Git

Git est un système de versionning. Il permet de garder toutes les versions des fichiers dans le but d'y revenir si besoin pour corriger des erreurs ou revoir une ancienne version de ces derniers.

Git est un système décentralisé, il n'a donc pas besoin de serveur distant pour fonctionner. Vous pouvez parfaitement l'exécuter sur votre machine localement.

Git gère ce que l'on appelle des objets : blob, tag, commit, tree.

Git a été développé par Linus Torvalds (auteur du noyau Linux). Git est un logiciel libre (GNU version 2). Git est le logiciel le plus populaire pour la gestion de version. La première version de Git remonte à 2005. Aujourd'hui on en est à la version 2.23.

Git permet de versionner ces fichiers mais également de collaborer avec d'autre développeurs ou d'autres personnes travaillant sur un projet nécessitant des versions de fichiers.

Dans ce cours nous utiliseront Git en ligne de commande. C'est important pour bien maitriser les concepts clés de ce logiciel.

Intérêt

Git est utilisé par beaucoup d'entreprises en informatique et surtout en développement de logiciel. Il permet le travail en équipe ou local pour tester ses propres idées sur un projet donné. On verra par la suite sur le chapitre sur les branches que Git est utilisé largement en développement pour son implémentation de la gestion des branches. Les branches vous permettrons de tester du code sans impacter votre projet et de développer en sécurisant le code de l'application.

Principe du fonctionnement de l'historique

Git index les fichiers que vous suivez avec une somme de contrôle calculé avec la fonction de hachage SHA-1. Le principe est simple si le fichier n'est pas modifier cette somme ne change pas. Git ne stocke qu'une seule fois dans son historique le fichier, par contre si le fichier change la somme de contrôle est différente alors Git garde dans son historique 2 fichiers le fichier avant modification et le fichier après modification.

```
commit 1                    commit 2

mon_fichier1.txt
                     ---->
mon_fichier2.txt [modifié]  mon_fichier2.txt
```

Nous le verrons mais on appelle commit une photographie de l'état de votre dépôt.

Décentralisé

Le fait que Git soit décentralisé permet de travailler sans un serveur distant. Si on suppose que vous avez installé Git sur votre machine alors vous pouvez créez un dépôt Git local et commencer à travailler avec Git. Mais vous pouvez aussi cloner ou faire un "fork" d'un dépôt distant et travailler en local sur le projet.

Dans tous les cas vous travaillez localement, vous faite vos commits (photographie des fichiers). Et si besoin vous pouvez proposer vos fichiers modifiés sur le dépôt distant.

Lorsqu'on clone ou fork un projet on possède alors sur sa machine tout l'historique du logiciel/projet.

Un des grands succès de Git repose sur le fait qu'il soit décentralisé.

Commit

Un commit c'est une photo à un instant t de l'état dans lequel se trouve votre application par exemple. Un commit sera toujours accompagné d'un texte qui explique ce que vous venez de faire.

Lorsque vous **commité** c'est que vous venez de finir une **feature**, par exemple un système d'authentification pour votre Web App. Vous ferez donc un commit que vous nommerez "Mise en place de l'authentification email/password dans la Web App".

Installation

Vous trouverez selon votre système à la page suivante un installeur de Git :

Download

- Pour l'installation sous Window choisissez "Use Git from git bash only"

Pour une installation sous Linux vous devez tapez les lignes de code suivantes :

```
sudo add-apt-repository ppa:git-core/ppa

sudo apt update

sudo apt install git
```

Configuration de Git

Vous devez pour pouvoir utiliser Git sur votre machine définir un utilisateur Git. Ce point est important car sans cela Git ne pourra pas fonctionner. En effet, Git est un système de versionning collaboratif et il doit donc identifier qui a fait les modifications des fichiers dans ses commits.

Pour l'utilisateur d'une machine :

```
# s'enregistre dans .gitconfig de votre compte user
$ git config --global user.name Tony
$ git config --global user.email tony@tony.fr

# Définir un éditeur pour écrire le texte du commit
# Normalement lors de l'installation automatique de Git
# sous Window ou Mac un éditeur est déjà défini
$ git config --global core.editor vim
```

Optionnel pour l'instant. Pour un dépôt particulier vous pouvez également définir un utilisateur spécifique :

```
# pour un dépôt en particulier, dans le fichier config du dossier .git/
$ git config --local user.name Alan
$ git config --local user.email alan@alan.fr
```

Zone de travail de Git

Ci-dessous on utilise git add et git commit qui sont deux commandes Git.

- Répertoir de travail *mon_fichier*
- Index : git add mon_fichier
- Dépôt : git commit -m "une feature"

Remarques : tant que vous n'avez pas ajouté votre fichier à l'index Git ne connait pas votre fichier (...), il n'est pas suivi en version. Une fois dans l'index il sera suivi en version.

git init

Pour initialiser un projet Git il faut en premier lieu tapez la ligne de commande suivante :

```
git init
```

Cette commande est importante car elle va créé le dossier .git/ qui est l'historique Git de votre projet. Vous devez impérativement lancer cette commande avant de commencer avec Git.

Pour voir si tout fonctionne bien vous vous placerez dans le dossier versionné et taperez la commande suivante :

```
git status
```

Exercice menu Pizza

Créez un dossier Pizza dans lequel vous allez créer un seul fichier pour l'instant calzone.txt, voici le contenu de ce fichier :

```
Pizza : Calzone
Coulis de tomate
mozzarella
jambon supérieur
champignons de Paris frais
oeuf
```

Ouvrez ce dossier avec un terminal et tapez la ligne de commande suivante, cette commande permet d'initialiser un dépôt Git :

```
git init
```

Si vous explorez le dossier Pizza maintenant vous pouvez voir un dossier particulier .git/. Ce dernier dossier contient l'ensemble de l'historique des versions de vos fichiers.

Tapez maintenant la commande suivante, vous verrez qu'il n'y a aucun commit et aucun fichier suivi :

```
git status
```

Résultat de la commande ci-dessus :

```
Sur la branche master

Aucun commit

Fichiers non suivis:
  (utilisez "git add <fichier>..." pour inclure dans ce qui sera validé)
        calzone.txt

aucune modification ajoutée à la validation mais des fichiers non suivis sont présents (util
```

Nous allons maintenant ajouter le fichier dans la zone d'index :

```
git add calzone.txt
```

Refaites un **git status** pour voir que maintenant git connait ce fichier.

Notez qu'une commande vous est proposée pour désindexer le fichier :

```
git rm --cached calzone.txt
```

Un git status est une précieuse commande elle vous donnera l'état du dépôt à un instant t et vous donnera également des information complémentaire.

Il existe une commande "plomberie" de Git qui permet de voir ce qu'il y a dans l'historique, on ne l'utilise jamais dans la pratique, mais celle-ci permet de voir comment Git crée ces objets :

```
# Commande linux
find .git/objects -type f
```

Vous devriez voir la clé de hachage créé par Git pour suivre ce fichier.

Nous allons maintenant créer notre premier commit, deux options s'offrent à vous :

```
# version courte
git commit -m "création du fichier calzone (ingrédients)"

# équivalent, dans ce cas vous avez un éditeur qui s'ouvre
# ce dernier vous permet d'expliquer
# plus pécisémment ce que vous venez de faire
git commit
```

Créez maintenant le fichier readme.md, ce fichier est un fichier classique dans un dépôt Git c'est le **README** fichier de documentation du projet.

Vous mettrez dans ce fichier le contenu suivant :

```
# Menu de nos Pizza

Bienvenu à la pizzeria du Chateau. Nos pizza sont réalisés avec des produits de saisons et b
```

Terminez : en faisant un index + commit

Puis visualiser les logs, de manière concrète vous verrez les clés de hachages ainsi que des informations relatives à chaque commit :

```
git log
```

Pour sortir de l'affichage tapez sur la lettre "q" de votre clavier.

Règles messages de commit

- un titre de 49 caractères max
- un texte plus long, répondez à la question pourquoi faire dans votre texte.
- on peut commiter avec un titre uniquement option -m

```
$ git commit -m "étudiant/tp: si terminé, attribuer note"

# Ouvrir l'éditeur et mettre un titre et texte plus long
$ git commit
```

Exercice ajoutez une recette

Vous allez maintenant ajouter une nouvelle recette dans votre dépôt Pizza.

Créez le fichier bellachao.txt avec le contenu suivant :

```
Pizza : Bellacho
Sauce tomate
mozzarella
chorizo
merguez & poivrons
```

Ajoutez le fichier **bellachao.txt** avec git add puis avec git commit sans l'option -m, vous allez rédiger un message de commit qui correspond aux bonnes pratiques.

Soit vous avez nano, soit vous avez vim :

- **nano** : écrivez votre commit puis faites un ctrl + x et confirmez que vous voulez enregistrer le message du commit.
- **vim** : Si vous n'êtes pas en mode insertion tapez simplement sur la touche i de votre clavier.

Pour enregistrer votre message de commit dans vim tapez sur la touche "Echap" et faites la combinaison de touches suivantes **:wq + Entrée**.

Vérifiez que vous avez bien fait votre commit avec l'option suivante :

```
$ git log --oneline
# ou une autre commande équivalente
$ git shortlog
```

Exercice ajoutez un ensemble de fichiers

Il est courant dans Git d'ajouter un ensemble de fichiers/dossiers, libraires, framework, . . .

Récupérez maintenat la liste des pizza de speciales (dossier) à l'adresse suivante et placez celles-ci dans le dossier Pizza :

specials

Pour indexer les nouveaux fichiers ou indexer les modifications vous pouvez tapez la commande suivante :

```
# commande pour voir l'état de votre dépôt
git status

# Ajoutez un ensemble de fichiers/dossiers ou modifs
git add .

# ou de manière équivalent
git add -A
```

Commandes d'aide dans Git

Voici quelques commandes pratiques pour gérer un dépôt Git

```
$ git status
$ git help [nom de la commande]
$ git log
$ git log --oneline # voir alias dans la configuration
$ git log master..origin/master # voir les différences entre deux branches
$ git log -5 # 5 derniers commits
$ git log -p -5 # log avec différence pour chaque commit sur les 5 derniers
$ git log --stat # stat sur les modifs par commits
$ git log --since=2.weeks # depuis deux semaines, --until existe également

# Blame qui a fait la modif !
$ git blame -L 40,60 readme.md # rechercher qui a fait les modifs par ligne -L
$ git blame --since=3.weeks -- readme.md # depuis 3 semaines avec auteurs des modifications
$ git blame -L "/^### /" readme.md # recherche avec expression régulière
```

Renommer un fichier

Si vous devez renommer un fichier dans Git vous devez utiliser la commande suivante pour que Git intègre cette modification à son historique :

```
git mv mon_ancien_fichier mon_nouveau_fichier

git status
```

Application modififer et ajouter

Ajoutez dans le dossier specials la pizza vegan_double.txt avec le contenu suivant :

```
Pizza : vegan double
Sauce tomate 2 fois plus
râpé vegan 2 fois plus
```

Puis renommez le fichier readme.md en README.md à l'aide de la commande vue précédemment. Faites alors un git status pour voir l'état de votre dépôt.

Une fois que vous avez tapé git mv, git a ajouté dans l'index le renommage :

```
Sur la branche master
Modifications qui seront validées :
  (utilisez "git restore --staged <fichier>..." pour désindexer)
    renommé :         readme.md -> README.md

Fichiers non suivis:
  (utilisez "git add <fichier>..." pour inclure dans ce qui sera validé)
    specials/vegan_double.txt
```

```
git commit -m "renommage de readme en README"

git status
```

Le git status devrait vous renvoyer ce qui suit :

```
Sur la branche master
Fichiers non suivis:
  (utilisez "git add <fichier>..." pour inclure dans ce qui sera validé)
    specials/vegan_double.txt

aucune modification ajoutée à la validation mais des fichiers non suivis sont présents (util
```

Mettez le dépôt dans l'état "copie de travail propre".

Modifier un fichier connu par Git

Modifiez le fichier vegan_double en ajoutant la ligne suivante :

```
options possibles : courgette, harico
```

Vous pouvez alors tapez la ligne de commande suivante :

```
git commit -am "un message court"
```

Supprimer un fichier

Supposez maintenant que nous voulions supprimer un fichier suivi par Git ? Dans ce cas il faudra utiliser la commande :

```
git rm mon_fichier
```

Exercice suppresion

Supprimer le fichier **vegan__double.txt**. Faites le depuis la racine de votre dépôt.

Faites bien un git status pour vérifier l'état de votre dépôt, normalement Git vous propose maintenant de faire un commit pour valider l'opération.

Les commandes que vous venez de faire sont en faite équivalente aux commandes suivantes :

```
rm nom_fichier
git add nom_fichier
```

Ne plus suivre un fichier

Ajoutez le dossier images suivants :

images

Vous vous apercevez que dans le dossier images vous souhaitez ne plus suivre le fichier incredients.jpg.

Tapez la commande suivante pour d'indexer l'image en question :

```
git rm --cached images/incredients.jpg
```

Configuration du .gitignore

Le fichier **incredients.jpg** va continuer d'apparaitre dans les messages de Git.

Créez le fichier **.gitignore** à la racine du dépôt et écrivez le code suivant dans ce fichier :

```
/images/incredients.jpg
```

Si vous faites maintentant un git status vous ne serez plus embêté par le fichier non suivi dans les messages de Git. Git ignorera maintenant totalement ce fichier.

Vous devez ajouter votre gitignore par contre à votre dépôt.

Une commande utile pour voir tous les fichiers suivis :

```
git ls-files
```

Remarques sur les patterns à utiliser pour exclure des fichiers :

```
vendor  --> ignore tous les dossiers vendor
```

```
/vendor  --> ignore le dossier vendor à la racine du dépôt

doc/foo/  --> matchera avec doc/foo mais pas a/doc/foo

foo/      --> matchera avec a/foo et foo

foo/*     --> matchera avec n'importe quel fichier autre qu'un "/"

*.txt     --> ignore tous les fichiers qui ont cette extension dans le dépôt : racine, dossi

foo/*.txt  --> ignore tous les fichiers de ce type dans les dossiers foo de l'application.

/*.txt   --> ignore juste au niveau racine du dépôt.

**/foo   --> ignore foo/a, bar/foo/a mais n'ignore pas foo/a/b

foo[0-9]  --> ignore foo0, foo1, ... foo9

foo   --> ignore a/foo, foo/a/b, a/b/foo, ...

foo/**/bar --> ignore foo/a/b/c/bar, foo/bar, ...
```