Dans ce cours, nous allons commencer par expliquer ce qu’est Git. Pour cela, nous allons définir ce qu’est un système de gestion de version et comprendre en quoi Git est différent des autres systèmes de gestion de version et pourquoi il est si populaire.

Nous allons ensuite découvrir les fonctionnalités principales de Git et apprendre à les utiliser. Nous apprendrons notamment à créer un nouveau dépôt Git avec la commande git init ou à cloner un dépôt existant avec git clone, à observer l’état d’un projet avec git status, à ajouter des fichiers à l’index avec git add, à valider des modifications avec git commit, etc.

……………………..

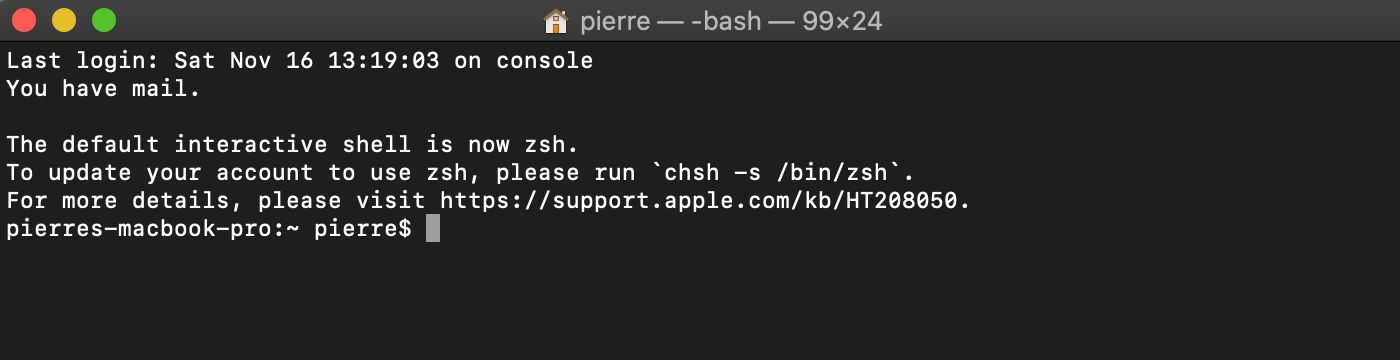
Dans cette leçon, nous allons commencer par définir ce qu’est une console et ce qu’est un langage en lignes de commande puisque ce sont les outils qu’on va utiliser pour travailler avec Git et allons installer et paramétrer le logiciel Git.

Utiliser Git : ligne de commande, console et interface graphique

On peut utiliser différents types d’interfaces pour installer et pour utiliser Git. Dans ce cours, nous allons utiliser la console et donc un langage en lignes de commande plutôt qu’une interface graphique. Nous faisons ce choix pour différentes raisons :

* Utiliser la ligne de commande est le seul moyen d’avoir accès à toutes les commandes Git ;
* Toutes les personnes vont avoir accès aux mêmes commandes avec la ligne de commande, ce qui signifie que tout le monde pourra suivre ce cours de la même façon
* Si vous savez comment utiliser la version en ligne de commande, vous saurez utiliser n’importe quelle autre interface graphique.

Commençons donc par rapidement présenter la console. Une console est une interface qui nous permet d’envoyer des commandes, c’est-à-dire des ordres directement à notre système (notre ordinateur) afin qu’il les exécute. La console, c’est cet écran noir qui nous parait très intimidant au premier abord :



Pour envoyer des commandes à notre système, nous utilisons un langage qui dépend du système utilisé. La plupart des commandes s’écrivent sur une ligne, d’où le terme de “langage en lignes de commande”.

Au début de l’informatique, les opérations se faisaient toutes en ligne de commande. Ensuite, des interfaces graphiques pour utilisateurs (GUI) ont été développées afin de rendre l’informatique plus accessible à tous. L’interface graphique utilisateur correspond à tout ce que vous voyez lorsque vous regardez votre ordinateur (les dossiers et fichiers sous forme d’icônes, etc.) et aux opérations auxquelles vous avez accès (ouvrir un fichier, le supprimer, le renommer…).

La particularité des interfaces graphique utilisateur par rapport à la console est qu’elles utilisent principalement la souris pour les opérations là où la console force les utilisateurs à n’utiliser que le clavier pour envoyer des commandes. Cependant, en arrière plan, ce sont les mêmes opérations qui se passent lorsqu’on utilise une GUI ou la console.

L’interface graphique utilisateur n’est finalement là que pour apporter un niveau d’abstraction supplémentaire pour l’utilisateur par rapport au langage machine et pour faciliter l’utilisation des ordinateurs.

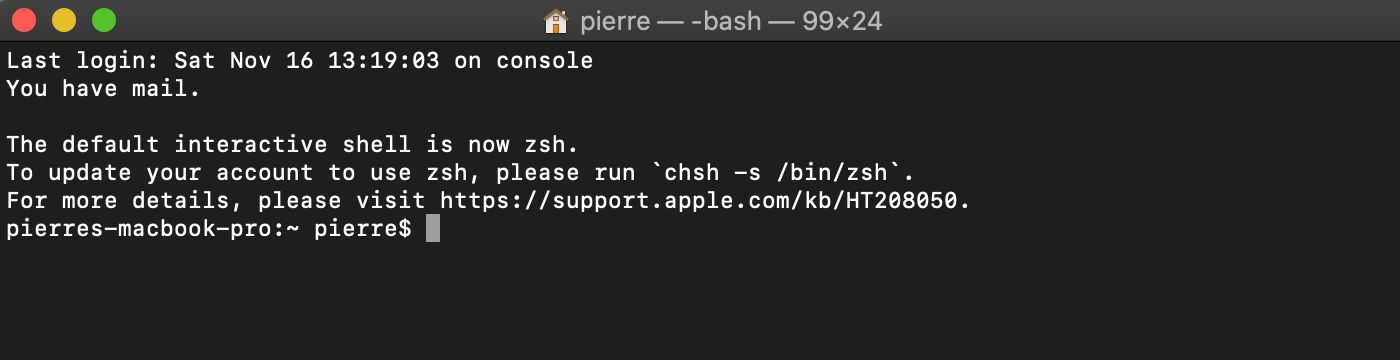
La question est donc : pourquoi utiliser la console plutôt que notre interface graphique utilisateur ? Tout simplement car certaines opérations sont beaucoup plus simples à réaliser via la console. Pour l’utilisation de Git, je vous renvoie aux raisons mentionnées ci-dessus.

On va donc pouvoir utiliser la console pour envoyer des ordres à notre système d’opération et pour effectuer différentes opérations courantes comme la création d’un dossier ou d’un fichier, sa suppression, le renommage etc. et également utiliser la console pour exécuter les commandes spécifiques de Git après l’avoir installé.

Les commandes de Git vont une nouvelle fois être les mêmes pour tous les systèmes (Windows, Mac OS, etc.). Les commandes liées aux opérations sur le système comme la création d’un dossier vont elles pouvoir être différentes d’un système à l’autre.

Pour ma part, j’utilise un Mac depuis des années et utilise donc des commandes UNIX tout comme Linux. Si vous utilisez Windows, je vous invite à installer un logiciel vous permettant d’avoir accès aux commandes UNIX ou à complètement changer de système d’opération si vous le pouvez. Pour ce cours, les différences de système ne devraient pas être gênantes car nous allons nous focaliser sur les commandes Git qui sont les mêmes pour tous, mais cela vous sera utile dans votre vie de développeur.

Commençons par ouvrir la console ensemble. Si vous êtes sous Mac, vous allez utiliser l’application Terminal disponible dans Applications -> Utilitaires -> Terminal. Si vous êtes sous Windows, vous allez utiliser l’Invite de commande ou PowerShell selon la version que vous possédez.



Arrivé ici, vous allez pouvoir taper toutes sortes de commandes pour demander à votre système d’effectuer des opérations. Notez que la plupart des commandes qu’on va taper vont être dépendantes de notre emplacement dans le système. Par exemple, pour ajouter un dossier sur notre bureau, il va déjà falloir naviguer jusqu’au bureau via notre console.

A titre d’information, les commandes (UNIX) de base à connaitre sont les suivantes :

* La commande pwd affiche le nom et chemin du répertoire courant. Elle permet de savoir où on se situe sur notre machine ;
* La commande ls affiche la liste des fichiers et répertories dans un répertoire ;
* La commande cd permet de changer de répertoire, c’est-à-dire de se déplacer d’un répertoire à un autre ;
* La commande mkdir permet de créer un nouveau répertoire (ou “dossier”) ;
* La commande touch permet de créer un fichier ;
* La commande mv permet de renommer ou de déplacer des fichiers.

Si vous désirez en savoir plus sur les commandes UNIX, je vous invite à lire mon cours sur le [shell Bash](https://www.pierre-giraud.com/shell-bash/).

Voilà tout pour ce petit aparté sur la ligne de commande qui devrait vous permettre de comprendre beaucoup plus rapidement et plus facilement la suite de ce cours.

Installation de Git

Entrons dans le vif du sujet et passons immédiatement à l’installation du logiciel Git. La façon la plus simple d’installer Git est de télécharger la dernière version sur le site officiel [http://git-scm.com/downloads](https://www.pierre-giraud.com/git-github-apprendre-cours/installer-git/%E2%80%9Chttp:/git-scm.com/downloads%22), d’ouvrir le fichier téléchargé et de suivre les instructions à l’écran en laissant toutes les valeurs par défaut.

Si vous êtes sous Windows, téléchargez plutôt la version de Git présente sur <https://gitforwindows.org/>. Cette version inclut un outil permettant d’émuler le comportement de Bash (le langage utilisé par Mac et Linux) et donc d’avoir accès aux mêmes commandes que moi.

Note : il est possible que la version de Git installée ne soit pas exactement la même que moi. Cela n’a aucune incidence et vous pouvez continuer à suivre le cours.

Paramétrage de Git

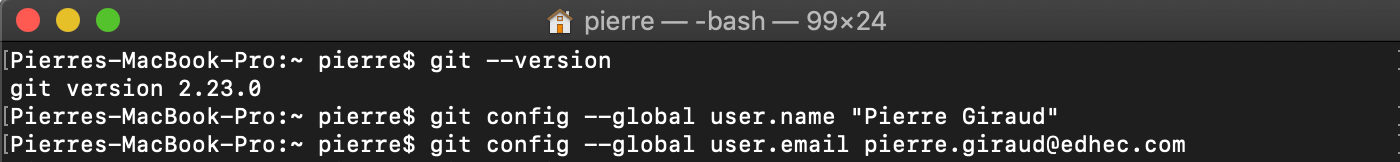
Une fois Git installé, nous allons paramétrer le logiciel afin d’enregistrer certaines données pour ne pas avoir à les fournir à nouveau plus tard.

Nous allons notamment ici renseigner un nom d’utilisateur et une adresse mail que Git devra utiliser ensuite.

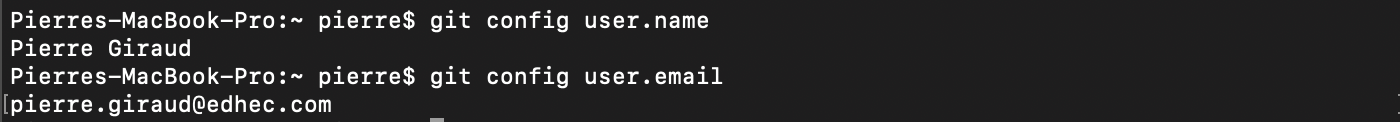
Pour faire cela, nous allons utiliser notre première commande Git qui est la commande git config. Cette commande permet de de voir et modifier les variables de configuration qui contrôlent tous les aspects de l’apparence et du comportement de Git.

Nous allons également passer une option --global à notre commande. Les options permettent de personnaliser le comportement par défaut de certaines commandes. Ici, l’option --global va nous permettre d’indiquer à Git que le nom d’utilisateur et l’adresse mail renseignés doivent être utilisés globalement (c’est-à-dire pour tout projet Git).

On va donc taper les commandes suivantes : git config --global user.name "Pierre Giraud" et git config --global user.email pierre.giraud@edhec.com à la suite pour renseigner un nom et une adresse email. Bien évidemment, utilisez votre propre nom et votre propre adresse email.



Pour vous assurer que vos informations ont bien été enregistrées, vous pouvez taper git config user.name et git config user.email. Les informations entrées devraient être renvoyées.



Voilà tout pour la configuration. Notez que certaines commandes Git ouvrent un éditeur, notamment celle qui vous demandent de saisir du texte. L’éditeur par défaut utilisé par Git est l’éditeur système qui est généralement Vi ou Vim.

J’utiliserai l’éditeur par défaut pour ce cours. Si vous souhaitez configurer un éditeur différent, vous pouvez entrer la commande git config --global core.editor nom\_de\_votre\_editeur.

Dans cette leçon, nous allons présenter le fonctionnement général de Git et définir des éléments de vocabulaire qui vont nous être utiles par la suite.

Démarrer un dépôt Git

Un “dépôt” correspond à la copie et à l’importation de l’ensemble des fichiers d’un projet dans Git. Il existe deux façons de créer un dépôt Git :

* On peut importer un répertoire déjà existant dans Git ;
* On peut cloner un dépôt Git déjà existant.

Nous allons voir comment faire cela dans la suite de ce cours. Avant cela, je pense qu’il est bon de comprendre comment Git conçoit la gestion des informations ainsi que le fonctionnement général de Git.

La gestion des informations selon Git

La façon dont Git considère les données est assez différente de la plupart des autres systèmes de gestion de version.

Git pense les données à la manière d’un flux d’instantanés ou “snapshots”. Grosso modo, à chaque fois qu’on va valider ou enregistrer l’état d’un projet dans Git, il va prendre un instantané du contenu de l’espace de travail à ce moment et va enregistrer une référence à cet instantané pour qu’on puisse y accéder par la suite.

Chaque instantané est stocké dans une base de donnée locale, c’est-à-dire une base de donnée située sur notre propre machine.

Le fait que l’on dispose de l’historique complet d’un projet localement fait que la grande majorité des opérations de Git peuvent être réalisées localement, c’est-à-dire sans avoir à être connecté à un serveur central distant. Cela rend les opérations beaucoup plus rapides et le travail de manière générale beaucoup plus agréable.

Les états des fichiers

Comment Git fait il pour suivre les modifications sur les fichiers d’un projet ? Pour comprendre cela, il faut savoir qu’un fichier peut avoir deux grands états dans Git : il peut être sous suivi de version ou non suivi.

Un fichier possède l’état “suivi” si il appartenait au dernier instantané capturé par Git, c’est-à-dire si il est enregistré en base. Tout fichier qui n’appartenait pas au dernier instantané et qui n’a pas été indexé est “non suivi”.

Lorsqu’on démarre un dépôt Git en important un répertoire déjà existant depuis notre machine, les fichiers sont au départ tous non suivis. On va donc déjà devoir demander à Git de les indexer et de les valider (les enregistrer en base). Lorsqu’on clone un dépôt Git déjà existant, c’est différent puisqu’on copie tout l’historique du projet et donc les fichiers sont tous déjà suivis par défaut.

Ensuite, chaque fichier suivi peut avoir l’un de ces trois états :

* Modifié (“modified”) ;
* Indexé (“staged”) ;
* Validé (“committed”).

Lors du démarrage d’un dépôt Git à partir d’un dépôt local, on demande à Git de valider l’ensemble des fichiers du projet. Un fichier est “validé” lorsqu’il est stocké dans la base de donnée locale. Lors du clonage d’un dépôt déjà existant, les fichiers sont enregistrés par défaut en base et ils sont donc validés par défaut.

Ensuite, lorsqu’on va travailler sur notre projet, on va certainement ajouter de nouveaux fichiers ou modifier des fichiers existants. Les fichiers modifiés vont être considérés comme “modifiés” par Git tandis que les nouveaux fichiers vont être “non suivis”. Un fichier modifié est considéré comme “modifié” par Git tant qu’il n’a pas été indexé.

On dit qu’on “indexe” un fichier lorsqu’on indique à Git que le fichier modifié ou que le nouveau fichier doit faire partie du prochain instantané dans sa version actuelle.

Enfin, lorsqu’on demande à Git de prendre l’instantané, c’est-à-dire lorsqu’on lui demande d’enregistrer en base l’état du projet actuel (c’est-à-dire l’ensemble des fichiers indexés et non modifiés), les fichiers faisant partie de l’instantané sont à nouveau considérés comme “validés” et le cycle peut recommencer.

Les zones de travail

Les états de fichiers sont liés à des zones de travail dans Git. En fonction de son état, un fichier va pouvoir apparaitre dans telle ou telle zone de travail. Tout projet Git est composé de trois sections : le répertoire de travail (working tree), la zone d’index (staging area) et le répertoire Git (repository).

Le répertoire de travail ou “working tree” correspond à une extraction unique (“checkout”) d’une version du projet. Les fichiers sont extraits de la base de données compressée située dans le répertoire Git et sont placés sur le disque afin qu’on puisse les utiliser ou les modifier.

La zone d’index ou “staging area” correspond à un simple fichier, généralement situé dans le répertoire Git, qui stocke les informations concernant ce qui fera partie du prochain instantané ou du prochain “commit”.

Le répertoire Git est l’endroit où Git stocke les méta-données et la base de données des objets de votre projet. C’est la partie principale ou le coeur de Git.

Le processus de travail va ainsi être le suivant : nous allons travailler sur nos fichiers dans le répertoire de travail. Lorsqu’on modifie ou crée un fichier, on peut ensuite choisir de l’indexer. Tant qu’un fichier n’est pas indexé, il possède l’état modifié ou est non suivi si c’est un nouveau fichier. Dès qu’il est indexé i.e que son nom est ajouté à la zone d’index, il possède l’état indexé. Finalement, on va valider (“commit”) la version indexée de nos fichiers pour les ajouter au répertoire Git.

Pour retenir ces informations, vous pouvez vous aider des schémas ci-dessous (source : git-scm.com).



