
Qualité de développement

Systeme de gestion de version : Git

Introduction

Le but des systèmes de gestion de version (*version control systems*) est principalement double :

1. Garder un historique des différentes versions d'un projet (au sens large)
2. Faciliter la collaboration entre plusieurs personnes travaillant sur le même projet

Leur rôle est d'automatiser ces tâches et de les rendre efficaces.

Historique des versions

L'historique des versions est un enregistrement des modifications apportées à un projet au fil du temps. Il permet de revenir à une version antérieure du projet, de comparer deux versions, de voir qui a fait quoi, etc.

Collaboration

La collaboration est facilitée par la possibilité de travailler sur le même projet en même temps, sans risque de conflit. Les modifications sont enregistrées dans un dépôt centralisé, et chaque personne peut récupérer les modifications des autres.

Git

Git est un système de gestion de version distribué. Il permet de gérer des projets de toutes tailles, de collaborer avec des milliers de personnes, et de gérer des projets de manière efficace.

Nous allons maintenant présenter quelques notions de base de git.

Dépôt

Un dépôt est un ensemble de fichiers et de répertoires, ainsi que l'historique des modifications apportées à ces fichiers. Il est possible de créer un dépôt à partir d'un dossier existant, ou de créer une copie (*cloner*) d'un dépôt existant.

Localement, un dépôt est un dossier contenant un dossier `.git` qui contient l'historique des modifications apportées au dépôt.

Remarque : l'accès et la modification du dépôt doit se faire (sauf rares exceptions) via les commandes git.

Commit

Un **commit** est une modification apportée à un dépôt. Il est possible (et conseillé !) de faire plusieurs commits dans un dépôt, et de revenir à une version antérieure du dépôt en annulant les commits. Cela correspond à une étape dans votre développement. Vous modifiez plusieurs fichiers pour développer une ou plusieurs fonctionnalités, et quand c'est terminé (ou quand vous quittez votre lieu de travail par exemple), vous faites un *commit* pour indiquer que vous êtes à telle ou telle étape dans votre travail.

Un commit est composé de :

- Un message de commit (exemple : "partie css terminée", "algo de tri des étudiants quasi-terminé, reste à bien tester")
- Un auteur
- La date de création
- Un identifiant unique
- Les modifications apportées au dépôt

Index

L'index est un espace temporaire contenant les modifications prêtes à être *committées*. Ces modifications peuvent être :

- création de fichier ;
- modification de fichier ;
- suppression de fichier.

Cela vous montre la différence entre votre travail (les fichiers dans vos dossiers locaux) et le dépôt local. Cela correspond aux modifications non encore validées/transférées dans le dépôt.

Verbes

Les verbes de git sont les commandes permettant de manipuler le dépôt. Les verbes les plus utilisés sont :

- `git init` : créer un dépôt
- `git clone` : créer une copie locale d'un dépôt distant
- `git add` : ajouter des modifications à l'index
- `git commit` : "créer un commit" (valider les modifications en cours, les transférer dans le dépôt local)
- `git push` : envoyer les commits sur le dépôt distant
- `git pull` : récupérer les commits du dépôt distant
- `git status` : afficher l'état du dépôt (les différences entre l'index et le dépôt local)
- `git log` : afficher l'historique des commits

Identifiants (à faire une seule fois)

Pour pouvoir utiliser git, il faut configurer son nom et son email. Cela permet de savoir qui a fait quoi.

```
git config --global user.name "John Doe"
git config --global user.email "john.doe@email.com"
```

Aide

Pour obtenir de l'aide sur une commande, il suffit d'ajouter `--help` à la fin de la commande.

```
git add --help
```

Création d'un dépôt git

Pour mettre en place un dépôt, il y a principalement deux manières :

1. Faire d'un répertoire existant un dépôt Git (ce qui est expliqué en cours)
2. **Cloner** un dépôt existant (*quelque part*) (ce qui sera fait en TD)

Création d'un dépôt git local

Pour créer un dépôt git, il faut se placer dans le répertoire à versionner et exécuter la commande `git init`.

```
% cd /User/njozefow/TypeScript
% git init
Initialized empty Git repository in /Users/njozefow/TypeScript/.git/
% ls -la
total 0
drwxr-xr-x   3 njozefow  staff   96 Nov 15 07:47 .
drwxr-xr-x+ 113 njozefow  staff 3616 Nov 15 07:46 ..
drwxr-xr-x   9 njozefow  staff  288 Nov 15 07:47 .git
```

La commande crée un répertoire `.git`. Le répertoire `.git` contient l'historique des modifications apportées au dépôt. **Remarque** : si le répertoire n'est pas vide, il faut d'abord ajouter les fichiers au dépôt avec `git add` (cf ci-dessous).

Modification d'un dépôt git local

Une modification peut être :

- l'ajout d'un nouveau fichier ;
- la suppression d'un fichier ;
- renommer / déplacer un fichier ;

- la modification du contenu d'un fichier.

Ces états sont établis par rapport au dernier commit.

Remarque : le terme *fichier* est utilisé pour désigner un **fichier** ou un **répertoire**.

Tester l'état des fichiers

Pour voir l'état des fichiers, il faut exécuter la commande `git status`.

```
% git status
On branch main

No commits yet

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

Ajout d'un nouveau fichier sous suivi de version

On crée un nouveau fichier dans le projet :

```
echo 'console.log("Hello, world!");' > main.ts
```

On peut voir que le fichier n'est pas suivi par git :

```
% git status
On branch main

No commits yet

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

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

Le fichier est marqué comme non suivi (*untracked*). Cela signifie qu'il ne sera pas pris en compte lors des commits et que ses modifications ne sont pas suivies.

Pour ajouter le fichier à l'index, il faut exécuter la commande `git add`.

```
% git add main.ts
% git status
On branch main

No commits yet

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

On peut ajouter d'autres fichiers. Par exemple, ajoutons l'ensemble des fichiers nécessaires à la mise en place d'un environnement de développement TypeScript. On a donc maintenant un répertoire avec les fichiers suivants :

```
% ls -la
total 48
drwxr-xr-x@  6 njozefow  staff   192 Nov 15 07:54 .
drwxr-xr-x+ 113 njozefow  staff  3616 Nov 15 07:46 ..
drwxr-xr-x  10 njozefow  staff   320 Nov 15 07:53 .git
drwxr-xr-x   3 njozefow  staff    96 Nov 15 07:54 .vscode
-rw-r--r--@  1 njozefow  staff  296 Nov 15 07:54 deno.jsonc
-rw-r--r--   1 njozefow  staff    30 Nov 15 07:51 main.ts
```

L'état du dépôt est le suivant :

```
% git status
On branch main

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   main.ts

Untracked files:
  (use "git add <file>..." to include in what will be committed)
  .vscode/
  deno.jsonc
```

On peut remarquer que les répertoires sont pris en compte comme des fichiers. Cela revient à suivre l'ensemble des fichiers contenus dans le répertoire.

Si on veut suivre les fichiers ajoutés, il faut les ajouter soit un par un, soit en utilisant le caractère `*` pour indiquer un ensemble de fichiers.

```
% git add .vscode/
% git add deno.jsonc
% git status
On branch main

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
   new file:   .vscode/settings.json
   new file:   deno.jsonc
   new file:   main.ts
```

Commit

Pour enregistrer les modifications, il faut exécuter la commande `git commit`.

```
% git commit -m "Initial commit"
[main (root-commit) cbc476c] Initial commit
3 files changed, 24 insertions(+)
create mode 100644 .vscode/settings.json
create mode 100644 deno.jsonc
create mode 100644 main.ts
```

L'option `-m` permet de définir un message de commit. Le message de commit est obligatoire. Il est conseillé de définir un message de commit explicite. Le message de commit est utilisé pour décrire les modifications apportées au projet. Vous pouvez trouver des conseils pour rédiger un bon message de commit [ici](#).

On peut voir que le commit a été effectué avec succès. Si on regarde l'état du dépôt, on peut voir que le dépôt est propre et à jour.

```
% git status
On branch main
nothing to commit, working tree clean
```

Historique des commits

Pour voir l'historique des commits, il faut exécuter la commande `git log`.

```
% git log
commit cbc476cc6db69f112040c43067c9609ed94da721 (HEAD -> main)
Author: John Doe <john.doe@email.com>
Date:   Wed Nov 15 07:58:11 2023 +0100

    Initial commit
```

Les informations affichées sont les suivantes :

- **commit** : identifiant du commit
- **Author** : auteur du commit
- **Date** : date du commit
- **message** : message du commit

Les nouvelles modifications sont prises en compte à partir du commit. Par exemple, si on ajoute le fichier **README.md**, le fichier n'est pas pris en compte dans le commit précédent.

```
% echo "Cours Git" > README.md
% git status
On branch main
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    README.md

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

Pour que **README.md** soit pris en compte dans les futurs commits, il faut l'ajouter à l'index.

```
% git add README.md
% git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   README.md
```

On peut voir que le fichier **README.md** est ajouté à l'index. Si on fait un nouveau commit, le fichier **README.md** sera pris en compte.

```
% git commit -m "feat: Add README.md"
[main cd5a979] feat: Add README.md
1 file changed, 1 insertion(+)
create mode 100644 README.md
```

```
% git status
On branch main
nothing to commit, working tree clean
% git log
commit bb3fc961672801dc0822f8b17588a6d552bf9858 (HEAD -> main)
Author: John Doe <john.doe@email.com>
Date:   Wed Nov 15 08:02:11 2023 +0100

    Add README.md

commit cbc476cc6db69f112040c43067c9609ed94da721
Author: John Doe <john.doe@email.com>
Date:   Wed Nov 15 07:58:11 2023 +0100

    Initial commit
```

Remarque : dans le cours, on peut avoir l'impression que dès que l'on fait une modification sur un fichier, on fait un commit. Cela n'est pas la pratique habituelle. En effet, on fait des commits lorsque l'on a des modifications significatives. Par exemple, on peut faire un commit lorsque l'on a terminé une fonctionnalité ou une tâche.

Modification d'un fichier

On peut modifier un fichier. Par exemple, on peut modifier le fichier `main.ts` en ajoutant une ligne.

```
% echo "console.log('Hello Git');" >> main.ts;
```

Si on regarde l'état du dépôt, on peut voir que le fichier `main.ts` a été modifié.

```
% git status
On branch main
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   main.ts

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

On peut voir que le fichier `main.ts` a été modifié mais n'est pas ajouté à l'index. Pour ajouter le fichier à l'index, il faut exécuter la commande `git add`.

```
% git add main.ts
% git status
```

```
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
       modified:   main.ts
```

On peut donc maintenant faire un commit pour enregistrer la modification du fichier `main.ts`.

```
% git commit -m "feat: Modify main.ts"
[main a3e34ae] feat: Modify main.ts
 1 file changed, 1 insertion(+)
% git status
On branch main
nothing to commit, working tree clean
% git log
commit be30c5223cca6b8c297ed9f915aa84c23588105b (HEAD -> main)
Author: John Doe <john.doe@email.com>
Date:   Wed Nov 15 08:05:24 2023 +0100

    Modify main.ts

commit bb3fc961672801dc0822f8b17588a6d552bf9858
Author: John Doe <john.doe@email.com>
Date:   Wed Nov 15 08:02:11 2023 +0100

    Add README.md

commit cbc476cc6db69f112040c43067c9609ed94da721
Author: John Doe <john.doe@email.com>
Date:   Wed Nov 15 07:58:11 2023 +0100

    Initial commit
```

Suppression d'un fichier

On peut supprimer un fichier. Par exemple, on peut supprimer le fichier `main.ts`.

```
% rm main.ts
% git status
On branch main
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
       deleted:   main.ts

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

On peut voir que le fichier `main.ts` a été supprimé mais n'est pas ajouté à l'index. Pour supprimer le fichier à l'index, il faut exécuter la commande `git rm`.

```
% git rm main.ts
rm 'main.ts'
% git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        deleted:    main.ts
```

Remarque : on peut utiliser directement la commande `git rm` pour supprimer un fichier et l'ajouter à l'index.

Renommer un fichier

On peut renommer un fichier. Par exemple, on peut renommer le fichier `README.md` en `README.txt`.

```
% mv README.md README.txt
% git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        deleted:    main.ts

Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        deleted:    README.md

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

Du point de vue de git, on peut voir que le fichier `README.md` a été supprimé et le fichier `README.txt` a été ajouté. Pour les mettre à l'index, il faut exécuter la commande `git add` pour l'un et `git rm` pour l'autre. On peut faire les deux en même temps avec la commande `git mv`.

Commençons par annuler la modification non indexée sur `README.md` :

```
% git restore README.md
```

pour `README.txt`, il faut juste le supprimer puisqu'il n'est pas suivi par Git.

```
% rm README.txt
```

On est bien revenu à l'état avant de renommer le fichier `README.md` en `README.txt`.

```
% git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
   deleted:    main.ts
```

Nous pouvons maintenant renommer le fichier `README.md` en `README.txt` en utilisant la commande `git mv` et le mettre directement à l'index.

```
% git mv README.md README.txt
% git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    renamed:   README.md -> README.txt
   deleted:    main.ts
```

Cependant, nous allons voir qu'il y a une manière de prendre en compte les modifications non indexées lorsque l'on fait un commit.

Ne pas suivre des fichiers ou répertoires

On ne peut pas vouloir suivre tous les fichiers et répertoires présents. Par exemple, supposons que l'on ait un fichier `todo.md` qui contient la liste des tâches à faire. On ne souhaite pas suivre les modifications de ce fichier dans le dépôt git.

```
% echo "1. Apprendre le cours sur git" > todo.md
```

Le fichier `todo.md` n'est pas suivi par git. Si on ne l'ajoute pas (`git add`), il ne sera pas pris en compte dans les commits. Mais on peut même le rendre invisible pour git en le mettant dans le fichier `.gitignore`.

```
% echo "todo.md" >> .gitignore
% git status
```

```
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    renamed:   README.md -> README.txt
    deleted:   main.ts

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

Commit avec modifications non indexées

On peut voir que le fichier `.gitignore` n'est pas ajouté à l'index. On pourrait l'ajouter avec la commande `git add`. Mais si pour un commit, on a plusieurs modifications non indexées, il est plus pratique de les prendre en compte toutes en même temps. Supposons ici que l'on modifie le fichier `README.txt` et que l'on crée un répertoire `src` avec un fichier `main.ts` :

```
% echo "Hello World" >> README.txt
% mkdir src
% echo "console.log('Hello World')" >> src/main.ts
```

On a trois modifications non indexées que l'on souhaite prendre en compte dans le commit :

```
% git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    renamed:   README.md -> README.txt
    deleted:   main.ts

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:  README.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
  .gitignore
  src/
```

Remarque : Git prend en compte la création du répertoire `src` et à travers lui, la création du fichier `main.ts`.

Plutôt que de faire `git add` pour chaque fichier, on peut utiliser la commande `git commit -a` qui va prendre en compte toutes les modifications non indexées.

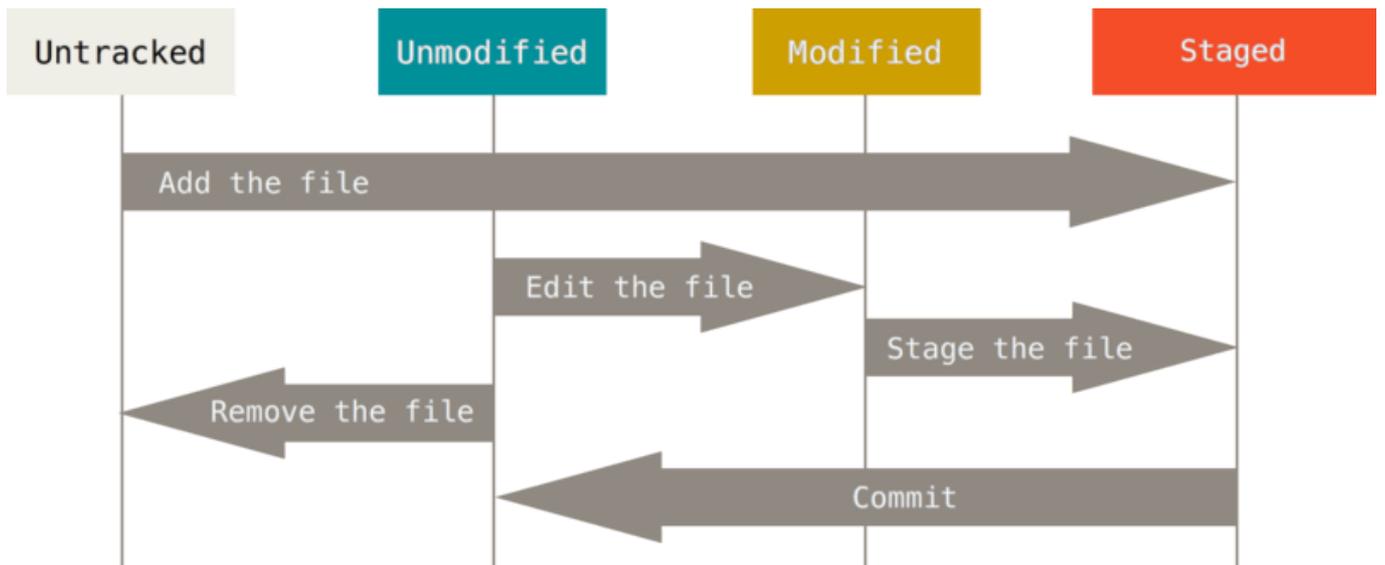
```
% git commit -a -m "add src directory + rename README.md + modify README.txt + deleted
main.ts + .gitignore"
[main 4d48afd] feat: add src directory + rename README.md + modify README.txt + deleted
main.ts
3 files changed, 2 insertions(+), 3 deletions(-)
delete mode 100644 README.md
create mode 100644 README.txt
delete mode 100644 main.ts
% git status
On branch main
Untracked files:
  (use "git add <file>..." to include in what will be committed)
  .gitignore
  src/

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

On voit que la modification du fichier `README.txt` est prise en compte dans le commit. Mais le fichier `.gitignore` n'est pas ajouté à l'index. Il en est de même pour le répertoire `src`. L'option `-a` met à l'index uniquement pour les fichiers qui sont déjà suivis. Pour les nouveaux fichiers, il faut donc d'abord les ajouter à l'index avec `git add`:

```
% git add .gitignore src
% git commit -a -m "feat: add src directory"
2 files changed, 3 insertions(+)
create mode 100644 .gitignore
create mode 100644 src/main.ts
% git status
On branch main
nothing to commit, working tree clean
```

Cycle des états d'un fichier



Revenir en arrière

Supposons que l'on efface le fichier `main.ts` et que l'on fasse un commit :

```
% git rm src/main.ts
rm 'src/main.ts'
% git commit -m "feat: delete main.ts"
[main 1b0b0a6] feat: delete main.ts
1 file changed, 1 deletion(-)
delete mode 100644 src/main.ts
```

On se rend compte par la suite que l'on a fait une erreur et qu'il faut récupérer le fichier `main.ts`. On peut le faire notamment en revenant à un commit antérieur. On a l'historique des commits avec la commande `git log`:

```
% git log
commit 2e2bb8f9f05bdac04dce2995dc2d1c21c424dc37 (HEAD -> main)
Author: John Doe <john.doe@email.com>
Date:   Wed Nov 15 08:22:17 2023 +0100

    delete main.ts

commit 1fb340dc7390f61e84c68733e7c083b95d6b69fa
Author: John Doe <john.doe@email.com>
Date:   Wed Nov 15 08:21:03 2023 +0100

    really add src and .gitignore

commit d9519a3197a4daf5b5789444ce58906430b7d435
Author: John Doe <john.doe@email.com>
```

```
Date: Wed Nov 15 08:18:57 2023 +0100
```

```
add src directory + rename README.md + modify README.txt + deleted main.ts +  
.gitignore
```

```
commit be30c5223cca6b8c297ed9f915aa84c23588105b
```

```
Author: John Doe <john.doe@email.com>
```

```
Date: Wed Nov 15 08:05:24 2023 +0100
```

```
Modify main.ts
```

```
commit bb3fc961672801dc0822f8b17588a6d552bf9858
```

```
Author: John Doe <john.doe@email.com>
```

```
Date: Wed Nov 15 08:02:11 2023 +0100
```

```
Add README.md
```

```
commit cbc476cc6db69f112040c43067c9609ed94da721
```

```
Author: John Doe <john.doe@email.com>
```

```
Date: Wed Nov 15 07:58:11 2023 +0100
```

```
Initial commit
```

On peut revenir à un commit à partir de son *numéro* :

```
% git reset --hard 1fb340dc7390f61e84c68733e7c083b95d6b69fa  
HEAD is now at 1fb340d really add src and .gitignore
```

On est bien revenu à l'état correspondant à ce commit :

```
% ls -la  
total 32  
drwxr-xr-x@ 9 njozefow staff 288 Nov 15 08:25 .  
drwxr-xr-x+ 113 njozefow staff 3616 Nov 15 08:28 ..  
drwxr-xr-x 13 njozefow staff 416 Nov 15 08:28 .git  
-rw-r--r-- 1 njozefow staff 8 Nov 15 08:16 .gitignore  
drwxr-xr-x 3 njozefow staff 96 Nov 15 07:54 .vscode  
-rw-r--r-- 1 njozefow staff 22 Nov 15 08:17 README.txt  
-rw-r--r--@ 1 njozefow staff 296 Nov 15 07:54 deno.jsonc  
drwxr-xr-x 3 njozefow staff 96 Nov 15 08:25 src  
-rw-r--r-- 1 njozefow staff 30 Nov 15 08:14 todo.md
```

Si on regarde l'historique des commits, on peut voir que l'on est revenu au commit choisi :

```
% git log
commit 1fb340dc7390f61e84c68733e7c083b95d6b69fa (HEAD -> main)
Author: John Doe <john.doe@email.com>
Date:   Wed Nov 15 08:21:03 2023 +0100

    really add src and .gitignore

commit d9519a3197a4daf5b5789444ce58906430b7d435
Author: John Doe <john.doe@email.com>
Date:   Wed Nov 15 08:18:57 2023 +0100

    add src directory + rename README.md + modify README.txt + deleted main.ts +
    .gitignore

commit be30c5223cca6b8c297ed9f915aa84c23588105b
Author: John Doe <john.doe@email.com>
Date:   Wed Nov 15 08:05:24 2023 +0100

    Modify main.ts

commit bb3fc961672801dc0822f8b17588a6d552bf9858
Author: John Doe <john.doe@email.com>
Date:   Wed Nov 15 08:02:11 2023 +0100

    Add README.md

commit cbc476cc6db69f112040c43067c9609ed94da721
Author: John Doe <john.doe@email.com>
Date:   Wed Nov 15 07:58:11 2023 +0100

    Initial commit
```

Remarque : si on utilise l'option `--hard`, toutes les modifications non indexées sont perdues. Si vous voulez les garder, il ne faut pas mettre cette option : `git reset <num>`.

Utilisation de Git avec un serveur distant

Jusqu'à maintenant, on a utilisé Git en local. On peut aussi utiliser Git avec un serveur distant. Cela permet de sauvegarder les modifications sur un serveur distant et de pouvoir travailler à plusieurs sur un même projet. Il existe de nombreux serveurs Git. On peut citer par exemple GitHub, GitLab, Bitbucket, etc. Dans le cours, nous allons utiliser le serveur gitlab de l'Université de Lorraine (<https://gitlab.univ-lorraine.fr>). Vous pouvez vous y connecter avec vos identifiants universitaires.

Remarque : vous ne pouvez disposer que de 5 dépôts sur le serveur GitLab de l'Université de Lorraine. Si vous en avez déjà 5, vous ne pourrez pas en créer de nouveaux et il vous faudra en supprimer un pour en créer un nouveau.

Création d'un dépôt distant

Une fois connecté sur le serveur GitLab, on peut créer un nouveau dépôt en *créant un nouveau projet* (en cliquant sur **New project** puis **Create blank project**). Vous devez alors donner un nom à votre projet. Vous pouvez aussi choisir la visibilité :

- **private** : il faut être invité pour voir le projet
- **public** : tout le monde peut voir et accéder au projet

Ici, nous allons le rendre **public** et l'appeler **coursGIT**. On va aussi décocher la case **Initialize repository with a README** car nous avons déjà un fichier **README.txt** dans notre dépôt local.

On obtient alors une page avec les informations du dépôt. Le dépôt est associé à une URL : <https://gitlab.univ-lorraine.fr/username/coursGIT.git>.

The screenshot shows the GitLab web interface for a newly created project named 'coursGIT'. The page displays a notification that the project was successfully created. Below this, there are options to 'Invite your team' and 'The repository for this project is empty'. A list of options for adding files is provided, including 'Clone', 'Upload File', 'New file', 'Add README', 'Add LICENSE', 'Add CHANGELOG', 'Add CONTRIBUTING', 'Set up CI/CD', and 'Add Wiki'. There are also 'Command line instructions' for setting up the repository locally, including global setup and creating a new repository.

Lier le dépôt local au dépôt distant

Pour lier le dépôt local au dépôt distant, on utilise la commande **git remote add** :

```
% git remote add origin https://gitlab.univ-lorraine.fr/username/coursGIT.git
```

On peut vérifier que le dépôt distant a bien été ajouté avec la commande **git remote -v** :

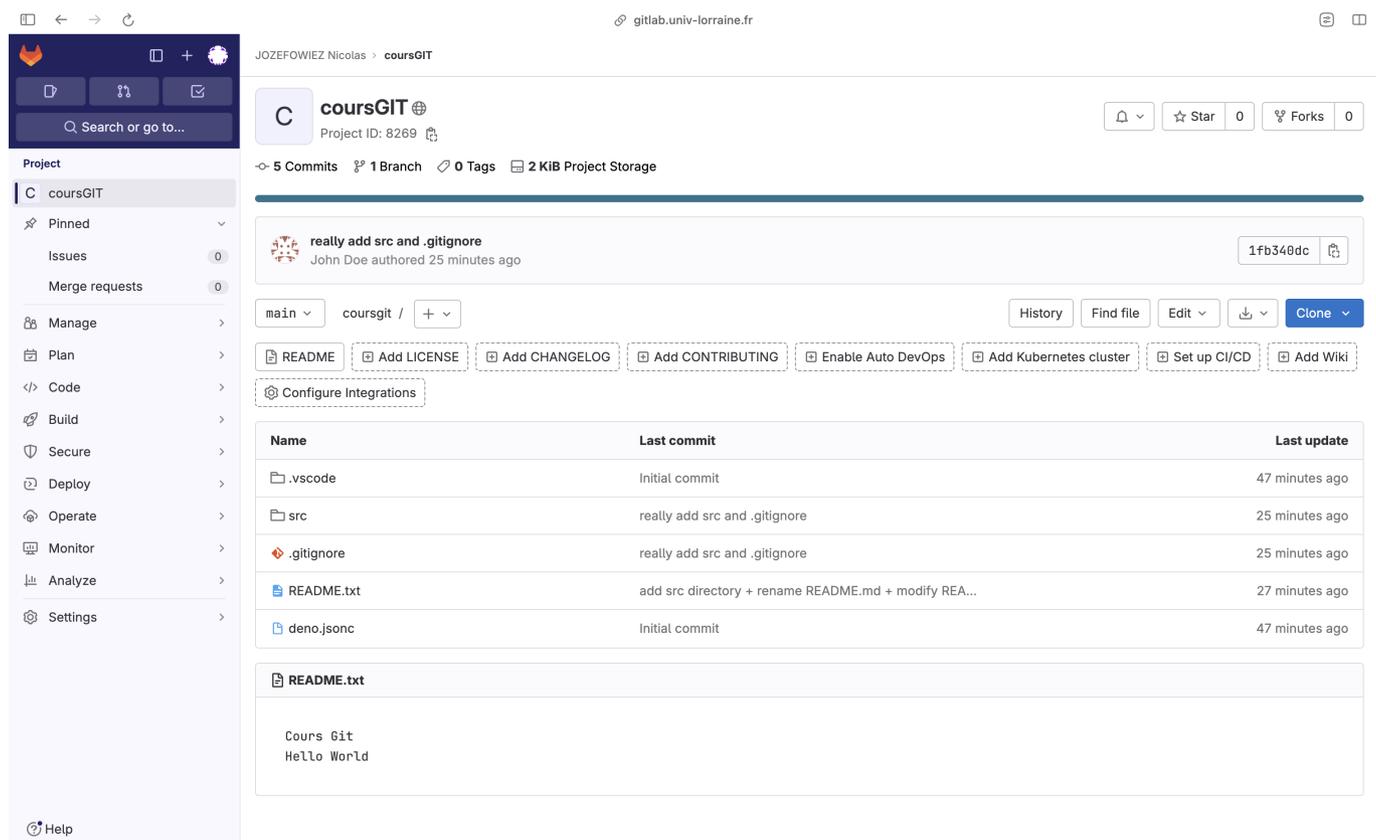
```
% git remote -v
origin https://gitlab.univ-lorraine.fr/username/coursGIT.git (fetch)
origin https://gitlab.univ-lorraine.fr/username/coursGIT.git (push)
```

Pousser les modifications sur le dépôt distant

Les modifications locales ne sont pas encore sur le dépôt distant. Pour les pousser, on utilise la commande `git push`:

```
% git push -u origin main
Enumerating objects: 20, done.
Counting objects: 100% (20/20), done.
Delta compression using up to 10 threads
Compressing objects: 100% (13/13), done.
Writing objects: 100% (20/20), 1.79 KiB | 1.79 MiB/s, done.
Total 20 (delta 3), reused 0 (delta 0), pack-reused 0
To https://gitlab.univ-lorraine.fr/jozefowi1/coursGIT
 * [new branch]      main -> main
branch 'main' set up to track 'origin/main'.
```

On peut voir que le dépôt distant a bien été mis à jour et que l'ensemble des commits que l'on a fait sur le dépôt local ont été poussés sur le dépôt distant (cliquer sur `x commits`):



The screenshot shows the GitLab web interface for the 'coursGIT' project. The project name is 'coursGIT' with Project ID: 8269. It shows 5 Commits, 1 Branch, 0 Tags, and 2 KiB Project Storage. A commit titled 'really add src and .gitignore' by John Doe is highlighted. Below the commit, there is a table of files and their last commit details.

Name	Last commit	Last update
.vscode	Initial commit	47 minutes ago
src	really add src and .gitignore	25 minutes ago
.gitignore	really add src and .gitignore	25 minutes ago
README.txt	add src directory + rename README.md + modify REA...	27 minutes ago
deno.jsonc	Initial commit	47 minutes ago

Below the table, there is a section for 'README.txt' with the content:

```
Cours Git
Hello World
```

On peut aussi voir que le dépôt local a été mis à jour avec la commande `git status` :

```
% git status
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
```

On a une information supplémentaire par rapport à la commande `git status` précédemment : `Your branch is up to date with 'origin/main'`. Cela signifie que le dépôt local est à jour avec le dépôt distant.

Exemple de modifications et de push

On va maintenant faire quelques modifications sur le dépôt local et les pousser sur le dépôt distant. On va par exemple ajouter un fichier `exo.ts` dans le dossier `src` et modifier le fichier `README.txt` et on va faire un commit :

```
% touch src/exo.ts
% echo "Ceci est un fichier d'exercice" >> README.txt
% git add src/exo.ts README.txt
% git commit -m "feat: add exo.ts + modify README.txt"
[main 04542fa] feat: add exo.ts + modify README.txt
2 files changed, 1 insertion(+)
create mode 100644 src/exo.ts
```

On peut voir que le dépôt local a été mis à jour avec la commande `git status` :

```
% git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
```

On a une information supplémentaire par rapport à la commande `git status` précédemment : `Your branch is ahead of 'origin/main' by 1 commit..` Cela signifie que le dépôt local est à jour mais qu'il y a une différence entre avec le dépôt distant. On a donc fait un commit sur le dépôt local qui n'est pas encore sur le dépôt distant. Si vous allez sur le dépôt distant, vous verrez que le fichier `exo.ts` n'est pas présent et que le fichier `README.txt` n'a pas été modifié.

On peut donc pousser les modifications sur le dépôt distant avec la commande `git push` :

```
% git push
Enumerating objects: 8, done.
Counting objects: 100% (8/8), done.
Delta compression using up to 10 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (5/5), 451 bytes | 451.00 KiB/s, done.
Total 5 (delta 1), reused 0 (delta 0), pack-reused 0
To https://gitlab.univ-lorraine.fr/jozefowi1/coursgit.git
 96ce157..04542fa main -> main
```

Si on regarde avec `git status`, on voit que le dépôt local et le dépôt distant sont synchroniser :

```
% git status
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
```

On peut voir que le dépôt distant a bien été mis à jour et que l'ensemble des commits que l'on a fait sur le dépôt local ont été poussés sur le dépôt distant (cliquer sur `x commits`).

Remarque : on n'est pas obligé de faire un `git push` à chaque fois qu'on fait un commit. On peut faire plusieurs commits et ensuite faire un `git push` pour pousser tous les commits sur le dépôt distant. C'est ce que l'on a fait par exemple sur le push initial.

Ressource

Ce cours est largement basé sur le livre *Pro Git* disponible gratuitement en ligne à l'adresse <https://git-scm.com/book/en/v2>.