



# TP 1 : Git

## Préparer un compte rendu

### 1- Introduction

Git est un logiciel de gestion de versions décentralisé. C'est un logiciel libre créé par Linus Torvalds, auteur du noyau Linux, et distribué selon les termes de la licence publique générale GNU version 2. En 2016, il s'agit du logiciel de gestion de versions le plus populaire qui est utilisé par plus de douze millions de personnes.

— <https://fr.wikipedia.org/wiki/Git>

Lors cette première séance du TP, nous allons initier à l'utilisation du Git et puis à la maintenance d'un répertoire de travail de manière structurée et ordonnée.

### 2- Premiers pas avec Git

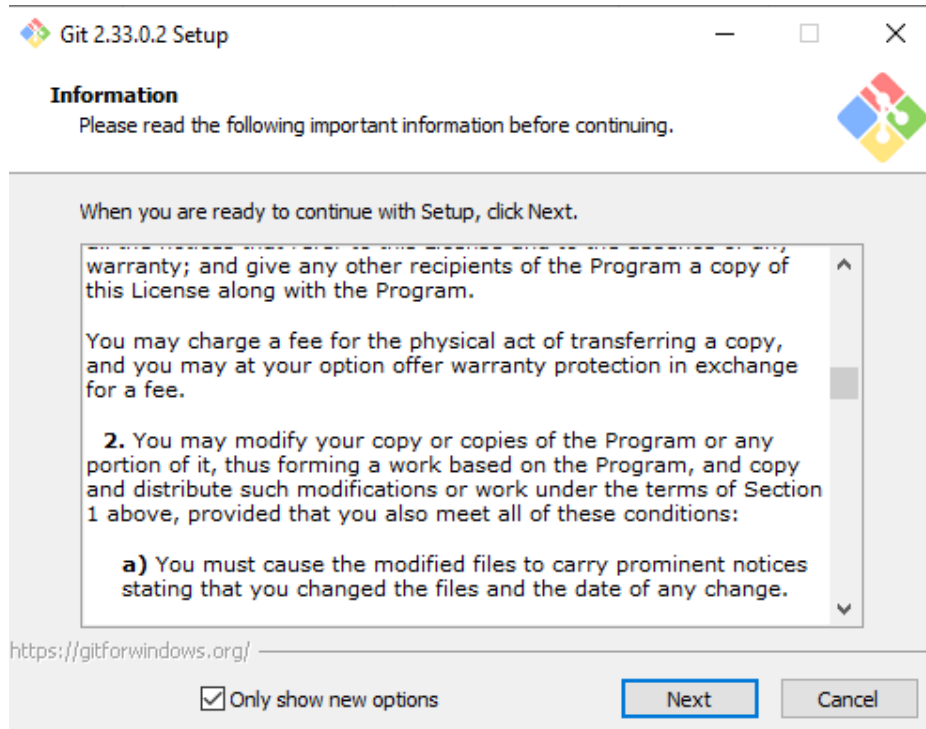
Git est un logiciel de contrôle de versions, il permet de sauvegarder l'historique du contenu d'un répertoire de travail. Pour ce faire l'utilisateur doit régulièrement enregistrer (en créant une révision ou commit) les modifications apportées au répertoire, il pourra ensuite accéder à l'historique de toutes les modifications et inspecter l'état du dossier à chaque révision.

Git a la particularité de permettre de créer une copie d'un répertoire de travail, working copy, et de synchroniser entre eux plusieurs copies du même répertoire, permettant la décentralisation du travail.

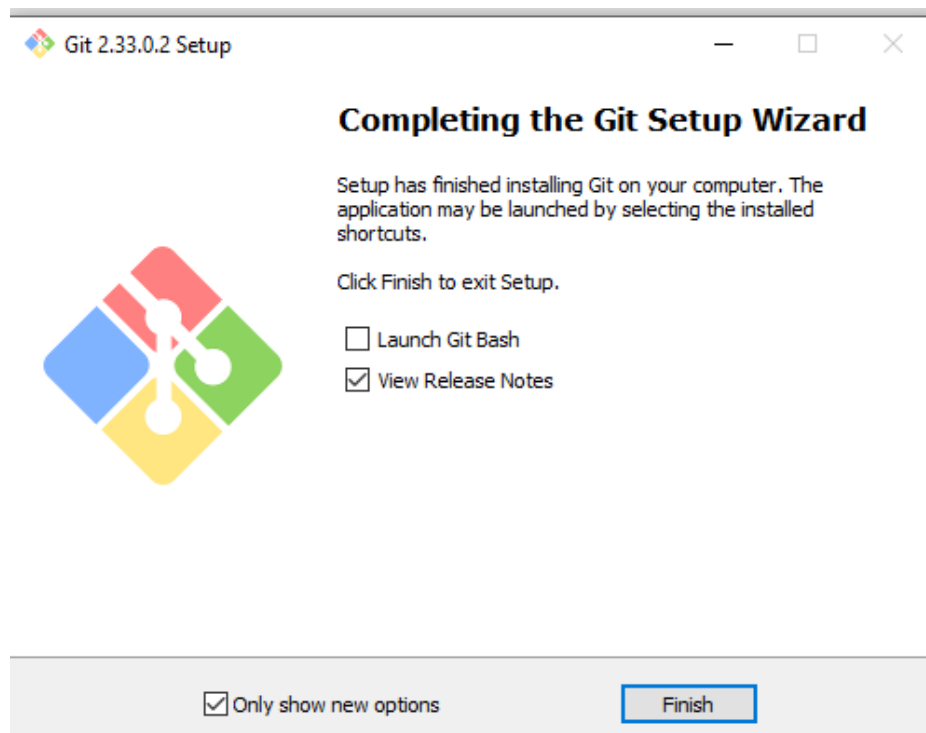
De plus, Git permet d'utiliser une ou plusieurs branches de développement et de fusionner entre elles ces branches.

## 2.1 Installation :

- a- Installer git à partir de ce lien : <https://git-scm.com/downloads>

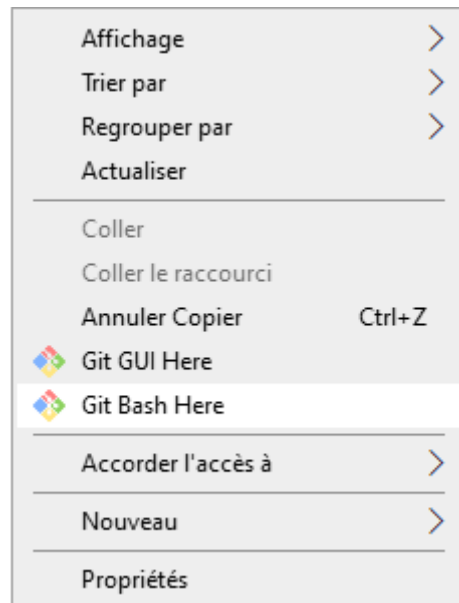


- b- Cliquez Next :

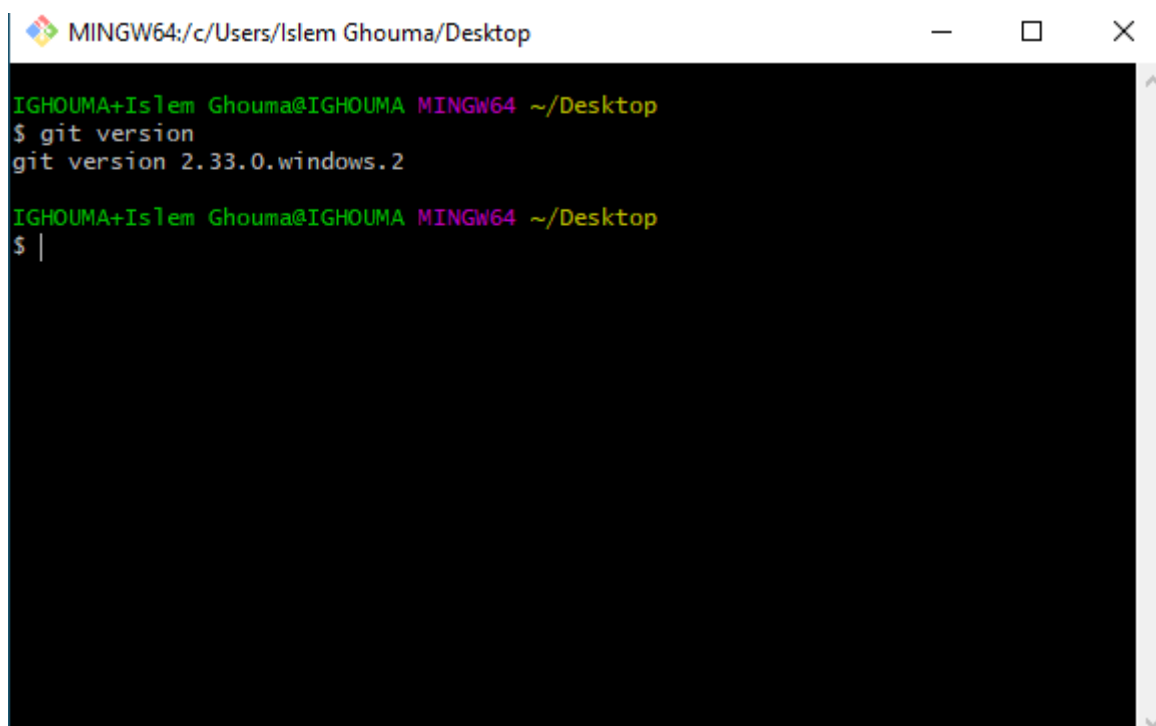


- c- Cliquez finish

d- Pour s'assurer que git est bien installer bouton droit et cliquez sur **Git Bash Here**



e- Tapez git version

A screenshot of a Windows terminal window titled 'MINGW64:/c/Users/Islem Ghouma/Desktop'. The prompt is 'IGHOUMA+Islem Ghouma@IGHOUMA MINGW64 ~/Desktop'. The command '\$ git version' has been entered, and the output is 'git version 2.33.0.windows.2'. The prompt '\$ |' is visible on the next line.

```
IGHOUMA+Islem Ghouma@IGHOUMA MINGW64 ~/Desktop
$ git version
git version 2.33.0.windows.2
IGHOUMA+Islem Ghouma@IGHOUMA MINGW64 ~/Desktop
$ |
```

## 2.2 GitHub :

GitHub est l'un des plus grands hébergeurs de dépôts Git et constitue le point central pour la collaboration de millions de développeurs et de projets. Une grande partie des dépôts Git



publics sont hébergés sur GitHub et de nombreux projets open-source l'utilisent pour l'hébergement Git, le suivi des erreurs, la revue de code et d'autres choses. Donc, bien que ce ne soit pas un sous-ensemble direct du projet open source Git, il est très probable que vous souhaiterez ou aurez besoin d'interagir avec GitHub à un moment ou à un autre dans votre utilisation professionnelle de Git.

#### a- Créer un compte GitHub

La première chose à faire consiste à créer un compte utilisateur gratuit. Allez tout simplement sur <https://github.com>, choisissez un nom d'utilisateur qui n'est pas déjà pris et saisissez une adresse électronique et un mot de passe, puis cliquez sur le gros bouton vert « Sign up for GitHub » (S'inscrire sur GitHub).



## Sign in to GitHub

Username or email address

Password

[Forgot password?](#)

Sign in

New to GitHub? [Create an account.](#)

#### Accès par SSH

Pour l'instant, vous avez la possibilité de vous connecter à des dépôts Git en utilisant le protocole <https://> et de vous identifier au moyen de votre nom d'utilisateur et de votre mot



de passe. Cependant, pour simplement cloner des projets publics, il n'est même pas nécessaire de créer un compte - le compte que nous venons de créer devient utile pour commencer à dupliquer (fork) un projet ou pour pousser sur ces dépôts plus tard.

Si vous préférez utiliser des serveurs distants en SSH, vous aurez besoin de renseigner votre clé publique. Si vous n'en possédez pas déjà une, référez-vous à Génération des clés publiques SSH.

Ouvrez Git Bash.

```
ssh-keygen -t rsa -b 4096 -C "votre_email@exemple.com "
```

email: -C label

default location: user home

generate defaulting to 2048-bit RSA (and SHA256)

```
MINGW64:/c/Users/Islem Ghouma
IGHOUMA+Islem Ghouma@IGHOUMA MINGW64 ~
$ ssh-keygen -t rsa -b 4096 -C "islem.ghouma@enis.tn"
```

```
MINGW64:/c/Users/Islem Ghouma
IGHOUMA+Islem Ghouma@IGHOUMA MINGW64 ~
$ ssh-keygen -t rsa -b 4096 -C "islem.ghouma@enis.tn"
Generating public/private rsa key pair.
Enter file in which to save the key (/c/Users/Islem Ghouma/.ssh/id_rsa):
```

Dans le dossier .ssh vous trouvez :

- `~/ .ssh/authorized_keys`: Contient une liste des clés publiques autorisées pour les serveurs. Lorsque le client se connecte à un serveur

Le serveur authentifie le client en vérifiant sa clé publique signée stockée dans ce fichier.

- `~/ .ssh/known_hosts`: Contient les clés d'hôte DSA des serveurs SSH auxquels l'utilisateur a accédé. Ce fichier est très important pour garantir que le client SSH connecte le bon serveur SSH.

Ajouter la clé publique dans GitHub.



## Exercice :

- 1- Pour démarrer un nouveau dépôt, utiliser **git init**
- 2- Configurer vos utilisateurs et vos courriels avec **git config**
- 3- Lancer **git config --list**
- 4- Utiliser echo pour créer 3 nouveaux fichiers avec comme contenu une ligne "Ligne 1"
- 5- Avec **git status**, git vous montre qu'il y a 3 nouveaux fichiers dans votre répertoire de travail qui sont "**untracked**" ou inconnu.
- 6- Pour ajouter ces fichiers au "staging" (première étape avant de faire un commit), il faut utiliser la commande **git add**.
- 7- Attention : cela n'ajoute pas le fichier à git, pour se faire il faut faire un "commit".
- 8- La commande **git status** vous montre l'état de votre dépôt
- 9- Le "staging" est la partie notée par "changes to be committed". Dans git, le "commit" correspond à la persistance dans la base de données git, et le commit prend les fichiers qui sont dans le staging.  
  
Pour créer un nouveau commit, utiliser la commande **git commit**.  
  
Ajouter le premier fichier dans une 1ere commit  
  
À partir du moment où un fichier est dans un commit, il est persisté dans git.
- 10- Ajouter fichier2 et fichier3 dans un 2e commit
- 11- Vous avez déjà fait 2 commits et la commande **git log** vous permet de les voir  
  
L'option **--name-status** donne la liste des fichiers modifiés (le A préfixé veut dire "added", vous avez aussi M pour "modified" et D pour "deleted")



12- La commande `git show` permet de voir le contenu des modifications, les lignes ajoutées sont préfixées d'un +, les lignes supprimées sont préfixées d'un - (ce sont les seules modifications possibles)

**Note :**

Le paramètre `HEAD` veut dire "commit courant"

La syntaxe `HEAD~1` permet de revenir un commit en arrière (`HEAD` est le commit courant, `HEAD~1` est le commit précédent, `HEAD~2` est le commit d'avant, etc.)

13- Utiliser **`git checkout`** pour récupérer la dernière version d'un fichier (si le fichier est modifié, on revient à la version du dernier commit)

14- Utiliser **`git revert`** pour créer un commit qui est l'inverse du commit (les lignes ajoutées seront enlevées et vice-versa)

15- Utiliser `git reset` pour supprimer le commit (attention : cette opération parfois est réversible, mais avec des commandes difficiles à utiliser, éviter l'usage)

**`git reset --hard HEAD~1`**

## **Les branches :**

16- Les branches permettent de développer des nouvelles fonctionnalités de manière indépendante, créer une nouvelle branche **`develop`** avec la commande **`git branch`**

Vous pouvez les voir comme des copies de votre espace de travail, entre lesquelles vous pouvez basculer rapidement, avec la commande **`git checkout`**

17- Pour l'instant, les 2 branches sont identiques, mais les modifications de l'une ne vont pas impacter l'autre.



On ajoute un fichier “fichierdev” dans la branche courante dans un nouveau commit

18- Est-ce que ce commit est présent dans master ?

19- Revenir au branche master

20- Créer un nouveau fichier dans master ‘masterfile’

21- Pour fusionner l’historique on utilise la commande git merge, en passant comme argument l’autre (ou les autres) branche à fusionner

22- Exécuter cette commande **git log --oneline --graph**

23- Analyser

24- On ajoute du contenu au fichier “fichierdev”, pour rappel, un fichier de même nom est dans la branche “develop”

25- On a maintenant :

Dans la branche “master”, le contenu de “fichierdev” est “Contenu master”

Dans la branche “develop”, le contenu de “fichierdev” est “Contenu develop”

Lorsqu’on fait le merge, git ne sait pas quelle version prendre

26- Afficher le contenu de fichier “fichierdev”

## Gestion des conflits

Un conflit se produit lorsque les modifications à fusionner sont incompatibles avec les modifications locales de la branche cible.

- Les fichiers avec des conflits contiennent des marqueurs de conflit
- Ils ne sont pas automatiquement ajoutés à l’index
- Résoudre le conflit
- Ajouter les fichiers à l’index
- Committer le résultat





## 27- Résoudre le conflit

Il faut dire à git que la résolution de conflit est terminée en ajoutant le fichier, puis en faisant un commit

28- `git remote add <nom> <url>`: permet de lier votre depot local (initié par init) a un depot distant (url sur GitHub)

29- Mettre le projet dans github

30- Créer une nouvelle branche release et faire un push

31- Supprimer la branche locale avec `git branch -d`

32- Supprimer la branche distante avec `git push -d`

33- Faire un pull request

## Autres Fonctionnalités

34- Créer un objet de type tag, contenant un nom et un commentaire avec la commande `git tag`

35- Transférer les tags séparément `git push --tags`

36- Prendre un commit dans une autre branche (une correction de bug) et l'appliquer à la branche courante avec la commande `git cherry-pick`

37- Identifier les auteurs avec la commande `git blame`

Pour chaque ligne d'un fichier montre le dernier commit et son auteur.

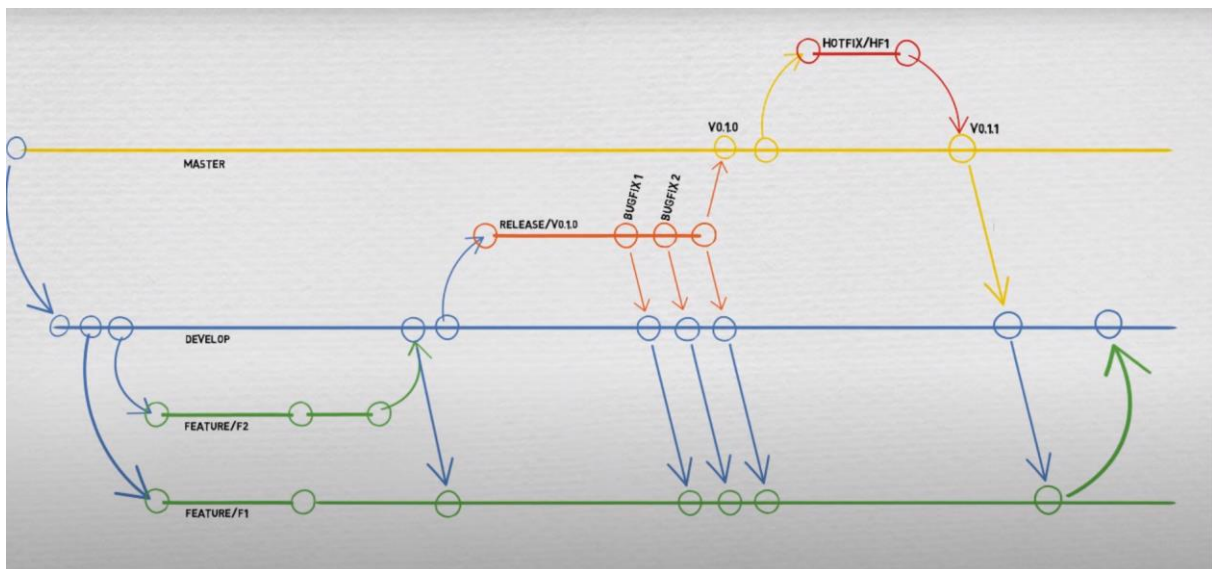
38- Récupère les commits distants sans les fusionner avec la commande `git fetch`

39- Mise à jour depuis un repository avec la commande `git pull`

- Récupère les branches distantes dans le repository local

- Fusionne la branche distante par d' défaut dans la branche courante et commite le résultat
- Peut produire un conflit.
- Résoudre le conflit et committer le résultat

## Git flow



- Une branche develop est créée à partir de master
- Une branche release est créée à partir de develop
- Les branches Feature sont créées à partir de develop
- Lorsque Feature est terminée, elle est fusionnée dans la branche de develop
- Lorsque la branche de publication est terminée, elle est fusionnée dans develop and master
- Si un problème dans master est détecté qu'une branche de hotfix est créée à partir du master
- Une fois Hotfix terminé, il est fusionné dans develop et master