**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 PHP 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 ou dossiers 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 "Ajout de nouveaux dossiers"

# 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 "Ajout 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 "Ajout 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 voir l’index :

git status

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.

Un dépôt Git est un entrepôt virtuel de votre projet. Il vous permet d'enregistrer les versions de votre code et d'y accéder au besoin.

L’avantage de GitHub, c’est que depuis quelques années GitHub est devenu le **book/portfolio des développeurs** ! Dans beaucoup de processus de recrutement, on vous demandera maintenant votre lien GitHub.

## Les alternatives

Concernant [GitLab](https://about.gitlab.com/), il est la principale alternative à GitHub depuis le rachat de GitHub par Microsoft ! Les anti-Microsoft ont même lancé le hashtag #MovingToGitLab ! GitLab fonctionne avec une version gratuite à installer sur son propre serveur ou une version cloud payante.

[BitBucket](https://fr.atlassian.com/software/bitbucket) est la version de Atlassian. Payante, elle plaira néanmoins aux habitués de la gestion de projet sous Atlassian. BitBucket conviendra aussi bien aux étudiants ou petites teams qu’aux grands groupes.

Parlons enfin de SourceForge, le petit dinosaure dans le domaine. SourceForge a été créé 10 ans avant les autres, afin de gérer à la base des projets open source. SourceForge intègre un outil de suivi des bugs et un répertoire de code intégré. Il n’est plus très populaire depuis ces dernières années.

# Récupérez du code d’un autre 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

ex : git clone https://github.com/prakhar1989/docker-curriculum.git

# Créez votre premier repository

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**.

Je vais créer le dépôt IT que j’ai déjà en local

# Envoyez votre code sur GitHub

Si tu veux synchroniser ton dépôt local avec celui qui est distant tu tapes :

git remote add origin <https://github.com/qeyss/IT.git>

Ensuite pour tout envoyer :

git push -u origin master

Cette commande demande à Git :

"Envoie mes modifs dans la branche master de mon remote origin."

* La branche **master** est la branche qui contient le code courant de votre repo GitHub. Ne vous préoccupez pas trop du terme "branche", on y reviendra par la suite.

Lorsque vous lancez la commande git push, il est possible que l'on vous demande vos identifiants GitHub. Renseignez-les dans le terminal !

# Récupérez des modifications

Si vous modifiez votre repo GitHub en ligne, ou si vous travaillez avec d'autres personnes dessus et qu'elles envoient leurs modifications locales sur le repo en ligne, votre code local ne sera plus à jour.

Pour récupérer en local les dernières modifications du repo GitHub, il vous faut utiliser la commande **git pull** :

git pull origin master

Envoie dans mon répertoire local les dernières modifications de la branche master située sur mon remote origin (qui correspond ici à GitHub).

Pensez à synchroniser régulièrement votre code local avec vos repos en ligne à l'aide des commandes git push et pull. C'est particulièrement important lorsque vous travaillez à plusieurs sur un projet, pour que tout le monde avance sur la même base !

## Récapitulatif lors d’une modification

1° Enregistrez les modifications avec un commit : git commit -a -m "Blablabla"

2° Faire un push pour envoyer tout ça sur GitHub sinon les deux dépôts ne seront pas identiques :

git push origin master

# Créez des branches

Les branches permettent de travailler sur des versions de code qui divergent de la branche principale contenant votre code courant.

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.

Nous allons prendre un cas un peu plus concret. Imaginons que vous ayez réalisé une superbe application bancaire pour M. Robert, et que M. Robert ait une superbe idée de cagnotte à ajouter à son application. Panique à bord dans votre tête ! Votre application fonctionne parfaitement, elle est en production, et y toucher serait prendre le risque de faire tout planter. Avec Git et ses fameuses branches, pas de soucis. Vous allez pouvoir créer une branche correspondant à l’évolution cagnotte et cela, sans toucher à votre application en production qui fonctionne parfaitement. Une fois que toutes vos modifications auront été testées, vous pourrez les envoyer en production sans crainte (et dans le pire des cas, revenir en arrière simplement) !

 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.

* 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

# Fusionnez des branches

Lorsque vous travaillez sur plusieurs branches, il va souvent vous arriver de vouloir ajouter  dans une branche A les mises à jour que vous avez faites dans une autre branche B. Pour cela, on se place dans la branche A :

git checkout brancheA

Puis on utilise la commande **git merge** :

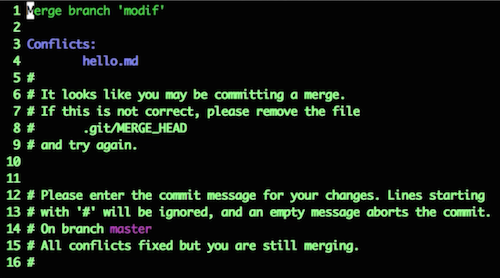
git merge brancheB

# Résolvez un conflit

Si vous avez deux fichiers au même nom mais différents, dans deux branches différentes, faudra les ouvrir et modifier l’un des deux pour les rendre identique puis faire un commit sans message :

git commit

Git va détecter que vous avez résolu le conflit et vous proposer un message de commit par défaut :



Vous pouvez  alors personnaliser le message du commit si vous le souhaitez. Dans notre cas, la résolution étant assez simple, nous allons garder le message proposé par défaut et le sauvegarder en tapant :x.

Git va alors vous confirmer que vos branches ont été fusionnées

# Retrouvez qui a fait une modification

Pour retrouver qui a modifié une ligne précise de code dans un projet, faire une recherche avec git log peut s'avérer compliqué, surtout si le projet contient beaucoup de commits. Il existe un autre moyen plus direct de retrouver qui a fait une modification particulière dans un fichier : la commande **git blame**.

git blame nomdufichier.extension

Cette commande liste toutes les modifications qui ont été faites sur le fichier ligne par ligne. À chaque modification est associé le début du **sha** du commit correspondant. Par exemple :

^05b1233 (Marc Gauthier 2014-08-08 00:31:02 1) # Une liste

Pour retrouver pourquoi cette modification a été faite, vous avez deux possibilités :

1. Faire un **git log** et rechercher le commit dont le sha commence par 05b1233.
2. Utiliser la commande **git show** qui vous renvoie directement les détails du commit recherché en saisissant le début de son sha :

git show 05b1233

On en revient à un point essentiel : pensez à écrire des messages clairs et précis lorsque vous faites vos commits. Cela vous facilitera la vie lorsque vous y reviendrez dessus plus tard, et la vie de vos collaborateurs

# Corrigez vos erreurs sur votre dépôt local

## J’ai crée une branche que je n’aurais pas du créer

# Corrigez vos erreurs sur votre dépôt distant

# Utilisez Git reset

# Corrigez un commit raté

# Ignorez des fichiers

# Evitez des commits superflus

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 commandes:

git init

Git commit

Git help

Git status

Git log

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 "Ajout 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