Formation Git

Table des matières

1. Introduction

Nous allons illustrer l'utilité de Git dans la pratique de votre futur métier via deux exemples:

- Le premier: Imaginez qu'un client vous demande de faire un site de ventes en ligne : vous le faites et il fonctionne nickel.

Pendant la période de Noël, il vous demande de faire de grosses modifications et bardaf, il ne fonctionne plus du tout : erreur 500… Le client vous demandera de remettre immédiatement le site à l'état précédent les modifications problématiques pour ne pas perdre trop de ventes en cette période très lucrative pour lui.

Si vous n'aviez pas un backup de la précédente version ou un système de gestion de l'historique des modifications, vous risquez d'être bien embêté…

C'est ici qu'entre en scène l'outil git.

- Le second: vous devez travailler sur un projet à plusieurs mais vous ne savez pas comment faire pour collaborer de manière efficace avec vos collègues pour la gestion du code et voir comment gérer/fusionner le travail fait par chacun. C'est ici qu'entre en scène git via l'intermédiaire d'outils comme GitHub, Gitlab, etc.

- Le dernier: ce cours. Je vous l'ai envoyé par email et partagé dans un Google Drive. Cependant, si je modifie ce cours comme je le fais en ce moment, vous n'aurez pas la dernière version. Il faudra que je vous l'envoie à nouveau par email et que je remette ce document sur le Drive. Alors qu'avec l'utilisation de Git et GitHub se problème de dernière version est facilement géré.

2. Git c'est quoi ?

Git a été créé en 2005 par le papa de Linux (1991) : Linus Torvalds pour le développement du noyau Linux. Il permet de faire le suivi des différentes modifications d'un projet et de revenir dans un état précédent. Et surtout, il permet de travailler à plusieurs sur un même projet.

Auparavant, les développeurs utilisaient Bitkeeper de la société Bitmover qui concédait une licence gratuite d'utilisation au développement du noyau Linux. Mais un développeur du Noyau commença a créé un client opensource pour accéder à Bitkeeper. Ce qui provoqua après de longues négociations la suppression de la licence gratuite aux développeurs du noyau Linux. Réaction que l'on peut comprendre dans l'absolu… C'est pourquoi Linus Torvalds coda Git pour éviter tout futurs problèmes de licence d'utilisation. Ce qui devait sans doute arriver tôt ou tard car les puristes du logiciel libres dont le grand gourou du logiciel libre Richard Stallman prônait depuis longtemps qu'il n'était pas acceptable que le noyau Linux utilise un logiciel propriétaire pour la gestion de son code source.

Git est le logiciel de versions le plus populaire, utilisé par plus de 56.000.000 de développeurs dans le monde. Il est un incontournable des grandes sociétés de logiciel. Grandement utilisé dans le développement de logiciels OpenSource via la plateforme GitHub qui a été rachetée par Microsoft.

3. Installation de Git

3.1 Windows

3.1.1 Téléchargement

Il vous faudra télécharger l'une des versions suivantes de git, 32 bits ou 64 bits:

1. version 32 bits: https://github.com/git-for-windows/git/releases/download/v2.35.1.windows.2/Git-2.35.1.2-32-bit.exe
2. version 64 bits: https://github.com/git-for-windows/git/releases/download/v2.35.1.windows.2/Git-2.35.1.2-64-bit.exe

3.1.2 Installation

- Double cliquez sur l'exécutable.
- Cliquez sur "Next" pour accepter la licence de type GNU.
- Cliquez sur "Next" pour accepter le chemin d'installation par défaut. (S'il ne vous convient pas, cliquez sur Browse et changez-le).
- Cliquez sur "Next" pour laisser les composants par défaut à installer.
- Cliquez sur "Next" pour laisser Git comme nom dans le menu Windows.
- Sélectionnez "Use Visual Studio Code as Git's default editor"
- Cliquez ensuite sur "Next".
- Cliquez sur "Next" pour laisser Git décider du nom de branche par défaut (master).
- Laissez l'option sélectionnée:"Git from the command line and also from 3-rd party software"
- Cliquez sur "Next"
- Cliquez sur "Next" pour utiliser OpenSSH pour ssh fournit avec git.
- Cliquez sur "Next"
- Laissez l'option "Use the OpenSSL Library"
- Cliquez sur "Next"
- Laissez l'option "Checkout Windows-style, commit Unix-style line endings"
- Cliquez sur "Next"
- Sélectionnez "Use Windows's default console window"
- Cliquez sur "Next"
- Laissez le choix par défault: Default.
- Cliquez sur "Next"
- Laissez le choix par défault: Git Credential Manager

- Cliquez sur “Next”
- Laissez le choix par défault: Enable file system caching
- Cliquez sur “Next”
- Ne cochez aucune case
- Cliquez sur “Install”
- Cliquez sur “Finish”

3.2 Mac Os

Essayez de taper git –version en ligne de commandes. Si git n’est pas installé, installez git via brew.

Pour installer homebrew ou abrégé en brew, tapez la ligne de commande dans un terminal:

```
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

L’installation de brew peut prendre un certain temps. Soyez patient. :-)

Homebrew vous permet d’installer ce dont vous avez besoin et qu’Apple n’a pas installé.

Ouvrez un nouveau terminal et tapez la commande suivante pour installer git:

```
brew install git
```

3.3 Linux - via apt (Debian, Ubuntu, Raspberry Pi OS)

Ouvrir un terminal et taper:

```
sudo apt install git
```

4. Nouveau Windows Terminal

Microsoft a remis aux goûts du jour son Terminal. Il l’a rendu beaucoup plus sexy et permet l’exécution de différents terminaux dans des onglets.

On peut avoir comme terminaux: cmd, powershell, Azul cloud shell ou si linux si vous avez installé une distribution.

4.1 Installation

Microsoft a fait évoluer son terminal. Et il est possible de l’installer via le store de Microsoft: https://www.microsoft.com/fr-be/p/windows-terminal/9n0dx20hk701?rtc=1&activetab=pivot:overviewtab

4.2 Lancement du terminal

Pour exécuter le programme, on effectue la combinaison des touches Windows + R et on y écrit wt appuyez sur la touche ENTER.

4.3 Changer le terminal par défaut

C’est Powershell qui est installé par défaut. On pourrait l’utiliser mais utilisera dans le cadre de notre cours ça sera l’invite de commandes (cmd).

Nous allons le changer:

1. Exécutez la combinaisaon de touches: Windows + R
2. Ecrivez wt et appuyez sur la touche ENTER.
3. Dans wt, faites la combinaison de touches CTRL + ,
4. Dans démarrage, changez le profil par défaut et mettez “Invite de commandes”
5. Validez en appuyant sur le bouton Enregistrer

4.4 Nouvel onglet

Il suffit de faire la combinaison de touches: CTRL + SHIFT + T

4.5 Naviguer entre onglets

Si vous voulez aller: - au premier onglet: CTRL + ALT + 1 - au deuxième: CTRL + ALTR + 2 - au troisième: CTRL + ALTR + 3 - etc…

Si vous essayez d’accéder à l’onglet n°9 alors que vous n’en avez que 5. Il ira à l’onglet numéro 5.

5. Utiliser la ligne de commandes

Il y a deux façons d’utiliser git: par interface graphique ou via la ligne de commandes. Le mieux c’est l’utilisation du terminal pour utiliser au mieux git. Nous allons voir quelques commandes de base.

Pour chaque commande on peut avoir de l'aide via la commande elle-même

- Sous Windows commande /?
- Sous Mac/Linux commande –help ou via les man (manuels) unix s'ils sont installés. Exemple: man mkdir

L'invite de commandes signifie que le terminal est en attente de commandes. Une commande reçoit de 0 à N paramètres. Par exemple la commande: git version

git est la commande et version est le paramètre. Vous validez cette commande et son paramètre via la touche ENTER.

Ouvrez un terminal: Sur windows via la combinaisaon des touches Windows + R et tapez wt suivit d'ENTER.

5.1. Connaître le répertoire courant

Le répertoire courant est simplement le répertoire où votre invite de commandes se trouve actuellement. Par défaut, lorsque vous lancez un invites de commandes il se lance dans le répertoire de votre utilisateur. Sous windows c'est par exemple: C:

Voyons comment connaître le répertoire courant dans lequel vous vous trouvez:

1. Sur Windows: on regarde ce qu'on appelle le prompt. Il est composé du chemin courant suivit du symbole >. Exemple c:\Windows> le chemin c'est c:\Windows où le c: indique le lecteur et Windows le répertoire. Ou bien on tape la commande echo %cd% ou encore tout simplement cd sans aucun paramètre: ça retourne le répertoire courant. Le plus simple étant donc la commande cd sans aucun paramètre.

2. Sur Mac: on a deux manières. Le prompt est composé du nom de l'ordinateur suivit d'un symbole : ensuite du chemin suivit d'un espace. Il y a ensuite le nom de l'utilisateur terminé par le symbole $. Il est possible que vous voyez le symbole ~ (tilde) dans le chemin. Le symbole tilde signifie le chemin vers votre répertoire utilisateur. Ensuite on peut utiliser la commande pwd qui va donner le répertoire courant. Si vous avez un tilde dans le prompt, vous verrez le chemin complet de l'utilisateur grâce à la commande pwd. Exemple si votre prompt est mcfly:~/Documents john$ la commande pwd donnera le résultat suivant: /home/john/Documents

3. Sur Linux: Même chose que Mac sauf que le prompt est ainsi composé du nom de l'utilisateur suivit du symbole @ ensuite du nom de l'ordinateur d'un symbole : et enfin le chemin terminé par le symbole $. Exemple: john@mcfly:~/Documents$ la commande pwd donnera le résultat suivant: /home/john/Documents

4. Lister les fichiers

   A. Sur Windows: On utilise la commande dir avec comme paramètre /w. Exemple: dir /w
   B. Sur Mac/Linux: On utilise la commande dir ou ls ou encore
5. Créer un répertoire

   A. Sur Windows: c'est la commande mkdir (make directory) qui peut être raccourcie par md. Exemple: mkdir cours_git ou md cours_git
   B. Sur Mac/Linux: c'est la commande mkdir. Exemple mkdir cours_git
6. Changer de répertoire

   Sur Windows et Mac/Linux, c'est la commande chdir ou cd (change directory). Exemple: cd cours_git

   Ou peut remonter dans le répertoire parent à l'aide de la commande cd ..

7. Supprimer un répertoire

   ATTENTION cette commande est à utiliser avec énormément de précautions.

   A. Sur Windows: C'est la commande rmdir (remove directory). Créons le répertoire toto: mkdir toto Supprimons ce répertoire: rmdir toto Cependant si le répertoire toto n'est pas vide, Windows vous le signalera et ne fera rien. Il faudra alors utiliser des paramètres à la commande rmdir à savoir /s (Supprime tous répertoires et fichiers inclus dans le répertoire à supprimer). Exemple: rmdir /s toto (Le système demandera une confirmation)

   B. Sur Mac/Linux: C'est aussi la commande rmdir mais on utilise généralement rm on peut coupler cette commande avec les paramètres -rf Exemple: rm -rf toto (le -r supprime le répertoire et sous repertoire de manière récursive. Le -f ne demande aucune confirmation d'effacement et pas d'erreur si le fichier n'existe pas) Si vous faites un rm -rf répertoire (faites très attention à ce que vous faites…)

8. Supprimer un fichier

   Comme pour la commande précédente faites attention à ce que vous effacez

   A. Sur Windows: C'est la commande del fichier exemple: del toto.txt
   B. Sur Mac: C'est la commande rm fichier exemple: rm toto.txt
9. Renommer un fichier/répertoire

   A. Sur Windows: C'est la commande ren (pour rename). Exemple: ren toto.txt toto2.txt
   B. Sur Mac: On utilise la commande mv (move). Exemple mv toto.txt toto2.txt
10. Afficher le contenu d'un fichier

   A. Sur Windows: type nomfichier exemple: type toto.txt affichera le contenu du fichier toto.txt
   B. Sur Mac: cat nomfichier exemple: cat toto.txt affichera le contenu du fichier toto.txt
11. Créer un fichier avec un contenu vide

   A. Sur Windows: type nul > toto.txt va créer un fichier vide avec comme nom toto.txt
   B. Sur Mac: touch toto.txt
12. Les redirections des commandes > et >>

   On peut rediriger l'entrée standard d'une commande qui est la console vers par exemple un fichier. On a deux types de redirections:

   1. Création du fichier avec > par exemple dir > listing.txt (redirige le résultat de la commande dans un nouveau fichier nommé listing.txt)
   2. Concaténation du résultat dans un fichier avec >> par exemple dir >> listing.txt (Si le fichier existe, le résultat de la commande dir est ajouté au fichier listing.txt sinon le fichier sera créé). Dans le temps (vieux suis, vieux je resterai), on pouvait rediriger vers une imprimante. Par exemple dir > prn
13. Rediriger la sortie d'une commande vers un programme

J'ai vu la difficulté que certains ont à relire ce que l'invite de commandes renvoie comme message/sortie.

En cherchant un peu/beaucoup j'ai trouvé deux solutions que vous pourriez utiliser.

12. Customiser son prompt

   Il correspond à la string(chaîne de caractères) qui se trouve à gauche du curseur dans l'invite de commandes.

   Sous Windows par défaut c'est c:\RépertoireEnCours> il est défini par la commande prompt pg

   Il est possible de customiser le prompt de l'invite de commandes:

   A. Sur Windows: C'est la commande prompt qui le permet. Pour certains, un prompt plus succinct est sans doute intéressant. Exemple: prompt $g Donnera que le symbole > pour votre invite de commandes

   B. Sur Mac: Il faut définir une variable d'environnement PS1. Exemple: PS1=">"

Pour Windows, il existe des variables prédéfinies que l'on peut utiliser. Pour les connaître faites un prompt /?

6. Qu'est-ce qu'un dépôt/repository Git ?

Un dépôt Git est en quelque une sorte de livre journal où l'on peut retrouver tout l'historique des modifications. Il vous permet d'enregistrer les différentes versions de votre code et d'y accéder au besoin.

7. Répertoire courant vs la zone d'indexation vs dépôt

Répertoire de travail = Lieu où sont stockés vos fichiers de travail.

Zone d'indexation = La zone d'index est un simple fichier, généralement situé dans le répertoire Git, qui stocke les informations concernant ce qui fera partie du prochain instantané. On l'appelle aussi des fois la zone de préparation.

Dépôt (repository) = répertoire portant le nom .git. Le répertoire Git est l'endroit où Git stocke les métadonnées et la base de données des objets de votre projet. Vous ne devez jamais aller modifier manuellement ce répertoire.

8. Chronologie des actions avant que vos modifications fassent partie de votre dépôt

L'utilisation standard de Git se passe comme suit :

- Vous modifiez des fichiers dans votre répertoire de travail.
- Vous indexez les fichiers modifiés, ce qui ajoute des instantanés (une sorte de photo si vous voulez) de ces fichiers dans la zone d'index.
- Vous validez ces modifications qui sont en zone d'index, ce qui a pour effet de basculer les instantanés des fichiers de l'index dans la base de données du répertoire Git.

9. Configurer les informations de l'utilisateur pour tous les dépôts locaux

Pour que l'on sache qui a fait une action sur un dépôt, il est de mise de configurer les variables globales suivantes:

a. git config –global user.name “nom” - Exemple: git config –global user.name “Johnny Piette”
b. git config –global user.email “adresse email”

Exemple: git config –global user.email “johnny.piette@gmail.com”

Vérification de la configuration de git globale.
git config –global –list

Pour supprimer une entrée de notre configuration globale, on utilise: git config –unset –global user.name (Ici ça va supprimer notre prénom et nom).

10. Création de notre premier dépôt local

1. Création d’un répertoire de travail: créer le répertoire cours_git
2. En ligne de commandes aller dans le répertoire de travail à l’aide de la commande cd: cd cours_git
3. Taper la commande git init
4. Taper la commande dir on constate qu’il n’y a rien.
5. A. Pour Windows: Recommençons avec dir /ah (on liste les fichiers ayant l’attribut h pour hidden/caché)
6. B. Pour Mac: ls -al
7. On constate que l’on a un répertoire .git caché, notre dépôt.

11. Status de notre dépôt

Taper la commande git status que voyez-vous ? git status va nous indiquer l’état de notre dépôt.

Y a-t-il des fichiers modifiés/créés/supprimés qui n’ont pas été indexés ?

12. Ajouter un fichier

Dans le répertoire cours_git, copiez les fichiers:

- genius.js
- genius.html
- display.js
- inutile.txt

13. Indexer/désindexer ses modifications

13.1 Indexer un fichier

On utilise la commande git add suivit du nom de fichier:

1. git status (qu’est-ce qui a changé ?) On voit qu’il est indiqué untracked files. Il y est indiqué nos fichiers genius.js, genius.html, display.js, inutile.txt
2. git add genius.js (on l’ajoute dans la zone d’index)
3. git status (il indique les changements à commiter. ici genius.js qui est marqué comme “new file”)
4. Faites de même pour genius.html
5. Pour les fichiers display.js et inutile.txt faites un git add . Le point ajoutera tous ce qui a été détecté par git comme nouveaux, modifiés, supprimés.

13.2 Désindexer un fichier

Pour désindexer un fichier on utilise la commande git reset.

1. git reset inutile.txt (on a retiré les modifications de la zone d’index)
2. git status

13.3 Indexer plusieurs fichiers

Si vous avez modifier plusieurs fichiers ou ajouté un répertoire contenant une centaine de nouveaux fichiers, il sera alors fastidieux d’indexer un à un les nouveaux répertoires/fichiers.

Si l’on veut ajouter tous les fichiers présents dans le répertoire en cours, on utilisera la commande git add * Le caractère * signifie tous les fichiers du répertoire:

```
git add *
```

Cepdendant, on utilisera la commande git add . avec le caractère . si l’on veut ajouter tous les fichiers présents dans le répertoire courant plus les répertoires et fichiers/répertoires présents dans ces répertoires.

```
git add .
```

13.4 Indexer des fichiers supprimés

Ca peut sembler paradoxal. Indexer des fichiers supprimés ? Mais enfin ! Ils ont été supprimés ! On s'en …., non ? :-)

Oui, c'est juste si on les a supprimés localement, c'est qu'ils ne sont plus nécessaires. Mais si ces fichiers font partie de notre dépôts alors il est sans doute intéressant d'indiquer qu'ils ont été supprimés. Car sinon, ils seront toujours référencés comme non effacés dans le dépôt.

Pour indiquer à votre dépôt qu'ils ont été supprimés vous pouvez utiliser soit:

- git add nomfichier1 nomfichier2 nomfichier3 etc
- git add * (mais ici vous allez indexer tous les fichiers modifiés, créés, supprimés du répertoire courant)
- git add . (mais ici vous allez indexer tous les fichiers modifiés, créés, supprimés du répertoire courant et ses répertoires)

Il peut sembler étrange d'utiliser le paramètre add à la commande git. Effectivement, add signifie ajouter et on vient de supprimer des fichiers/répertoires ! Il faut penser que l'on ajoute cette modification (prise au sens large) dans la zone d'index.

Maintenant vous avez peut-être une bonne raison de ne pas indexer ces suppressions…

14. Notre premier commit

1. git commit -m "Ajout du fichier genius.js, genius.html, display.js"
2. git status (nothing to commit)
3. git tag v1 (Je vous expliquerai pourquoi plus tard)

15. Modifications d'un fichier

Première modification

1. Ouvrez votre programme genius.js avec vscode
2. Ajoutez une fonctions Multiply qui multiplie deux nombres.
3. Affichez le résultat s'inspirant du code dans genius.html pour afficher le résultat de la multiplication.
4. Testez votre programme jusqu'à ce qu'il fonctionne. :)
5. git status (git nous indique que notre fichier a été modifié)
6. git diff genius.js (Ca vous affichera les modifications: les ajouts (+) ou suppressions (-) de lignes)
7. git add genius.js
8. git add genius.html
9. git commit -m "Ajout de la multiplication" (Essayer de ne pas taper des accents)
10. git status (essayez de toujours faire un status par précaution)
11. git tag v2 (Je vous expliquerai pourquoi plus tard)

Deuxième modification

1. Ouvrez votre programme genius.js avec vscode
2. Ajoutez une fonctions Divide qui divise deux nombres.
3. Affichez le résultat s'inspirant du code dans genius.html pour afficher le résultat de la division.
4. Testez votre programme jusqu'à ce qu'il fonctionne. :)
5. git status (git nous indique que notre fichier a été modifié)
6. git diff genius.js (Ca vous affichera les modifications: les ajouts (+) ou suppressions (-) de lignes)
7. git add genius.js
8. git add genius.html
9. git commit -m "Ajout de la division" (Essayer de ne pas taper des accents)
10. git status (essayez de toujours faire un status par précaution)
11. git tag v3 (Je vous expliquerai pourquoi plus tard)

16. Voyage dans l'historique d'un dépôt

Un ensemble de commits reliés entre eux par un pointer constitue ce qu'on appelle une branche. Ici on est sur la branche master. C'est la branche principale. Un commit est constitué:

- d'un identifiant unique appelé SHA1 constitué de 40 caractères.
- un ensemble de modifications.
- un commentaire décrivant le commit qui vient de la commande commit -m "votre commentaire"
- les informations sur l'auteur (on les a donnés au point VI avec la commande git config –global user.name "nom" ainsi que de la commande git config –global user.email "adresse email")
- une date de création.
- Liste de son/ses parent/s (Allant de 0 à N parents).

Voyons l'historique de notre dépôt: 1. Tapez git log 2. git show SHA-1 (Pour un commit ayant ce SHA-1). Certains auront peut-être des problèmes pour copier/coller le SHA1. Ce n'est pas grave car nous avons tagué/nommé nos commits:v1, v2, v3. 3. git show v1 4. git show v2 5. git show master (montre les dernières modifications du commit avec le tag master)

17. Naviguer dans l'historique de notre dépôt

1. git log
2. Identifier le premier commit ainsi que son SHA1.
3. git checkout SHA-1 ou un tag (nous nous sommes déplacés sur un commit, nous avons remonté le temps) [Musique de Retour vers le Futur]
4. git log (Nous n'avons que l'histoire du tout premier commit)
5. Ouvrez genius.js et regardez ce fichier qui est différent de celle de master.
6. git checkout master (on revient sur le dernier commit appelé master).
7. Ouvrez genius.js

Essayez de revenir sur les différentes versions de notre programme genius.js et regardez si on est bien revenu dans une version précédente à l'aide des SHA1 ou des tags.

18. Les tags

Certains ont remarqué qu'il est plus aisé de créer des tags pour naviguer dans l'historique et pour revenir à une version antérieur de notre dépôt. Cependant on ne crée pas des tags pour tous les commits. On le fait quand c'est nécessaire. Je vous ai fait créer des tags à chaque commit pour que certains aient plus facile d'utiliser des tags que des SHA1.

On peut avoir plusieurs tags pour un même commit.

Donc pour créer un tag, on utilise la commande git tag montag. Pour supprimer un tag, on utilise la commande git tag -d montag

19. Le modèle distribué

Ici, nous avons travaillé sur notre dépôt local: tout est stocké dans notre ordinateur. Il existe différents modèles de gestion:

1. Modèle centralisé (svn, cvs): Tout est stocké sur un serveur central qui contrôle toute la base du code.

2. Modèle distribué (git, mercurial): Tous les développeurs ont une copie de base du projet. L'intérêt c'est que l'on ne doit pas être en permanence connecté au serveur pour travailler.

   Nous avons notre dépôt git hébergé sur un serveur (GitHub, GitLab) qui comprend un certains nombres de commits. Si des développeurs veulent travailler sur notre dépôt distant, les développeurs vont devoir dupliquer notre dépôt. Cela va dupliquer l'intégralité du dépôt et le .git et donc de l'ensemble des commits. Prenons deux développeurs: Sophie et Simon. Si Sophie travaille sur sa version locale, elle va faire un commit. Et envoyer sur le dépôt distant, l'ensemble des commits qu'elle a réalisés. Simon ayant la version de base pourra se synchroniser avec le dépôt distant et récevoir les modifications faites pour Sophie.

20. GitHub

Git est donc un outil de suivi des modifications d'un dépôt : il en contient son historique. GitHub est un hébergeur web de dépôts distants. C'est en quelque sorte un front end web Git pour des projets distants hébergés sur GitHub.

Mais il offre beaucoup plus que Git :

- Créer un wiki pour le dépôt.
- Contient un logiciel de suivi de problèmes.
- Possibilité de cloner le dépôt hébergé sur GitHub en local.
- Créer des gist : morceau de code que l'on peut partager, éditer, commenter, etc.
- Suivi de bugs avec les issues.
- Un wiki dédié au projet

La plupart des projets open sources sont sur GitHub : En 2020, 190 millions de dépôts dont 28 millions sont publics.

Racheté par Microsoft : 7 Milliards de dollars. Certains ont quitté car le vilain Microsoft a racheté GitHub : En 2018, 100.000 projets sur 75 millions ont quitté GitHub pour GitLab.

Il y a plus de 56 millions de développeurs et plus de 3 millions d'organisations qui l'utilisent.

Vous pouvez créer des dépôts publics et des dépôts privés. Avant on avait une limitation pour les dépôts privés. Maintenant GitHub permet de créer un nombre illimités de dépôts privés. :)

Apparemment, vous n'auriez pas de limite de taille par dépôt mais on conseille que cela soit entre 500MB et 1GB. La taille max d'un fichier ne peut faire plus de 100MB.

21. Cloner un dépôt hébergé sur GitHub avec la commande git clone

Cloner un dépôt signifie de faire une copie parfaite d'un dépôt distant. Sur GitHub, il y a énormément de dépôts publics.

Nous allons dans un premier temps clone les notes de cours qui se trouvent sur GitHub.

La commande qui permet de faire un clone d'un dépôt distant est la commande git clone suivie du nom du dépôt à clone.

1. Ouvrez un invite de commandes/terminal.
2. Créez un répertoire nommé mesdepots: mkdir mesdepots
3. Allez dans ce répertoire: cd mesdepots
4. Clonez le dépôt distant de notre cours git: git clone https://github.com/ZamBoyle/Eqla_Git.git
5. Faites un dir, vous devriez y voir un répertoire nommé Eqla_Git
6. Allez dans le répertoire: cd Eqla_Git
7. Affichez les fichiers: dir (pour Windows) ou ls (pour Mac)
8. Je vais créer un fichier sur le dépôt GitHub.
9. Mettez à jour votre dépôt: git pull
10. Chaque semaine, vous ferez un git pull dans le dépôt local de notre cours et le cours se mettra à jour depuis GitHub.

22. Présentation de la commande gh

gh est un outil en ligne de commandes qui permet de gérer vos dépôts sur GitHub.

L'utilisation de cet outil vient du constat qu'un stagiaire m'a dit que le site internet GitHub n'était pas du tout accessible.

Si vous tapez gh tout seul dans la console, gh vous affichera son menu d'aide et ses différents paramètres. Il faut voir gh comme un ensemble de commandes différentes que l'on appelle via des paramètres.

Les commandes principales sont: - gh repo pour la gestion des repos - gh gist pour la gestion des gists - gh issue pour la gestion des issues (problèmes rencontrés, demande d'amélioration, etc) - gh pr pour la gestion des pull requests - gh auth qui permet l'authentification que nous allons voir plus loin.

23. Installation de gh

23.1 Windows

Téléchargez le fichier à cette adresse: https://github.com/cli/cli/releases/download/v2.5.2/gh_2.5.2_windows_amd64.msi

23.2 Mac Os

```
brew install gh
```

23.3 Ubuntu

```
curl -fsSL https://cli.github.com/packages/githubcli-archive-keyring.gpg | sudo dd of=/usr/share/keyrings/githubcli-archive-keyring.gpg
echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/githubcli-archive-keyring.gpg] https://cli.github.com/packag
es stable main" | sudo tee /etc/apt/sources.list.d/github-cli.list > /dev/null
sudo apt update
sudo apt install gh
```

24. Authentification sur GitHub via gh

La procédure d'Authentification suivante ne devra être faite qu'une fois: gh retiendra nos credentials(=Informations d'Authentification).

1. Ouvrez un invite de commandes/terminal.
2. Tapez gh version si gh est bien installé vous devriez avoir le numéro de version de gh. Par exemple: gh version 1.8.1
3. Tapez gh auth login -s delete_repo Normalement on utilise gh auth login mais on va grâce -s delete_repo nous donner le droit de supprimer des repos via gh.
4. Appuyez sur ENTER à la question "What account do you want to log into ?". Ca va sélectionner Github.
5. Appuyez sur ENTER à la question "What is your prefered protocol for Git operations ?". Ca va sélectionner HTTPS
6. Appuyez sur ENTER à la question "Authenticate git with your GitHub credentials ? Ca va sélectionner Y pour yes.
7. Appuyez sur ENTER à la question "How would you like to authenticate GitHub CLI ?" Ca va sélectionner "Login with a web browser"
8. Copiez le one-time code qui s'affiche: par exemple 20B6-5A57
9. Appuyez sur ENTER pour ouvrir un navigateur sur GitHub
10. Sur la page web collez le one-time code.
11. Cliquez sur Continue
12. Cliquez sur Authorize GitHub pour autoriser gh à gérer nos dépôts sur GitHub:
13. Entrez votre mot de passe.
14. Cliquez sur "Confirm password".
15. Revenez dans l'invite de commandes.
16. Appuyez sur ENTER pour continuer suite à l'Authentification réussie sur GitHub.
17. Et voilà !!!!! :)

25. Creation d'un gist public

Un gist est simplement un partage d'informations. Ca peut être un programme PHP, du c, un listing, du texte, etc. Bref ça doit être du texte mais vu que l'on est sur GitHub, c'est principalement du code. ;) Via GitHub on peut directement écrire son gist.

L'intérêt d'un gist, c'est qu'il ne fait pas partie d'un dépôt. C'est du code que vous partagez par exemple avec un formateur ou une connaissance. Si ce code est bugué, on pourra éventuellement vous aider via les commentaires.

S'il est public, tout le monde peut y accéder. S'il est privé, il faudra avoir son url pour y accéder car il ne sera pas trouvable via la recherche GitHub pour tout le monde. Il est possible à toute personne d'ajouter un commentaire et d'y attacher des fichiers à ce commentaire.

C'est un outil vraiment fort utile. Vous pourriez donc l'utiliser avec vos formateurs quand vous rencontrez un problème. Vous donnez l'url de votre gist à la personne.

1. Ouvrez un invite de commandes/terminal.
2. Allez dans un répertoire où vous avez du code PHP.
3. Tapez la commande gh gist create-d "Ceci est mon premier gist" -p Le paramètre -d donne la description Le paramètre -p met le gist en public.
4. Vous recevez une url pour le partager.
5. Pour vérifier qu'il a été créé: gh gist list Votre nouveau gist devrait être listé.

26. Cloner un dépôt hébergé sur GitHub avec la commande gh repo clone

Cette fois-ci nous allons le faire via la commande gh repo clone En fait l'intérêt principal d'utiliser gh repo clone c'est de rapidement cloner ses repos mais il peut aussi clone d'autres repos mais autant utiliser alors la commande git clone.

1. Ouvrez un invite de commandes/terminal.
2. Allez dans le répertoire mesdepots: cd mesdepots
3. clonez le dépôt distant de notre cours git: gh repo clone https://github.com/ZamBoyle/Eqla_Git.git EqlaBis On va nommer le répertoire EqlaBis car nous avons déjà un répertoire Eqla_git
4. Faites un dir, vous devriez y voir un répertoire nommé EqlaBis
5. Allez dans le répertoire: cd EqlaBis
6. Affichez les fichiers: dir (pour Windows) ou ls (pour Mac)
7. Je vais créer un fichier sur le dépôt GitHub.
8. Mettez à jour votre dépôt: git pull
9. Chaque semaine, vous ferez un git pull dans le dépôt local de notre cours et le cours se mettra à jour depuis GitHub.

27. Créer un dépôt sur GitHub

Ici, on va créer un dépôt sur GitHub et en même temps notre dépôt local.

1. Ouvrez un invite de commandes/terminal.
2. Allez dans le répertoire mesdepots: cd mesdepots
3. Tapez la commande gh repo create MonDepotGitHub –public –confirm Cette commande va créer un dépôt local et distant (GitHub) public. Le paramètre –confirm donne notre accord aux questions suivantes:Voulez-vous créer un dépôt sur GitHut ? Voulez-vous créer un dépôt local ?
4. Vérifiez que le dépôt local a été créé: la commande dir ou ls devrait nous indiquer la présence du répertoire MonDepotGitHub
5. Entrez dans le répertoire MonDepotGitHub: cd MonDepotGitHub
6. Regardez le statut de votre dépôt: git status
7. Vérifier si GitHub a bien été configuré dans notre dépôt: git remote -v On constate que nous avons deux remotes un pour le fetch et un pour
8. Ajoutez des fichiers genius.js et un autre de votre choix dans ce répertoire.
9. Regardez le statut de votre dépôt: git status
10. Ajoutez tous les fichiers dans la zone d'index: git add * Ca copiera tous les fichiers du répertoire courant. Si on utilise git add . ça ajoutera tous les fchiers du répertoire courant plus les répertoires et sous réptertoires.
11. Commitez le tout: git commit -m "Initial Commit"
12. git push -u origin master On envoie nos modifications sur le remote origin (ici GitHub) de la branche master.

28. Le fichier .gitignore

Il peut arriver que dans votre dépôt que vous travaillez sur des fichiers temporaires pour vos tests et programmes. Il arrive aussi qu'en fonction de votre environnement de développement, celui-ci génère un nombre impressionnant de fichiers. Il est dès lors peu intéressant, de commiter des fichiers "non utiles" pour notre projet. Via l'interface web on peut dire qu'on veut un gitignore pour python, visual studio, Unity, R, etc.

Maintenant, il peut arriver que vous travailliez dans un répertoire temporaire. Pour vos tests, essais et que vous n'avez pas envie de pousser ce répertoire sur GitHub. On ajoutera dans le .gitignore l'information suivante tmp/*

On pourrait aller dans le répertoire .git/info (celui dans lequel il ne faut jamais aller) et ouvrir le fichier exclude. L'intérêt de faire ça, c'est que personne n'est au courant que vous avez un répertoire tmp/ avec des fichiers dedans. En effet, si une personne regarde le .gitignore elle verra la présence du répertoire.

Nous allons en créer un à la main car nous sommes maintenant des ninjas de la ligne de commandes.

1. Ouvrez un invite de commandes/terminal.
2. Allez dans votre répertoire mesdepots
3. Créez un nouveau dépôt: gh repo create gitignoreTests –public –confirm
4. Allez dans ce répertoire: cd gitignoreTests
5. Dans ce répertoire gitignoreTests, créez un fichier .gitignore avec visual studio code. Attention que le fichier commence par un point.
6. Ajoutez dans le fichier .gitignore les deux lignes suivantes: *.temp password.txt (On ne met évidemment pas un mot de passe en clair dans un fichier mais plutot dans un gestionnaire de mot de passes: par exemple keepass)
7. Sauvegardez ce fichier .gitignore
8. Créez un fichier nommé test.php: type nul > test.php ou sur Mac touch test.php
9. Vérifiez qu'il a été créé: dir
10. Faites un git status: notre fichier test.php est indiqué comme nouveau et n'étant pas dans la zone d'index.
11. Créez un fichier nommé password.txt: type nul > password.txt
12. Faites un git status: que constatez-vous ?
13. Créez un fichier nommé dump.tmp: type nul > dump.tmp
14. Faites un git status: que constatez-vous ?
15. Créez un fichier nommé password.tata: type nul > password.tata
16. Faites un git status: que constatez-vous ? Que pourrions-nous faire pour considérer tous les fichiers commençants par password ?
17. Faites un git add *
18. Faites un git commit -m "commit de test"
19. Faites un git push -u origin master

29. Le fichier README.md et le format Markdown

Ce fichier donne des informations sur le dépôt en question.

Quand vous avez créé un fichier README.md son contenu est directement affiché sur la page du dépôt. Un peu comme un index.html sur un site internet.

Ce fichier a une structure bien particulière est se base sur Markdown. Markdown permet de faire une belle mise en page.

L'étude de Markdown n'entre dans pas dans ce cours mais je vais vous donner quelques exemples.

29.1 Mettre un titre h1

On commence la ligne par un #

```
# Ceci est un titre 1

## Ceci est un titre 2
```

29.2 Afficher du code

On utilise ```PHP avant notre code et après notre code ```. L'intérêt est d'avoir un affichage du code propre et avec une colorisation syntaxique.

Exemple pour du code PHP:

````
```php
name = input("Quel est votre nom ?")
firstname = input("Quel est votre prenom ?")
```
````

Donnera le résultat suivant (voir le résultat sur github et pas via un pdf/html):

```
name = input("Quel est votre nom ?")
firstname = input("Quel est votre prenom ?")
```

On a plusieurs languages supportés: c, php, js, html, etc…

29.3 Mettre en gras

On encadre avec ** ou __ce qu'on veut mettre en gras. Une partie de ce texte est en **gras*_.

29.4 Mettre en italic

On encadre avec _ ou _ ce qu'on veut mettre en italique. Une partie de ce texte est en*italic*.

29.5 Liens hypertextes

Il est courant de donner des liens hypertextes dans le fichier README.md

a. Mettre le lien directement: http://www.google.be

b. Mettre entre crochet notre texte décrivant le lien hypertextLien vers Google
c. Description du lien plus un title Lien vers Google
d. Pointer vers un fichier Ce lien pointe vers un fichier de votre dépôt

29.6 Les tableaux

```
Colonne 1|Colonne 2|Colonne 3
---|---|---
Cellule 1|Cellule 2|Cellule 3
Cellule 4|Cellule 5|Cellule 6
Cellule 7|Cellule 8|Cellule 9
```

Aura pour résultat:

Colonne 1	Colonne 2	Colonne 3
Cellule 1	Cellule 2	Cellule 3
Cellule 4	Cellule 5	Cellule 6
Cellule 7	Cellule 8	Cellule 9

On constate que pour faire un tableau, qu'il faut séparer le nom des colonnes par des |.

Ensuite on met autant de triple tirets (---) qu'on a de colonnes séparés par un |.

On a ensuite nos colonnes séparées par un |

30. Pousser ses tags sur Github

Par défaut, les tags ne sont pas explicitement poussés. Il faut le faire via la commande

git push origin v1 enverra le tag v1 sur le bon commit.
ou encore
git push origin –tags enverra tous les tags

31. Les issues

C'est une partie intéressante de GitHub. On peut signaler un bug ou une demande de fonctionalité via ce qu'on appelle les issues.

Si votre dépôt est public, vous pouvez avoir des utilisateurs qui demanderont de l'aide ou une fonctionalité de votre programme PHP par exemple.

Pour créer une issue on rentre dans le dépôt local où l'on veut créer une issue.

1. cd mesdepots
2. gh repo create testsIssues –public –confirm
3. cd testsIssues
4. La commante suivante va créer une issue directement sur GitHub pour notre dépôt fraîchement créé: gh issue create -t "Fichier README.md est manquant" -b "Il manque le fichier README.md en effet !"
5. Vérifions que l'issue à été créée: gh issue list
6. Créer un fichier README.md
7. Mettez en titre 1 le texte suivant: dépôt TestsIssues
8. Ajoutez le texte suivant: Ce depôt ne sert que pour tester les issues.
9. Enregistrez-le
10. Faites un git status
11. Faites un git add _ (Rappel _ mets tous les fichiers du répertoire courant)
12. Faites un git commit -m "#1: Ajout du fichier README.md" (Le #1 indique que le commit porte sur l'issue 1. Lorsque l'on consultera sur GitHub l'issue on vera une référence à notre commit)
13. Poussez votre modification sur GitHub un git push -u origin master
14. Visualisation de l'issue sur GitHub: gh issue view 1 –web
15. cloture d'une issue via gh de notre issue: gh issue close 1 (Notre issue sera cloturée)
16. Vérifions que l'issue à été cloturée: gh issue list (Vous devriez avoir le message suivant: There are no open issues)

32. Informations de modifications d'un fichier

Pour voir toutes les modifications faites sur un fichier, on utilise la commande git blame. Par exemple:

1. cd mesdepots
2. cd Eqla_Git
3. cd Theo
4. git blame git.txt ou git blame -l git.txt pour avoir le sha1 complet du commit.

Ca nous affiche les commits, la date de modification, l'auteur et la modification (+ ou -).

33. Git stash: garder des modifications non commitées

Il peut arriver que nous ayont besoin de retourner à une version antérieure de notre dépôt avec la commande git checkout V1 (ou via un sha1).

Cependant, vous avez fait des modifications mais n'avez pas envie de commiter celles-ci car vous n'avez pas terminé.

Dans ce cas, si vous faites un git checkout V1 git vous dira qu'il ne peut revenir à une version antérieure car vous avez des modifications non commitées. Il vous indiquera que vous pouvez aussi faire un git commit ou un git stash.

Pour le stash, vous faites un git stash save "Un message portant sur vos modifications". Ca va indexer ça dans notre zone de Stash. Heinnnn ??? Une zone de stash ? Une zone en plus de la zone d'index ? Oui mais à la différence que la zone d'index peut être commitée et non la zone de stash.

Affichons les stashs présents: git stash list Ensuite, vous pouvez revenir à votre version V1: git checkout V1

Revenons à notre tag master: git checkout master Affichons les stashs présents: git stash list

Rechargeons nos modifications de la zone de stash: git stash pop 0 (0 car c'est le premier stash et en informatique bcp de choses ont comme premier index le 0)

Si vous avez terminés avec vos modifications ajoutez-les à la zone d'index, commitez et envoyez sur Github si nécessaire.

34. Faire un git pull avec une modification en local et distante

Lors du rapatriement du cours un élève m'a dit: monsieur je ne sais pas faire un git pull origin car ça génère une erreur !

En effet, celui-ci a modifié le fichier théorie git.txt pour sans doute corriger des éventuelles erreurs.

On verra ce que nous pourrons faire au chapitre suivant.

35. Modifications conflictuelles

Vous avez un dépôt sur GitHub.

Vous bossez sur votre dépôt à la maison depuis votre dépôt local A sur votre ordinateur.

Votre collègue bosse sur son dépôt au boulot depuis son dépôt local B sur l'ordinateur de son employeur.

A la maison, vous modifiez votre dépôt local A et vous poussez vos modifications sur GitHub.

Votre collègue modifie son dépôt local B et il va poussez ses modifications sur GitHub.

Il va rencontrer une erreur:

```
hint: Updates were rejected because the remote contains work that you do hint: not have locally. This is usually caused by another repo
sitory pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

Il faut savoir que git a toujours connaissance du parent de tout commit. Quand vous avez poussez vos modifications du dépôt A sur GitHub, le dépôt B n'a pas le dernier commit de GitHub. Hors, git va constater que le dépôt B n'est pas à jour. git propose de faire un git pull auparavant sur le dépôt B.

Le pull s'effectue mais git nous signale qu'il va y avoir un ou des conflits et qu'il a "mergé" (fusionné) nos fichiers modifiés locaux avec les distants.

```
Auto-merging Theo/git.txt
CONFLICT (content): Merge conflict in Theo/git.txt
Automatic merge failed; fix conflicts and then commit the result.
```

Deux cas se posent à vous:

- Vous ne voulez pas merger et vous voulez revenir à l'étape avant vos modifications. Faites un git merge –abort et vous reviendrez à l'étape précédente.
- Vous voulez analyser le merge effectué: éditez le/les fichiers en question. On va supposer que vous voulez garder le merge. On doit maintenant manuellement valider les merges(fusions) en éditant le fichier. Dans un merge il y a 5 parties:
  - le début (<<<<<<< HEAD),
  - Des modifications locales
  - une suite des caractères pour séparer les modifications locales et distantes: =======
  - Les modifications distantes.
  - la fin (>>>>>>>)

A vous de voir ce qui a lieu de garder dans les modifications. Une fois que c'est fait, vous devez aussi supprimer le <<<<<<< HEAD, les ====== et le >>>>>>>> Vous enregistrez vos modifications. Vous ajoutez dans la zone d'index, vous commitez et vous poussez vos modifications sur GitHub.

Maintenant notre dépôt distant est à jour. Mais le dépôt B va avoir quelques problèmes car il va vouloir faire un git push qui va ne pas fonctionner car le dépôt A a déjà pushé ses modifs.

dépôt local A : C1(A) ===> C2(B) ===> C3(A) ===> C4(A)
GitHub : C1(A) ===> C2(B) ===> C3(A) ===> C4(A)
dépôt local B: C1(A) ===> C2(B) ===> C3(A) ===> C4(B)

Le dépôt B: fait un git pull mais constate qu'il y a un merge et qu'il devra faire un commit qui va compliquer la lecture de la branche master.

On fera un git merge –abort

On fera ensuite un git pull –rebase (comme ça le commit parent sera C4(A) et le commit du dépôt B deviendra C5(B)

dépôt local A : C1(A) ===> C2(B) ===> C3(A) ===> C4(A)
GitHub : C1(A) ===> C2(B) ===> C3(A) ===> C4(A)
dépôt local B: C1(A) ===> C2(B) ===> C3(A) ===> C4(A) ==> C5(B)

Mais en faisant ça, on auto merge donc à vous de vérifier ce qu'il y a lieu de faire avant de faire le commit C5(B) et le push.

dépôt local A : C1(A) ===> C2(B) ===> C3(A) ===> C4(A)
GitHub : C1(A) ===> C2(B) ===> C3(A) ===> C4(A) ==> C5(B)
dépôt local B: C1(A) ===> C2(B) ===> C3(A) ===> C4(A) ==> C5(B)

Et si le dépôt A fait un git pull:

dépôt local A : C1(A) ===> C2(B) ===> C3(A) ===> C4(A) ==> C5(B)
GitHub : C1(A) ===> C2(B) ===> C3(A) ===> C4(A) ==> C5(B)
dépôt local B: C1(A) ===> C2(B) ===> C3(A) ===> C4(A) ==> C5(B)

6. Ajouter un dépôt local sur GitHub

36.1 Via Git

36.2 Via Git

7. Les branches dans Git
