1. Qu’est qu’un système de contrôle de version en informatique ?

C’est un outil (logiciel) permettant d’enregistrer, de suivre et de gérer plusieurs versions d’un fichier ou d’un code source. Il permet d’établir un historique de toutes les modifications effectuées sur un élément, pour ainsi avoir la possibilité de récupérer une version antérieure selon la date et l’heure de la sauvegarde, et ce en cas d’erreur ou de problème sur une version actuelle.

2. Quelle est l’utilité d’un tel système ?

Voici les principaux avantages que vous devez attendre du contrôle de version:

- Un historique complet des changements à long terme de chaque fichier. Autrement dit, le moindre changement effectué par de nombreuses personnes au fil des ans. Les changements comprennent la création et la suppression de fichiers, ainsi que l'édition de leur contenu. Les différents outils VCS se distinguent par la manière dont ils gèrent le renommage et le déplacement des fichiers. Cet historique doit également comprendre l'auteur, la date et des notes écrites sur l'objet de chaque changement. Le fait de disposer de l'historique complet permet de revenir aux versions précédentes pour aider à l'analyse des causes profondes des bugs, ce qui est crucial lorsqu'il est question de corriger des problèmes dans les anciennes versions d'un logiciel. Si des développeurs travaillent activement sur le logiciel, presque toute version du logiciel peut être considérée comme « ancienne ».
- Branching et merges. Le fait que les membres d'une équipe travaillent simultanément est une évidence, mais il peut être intéressant pour les personnes autonomes de pouvoir travailler sur des flux de changements indépendants. La création d'une « branche » dans des outils VCS permet de

garder plusieurs flux de travail indépendants les uns des autres, tout en offrant la possibilité de merger ces tâches, ce qui permet aux développeurs de vérifier que les changements sur les différentes branches n'entrent pas en conflit. De nombreuses équipes de développement adoptent une pratique de branching pour chaque fonctionnalité et/ou pour chaque version. Il existe de nombreux workflows différents parmi lesquels les équipes peuvent choisir lorsqu'elles décident d'utiliser les installations de branching et de merges dans des outils VCS.

Le **premier objectif** d'un gestionnaire de version est de garder un historique des différentes mises à jour d'une application ou d'un logiciel. Le **second objectif** permet un meilleur travail collaboratif : gestion de plusieurs versions du code source.

3. Citez des outils de contrôle de version

Gestion de version centralisée

Il n'y a qu'un seul dépôt disponible, on parle de dépôt centralisé. Ce système de gestion centralisé présente des avantages en termes de fiabilité, mais aussi des inconvénients en termes d'échanges entre les dépôts et entre les copies locales impossibles et avec un temps de mise à jour long.

Exemples : CVS (Concurrent Version Système), Apache Subversion

Gestion de version décentralisée

Ici, les développeurs possèdent leurs propres dépôts et leur propre copie locale. L'intérêt d'un gestionnaire de version décentralisée est de permettre la communication entre les dépôts locaux : possibilité de cloner ces dépôts et de « lire/écrire » depuis un dépôt vers un autre. À la différence du système de gestion de version centralisée, il est possible de travailler hors connexion. Aussi, chacun peut travailler à son rythme sur son dépôt local et permet d'avoir plus de sécurité, car il n'y a pas qu'un seul dépôt de référence, mais plusieurs.

Exemples : : GNU Arch, Git, et Mercurial

4. Présentation de Git :

Définition :

GIT est un système de contrôle de versions gratuits et Open-Source, créé en 2005 par Linus **Torvalds**, le créateur de Linux.

Caractéristiques

Historiques des pistes

Gratuit et Open Source

Prend en charge le développement non linéaire

Créer des sauvegardes

Evolutif

Prend en charge la collaboration

La ramification est plus facile

Un dépôt Git: local et distant

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.

Un dépôt local est un dépôt Git qui est **stocké sur votre ordinateur.**

Un dépôt distant est un dépôt Git qui est **stocké sur un ordinateur distant.**

Le dépôt distant est généralement utilisé par les équipes comme **un dépôt central** dans lequel chacun pousse les modifications depuis son dépôt local et à partir duquel chacun récupère les modifications vers son dépôt local.

Notion de branche

Dans Git, les branches font partie de votre processus de développement quotidien. Les branches Git sont en fait un pointeur vers un instantané de vos modifications. Lorsque vous souhaitez ajouter une nouvelle fonctionnalité ou corriger un bogue, quelle que soit sa taille, vous générez une nouvelle branche pour encapsuler vos modifications.

> **Une branche dans Git est simplement un pointeur léger et déplaçable vers un de ces commits. La branche par défaut dans Git s'appelle master . Au fur et à mesure des validations, la branche master pointe vers le dernier des commits réalisés. À chaque validation, le pointeur de la branche master avance automatiquement.**

Commandes essentiels

La commande cd sert à se déplacer dans un répertoire

mkdir pour créer un répertoire vide

touch pour créer un fichier vide

Pour initialiser un dépôt Git, on utilise la commande git init . Cela crée un sous répertoire .git qui contient un ensemble de fichiers qui vont permettre à un dépôt Git de fonctionner.

Lorsqu'on utilise git init, Git nous renvoie un message en nous informant que le dépôt Git a bien été initialisé et qu'il est vide.

On peut utiliser ici la commande git status pour déterminer l'état des fichiers de notre répertoire.

Pour indexer des fichiers, on utilise la commande git add. On peut lui passer un nom de fichier pour indexer le fichier en question, le nom d'un répertoire pour indexer tous les fichiers du répertoire d'un coup

Lorsqu'on utilise git commit sans argument, une nouvelle fenêtre s'ouvre en utilisant l'éditeur par défaut .

La commande git branch permet en fait bien plus que la simple création et suppression de branches. Si vous la lancez sans argument, vous obtenez la liste des branches courantes

Gestion de version : définition & objectifs

C’est un outil (logiciel) permettant d’enregistrer, de suivre et de gérer plusieurs versions d’un fichier ou d’un code source. Il permet d’établir un historique de toutes les modifications effectuées sur un élément, pour ainsi avoir la possibilité de récupérer une version antérieure selon la date et l’heure de la sauvegarde, et ce en cas d’erreur ou de problème sur une version actuelle.

Le **premier objectif** d’un gestionnaire de version est de garder un historique des différentes mises à jour d’une application ou d’un logiciel.
Le **second objectif** permet un meilleur travail collaboratif : gestion de plusieurs versions du code source.

Gestion de version centralisée

Il n'y a qu'un seul dépôt disponible, on parle de dépôt centralisé. Ce système de gestion centralisé présente des avantages en termes de fiabilité, mais aussi des inconvénients en termes d'échanges entre les dépôts et entre les copies locales impossibles et avec un temps de mise à jour long.
Exemple de logiciels de gestion de version centralisée : CVS (Concurrent Version Système) et Apache Subversion.

Gestion de version décentralisée

Ici, les développeurs possèdent leurs propres dépôts et leur propre copie locale. L'intérêt d'un gestionnaire de version décentralisée est de permettre la communication entre les dépôts locaux : possibilité de cloner ces dépôts et de « lire/écrire » depuis un dépôt vers un autre. À la différence du système de gestion de version centralisée, il est possible de travailler hors connexion. Aussi, chacun peut travailler à son rythme sur son dépôt local et permet d'avoir plus de sécurité, car il n'y a pas qu'un seul dépôt de référence, mais plusieurs.
Exemple de logiciels de gestion de version centralisée : GNU Arch, Git et Mercurial.

Qu'est-ce que Git ?

Git est un logiciel de versioning créé en 2005 par Linus Torvalds, le créateur de Linux.

Git permet de coordonner le travail entre plusieurs personnes en conservant un historique des changements effectués sur des fichiers.

Git permet à différentes versions d'un même fichier de coexister. Les développeurs travaillant avec Git ont accès à l'historique des modifications pour l'intégralité du projet et peuvent ainsi savoir quels changements ont été fait par rapport à leur version des fichiers, qui a fait ces changements, etc.

Qu'est-ce que GitHub ?

Dans le langage des systèmes de gestion de version, la copie de l'intégralité des fichiers d'un projet et de leur version située sur le serveur central est appelé un dépôt. Git appelle également cela "repository" ou "repo" en abrégé.

GitHub est un service en ligne qui permet d'héberger des dépôts ou repo Git. C'est le plus grand hébergeur de dépôts Git du monde.

Une grande partie des dépôts hébergés sur GitHub sont publics, ce qui signifie que n'importe qui peut télécharger le code de ces dépôts et contribuer à leur développement en proposant de nouvelles fonctionnalités.

!!! Git est un logiciel de gestion de version tandis que GitHub est un service en ligne d'hébergement de dépôts Git qui fait office de serveur central pour ces dépôts.

Installation de Git

La façon la plus simple d'installer Git est de télécharger la dernière version sur le site officiel http://git-scm.com/downloads, 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.

Pour la configuration de 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.

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.

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.

Créer un dépôt Git à partir d'un répertoire existant

La commande `cd` sert à se déplacer dans un répertoire

`mkdir` pour créer un répertoire vide

`touch` pour créer un fichier vide

Pour initialiser un dépôt Git, on utilise ensuite la commande `git init .` Cela crée un sous répertoire `.git` qui contient un ensemble de fichiers qui vont permettre à un dépôt Git de fonctionner.

Lorsqu'on utilise `git init`, Git nous renvoie un message en nous informant que le dépôt Git a bien été initialisé et qu'il est vide.

On peut utiliser ici la commande `git status` pour déterminer l'état des fichiers de notre répertoire.

Pour indexer des fichiers, on utilise la commande `git add`. On peut lui passer un nom de fichier pour indexer le fichier en question, le nom d'un répertoire pour indexer tous les fichiers du répertoire d'un coup ou encore un "fileglob" pour ajouter tous les fichiers correspondant au schéma fourni.

Lorsqu'on utilise `git commit` sans argument, une nouvelle fenêtre s'ouvre en utilisant l'éditeur par défaut .

Notion de branche

Une branche dans Git est simplement un pointeur léger et déplaçable vers un de ces commits. La branche par défaut dans Git s'appelle master . Au fur et à mesure des validations, la branche master pointe vers le dernier des commits réalisés. À chaque validation, le pointeur de la branche master avance automatiquement.

Travailler avec des dépôts distants

Pour pouvoir collaborer sur un projet Git, il est nécessaire de savoir comment gérer les dépôts distants. Les dépôts distants sont des versions de votre projet qui sont hébergées sur Internet ou le réseau d'entreprise. Vous pouvez en avoir plusieurs, pour lesquels vous pouvez avoir des

droits soit en lecture seule, soit en lecture/écriture. Collaborer avec d'autres personnes consiste à gérer ces dépôts distants.

Afficher les dépôts distants

Pour visualiser les serveurs distants que vous avez enregistrés, vous pouvez lancer la commande `git remote`. Elle liste les noms des différentes références distantes que vous avez spécifiées. Si vous avez cloné un dépôt, vous devriez au moins voir l'origine `origin` — c'est-à-dire le nom par défaut que Git donne au serveur à partir duquel vous avez cloné
Vous pouvez aussi spécifier `-v`, qui vous montre l'URL que Git a stockée pour chaque nom

Cloner un dépôt Git

La deuxième façon de démarrer un dépôt Git est de cloner localement un dépôt Git déjà existant. Pour cela, on va utiliser la commande `Git clone`.

Renommer un fichier dans Git

On peut également renommer un fichier de notre projet depuis Git en utilisant cette fois-ci une commande `git mv ancien-nom-fichier nouveau-nom-fichier`.

Consulter l'historique des modifications Git

La manière la plus simple de consulter l'historique des modifications Git est d'utiliser la commande `git log`. Cette commande affiche la liste des commits réalisés du plus récent au plus ancien. Par défaut, chaque commit est affiché avec sa somme de contrôle SHA-1, le nom et l'e-mail de l'auteur, la date et le message du commit.

Ecraser et remplacer un commit

Parfois, on voudra annuler une validation (un commit), notamment lorsque la validation a été faite en oubliant des fichiers ou sur les mauvaises versions de fichiers.

La façon la plus simple d'écraser un commit est d'utiliser la commande `git commit` avec l'option `--amend`. Cela va pousser un nouveau commit qui va remplacer le précédent en l'écrasant.