

# TP GIT

Dans ce TP, nous allons les fonctionnalités de GIT les plus importantes/utilisées. Nous allons commencer par travailler localement pour tester les différentes fonctionnalités de base. Dans la deuxième partie nous verrons les fonctionnalités avancées. Pour finir, dans la troisième nous verrons comment utiliser GIT pour collaborer sur un même projet.

Toutes les commandes sont à taper dans un terminal. Notez qu'il est possible et recommandé pour plus de simplicité d'utiliser le terminal d'un environnement de développement intégré (IDE) tel que Rstudio et Vscode.

## 1 Installation et configuration

Dans un premier temps, nous allons vérifier que git est installé et faire les premières configurations.

### 1.1 Installation

Vérifiez la version de git installée avec la commande :

**git -v**

La commande ci-dessus affiche la version de git. Si le terminal affiche un message d'erreur, cela signifie que git n'est pas installé. Il faut alors le télécharger depuis <https://git-scm.com/downloads> et l'installer avec les paramètres par défaut.

### 1.2 Configuration de l'identité

Si cela n'a pas été fait auparavant, il faut renseigner le nom d'utilisateur et l'email. Ces informations sont importantes, notamment quand on travaille en équipe, pour garder une trace de la provenance des commits.

**git config --global user.email "you@example.com"**

**git config --global user.name "Your Name"**

Remplacez par vos identifiants personnels et tapez ces commandes dans votre terminal.

## 2 Premiers pas avec GIT

Dans cette première partie, nous allons voir les notions de base de GIT. Notamment pour gérer les versions d'un projet, dans un contexte mono-utilisateur.

### 2.1 Création d'un dépôt local

Créer un nouveau dossier (à partir de l'explorateur ou bien directement à partir du terminal avec la commande :

**mkdir "nom du dossier"**

A partir du terminal, naviguez vers ce dossier :

**cd "chemin du dossier"**

Ce dossier nouvellement créé représente votre zone de travail. Il faut maintenant créer la zone de transit et le dépôt local avec la commande :

### **git init**

Cette commande va initialiser votre dépôt. En pratique, un dossier caché nommé « **.git** » est créé, ce dossier contient des fichiers de configuration, c'est dans ce dossier où seront stockés tous vos commits.

Affichez avec **git status** l'état dans lequel se trouve votre dépôt.

```
>> git status
On branch master

No commits yet

nothing to commit (create/copy files and use "git add" to track)
```

Ce message indique que la zone de travail est à jour avec le dépôt local.

## 2.2 Premier commit

Créez un premier fichier « **premier\_fichier.txt** » et mettez-y du texte de votre choix (idéalement sur plusieurs lignes).

Vérifiez à nouveau l'état de votre zone de travail avec **git status**.

```
>> git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    premier_fichier.txt

nothing added to commit but untracked files present (use "git add" to track)
```

Ce message indique qu'il existe un fichier dans la zone de travail qui n'est ni indexé (dans la zone de transit) ni enregistré (dans le dépôt local).

Ajoutez le fichier créé à la zone de transit avec la commande :

### **git add "nom du fichier"**

En remplaçant **"nom du fichier"** par le nom de votre fichier.

Vérifiez à nouveau l'état de votre zone de travail avec **git status**.

```
>> git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   premier_fichier.txt
```

Ce message indique que le fichier est dans la zone de transit et est prêt à être enregistré (commit) dans le dépôt local.

Enregistrez les changements de la zone de transit dans le dépôt local avec la commande :

### **git commit -m "message de commit"**

Les messages de commits sont importants pour avoir une vision plus claire des modifications qu'apporte un commit. Assurez-vous de toujours écrire des messages clairs et courts (< 65 caractères). Evitez les messages de commit du type « modification du fichier x à la ligne y », mettez plutôt des messages du type « ajout de fonctionnalité x », « correction du bug y »...etc.

```
>> git commit -m "premier commit"
[master (root-commit) 736b2fd] premier commit
1 file changed, 3 insertions(+)
create mode 100644 premier_fichier.txt
```

Maintenant un **git status** vous affiche ce message :

**nothing to commit, working tree clean.**

Cela veut dire que votre zone de travail est à jour avec le dépôt local.

## 2.3 Visualisation des commits

Vous pouvez voir la liste de vos commits avec la commande :

**git log**

```
>> git log
commit 736b2fd2ddd48bea7863dfb777953673472a6450 (HEAD -> master)
Author: Riad Ladjel <riad.ladjel@quadratic-labs.com>
Date: Sun Feb 4 10:57:40 2024 +0100

premier commit
```

Pour l'instant, un seul commit est présent. **git log** affiche les informations importantes d'un commit :

- L'identifiant
- L'auteur
- La date
- Et le message du commit

Notes :

- Si le contenu du message affiché par **git log** ne tient pas sur tout le terminal, il faut appuyer sur les touches espace ou entrer de votre clavier pour faire défiler le texte. Pour quitter l'affichage, il faut appuyer sur la touche « q ».
- La commande **git log** affiche les commits du plus récent en premier au plus ancien.

## 2.4 Visualisation des différences

GIT nous donne la possibilité de voir les différences apportées à nos fichiers entre les différentes zones. Nous allons tester les différentes commandes.

Modifiez votre fichier en enlevant une ligne et en ajoutant une autre à la fin de celui-ci, puis affichez les différences avec la commande :

**git diff**

GIT affiche en rouge les lignes retirées et en vert celles rajoutées.

```
>> git diff
diff --git a/premier_fichier.txt b/premier_fichier.txt
index 2dedbc1..f5ef272 100644
--- a/premier_fichier.txt
+++ b/premier_fichier.txt
@@ -1,3 +1,3 @@
 1ere ligne pour tester
 une autre ligne
-3eme ligne...
\ No newline at end of file
+Ajout d'une nouvelle ligne
\ No newline at end of file
```

Ajoutez vos modifications à la zone de transit avec la commande :

**git add "nom du fichier"**

Maintenant la commande **git diff** n'affiche plus rien. Comme vos modifications sont déjà dans la zone de transit, il n'y a plus aucune différence.

Il est possible de voir la différence entre la zone de transit et le dépôt avec la commande :

**git diff --staged**

Faites de nouveaux commits, en ajoutant d'autres fichiers et/ou en modifiant le premier. Faites des **git add** et **git commit** après chaque modification et observez les différences avec les différentes commandes **git diff**.

Notes :

- Les commandes **diff** n'affichent les différences que si le fichier a déjà été indexé auparavant (mis dans la zone de transit ou le dépôt local) elles n'affichent rien pour les fichiers nouvellement créés (untracked).
- Il existe une commande « raccourcis » qui permet de faire **git add** et **git commit** en une seule étape « **git commit nom\_du\_fichier -m "message du commit"** ». Comme pour **git diff**, cette commande ne fonctionne que pour les fichiers déjà dans le dépôt.
- Il est possible d'ajouter toutes les modifications à la zone de transit en une seule fois sans spécifier les noms des fichiers modifiés. Il s'agit de « **git add .** ».

### 3 Utilisation avancée de GIT

Dans cette deuxième partie, nous allons voir l'utilisation des fonctionnalités avancées de GIT (gestion des changements, utilisation des branches, gestion de conflits).

#### 3.1 Gestion des changements

Dans les développements au quotidien, il arrive de se tromper et/ou de vouloir changer d'avis après avoir modifié un fichier ou même après l'avoir enregistré dans le dépôt local. GIT gère bien ces changements entre les différentes zones.

##### 3.1.1 Zone de travail-Dépôt local

Modifiez des fichiers dans votre zone de travail et regardez son état avec **git status**.

Pour annuler les modifications apportées à un fichier dans la zone de travail et le remettre à l'état du dernier commit, utilisez la commande :

**git checkout -- "nom du fichier"**

Essayez la commande et refaites un **git status** pour voir l'état de votre zone de travail. Regardez aussi les fichiers que vous aviez modifiés, les changements ont disparu.

Note : cette commande efface définitivement les modifications, il ne sera plus possible de les récupérer après.

##### 3.1.2 Zone de travail-Zone de transit

Il arrive aussi de vouloir retirer des fichiers de la zone de transit après les y avoir mis.

Modifiez un fichier puis ajoutez-le à la zone de transit (**git add "nom du fichier"**)

Regardez l'état de votre zone de travail avec **git status**. Les modifications sont dans la zone de transit.

Retirez les modifications ajoutées avec la commande suivante :

**git restore --staged "nom du fichier"**

Si vous affichez l'état de votre zone de travail avec **git status**, vous allez voir que les fichiers ont été retirés de la zone de transit mais que les modifications apportées sont toujours présentes.

### 3.1.3 Appliquer l'état d'un ancien commit à un fichier

Il est possible d'appliquer l'état d'un commit précédent sur un fichier. Affichez la liste des commits avec la commande **git log**.

Récupérez l'identifiant d'un commit précédent que vous voulez appliquer à un de vos fichiers (quand il n'y a pas de doublon, les cinq premiers caractères d'un identifiant suffisent).

Modifiez l'état d'un fichier avec la commande :

**git checkout "id du commit" "nom du fichier"**

Le fichier est maintenant au même état qu'il était au commit que vous avez choisi. Vérifiez l'état de la zone de travail avec **git status**. Les modifications apportées sont déjà dans la zone de transit. Vous avez la possibilité de soit enregistrer ces changements avec **git commit**. Ou bien de les annuler (en les enlevant de la zone de transit **git restore --staged "nom du fichier"**. Ensuite **git checkout -- "nom du fichier"** pour annuler les modifications.

## 3.2 Gestion des branches

Les branches sont l'une des fonctionnalités les plus importantes de GIT. Elles permettent d'avoir en parallèle plusieurs versions d'un même projet. Elles sont aussi utilisées pour garder une version stable pendant qu'on développe de nouvelles fonctionnalités dans notre projet en parallèle.

### 3.2.1 Création de branches

Créez une nouvelle branche avec la commande :

**git branch "nom de la branche"**

Affichez la liste des branches existante avec la commande :

**git branch**

```
>> git branch
* master
  nouvelle_branche
```

Git branch affiche les branches d'un projet. Le signe « \* » indique la branche actuelle du projet (parfois c'est indiqué en vert aussi, comme dans la figure ci-dessus).

Pour changer de branche, on utilise la commande :

**git switch "nom de la branche"**

Mettez-vous dans la branche nouvellement créée.

Note : il existe un raccourci pour créer une branche et d'y basculer directement, c'est la commande :

**git checkout -b "nom de la branche"**

### 3.2.2 Premiers commits sur une branche

Ajoutez des fichiers et modifiez en quelques-uns (n'oubliez pas de faire des **add** et des **commit**).

Visualisez vos nouveaux commits avec git log.

```

commit c860f32e3d431d5bb27021e7235858adf70b75c1 (HEAD -> nouvelle_branche)
Author: Riad Ladjel <riad.ladjel@quadratic-labs.com>
Date: Sun Feb 4 14:45:05 2024 +0100

    commit sur la nouvelle branche

commit a83722c70842f2f993303b22ad5da1737a93be40 (master)
Author: Riad Ladjel <riad.ladjel@quadratic-labs.com>
Date: Sun Feb 4 12:23:32 2024 +0100

    5eme commit

commit f24d6a76633ddf950efdba32427df83b1129bd95
Author: Riad Ladjel <riad.ladjel@quadratic-labs.com>
Date: Sun Feb 4 11:33:40 2024 +0100

    4eme commit

commit 7ecbede0311af9484441f36c985d327d93b71bb8
Author: Riad Ladjel <riad.ladjel@quadratic-labs.com>
Date: Sun Feb 4 11:33:07 2024 +0100

    3eme commit

```

A côté de l'identifiant de votre dernier commit, vous allez voir (**HEAD -> nom de la branche**). Cela indique que ce commit n'est présent que sur cette branche. (**HEAD** fait référence à la branche sur laquelle pointe votre projet). La notation (**master**) à côté de l'avant dernier commit de la figure ci-dessus indique que ce commit (et ceux qui le précèdent dans cet exemple) sont issus de la branche master.

Changez maintenant de branche pour vous rendre dans la branche master<sup>1</sup> avec la commande :

**git switch master**

Faites maintenant un **git log**. Les commits de l'autre branche ne sont plus visibles. Plus encore, vous ne voyez plus les modifications apportées sur l'autre branche. Comme dit en cours, les modifications d'une branche n'affectent pas les autres branches.

Ajoutez un fichier sur la branche master puis faites un **add/commit**. (Ne modifiez pas un fichier modifié dans l'autre branche, ça risque de générer des conflits. Notion qu'on verra un peu plus loin).

Maintenant, chacune de vos branches contient des commits qui n'existent pas dans l'autre branche. Dans la vie d'un projet informatique, ça ressemble au développement de deux versions d'un projet en parallèle.

### 3.2.3 Fusion de branches

On va maintenant fusionner les deux branches. Pointez votre projet sur la branche master (**git switch master**) et exécutez la commande :

**git merge "nom de la branche à fusionnée"**

Visualisez maintenant le log avec **git log**. Les commits de l'autre branche sont maintenant fusionnés avec la branche master.

Note : **git merge** n'affecte que la branche sur laquelle vous avez fait la fusion. Si vous changez de branche, vous n'y trouverez pas les modifications apportées à la branche master.

Maintenant que les modifications de la nouvelle branche sont fusionnées, nous n'avons plus besoin de cette dernière. Basculez vers la branche master si vous n'y êtes pas déjà et supprimez l'autre branche avec la commande :

**git branch -d "nom de la branche"**

Visualisez le résultat avec **git branch**.

---

<sup>1</sup> Selon votre configuration, la branche principale peut s'appeler main.

### 3.3 Gestion des conflits

Quand vous modifiez différemment la même partie d'un même fichier sur deux branches différentes. Git n'est pas capable de fusionner automatiquement les modifications. Il faut alors résoudre ces conflits manuellement.

Créez une nouvelle branche et pointez vers cette branche (utilisez le raccourci **git checkout -b "nom de la branche"**).

Modifiez un fichier existant et enregistrez vos modifications (**add/commit**).

Basculez vers la branche master et modifiez le même fichier puis enregistrez vos modifications.

Maintenant fusionnez les deux branches avec **git merge "nom de la nouvelle branche"**.

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

Le message ci-dessus s'affiche, indiquant qu'il y a eu un conflit lors de la fusion. La commande **git status** permet d'afficher les fichiers qui sont en conflit.

```
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)
    both modified:   premier_fichier.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

Ouvrez le fichier indiqué par **git status**. Le fichier contient maintenant des indications sur les parties du code qui sont en conflit. Les modifications issues de la branche sur laquelle on se trouve est entouré de « <<<<<< HEAD » et « ===== ». Et ceux de la branche entrante est entouré de « ===== » et « >>>>>> nom de la branche ». Voir exemple ci-dessous.

```
1ere ligne pour tester
une autre ligne
Ajout d'une nouvelle ligne
<<<<<< HEAD
Contenu issu de la branche master
=====
Contenu issu de la nouvelle branche
>>>>>> autre_branche
```

Pour résoudre le conflit, il suffit de choisir quel contenu garder (vous pouvez aussi choisir de garder les deux ou aucun) en fonction de vos besoins. Et de supprimer les symboles « <<<<<< HEAD », « ===== » et « >>>>>> nom de la branche ». Il vous faut ensuite faire un nouveau **add/commit** pour enregistrer vos modifications.

Note : si vous utilisez un IDE comme VsCode, le fichier en conflit est affiché comme suit :

```
Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes
<<<<<< HEAD (Current Change)
Contenu issu de la branche master
=====
Contenu issu de la nouvelle branche
>>>>>> autre_branche (Incoming Change)
```

Vous avez donc la possibilité de régler facilement le conflit en cliquant sur une des options en haut.

## 4 Collaboration à travers des dépôts distants

L'un des intérêts majeurs de GIT est de pouvoir travailler ensemble sur un projet sans se marcher dessus. Dans cette partie, nous allons voir comment utiliser des serveurs distants comme GitHub.

Cette partie est à faire par binôme ou plus. Quand une question est marquée Utilisateur A, cela veut dire qu'il n'y a qu'un seul utilisateur qui exécute la tâche (évidemment l'utilisateur devra être le même tout au long de la suite du TP). Inversement, si une question est marquée Utilisateur B, elle est à faire par le ou les autres membres du groupe (sauf Utilisateur A). Si rien n'est noté, la tâche est à faire par tous les membres du groupe.

### 4.1 Mise en place de l'environnement

#### 4.1.1 Création de compte

Nous allons utiliser GitHub comme dépôt distant, commencez par créer des comptes avec vos adresses mails sur <https://GitHub.com>.

#### 4.1.2 Etablissement d'un canal sécurisé

Comme on l'a vu dans le cours, pour pouvoir échanger des fichiers avec le serveur, il faut établir un canal de communication sécurisé en utilisant **ssh**. Pour ce faire, tapez **ssh** dans un terminal. Si un message d'erreur s'affiche, cela voudra dire que **ssh** n'est pas installé et il faut donc le faire. Sinon créez une paire de clés publique/privée avec la commande :

**ssh-keygen**


Appuyez sur la touche entrer sans rien modifier pour les questions qui s'affichent.

Cette commande va générer deux fichiers dans le répertoire **.ssh** de votre dossier personnel. **id\_rsa** et **id\_rsa.pub**. Le premier contient votre clé privée, il ne faut jamais la partager. Le second contient la clé publique. C'est celle que vous devez enregistrer sur le serveur à l'étape suivante.

Note : vous pouvez trouver le chemin de la clé publique qui vient d'être générée dans le message affiché par la commande précédente, par exemple :

**Your public key has been saved in C:\Users\username\.ssh/id\_rsa.pub**

#### 4.1.3 Enregistrement de la clé publique

Ajoutez la clé publique que vous venez de créer à votre compte GitHub. Pour cela, allez dans l'onglet paramètres (ou **settings**) en cliquant sur le rond en haut à droite de votre page d'accueil GitHub. Naviguez ensuite vers la section **SSH and GPG keys** et cliquez sur . Donnez un titre à votre clé et copiez le contenu de **id\_rsa.pub**.

Note : la génération de clés et l'enregistrement de la clé publique ne sont à faire qu'une seule fois par machine et non par projet. C'est-à-dire que si vous changez de machine, il faut recréer une paire de clés et enregistrer la clé publique sur le serveur. Sinon, si vous restez sur la même machine, vous pouvez participer à autant de projet que vous le souhaitez sans régénérer de nouvelles clés.


### 4.2 Premières étapes

Toutes les tâches de cette partie sont à faire par l'utilisateur A.

#### 4.2.1 Création d'un premier projet

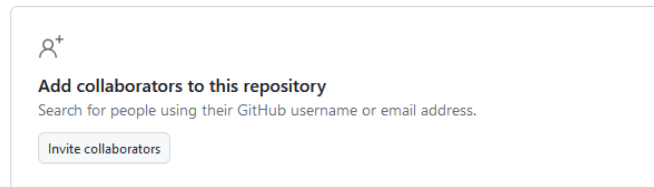
L'utilisateur A doit créer un premier projet qui sera utilisé par les autres membres du groupe pour partager leur code.



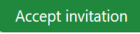


Créez un nouveau projet sur GitHub en appuyant sur . Entrez un nom pour votre projet (**Repository name\***) et laissez les autres paramètres par défaut. Par défaut, votre dépôt est public, c'est-à-dire que tout le monde a accès à votre code (droits de lecture).

#### 4.2.2 Inviter les collaborateurs

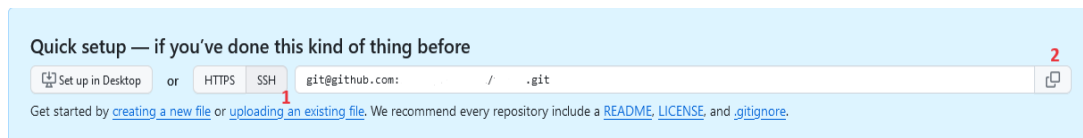
Maintenant que votre projet est créé, il faut inviter vos collaborateurs pour pouvoir y contribuer. Sur la page de votre projet appuyer sur « invite collaborators » (voir figure ci-dessous)



Dans la nouvelle page qui s'ouvre appuyez sur  et ajoutez vos collaborateurs. De leur côté, les collaborateurs doivent se rendre dans le compte et accepter l'invitation en cliquant sur  en haut à droite, puis en ouvrant le message d'invitation et en cliquant sur .

#### 4.2.3 Lier le dépôt distant à un dépôt local

Une fois votre projet créé, une nouvelle page va s'afficher. Cliquez sur ssh (1) et ensuite copiez le lien qui s'affiche (2) (voir figure ci-dessous).



Pour lier le dépôt distant à un dépôt local, ouvrez un terminal et lancez la commande :

**git clone "url copiée "**

#### 4.2.4 Premier commit

Allez dans le dossier qui vient d'être créé par la commande précédente et ajoutez des fichiers puis enregistrez les dans votre dépôt local (**git add, git commit**).

#### 4.2.5 Synchronisation de votre travail sur le dépôt distant


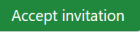
Allez voir dans votre projet sur GitHub, les modifications apportées localement n'existent pas encore et c'est normal, vous n'avez pas encore fait de synchronisation.

Pour synchroniser votre dépôt local avec le dépôt distant, utilisez la commande

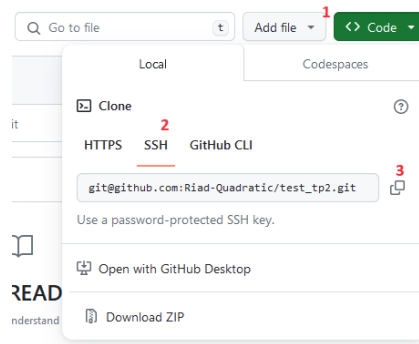
**git push**

Allez voir maintenant votre projet sur GitHub, vous allez voir les fichiers que vous venez de créer.

### 4.3 Contribution des collaborateurs

Cette partie est à faire par l'utilisateur B. Si cela n'a pas déjà été fait, acceptez l'invitation en cliquant sur  en haut à droite, puis en ouvrant le message d'invitation et en cliquant sur .

Allez dans le projet que vous venez de rejoindre et récupérez son URL. Pour ce faire, appuyez sur Code, ensuite SSH et enfin copiez l'URL. (Voir figure ci-dessous).



Dans un terminal, tapez la commande :

**`git clone "url du projet"`**

Vous avez maintenant une copie du projet localement. Ajoutez un fichier puis enregistrez le dans votre dépôt local (**`git add/ git commit`**).

Synchronisez maintenant vos modifications avec le dépôt distant avec la commande :

**`git push`**

#### 4.4 Récupération du travail des collaborateurs

##### **Utilisateur A :**

Récupérez une copie des modifications de vos collaborateurs localement avec la commande :

**`git pull`**

Votre zone de travail se met à jour avec les modifications apportées par vos collaborateurs. Vous pouvez voir la liste des commits avec **`git log`**.

Ajoutez maintenant un fichier, puis enregistrez le dans le dépôt distant (**`git add/ git commit/ git push`**).

##### **Utilisateur B :**

Récupérez une copie des modifications de l'utilisateur A localement avec la commande :

**`git pull`**

Inversez les rôles et faites d'autres tests de **push** et **pull**.

#### 4.5 Aller plus loin

Quelques idées pour aller plus loin :

- Explorer les différentes fonctionnalités qu'on a vu précédemment (branches, retour en arrière) avec les dépôts distants.
- Générer un conflit (en modifiant par exemple le même fichier et faire des push) et le résoudre.
- Essayer les interfaces graphiques qu'offrent les IDE comme VsCode ou Rstudio.
- Essayer la commande **`git stash`** qui permet de mettre du travail de côté.
- Utilisation des fichiers `.gitignore` pour spécifier la liste des fichiers à ne pas inclure.