Workshop GIT

Par Kristina Matrosova

PLAN

01 Présentation des outils

02 Installation

03 Configuration

04 Nouveau projet

05 GitHub Desktop

06 Projet existant en local

07 Introduction au travail collaboratif

MÉMO COMMANDES WINDOWS

Principales commandes terminal (Windows)

- ➔ `cd  nom_de_dossier` - accéder à un dossier
- ➔ `cd ..` - revenir un dossier en arrière
- ➔ `cd` - revenir à la racine
- ➔ `dir` - afficher les fichiers du dossier actuel
- ➔ `dir nom_de_dossier` - afficher les fichiers d'un dossier
- ➔ `md nom_de_dossier` - créer un dossier
- ➔ `cls` - nettoyer le terminal

MÉMO COMMANDES (MAC / UNIX)

Principales commandes terminal (Mac / UNIX)

- ➔ `cd nom_de_dossier` - accéder à un dossier
- ➔ `cd ..` - revenir un dossier en arrière
- ➔ `cd` - revenir à la racine
- ➔ `ls` - afficher les fichiers du dossier actuel
- ➔ `ls nom_de_dossier` - afficher les fichiers d'un dossier
- ➔ `mkdir nom_de_dossier` - créer un dossier
- ➔ `clear` - nettoyer le terminal

MÉMO COMMANDES GIT

Principales commandes Git

- ➔ `git clone` - copier un repository GitHub en local
- ➔ `git init` - initialiser un dossier avec les fichiers de configuration git
- ➔ `git add fichier.ext` - ajouter un fichier
- ➔ `git add -A` - ajouter tous les fichiers
- ➔ `git commit -m “message”` - soumettre les modifications
- ➔ `git branch -M branchname` - sélectionner la branche pour appliquer les modification (main = branche principale)
- ➔ `git remote add origin https://github.com/username/projectname.git` - rattacher le dossier local avec le repository git
- ➔ `git push` - appliquer les modifications
- ➔ `git push -u origin main` - appliquer les modifications à la branche d’origine
- ➔ `git pull` - récupérer la dernière version du projet
- ➔ `git status` - afficher la liste des fichiers modifiés

PRÉSENTATION DES OUTILS

01

PRÉSENTATION DES OUTILS

Qu'est-ce que Git?

Git est un **logiciel de versionning** qui permet de **stocker un ensemble de fichiers** en conservant la **chronologie de toutes les modifications** qui ont été effectuées dessus.
Il fait partie de la famille des VCS dit décentralisés car dans son fonctionnement chaque développeur va avoir en local une copie complète de l'historique de son code source (dépôt ou repository en anglais).

Pourquoi utiliser un logiciel de versioning?

1. **Conserver un historique des modifications** effectuées sur un projet afin de pouvoir rapidement identifier les changements effectués et de revenir à une ancienne version en cas de problème.

2. Faciliter la **gestion de projet en équipe** de manière à ce que :
    a. Les fonctionnalités sur lesquelles travaille un membre du projet ne rentrent pas en conflit avec les fonctionnalités sur lesquelles travaillent les autres
    b. Chacun sache sur quoi travaillent les autres membres du projet afin de travailler correctement
    c. Chacun possède une version actualisée du projet pour tester et implémenter ses fonctionnalités

Dans ce premier workshop nous nous intéresserons uniquement à l'utilisation individuelle de Git. Une deuxième session pourra être organisée sur demande pour étudier les spécificités du travail collaboratif.

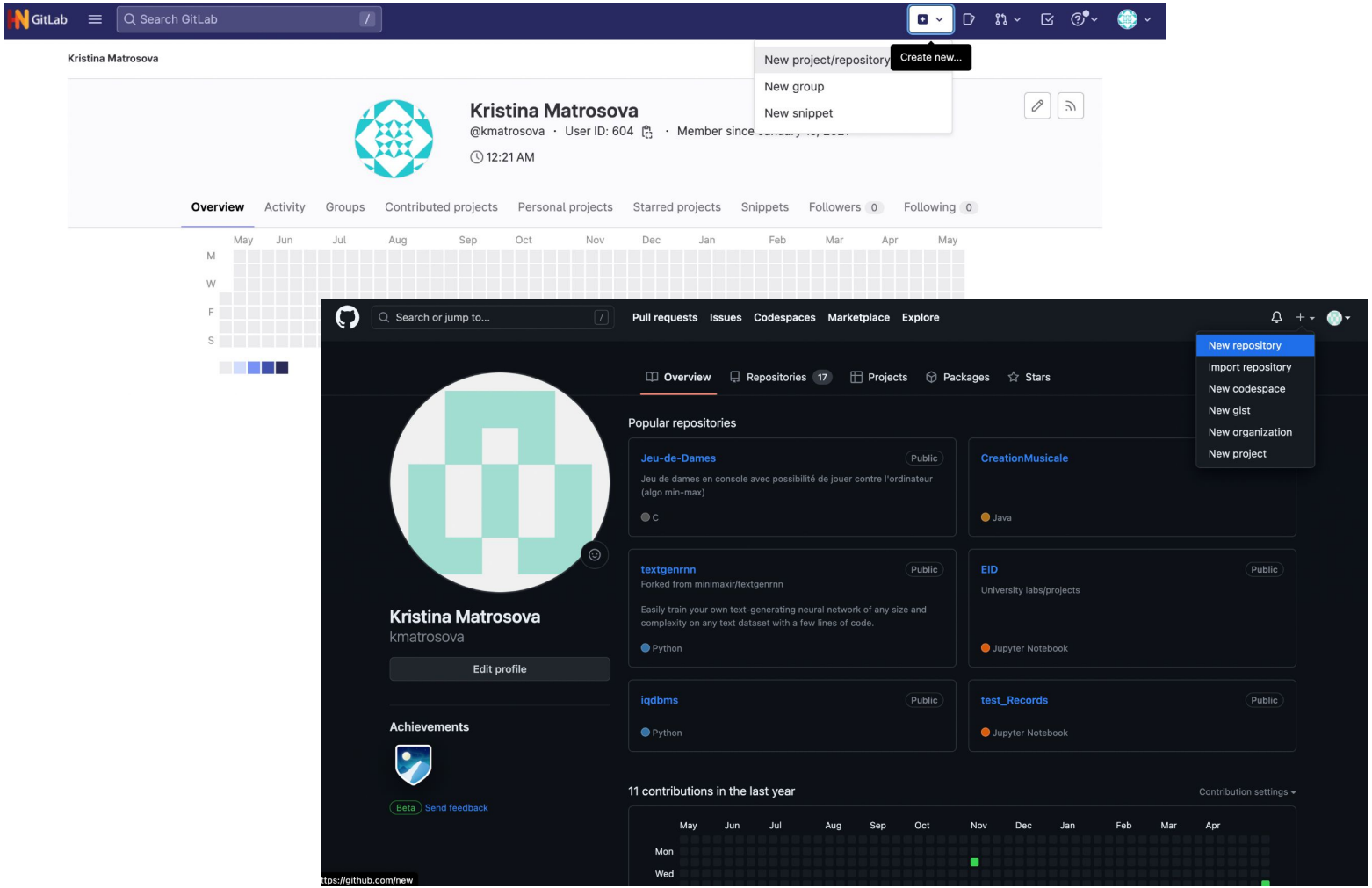
Qu'est-ce que Github?

Github est un service en ligne qui permet entre autres d'héberger des dépôts Git. Il permet notamment de :

- **Partager du code source** avec d'autres développeurs et chercheurs.
- **Signaler et gérer les problèmes** ou bugs de votre code source via les issues.
- **Proposer des évolutions** pour un projet open source.

D'autres plateformes de ce type existent (GitLab, Bitbucket etc.), mais GitHub reste la plus utilisée à ce jour, et est notamment disponible pour la plupart des OS (Windows, Mac, Linux) et est compatible avec de nombreux IDE. Nous avons donc choisi de travailler avec Github dans le cadre de ce workshop.

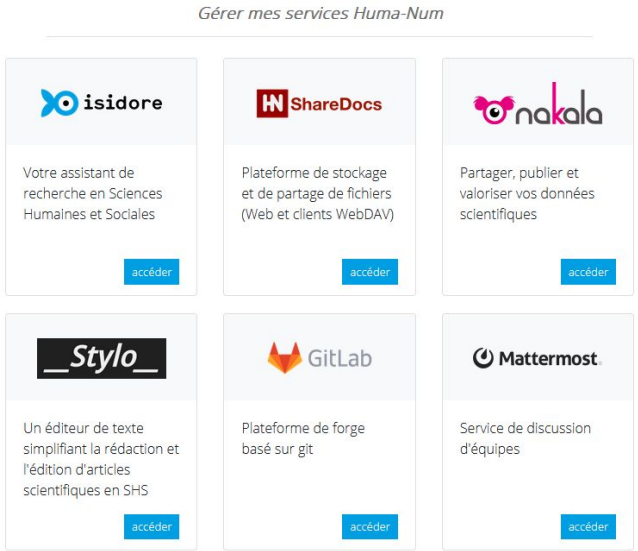
PRÉSENTATION DES OUTILS



L'instance GitLab d'HumaNum

- Demandez l'ouverture d'un compte HumanID : **https://humanid.huma-num.fr/**
- Vous pouvez ensuite demander l'accès aux différents services d'Huma-Num

La création du compte est très rapide si vous possédez un email institutionnel de l'ESR



INSTALLATION

02

Prérequis

Avant de commencer le tutoriel, assurez-vous d'avoir créé un compte sur le site https://github.com/.

INSTALLATION

Installation de Git sous Windows :

1. Téléchargez la dernière version sur https://gitforwindows.org/.
2. Lancez l'installation et suivez les instructions.
3. Ouvrez un terminal, entrez la commande `git version` pour vérifier que l'installation s'est bien passée.

Installation de Git sous MacOS :

1. La plupart du temps, Git est déjà installé sur MacOS. Ouvrez un terminal et entrez la commande `git version` afin de vérifier si c'est le cas.
2. Si ce n'est pas le cas, téléchargez la dernière version de Git en suivant ce lien.
3. Lancez l'installation et suivez les instructions.
4. Dans le terminal, entrez la commande `git version` pour vérifier que l'installation s'est bien passée.

INSTALLATION

Installation de Git sous Linux :

1. Sous Debian/Ubuntu :
    a. Dans le terminal, entrez la commande `sudo apt-get update` afin de s'assurer que l'installateur `apt` est à jour.
    b. Installez Git avec la commande `sudo apt-get install git-all`.
2. Sous Fedora :
    a. Installez Git en entrant la commande `sudo dnf install git-all` dans le terminal.
3. Entrez la commande `git version` pour vérifier que l'installation s'est bien passée.

CONFIGURATION

03

CONFIGURATION

Une fois que Git est installé, il va falloir faire quelques configurations.

Tout d'abord, nous allons **configurer le username et l'email**. Dans le terminal, entrez les commandes suivantes (en remplaçant les variables entre guillemets) :

➔ git config --global user.name "mon username"
➔ git config --global user.email "mon email"

Vous pouvez ensuite taper *git config --global --list* pour vérifier que la configuration a bien été prise en compte.

```
(base) kmatrosova@FRLPMC2823 ~ % git config --global --list
user.name=kmatrosova
user.email=tina.matrossova@gmail.com
```

CONFIGURATION

Nous allons utiliser Git depuis le terminal. Par défaut, à chaque interaction entre votre machine locale et GitHub, il vous sera demandé de vous authentifier avec vos identifiants. Depuis 2021, GitHub a remplacé (pour les opérations en terminal uniquement) le mot de passe par un **Personal access token** : c'est une sorte de mot de passe temporaire pour l'authentification durant l'utilisation en ligne de commande. Il permet d'avoir une sécurité plus élevée que celle du mot de passe, car il est généré aléatoirement et renouvelé régulièrement.

Néanmoins, cela peut être pénible d'entrer ses identifiants plusieurs fois dans la journée. Pour éviter cela, il est possible de configurer une paire de clés **SSH**. Ce moyen permet de créer une connexion durable et sécurisée entre votre machine locale et votre compte GitHub.

Connexion en SSH

Les clés SSH servent à authentifier les utilisateurs sans utiliser de noms d'utilisateur ni de mots de passe. Au lieu de cela, les utilisateurs disposent d'une paire de clés SSH de confiance qui les authentifie comme la personne qu'ils disent être.

La **clé publique** permet de chiffrer un message ou un document, tandis que la **clé privée** permet de le déchiffrer. Ceci garantit que seul le destinataire choisi peut déchiffrer et lire le contenu.

Création d'une clé SSH

1. Vérifiez si vous avez déjà configuré une clé SSH. Pour cela, allez dans le dossier `~/.ssh/` avec la commande `cd ~/.ssh/`, puis vérifiez si les fichiers `~/.ssh/id_rsa` and `~/.ssh/id_rsa.pub` existent en entrant la commande `ls` .
2. Si ce n'est pas le cas, générez une paire de clés avec la commande suivante (en remplaçant votre email) : `ssh-keygen -t rsa -C "mon email"` .
3. On vous demandera d'entrer l'emplacement où vous voulez stocker votre clé. Vous pouvez entrer le chemin vers le dossier de votre choix, ou cliquer ⏎ pour sauvegarder la clé à la racine.
4. Ensuite on vous demandera d'entrer une passphrase. Celle-ci représente une deuxième couche de sécurité : si une autre personne à accès à votre ordinateur, elle ne pourra pas utiliser la connexion par ssh sans connaître la passphrase. Cette étape n'est pas obligatoire, mais fortement recommandée.

CONFIGURATION

```
(base) kmatrosova@FRLPMC2823 ~ % ssh-keygen -t rsa -C tina.matrossova@gmail.com
Generating public/private rsa key pair.
Enter file in which to save the key (/Users/kmatrosova/.ssh/id_rsa):
Created directory '/Users/kmatrosova/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /Users/kmatrosova/.ssh/id_rsa
Your public key has been saved in /Users/kmatrosova/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:s5M+KX0TPFxPq0yIBA1UwNQzyl/wdxHEZrRBM4CqKQA tina.matrossova@gmail.com
The key's randomart image is:
+---[RSA 3072]----+
|     +=*.  ..*O. |
|E     o * .   == |
| .   . o *   o.. |
|  .   o o o o o  |
|   .   =S= + + . |
|    . o o+* . o  |
|     . .+. = .   |
|      ..+.o o    |
|       ..o .     |
+----[SHA256]-----+
```

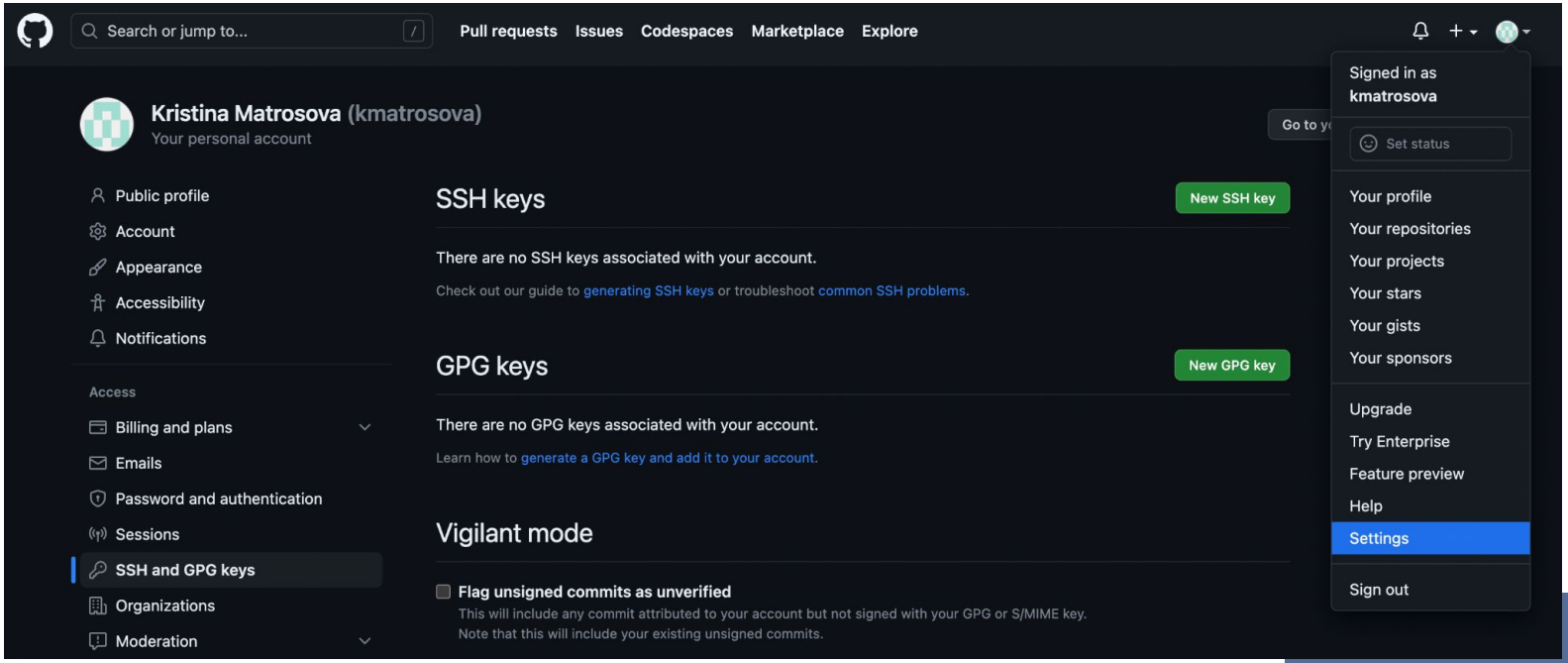
Connexion en SSH

Afficher votre clé publique avec la commande `cat ~/.ssh/id_rsa.pub` .

```
(base) kmatrosova@FRLPMC2823 ~ % cat ~/.ssh/id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABgQCy7j/FjWImzOkEfD02LFEgjVGpkHqS+bFj+Lui8nTMJPhEnVQojyxkkhHg
mkD1eEgy9vAskAnsR8LFYUC3jQQtbOrlZuZ9KVTlPkRFUDjTJbRiTlymN8XrFw2Cg8X1OH57iSG4IQ824Zo28qARam2vX3fW
tYFVUZOLHCHyVCjhAF2pXxVvZtZnALczXhAMbJwW7sXO51sfoLlgd/Byp8zlUXugfYee8ysIjGlO+BOzUvRzvnIJAVK/jtmD
DiVXFMCBVYhKhmpI7RzgM0Q2ZwT/uL18Z29DCqSsnoLLqGigUOGvPmAAgqkrjheqpE+S4ZtzXUqeAHdWN6XDKkBnNp1WP8hN
RNE0gKHPDA3BMNs47heG0glZqtif9HUcxA51iYNjFloUEp4eGOnQSgy6dhWY4McgLCE3OJNXhNDV2j5txX8hlXqN7G14mKHU
WH0nMDNGjoexS06yAgcjl0v46qfBzmgHq58IepeiaI5C6/3fTueMcliW5UjXwNwmAqQOgoM= tina.matrossova@gmail.c
om
```

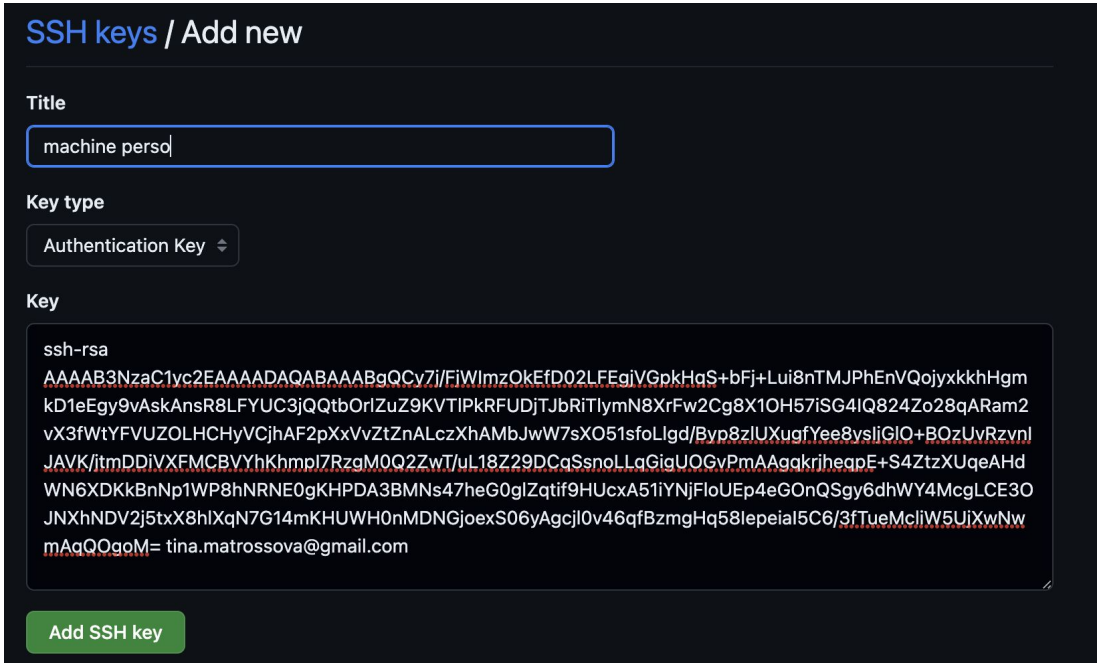
Connexion en SSH

Connectez-vous à GitHub, puis allez dans `Settings → SSH and GPG Keys → New SSH key`.



Connexion en SSH

Donnez un nom à votre clé (“Machine perso” par exemple) et copiez-collez votre clé publique dans le champ dédié.



Connexion en SSH

Tapez la commande `ssh -T git@github.com` puis suivez les instructions (entrez `yes`, puis votre passcode ssh).

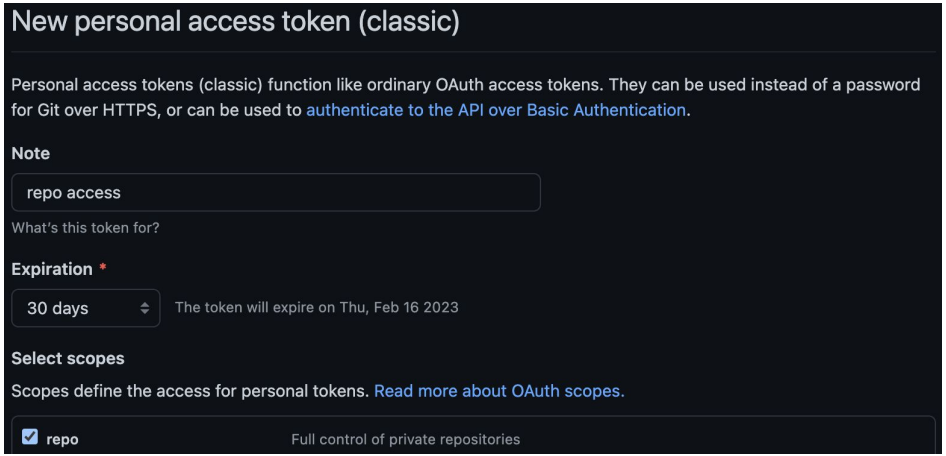
Le message suivant devrait s'afficher :

`Hi username! You've successfully authenticated, but Github does not provide shell access.`

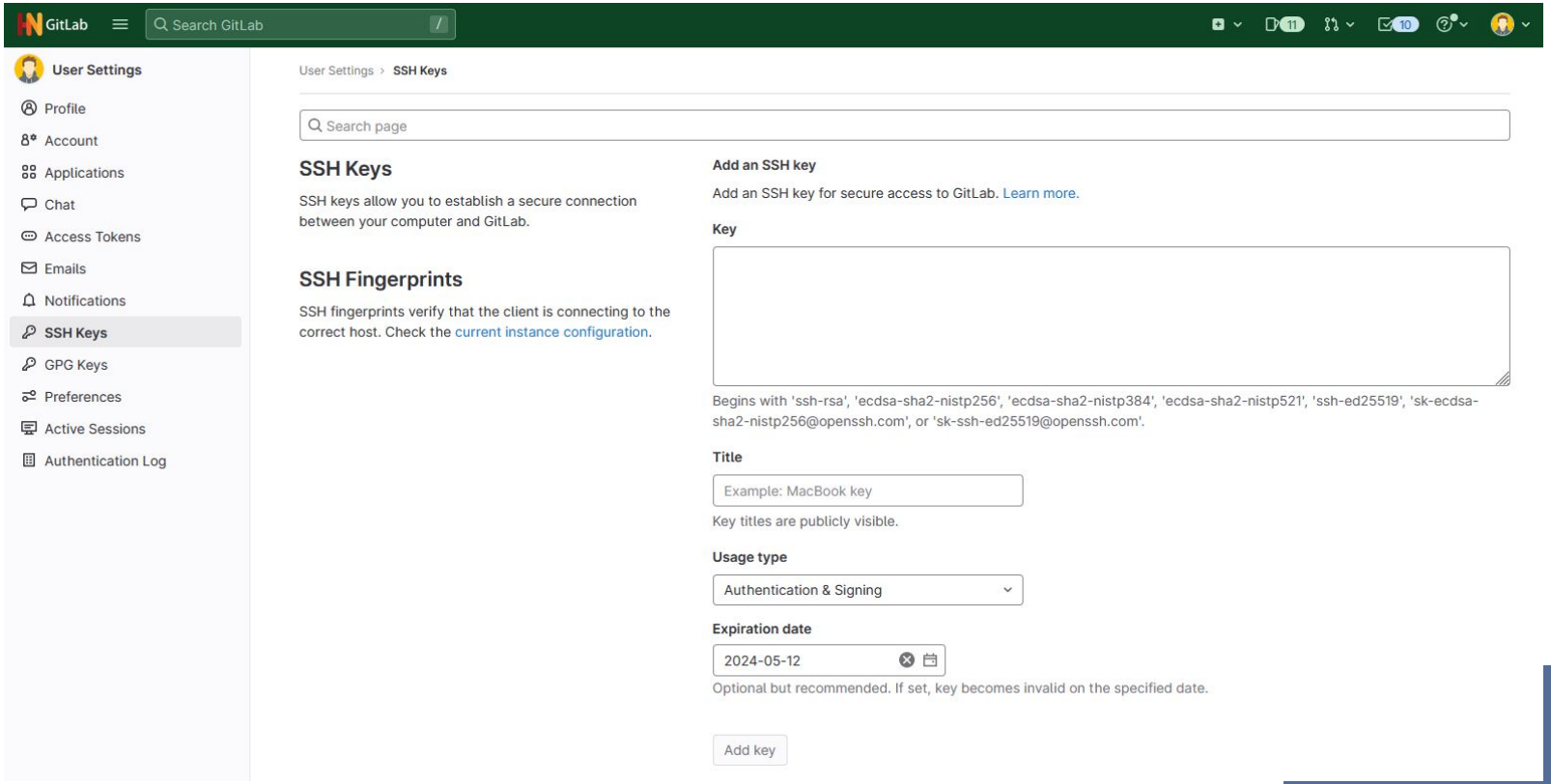
```
(base) kmatrosova@FRLPMC2823 ~ % ssh -T git@github.com
The authenticity of host 'github.com (140.82.121.4)' can't be established.
ED25519 key fingerprint is SHA256:+DiY3wvvV6TuJJhbpZisF/zLDA0zPMSvHdkr4UvCOqU.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'github.com' (ED25519) to the list of known hosts.
Enter passphrase for key '/Users/kmatrosova/.ssh/id_rsa':
Hi kmatrosova! You've successfully authenticated, but GitHub does not provide shell access.
```

Connexion avec un token HTTPS

1. Connectez vous à GitHub, puis allez dans `Settings → Developer settings → Tokens (classic)`
2. Cliquez sur `Generate new token (classic)`
3. Donnez un `nom` à votre token, sélectionnez une `durée de validité` et cochez les cases correspondantes au `cadre d'utilisation` du token. Dans ce cours, nous allons uniquement faire de la gestion de repository, donc nous avons besoin de cocher la case `repo`.
4. Copiez ensuite le token généré et sauvegardez-le dans un lieu sûr. Vous pouvez désormais l'utiliser comme mot de passe lorsqu'il vous sera demandé en ligne de commande sur votre machine.



GitLab - Connexion en SSH

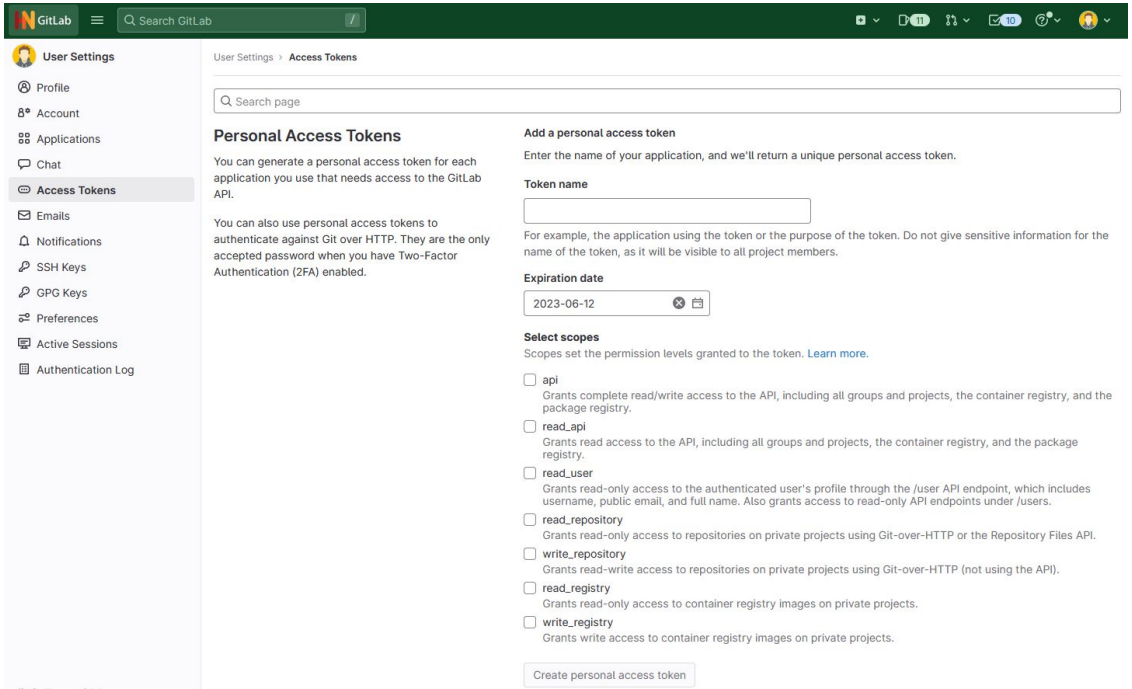


CONFIGURATION

GitLab - Connexion en SSH

Tapez la commande `ssh -T git@gitlab.huma-num.fr` puis suivez les instructions...

GitLab - Connexion avec un token HTTPS



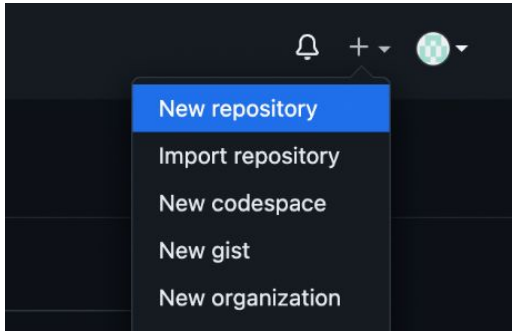
NOUVEAU PROJET

04

EXEMPLE D'UTILISATION

Créer un projet depuis GitHub

Nous allons maintenant créer un nouveau projet. Allez dans `New repository` dans le menu déroulant `+` en haut à droite.

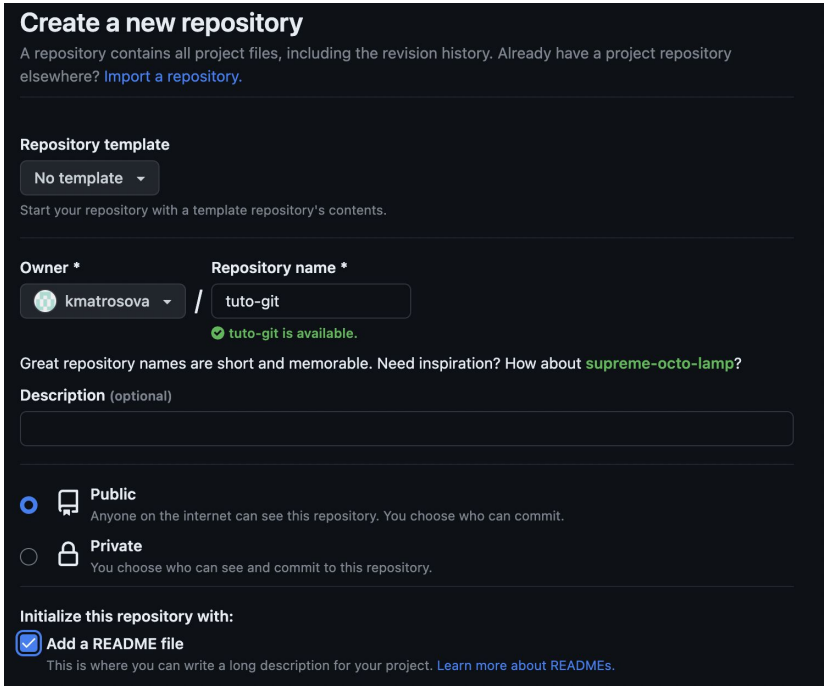


EXEMPLE D'UTILISATION

Créer un projet depuis GitHub

Nous allons appeler notre projet `tuto-git`. Ce projet sera `publique` (visible par tous les utilisateurs).

Cochez la case `Add a README file` - nous allons générer un fichier qui servira de descriptif de notre projet. On pourra y mettre des informations utiles à toute personne qui serait intéressée par notre projet.



EXEMPLE D'UTILISATION

Créer un projet depuis GitHub

Une fois le projet créé, allez dans votre terminal. Déplacez vous dans le dossier de votre choix (Documents par exemple) avec la commande `cd`, puis tapez `git clone git@github.com:username/tuto-git` (en remplaçant username par le votre). Tapez `ls` (`dir` sous Windows) - un dossier tuto-git est apparu sur votre machine. Si vous tapez `ls tuto-git` (`dir tuto-git` sous Windows) vous verrez que le fichier README est bien dans le dossier.

```
(base) kmatrosova@FRLPMC2823 ~ % cd Documents
(base) kmatrosova@FRLPMC2823 Documents % git clone git@github.com:kmatrosova/tuto-git
Cloning into 'tuto-git'...
Enter passphrase for key '/Users/kmatrosova/.ssh/id_rsa':
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (3/3), done.
(base) kmatrosova@FRLPMC2823 Documents % ls
1MMMM           Docs            PhD             Zoom            tuto-git
(base) kmatrosova@FRLPMC2823 Documents % cd tuto-git
(base) kmatrosova@FRLPMC2823 tuto-git %
```

EXEMPLE D'UTILISATION

Faire des modifications en local

Notre projet contiendra un fichier texte `test.txt`. Il contiendra la phrase `“Ceci est un fichier test.”`. Créez ce fichier dans le repository tuto-git sur votre machine.

EXEMPLE D'UTILISATION

Soumettre les modifications

Nous venons de modifier le projet en ajoutant un fichier et voulons maintenant soumettre ces modifications.

Ajoutez le fichier créé avec `git add test.txt` (utiliser `git add -A` s'il y à plusieurs fichiers à ajouter).

Si vous voulez systématiquement ajouter tous les nouveaux fichiers, sauf quelques exceptions, vous pouvez créer un fichier `.gitignore`. Cela peut-être le cas par exemple si vous mettez un dataset dans votre dossier pour tester votre code en local, mais que vous n'avez pas envie de publier le dataset sur GitHub. Les fichiers qui commencent par un point sont des fichiers cachés, pour les voir vous devez entrer la commande `ls -a`, le `-a` veut dire qu'on veut voir tous les fichiers qu'il y a dans le dossier actuel. Vous pouvez créer votre fichier `.gitignore` dans n'importe quel éditeur de texte, et vous pouvez y mettre les noms des fichiers/dossiers concernés (un par ligne).

EXEMPLE D'UTILISATION

Exemple d'utilisation d'un fichier .gitignore

Par exemple, vous avez plusieurs fichiers csv qui contiennent des données sensibles. Vous pouvez les ignorer en écrivant `*.csv`. Le symbole `*` signifie qu'on veut exclure tous les fichiers.

Si les données sont toutes contenues dans le dossier nommé data, vous pouvez les exclure avec `data/*`.
Vous pouvez être plus précis en indiquant le dossier et l'extension en même temps comme ceci : `data/*.csv`.

Supposons que vous voulez exclure tous les fichiers du même type sauf un, qui s'appelle par exemple `special_data.csv`. Vous pouvez d'abord mettre `*.csv`, puis à la ligne `!special_data.csv`. Le symbole `!` désigne la négation.

Vous pouvez également ajouter des commentaires, en commençant la ligne par `#`.

Ce type de syntaxe repose sur le principe des **expressions régulières**. Voilà un bon tuto si vous voulez en savoir plus : https://www.lucaswillems.com/fr/articles/25/tutoriel-pour-maitriser-les-expressions-regulieres

EXEMPLE D'UTILISATION

Exemple d'utilisation d'un fichier .gitignore

Voilà un exemple de fichier .gitignore.

```
# Fichiers à ignorer :  ← commentaire
requirements.txt        ← le fichier requirements.txt
*.log                   ← tous les fichiers d'extension .log contenus à la racine du projet
log/*.log               ← tous les fichiers d'extension .log contenus dans le dossier log
data/*                  ← tous les fichiers contenus dans le dossier data
!data/subset.csv        ← sauf le fichier subset.csv
```

EXEMPLE D'UTILISATION

Soumettre les modifications

Soumettez les modifications avec `git commit -m "add test.txt"` ("add test.txt" ici est le nom du commit). Il est préférable de choisir un nom cohérent avec les modifications faites afin de pouvoir facilement retracer les changements dans le futur et car il s'agit également d'un nom que tout le monde verra si le dépôt est public.

Prenez l'habitude de faire un `commit` à chaque fois que vous avez fini de travailler sur une partie de votre code, c'est un bon moyen d'organiser son travail et vous pourrez facilement retracer l'historique des modifications dans le futur. C'est d'autant plus important d'adopter cette habitude dans les projets collaboratifs.

```
(base) kmatrosova@FRLPMC2823 tuto-git % git commit -m "add test.txt"
[main bc0977b] add test.txt
 1 file changed, 1 insertion(+)
 create mode 100644 test.txt
```

EXEMPLE D'UTILISATION

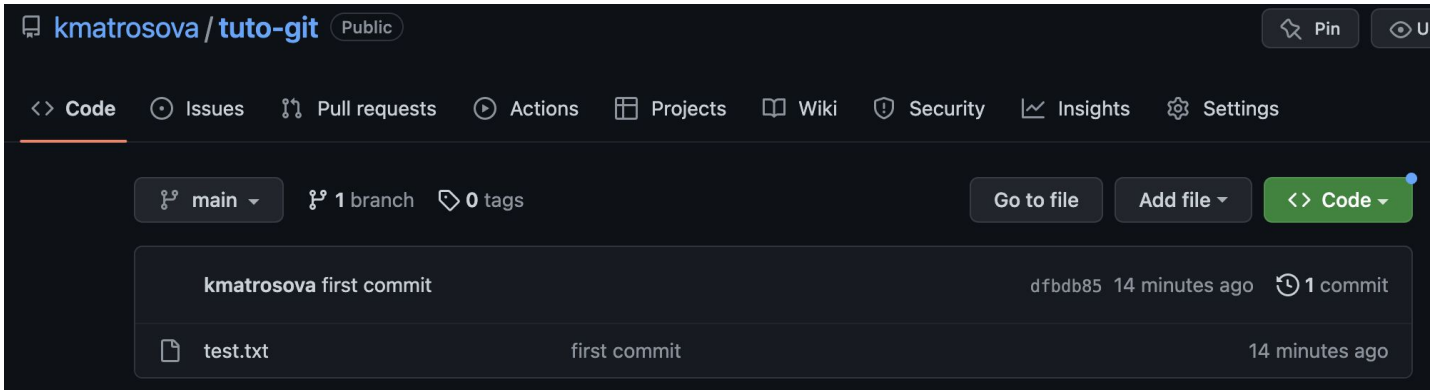
Soumettre les modifications

Appliquez les modifications avec `git push`. Notez que si vous avez choisi l'authentification par token, il faudra le rentrer en tant que mot de passé à cette étape, précédé de votre username.

```
(base) kmatrosova@FRLPMC2823 tuto-git % git push
Enter passphrase for key '/Users/kmatrosova/.ssh/id_rsa':
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 301 bytes | 150.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To github.com:kmatrosova/tuto-git
   de1e1fd..bc0977b  main -> main
(base) kmatrosova@FRLPMC2823 tuto-git %
```

EXEMPLE D'UTILISATION

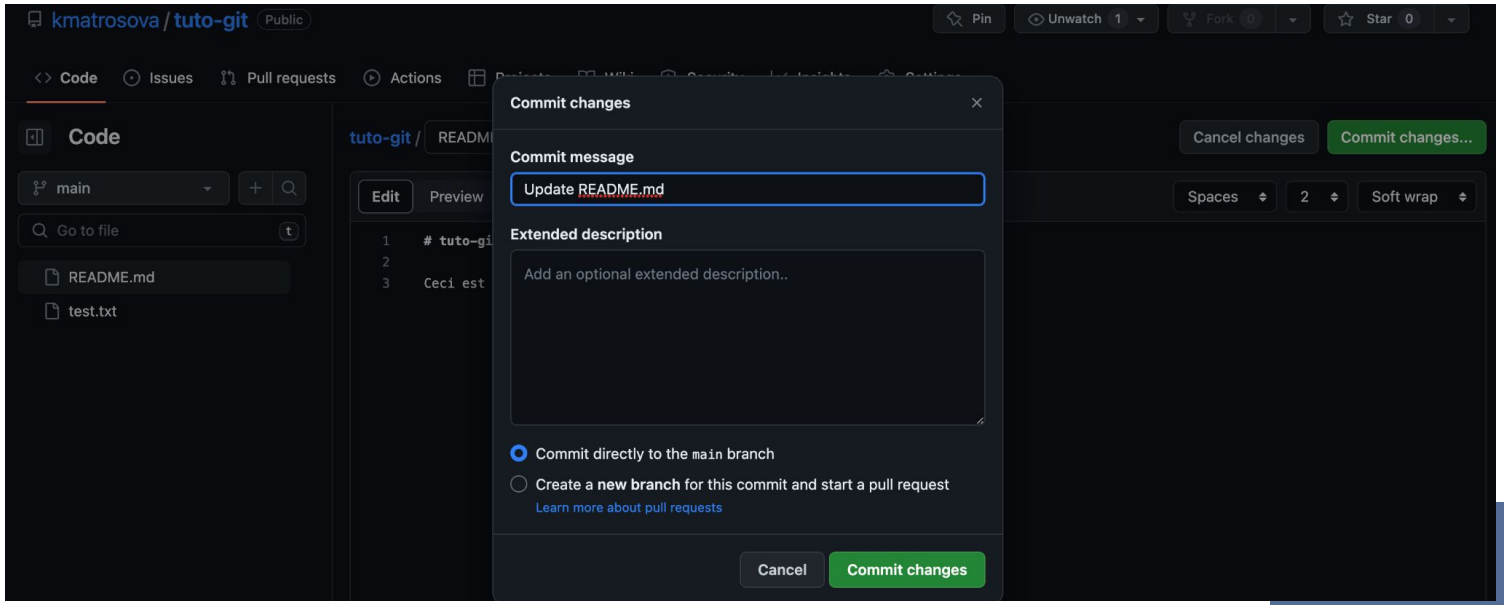
Vous pouvez voir que le fichier `test.txt` est apparu dans le projet sur GitHub.



EXEMPLE D'UTILISATION

Faire des modifications depuis GitHub

Vous avez également la possibilité de modifier des fichiers directement sur GitHub. Allez dans le fichier README et ajoutez la phrase “Ceci est un projet test git.”



EXEMPLE D'UTILISATION

Mettre à jour le projet en local

Pour que ces modifications soient prises en compte sur votre machine locale, entrez la commande `git pull` dans votre terminal.

En général, cette commande est essentielle pour s'assurer que le projet est à jour sur votre machine. Il est recommandé de l'utiliser régulièrement, par exemple une fois par jour avant de procéder à de nouvelles modifications sur le projet.

```
(base) kmatrosova@FRLPMC2823 tuto-git % git pull
Enter passphrase for key '/Users/kmatrosova/.ssh/id_rsa':
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 699 bytes | 174.00 KiB/s, done.
From github.com:kmatrosova/tuto-git
   bc0977b..3da033f  main       -> origin/main
Updating bc0977b..3da033f
Fast-forward
 README.md | 4 +++-
 1 file changed, 3 insertions(+), 1 deletion(-)
(base) kmatrosova@FRLPMC2823 tuto-git %
```

GITHUB DESKTOP

05

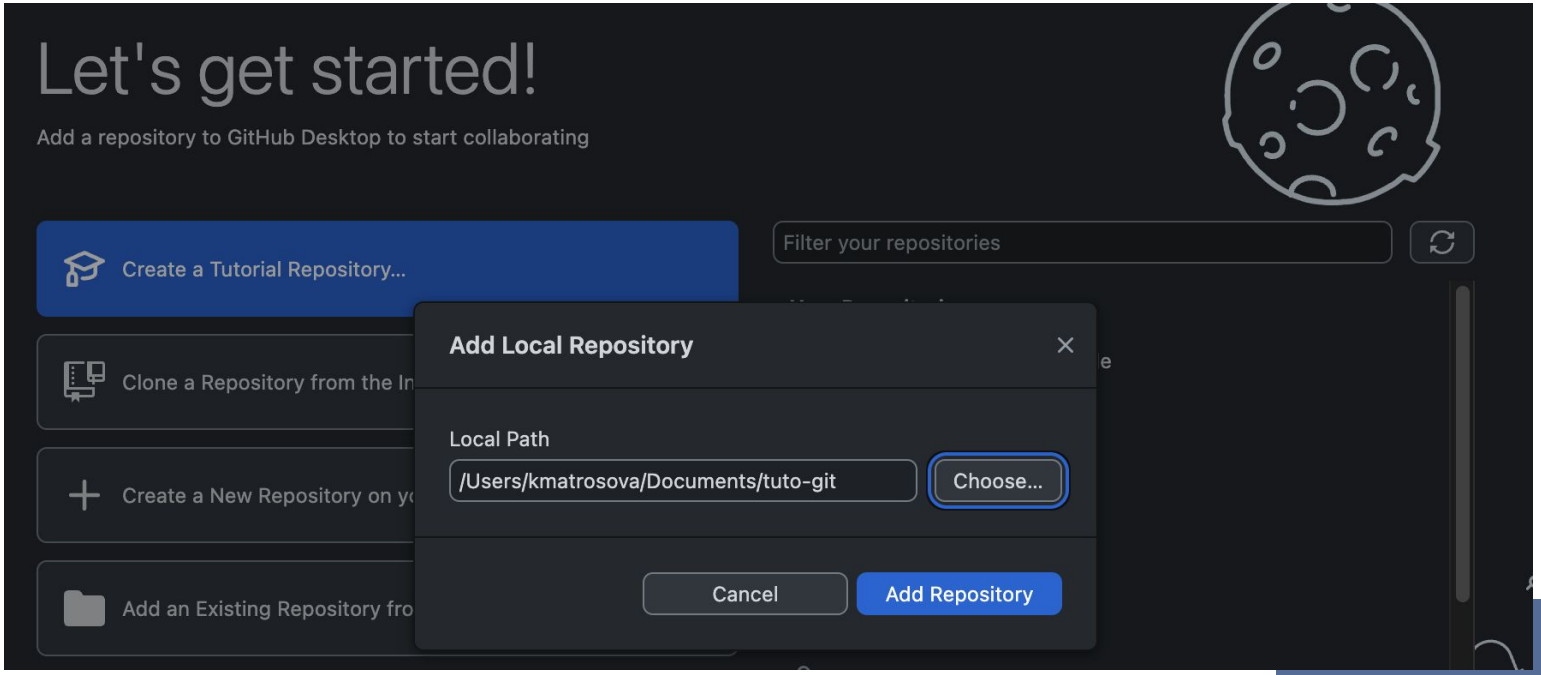
GITHUB DESKTOP

GitHub propose également une interface graphique GitHub Desktop, disponible sous Windows, Mac et Linux.

Vous pouvez la télécharger ici https://desktop.github.com/. Installez en suivant les instructions.

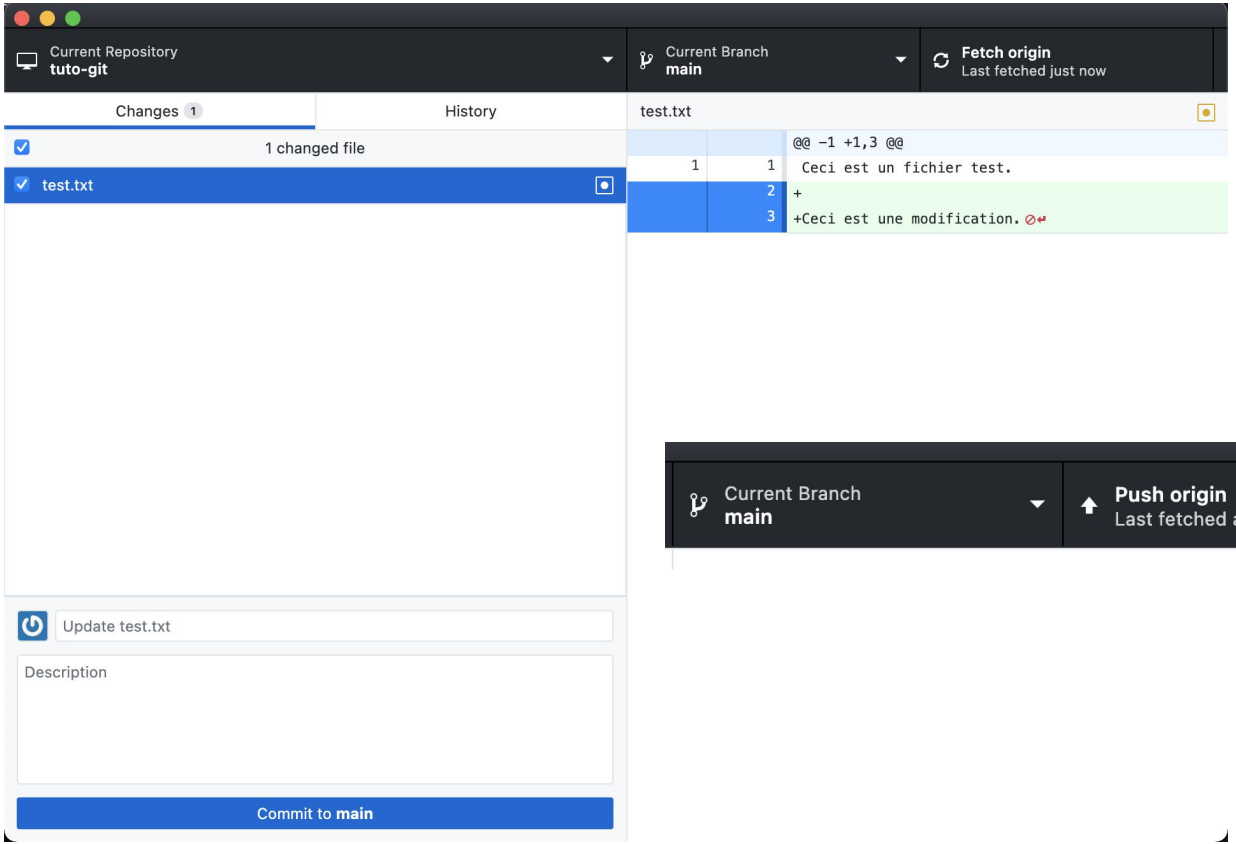
GITHUB DESKTOP

Lancez GitHub Desktop et ajoutez le projet en cliquant sur Add → Add Existing Repository.



GITHUB DESKTOP

Ajoutez une phrase au fichier test.txt. Vous allez voir que le changement apparaît sur l'interface. Vous pouvez désormais cliquer sur Commit to main (n'oubliez pas de donner un nom qui fait du sens à votre commit), puis cliquez sur Push origin. Cliquez sur Fetch origin pour récupérer la dernière version du projet.



AJOUTER UN PROJET EXISTANT

AJOUTER UN PROJET EXISTANT

Initialiser un projet depuis votre machine

Il est possible que vous ayez déjà des projets en cours sur votre machine pour lesquels vous aurez envie d'utiliser Git. Vous pouvez alors suivre les étapes suivantes :

1. Créez un repository vide sur GitHub (ne l'initialisez pas avec un README et/ou autres fichiers pour éviter les conflits).

2. Sur votre machine, allez dans le dossier du projet et entrez les commandes suivantes
    a. `git init` (initialiser le dossier avec les fichiers de configuration git)
    b. `git add -A`
    c. `git commit -m "first commit"`
    d. `git branch -M main`
    e. `git remote add origin git@github.com:username/projectname.git` (en changeant username et projectname par votre identifiant GitHub et le nom du repository sur GitHub)
    f. `git push --set-upstream origin main`

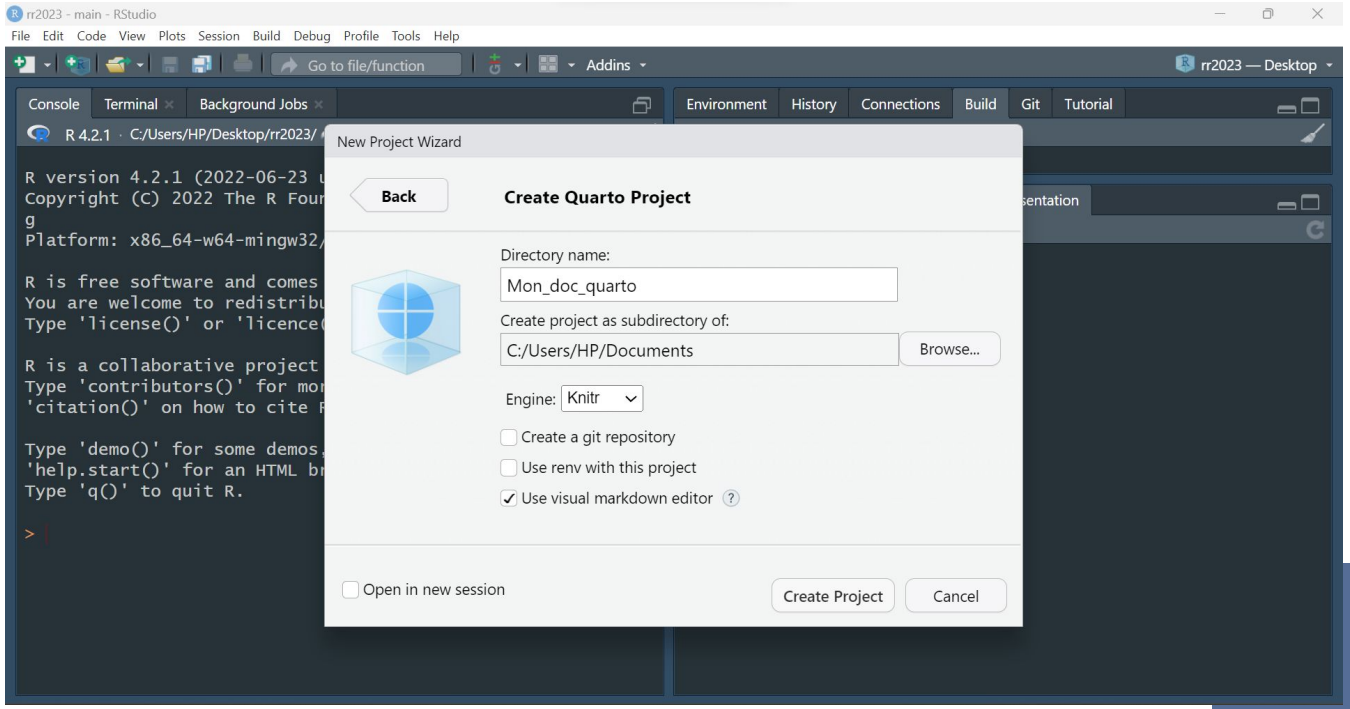
    Dans le cas de l'erreur `remote origin already exists`, utilisez la commande `git remote remove origin`, puis reprenez à l'étape e.

3. Vérifiez que vos fichiers sont apparus dans le repository sur GitHub. Vous pouvez désormais utiliser les commandes git habituelles.

EXEMPLE AVEC UN PROJET RSTUDIO

Exemple pratique

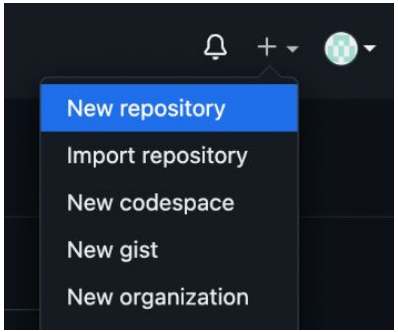
Créez projet Quarto avec Rstudio : File → New Project → New Directory → Quarto Project

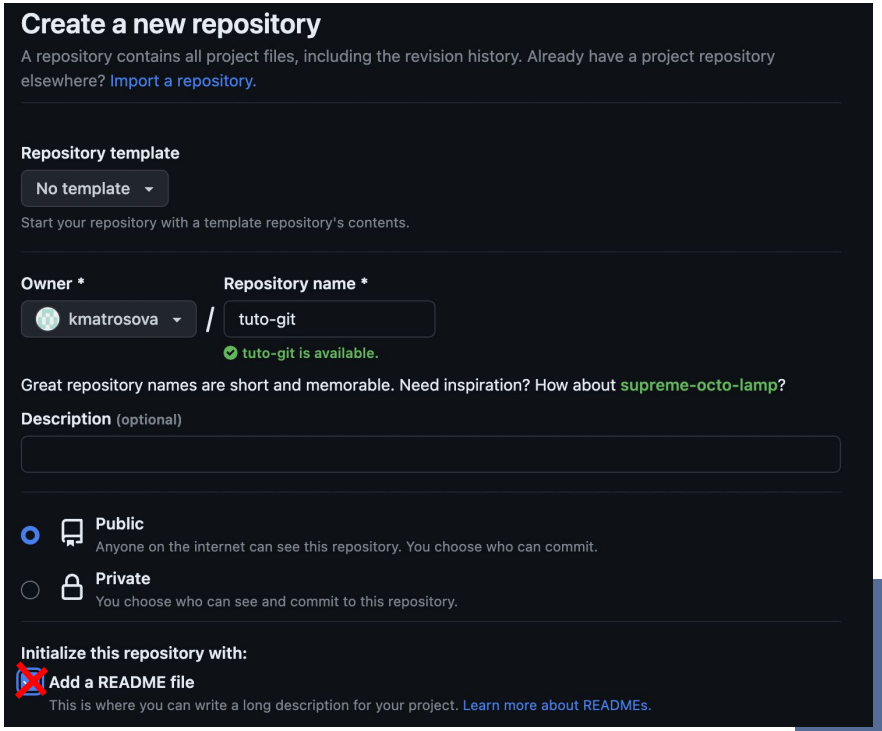


EXEMPLE AVEC UN PROJET RSTUDIO

Exemple pratique

Créez un nouveau dépôt sur GitHub.





EXEMPLE AVEC UN PROJET RSTUDIO

Exemple pratique

Ajouter le projet (quarto) existant dans le nouveau répertoire créé :

- Allez dans le dossier du projet. Exemple : `cd .\Documents\Mon_doc_quarto\`
- Entrez les commandes suivantes :

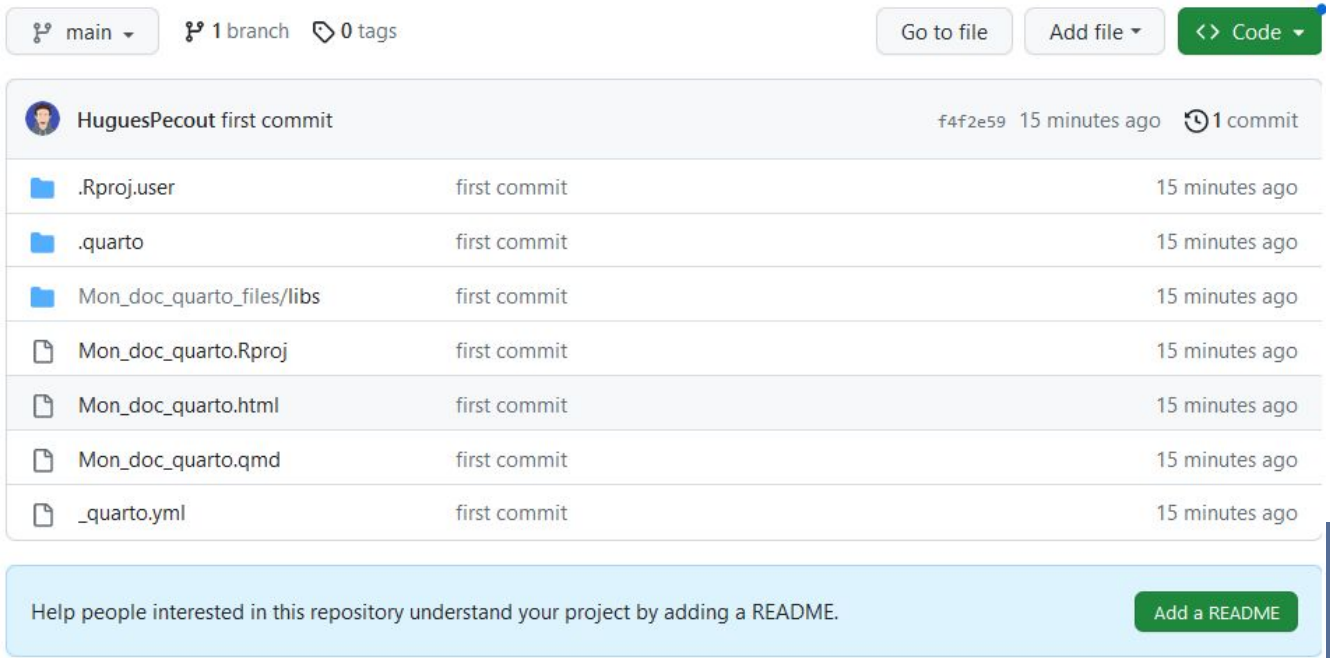
    a. `git init` (initialiser le dossier avec les fichiers de configuration git)
    b. `git add -A`
    c. `git commit -m "first commit"`
    d. `git branch -M main`
    e. `git remote add origin git@github.com:username/projectname.git`
    f. `git push --set-upstream origin main`

Dans le cas de l'erreur `remote origin already exists`, utilisez la commande `git remote remove origin`, puis reprenez à l'étape e.

EXEMPLE AVEC UN PROJET RSTUDIO

Exemple pratique

Vérifiez que vos fichiers sont apparus dans le repository sur GitHub.



(SERVIR) DÉPLOYER UN FICHIER HTML

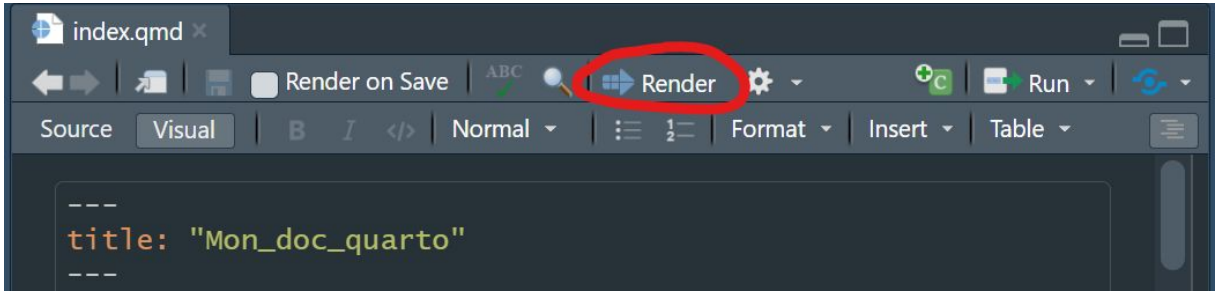
07

DÉPLOYER UN FICHIER HTML

Exemple Pratique

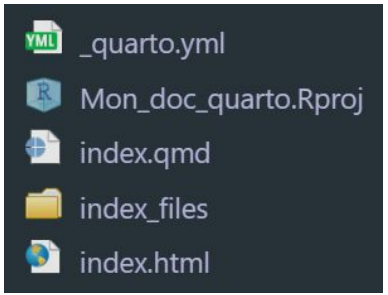
Il existe plusieurs façons pour déployer un fichier HTML via GitHub (ou GitLab). **Nous présentons ici la solution la plus simple et basique**.

1. Commencez par renommer votre fichier quarto `index.qmd`.
2. Compiler le notebook quarto en format HTML en cliquant sur `Render`



Le document Quarto en format HTM s'affiche dans le `Viewer` et le fichier `index.html` a été enregistré à la racine du répertoire projet.

Le fichier HTML à déployer doit obligatoirement se nommer `index.html`

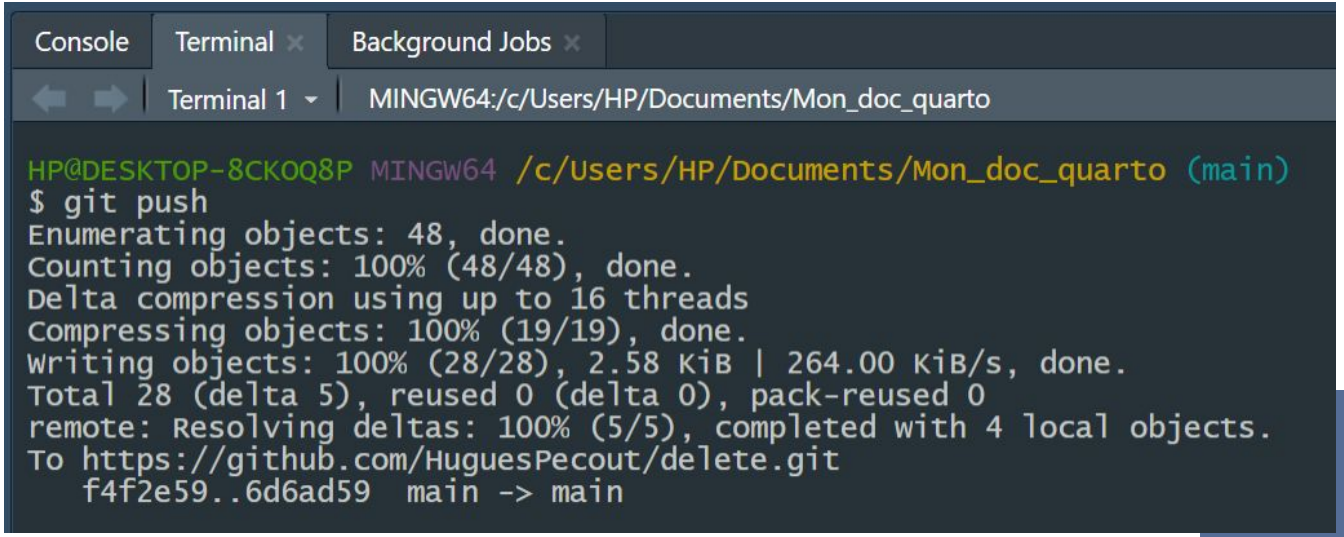


DÉPLOYER UN FICHIER HTML

Exemple Pratique

Ajouter ces nouveaux fichiers (et les modifications) sur le dépôt GitHub :

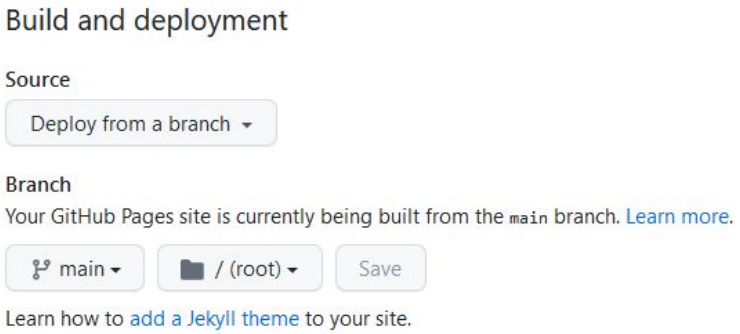
a. `git add -A`
b. `git commit -m "add render result of my quarto doc"`
c. `git push`



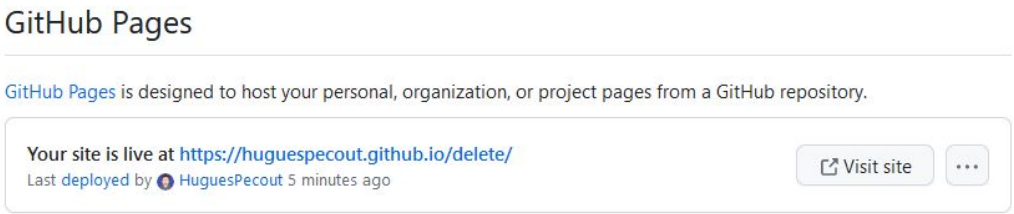
DÉPLOYER UN FICHIER HTML

Exemple Pratique

1. Sur la page du dépôt GitHub, allez dans `Settings → pages`, sélectionnez la branche `main`, enregistrez.

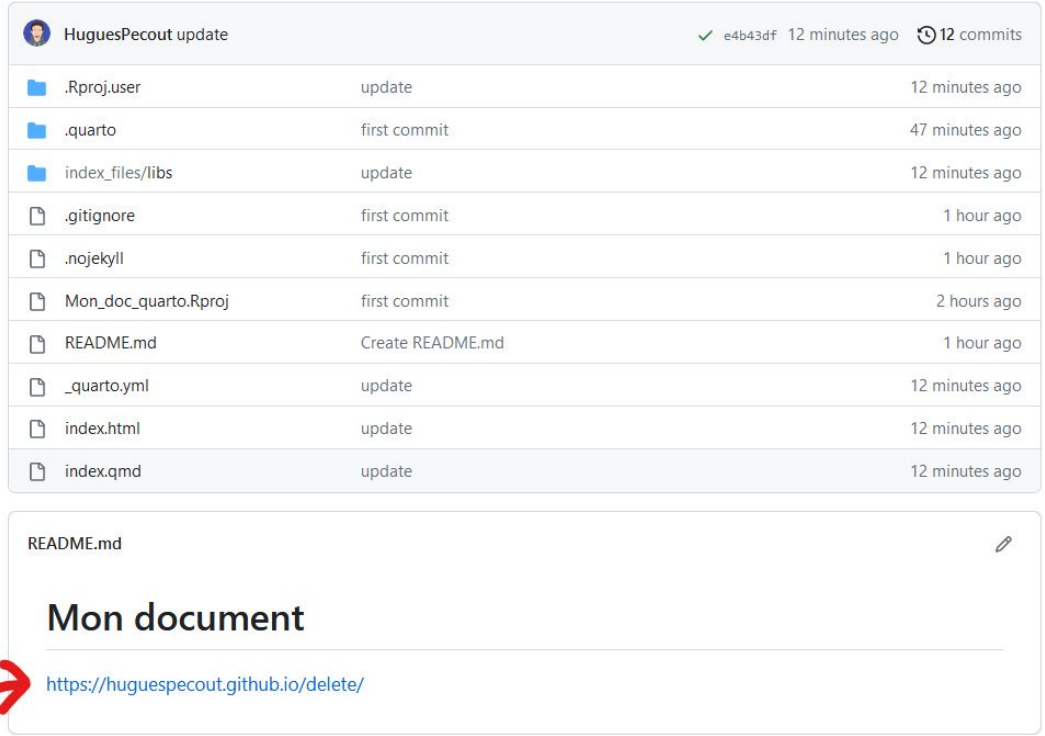


2. Rafraichissez la page quelques instants après... Votre page est en ligne !



Exemple Pratique

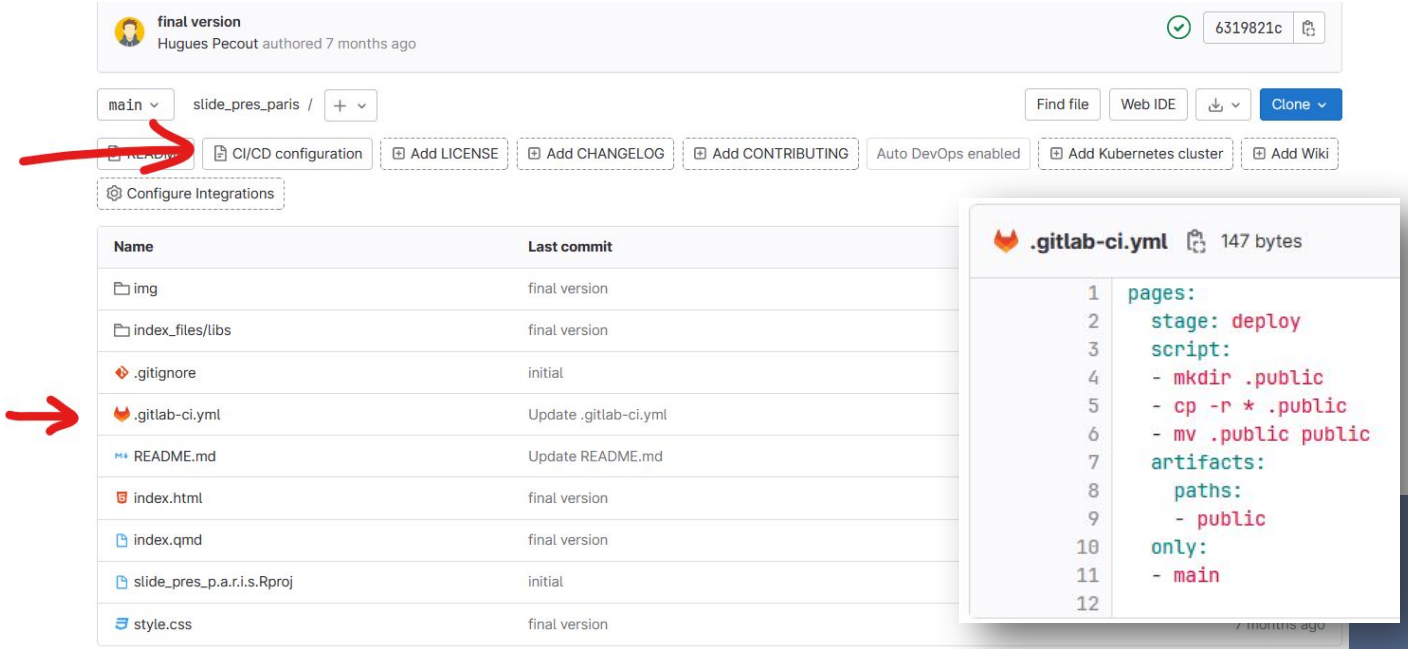
Vous pouvez ajouter le lien de la page déployée par GitHub en ajoutant l'URL dans le fichier README.md pour faciliter l'accès depuis le dépôt du code source du projet.



DÉPLOYER UN FICHIER HTML

Déployer un HTML avec GitLab

La méthode de déploiement de documents HTML avec GitLab est différente. Il suffit d'ajouter un fichier appelé `.gitlab-ci.yml` à la racine de votre dépôt, qui contient la configuration de l'intégration (ou déploiement continu, CI/CD). Exemple :



INTÉGRATION CONTINUE ET COLLECTIVE

DÉPLOYER UN FICHIER HTML

Intégration continue collective - Mise en pratique

Nous allons apporter, tour à tour, des modifications à un document Quarto déployé via Github.

Pour cela, **les droits de modifications d'un dépôt contenant un document Quarto vous ont été ouverts**.

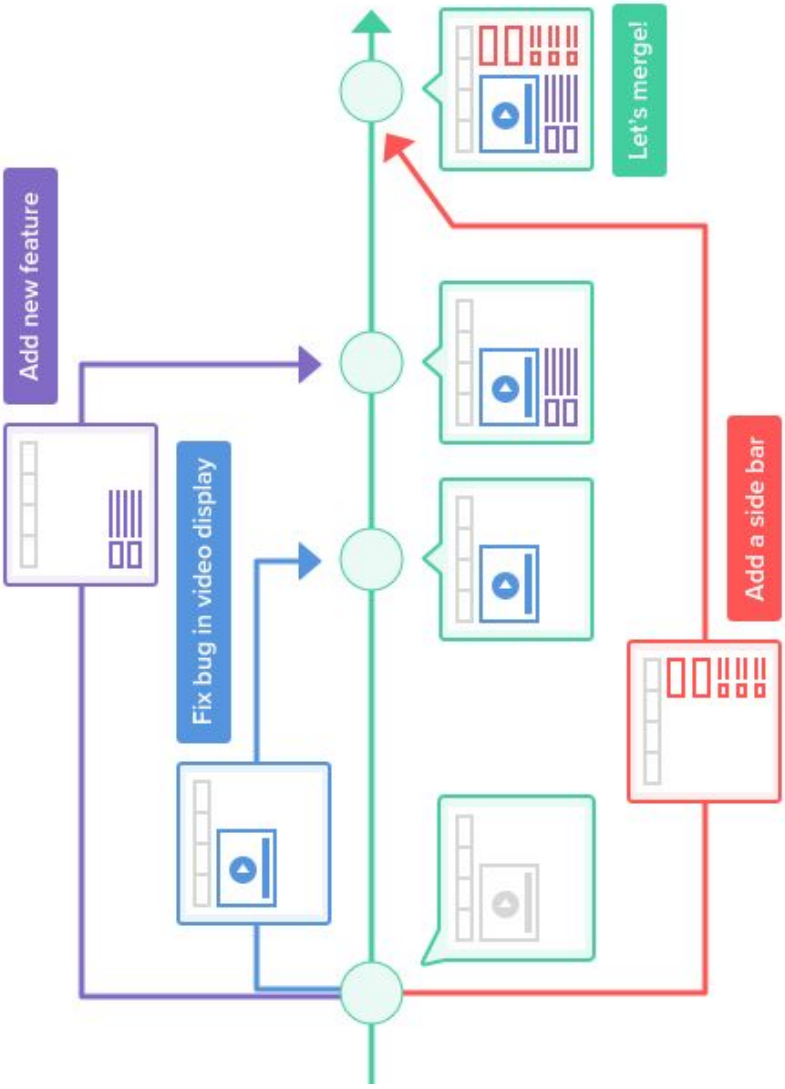
1. Cloner ce dépôt https://github.com/HuguesPecout/delete sur votre machine :
    a. `cd Documents`
    b. `git clone git@github.com:HuguesPecout/delete.git`

2. Chaque participant.es, à tour de rôle :
    a. Si des modifications ont eu lieu sur le dépôt depuis votre clone : `git pull`
    b. Apportez une modification au fichier `index.qmd`
    c. `Render` le fichier index.qmd
    d. `git add -A`
    e. `git commit -m "modification du titre"`
    f. `git push`

Les modifications sont rapidement répercutées sur le fichier HTML déployé en ligne.

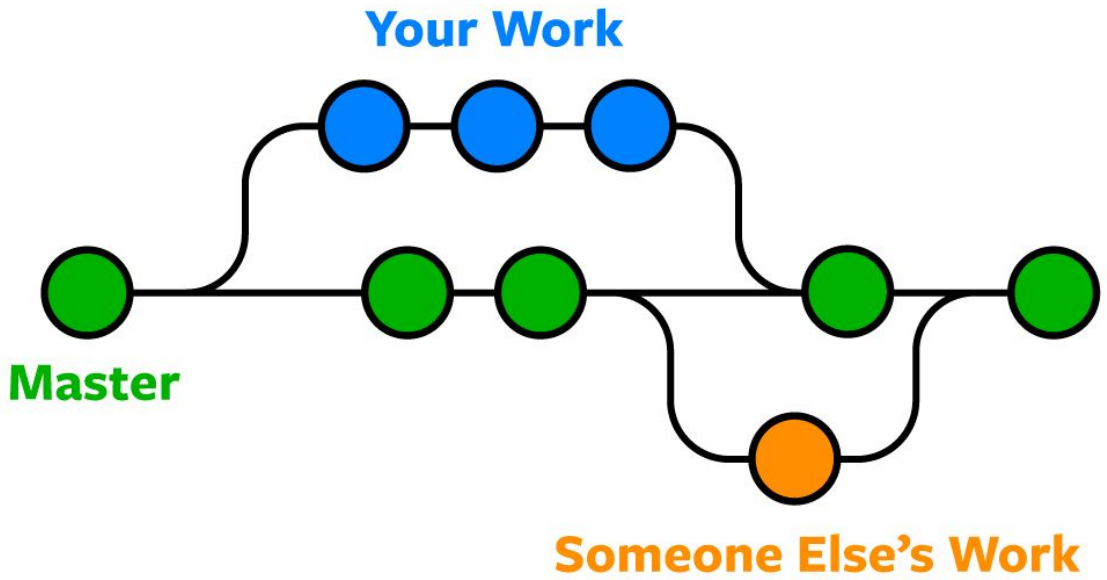
INTRODUCTION AU TRAVAIL COLLABORATIF

INTRODUCTION AU TRAVAIL COLLABORATIF



Branches

Créer une branche signifie diverger de la ligne principale de développement et continuer à travailler sans impacter cette ligne. Cela se fait lors d'un travail en groupe, mais il est aussi possible de créer sa propre branche sur n'importe quel repository publique, afin de l'utiliser pour ses propres besoins, ou bien pour apporter une contribution au projet initial.



INTRODUCTION AU TRAVAIL COLLABORATIF

Une PullRequest - PR est une demande de relecture de code par un pair développeur avant le merge d'une branche sur une autre.

On appelle merge le fait de réunir le code de deux branches sur une unique branche.

Lorsque deux personnes apportent des modifications sur le même fichier dans deux branches différentes, puis essayent de les merge, cela peut créer un conflit.