**Accompagnement du projet numérique**

**TD 2 : 28/09/2021**

Définition : Git est un logiciel de **gestion de versions décentralisé**. Git permet donc de **suivre l’évolution d’un fichier**, de sa création et sa dernière sauvegarde tout en conservant une trace de l’ensemble des modifications apportées à ce fichier. Très utilisé par les développeuses et développeurs. Egalement très utile dans la gestion d’un projet numérique lorsque l’on travaille avec des données textuelles (encodage d’un corpus par exemple).

## Historique et commandes de base

On comprend facilement que l’on est ici face à plusieurs problèmes :

* multiplication du nombre de versions d’un seul et même fichier
* comprendre à quoi correspond chacune des versions du fichier
* pouvoir revenir en arrière, avoir une trace des changements effectués

... sans parler de la difficulté que cela peut représenter dans la gestion d’un projet de groupe.

Git est un **outil de versionnage**.

* Créé en avril 2005 par le **créateur du noyau Linux** Linus Torvalds et par Junio Hamano
* Sous licence libre GNU GPLv2
* Version actuelle : 2.33 (date d’août 2021)
* En 2016, on estime que Git est utilisé par plus de 12 millions de personnes

Il existe cependant d’autres outils de versionnages :

* 1 CVS
* 2 SVN
* 3 Bazaar

## Les principes fondamentaux de Git

Voici les principes fondamentaux de Git :

Travail dans un *repository* (dépôt en français) ce qui correspond en réalité à un dossier sur votre ordinateur.

Il contient un dossier (souvent caché) .git qui contient toutes les archives enregistrées.

Contrairement à Dropbox ou Google Drive, pour qu’une modification soit archivée, il faut que cela **soit explicité**. C’est vous qui décidez à quel moment vous souhaitez enregistrer les changements effectués :

1. Ces modifications archivées sont appelées commit
2. Elles portent un message enregistré par l’utilisateur
3. Elles peuvent comporter un ou plusieurs fichier(s)/dossier(s)
4. Les fichiers qui ont subi des modifications doivent y être ajoutés explicitement (vous pouvez choisir d’ajouter tous les fichiers modifiés, ou seulement une partie d’entre eux)

Lorsque l’on travaille avec Git, on distingue trois états différents pour chaque fichier :

* un état de **travail** : le fichier a subi des modifications, mais nous ne l’avons pas encore ajouté (add) à un futur commit
* un état de **futur enregistrement** : le fichier a été ajouté (add) à un commit, mais le commit n’a pas été finalisé avec un message (on parle alors de *staging area*)
* un état **archivé** : le fichier a subi des modifications enregistrées et n’a pas été modifié depuis lors.

Les principales commandes Git :

Initialisation d’un répertoire Git : **git init**

Ajout de modifications : **git add** **[Nom de votre fichier]** ou **git add -A** (tous les changements sont concernés)

Etat du dépôt´ : **git status**

Enregistrement de modifications : **git commit -m ”Message du commit”** (N’oubliez jamais le ’-m’ !)

Historique du repository : **git log**

Différence entre l’état archivé et l’état actuel : **git diff** pour une différence détaillée, **git status** pour un point général.

Précisions techniques :

Contrairement à ce que l’on pourrait penser, Git n’archive ou **ne sauvegarde pas** vos fichiers. Ce qu’il sauvegarde, ce sont les modifications apportées à des fichiers.

Par exemple, si je crée un fichier test.txt qui contient ”test” en ligne 1, il enregistrera :

- Nouveau (toto.txt)

- Content (toto.txt, ”toto”)

Le fichier .gitignore

Un fichier bien utile

Git c’est bien, mais on ne veut pas toujours mettre à disposition toutes ses sources. Par exemple, le fichier qui comprend les mots de passe des bases de données, des fichiers caches ou compilés, etc.

Git fournit bien évidemment un outil pour ça : le fichier .gitignore.

**Où stocker ce fichier ?**

Ce fichier se nomme forcément .gitignore (Il commence donc par un point !).

Il se trouve à la racine du dépôt par défaut mais vous pouvez spécifier plusieurs gitignore qui auront toujours un effet sur le dossier courant et ses descendants.

**Quelle syntaxe utiliser ?**

**\*.txt** : cette ligne permettra d’ignorer tous les fichiers textes où qu’ils soient

**dossier/** : cette ligne ignorera l’ensemble du contenu de dossier et par extension, le dossier lui-même (Git ne conserve pas les dossiers vides)

**dossier/\*.jpg** : cette ligne ignorera les \*.jpg dans le dossier. Par contre, si le dossier a des enfants (dossier/sousdossier1) et des jpg à l’intérieur, ils seront versionnés (penser au principe de récursivité)

**mdp.txt** : cette ligne permet d’ignorer le fichier mdp.txt dans le dossier principal

## Les branches

**Une branche sur Git, c’est quoi ?**

Les branches sont comme des **”Sauvegarder sous...”** pour l’ensemble du dépôt (sauf qu’on peut les rejoindre/fusionner plus facilement !)

La branche par défaut s’appelle **main** (avant : master)

Dans git status, la branche main doit être affichée dans votre dossier de notes

Offre la possibilité de travailler sur différents problèmes en parallèle.

Possibilité de travailler sur des problèmes différents en même temps et de changer de tâche rapidement.

* Une branche main
* Une branche bug 1
* Une branche bug 2 parce que celui-ci est plus pressé

Les commandes liées aux branches

Création d’une branche : **git branch** [Nom de votre branche]

Aller d’une branche à l’autre : **git checkout** [Nom de la branche où vous voulez aller]

Créer une branche tout en s’y déplaçant : **git checkout -b** [Nom de la nouvelle branche]

Fusionner une branche avec une autre : **git merge** [branche dont on veut les modifications] (on fusionne toujours depuis la branche réceptrice, celle sur laquelle on veut continuer de travailler ensuite).

## GitHub

Ce qu’on a vu jusqu’à présent, c’est très bien mais ça ne permet pas le travail en équipe (et donc l’accompagnement de projets)

L’idée est de pouvoir travailler à plusieurs sur un même fichier/dossier et de garder une trace de l’ensemble des changements effectués (surtout si l’on cherche à avoir une version précédente d’un fichier corrompu).

Il existe un ensemble de solutions pour cela : des plateformes en ligne à l’instar de GitHub.

Les services en ligne de Git

Des entreprises et des structures du public fournissent des serveurs centralisés pour git :

**GitHub** qui est le plus à la mode et que nous utiliserons

**GitLab** qui est le concurrent le plus sérieux de GitHub aujourd’hui (et proposé par Huma-Num)

**Bitbucket**

Un peu de vocabulaire

**remote** : serveur distant permettant la synchronisation manuelle de votre dépôt

**origin** : nom du serveur principal, un peu comme main est la branche principale. Pour l’instant, nos serveurs n’en ont pas

**push** : envoyez les modifications effectuées sur un serveur donné

**pull** : retrouvez les informations depuis un serveur donné

**clone** : copie sur votre PC d’un dépôt trouvé en ligne

**fork** : dérivé d’un dépôt d’un-e autre développeur-se

**upstream** : par convention, nom d’un second serveur, généralement le serveur source du fork

**Pourquoi utiliser GitHub ?**

GitHub a des défauts (pas open source par exemple) mais reste l’outil le plus moderne pour de la gestion de code versionné en équipe

Il propose un environnement complet :

* Gestion d’équipe
* Connexions `a d’autres applications (possibilité de créer un petit site web : GitHub Pages)
* Gestion de ticket
* Gestion de fusion de branches
* Gestion de projets (To do list, etc.)

Création d’un compte GitHub

Création de compte :

Il est temps de vous créer un compte GitHub (si ce n’est pas déjà fait) ainsi qu’une organisation pour l’ensemble de la classe.

https://github.com/signup

Créez votre premier dépôt

Sur GitHub, veuillez créer un premier dépôt pour y mettre vos notes de cours.

Cliquez sur *New*

**Surtout ne pas cocher « initialiser le repository »**

Remplissez toutes les informations demandées

Suivez les instructions affichées `a l’écran

En local, sur votre ordinateur, assurez-vous d’avoir un dossier dans lequel vous avez initié un dépôt Git

**Comprenez-vous les effets de chacune de ces instructions ?**

Apportez des modifications à votre dépôt

A présent, vous pouvez modifier vos fichiers en local, indiquer à Git que vous avez fait des modifications, les ajouter à Git puis, enfin, envoyer ces modifications sur un serveur distant : votre dépôt GitHub.

* Modifiez votre fichier .txt
* **git status** : qu’est-ce qui s’affiche ?
* **git add [nom de votre fichier]**
* **git commit -m** « message de votre commit »
* **git pull**

Vérifiez que votre fichier modifié est bien sur GitHub.

Clôner un dépôt GitHub

Essayons de cloner un dépôt en ligne à présent

Un dépôt git en ligne est peu ou prou la même chose qu’un dépôt git local sauf que vous ne pouvez pas commit directement dessus. Il s’agit de l’archive « .git » de votre dépôt avec la capacité de se connecter et de synchroniser ces informations. Tous les dépôts git ne sont pas disponible en écriture. Il se peut qu’un repository git distant soit protégé pour n’être complété que par tel groupe de personne. Cependant, cela n’empêche pas de les cloner et de travailler dessus localement. Vous ne pourrez tout simplement pas modifier le Git distant (remote). Tout ce qui peut empêcher quelqu’un de cloner votre dépôt GitHub, c’est de le laisser en privé.

Quelques bonnes pratiques

A faire !`

* Toujours faire un **git status** en se connectant sur son dossier
* Suivi d’un **git pull** si vous avez travaillé dessus depuis un autre poste ou depuis votre PC
* Essayez le plus possible de ne pas travailler dans la master mais dans des branches focalisées sur un objectif dès que votre projet commence à grossir

Documenter son dépôt

Il existe une manière très simple de documenter votre projet et plus précisément votre dépôt GitHub, écrire un fichier README.md.

Il s’affiche automatiquement lorsque l’on consulte votre dépôt et doit idéalement comporter un certain nombre d’informations (en fonction de la nature de votre dépôt) :

* créateur/créatrice du dépôt
* que contient le d´epˆot ?
* explications dossier par dossier
* licence
* dépôt encore maintenu ?

Le format Markdown

**Markdown est un langage de balisage**, comme HTML ou LaTeX, mais c’est un langage de balisage léger.

HTML : <p>Ce<b>Mot</b> en gras et celui-ci <strong>aussi</strong>.</p>

LaTeX : Ce \textbf{Mot} en gras.

Markdown : Ce \*\*Mot\*\* est en gras.

Exercice : Rédigez votre propre README.md

Préparez, en local, un fichier en Markdown qui explique ce que contient votre dépôt GitHub. Essayez le gras, l’italique, les listes à puce, les url, etc.

Une fois votre fichier rédigé, ajoutez-le à votre dépôt GitHub.