du logiciel.   
Exemples de logiciels de versioning utilisant un modèle centralisé : SVN, CVS.

Tu dois voir accès à internet afin d’accéder à l’historique des différentes versions.

Le projet est centralisé sur un seul serveur.

**• Modèle** **distribué** : toutes les machines ont accès à la base de code, pas besoin de passer par un serveur central.   
Exemples de logiciels de versioning utilisant un modèle distribué : Git, Mercurial, Bazaar.

En distribué toutes les modifications sont en locales

**Installez Git**

**Mac ou Linux**

Téléchargez la dernière version de Git sur : http://git-scm.com/downloads

Dans le terminal :

git config --global user.name "Votre nom ou pseudo"

git config --global user.email "Votre@email.com"

Pour vérifier que tout va bien, relancez votre console et tapez simplement 'git'. Si l’installation a fonctionné, vous devriez voir du texte en anglais expliquant l’utilisation de Git.

#### Windows

Rendez vous sur [http://msysgit.github.io](https://msysgit.github.io) et téléchargez la dernière version disponible.

Maintenant vous allez pouvoir ouvrir l’application “git bash” qui se situe maintenant dans votre menu Démarrer.

git config --global user.name "Votre nom ou pseudo"

git config --global user.email "votre@email.com"

Pour vérifier que tout va bien, relancez votre console et tapez simplement 'git'. Si l’installation a fonctionné, vous devriez voir du texte en anglais expliquant l’utilisation de Git.

**Faites votre premier commit**

Pour commencer, créez **un nouveau dossier** ou prenez un dossier existant et positionnez vous dedans avec la console.

Pour activer un dossier comme repository Git, il suffit de se placer dans ce dossier avec le Terminal puis d'utiliser la commande **git init**.

Pour gérer un repository, Git génère un index de tous les fichiers dont il doit faire le suivi. Lorsque vous créez un fichier dans un repository, vous devez donc l'ajouter à l'index Git à l'aide de la commande **git add nomDeVotreFichier.extension**.

Pour gagner du temps, vous pouvez ajouter tous les fichiers dans le répertoire courant en tapant

git add .

Lorsque vous modifiez votre repository, vous devez demander à Git d'enregistrer vos modifications en faisant un **git commit**. L'option-m vous permet de lui envoyer un message décrivant les modifications effectuées, ce qui s'avèrera très utile pour vous par la suite, exemple :

git commit -m "Ajouté ma checklist-vacances.md (woohoo!) »

**Lisez l’historique**

La commande git log affiche la liste de tous les commits que vous avez réalisés !

Le sens de lecture du log est le suivant : on part du commit le plus récent en haut de la liste, au commit le plus vieux en bas de la liste.

L’historique des commits:

Dans la liste de cet historique, chaque commit est répertorié avec :

• son **SHA** : son identifiant unique, qui se présente sous forme d'une longue chaîne de caractères et de nombres. Par exemple : "87753191cef0bdb955a4cb4ff841f7c2cce4cb1c".

• son auteur : qui a fait le commit (utile lorsque vous travaillez à plusieurs sur un projet !)

• sa date

• sa description : vous vous rappelez, c'est le message de description que vous indiquez avec l'option -m lorsque vous faites votre git commit ? Exemple, avec le fichier que l'on a "commité" dans le chapitre précédent.

Jusqu'ici, lorsque vous mettez à jour un fichier dans votre repository, vous devez procéder en deux étapes :

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

git add checklist-vacances.md

1. Faire un commit qui décrit la mise à jour de votre fichier avec la commande  git commit.
2. git commit -m "Ajouté itinéraire dans checklist-vacances.md"

Et bien, 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"

L'option   -a demande à Git de mettre à jour les fichiers déjà existants dans son index. Pratique, non ?

**Positionnez-vous sur un commit donné.**

Pour vous positionner sur un commit donné de votre historique, il vous suffit d'utiliser la commande **git checkout** de la façon suivante :

git checkout + SHA du commit: Permet de revenir à la version précédente avant modification, on peut la voir mais pas la modifier.

Pour revenir à votre branche principale (au commit le plus récent), on utilise la même commande :  git checkout master

Supprimer un commit ?

On ne peut pas vraiment "supprimer" un commit, mais on a plusieurs options pour l'annuler. Cependant, ces options ont des limites et sont à utiliser avec prudence et parcimonie !

Je vous présente une de ces options : vous pouvez "revert" un commit, c'est-à-dire créer un nouveau commit qui fait l'inverse du précédent, avec la commande suivante :

git revert + SHA du commit

Annuler tous les changements effectués et encore non commités :

git reset --hard

Pour supprimer l’index :

git reset

**Découvrez les remotes**

Lorsque vous travaillez sur un projet sur votre machine, il est important d'avoir un backup de votre code sur une autre machine, au cas où la vôtre tombe en panne par exemple. Une fois que vous avez travaillé sur votre code et effectué vos commits, vous allez donc les envoyer sur un **remote**, c'est-à-dire une autre machine

Github qu’est-ce que c’est ?

GitHub est un service en ligne qui permet d'héberger ses repositories de code. GitHub est un outil gratuit pour héberger du code open source, et propose également des plans payants pour les projets de code privés. C'est le numéro 1 mondial et il héberge plus d'une dizaine de millions de repositories !

Pour l’utiliser il faut créer un compte sur : https://github.com/

Avec Github vous allez pouvoir notamment :

* 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.

Git branch dev

Git checkout -b dev

La branche c’est la possibilité d’avoir une timeline parallèle, la branche principale est dite master.

Rappel des comandes:

git init

Git commit

Git help

Git status

Git og

Git log —online

Git diff

Un projet via Git à 3 étapes essentielles:

1) Git init

Pour initialiser un Répertoire de travail

2) Git add

Pour ajouter des fichiers ou dossiers à l’index afin de les surveiller.

Tous ces fichiers forment l’index ou le stage.

3) Git commit

Pour ajouter un commit.

**git commit**. L'option -m vous permet de lui envoyer un message décrivant les modifications effectuées, ce qui s'avèrera très utile pour vous par la suite, you'll see!  Par exemple :

git commit -m "Ajouté ma checklist-vacances.md (woohoo!) »

Git log affiche la liste de tous les commits que vous avez réalisés !

Git indexe les fichiers d'après leur [somme de contrôle](https://fr.wikipedia.org/wiki/Somme_de_contr%C3%B4le) calculée avec la [fonction de hachage](https://fr.wikipedia.org/wiki/Fonction_de_hachage) [SHA-1](https://fr.wikipedia.org/wiki/SHA-1). Quand un [fichier](https://fr.wikipedia.org/wiki/Fichier_informatique) n'est pas modifié, la somme de contrôle ne change pas et le fichier n'est stocké qu'une seule fois. En revanche, si le fichier est modifié, les deux versions sont stockées sur le disque.

Pour activer un dossier comme repository Git, il suffit de se placer dans ce dossier avec le Terminal puis d'utiliser la commande **git init**. Pour gérer un repository, Git génère un index de tous les fichiers dont il doit faire le suivi. Lorsque vous créez un fichier dans un repository, vous devez donc l'ajouter à l'index Git à l'aide de la commande **git add nomDeVotreFichier.extension**. Par exemple :

git add villes.txt