



Git et Github

GIT, UN LOGICIEL DE GESTION DE VERSIONS DÉCENTRALISÉ
GITHUB, UN SERVICE WEB D'HÉBERGEMENT ET DE GESTION DE DEVELOPPEMENT GIT

FORMATION DISPENSÉE À L'INSTITUT HISTORIQUE ALLEMAND PAR HIPPOLYTE SOUVAY

LE 9 FÉVRIER 2022

Introduction : Git

Git est un logiciel de gestion de versions décentralisé.

Il s'agit d'un logiciel libre créé par Linus Torvalds (créateur du noyau Linux) en 2005.

Il a été pensé pour pouvoir fonctionner de manière décentralisé : il fonctionne en [peer-2-peer](#).

Git indexe les fichiers d'après leur somme de contrôle.

Une [somme de contrôle](#), c'est une séquence courte de données numériques calculée à partir d'un bloc de données plus important.

Si la somme de contrôle est différente, deux versions du fichier sont stockées.

À l'origine, git s'utilisait uniquement en ligne de commande.

Dépôt local et dépôt distant

Les dépôts locaux sont des images du dépôt distant. Le plus souvent, c'est sur ces dépôts locaux qu'on travaille.

On peut mettre à jour son dépôt local depuis un dépôt distant, et inversement.

Cela permet à plusieurs individus de travailler ensemble sur un même document tout en sachant qui a fait quoi.

Bien sûr cela sous-entend d'être discipliné et rigoureux car il faut veiller à mettre à jour son dépôt local à chaque nouvelle session de travail et mettre à jour le dépôt distant lorsqu'on a effectué des modifications.

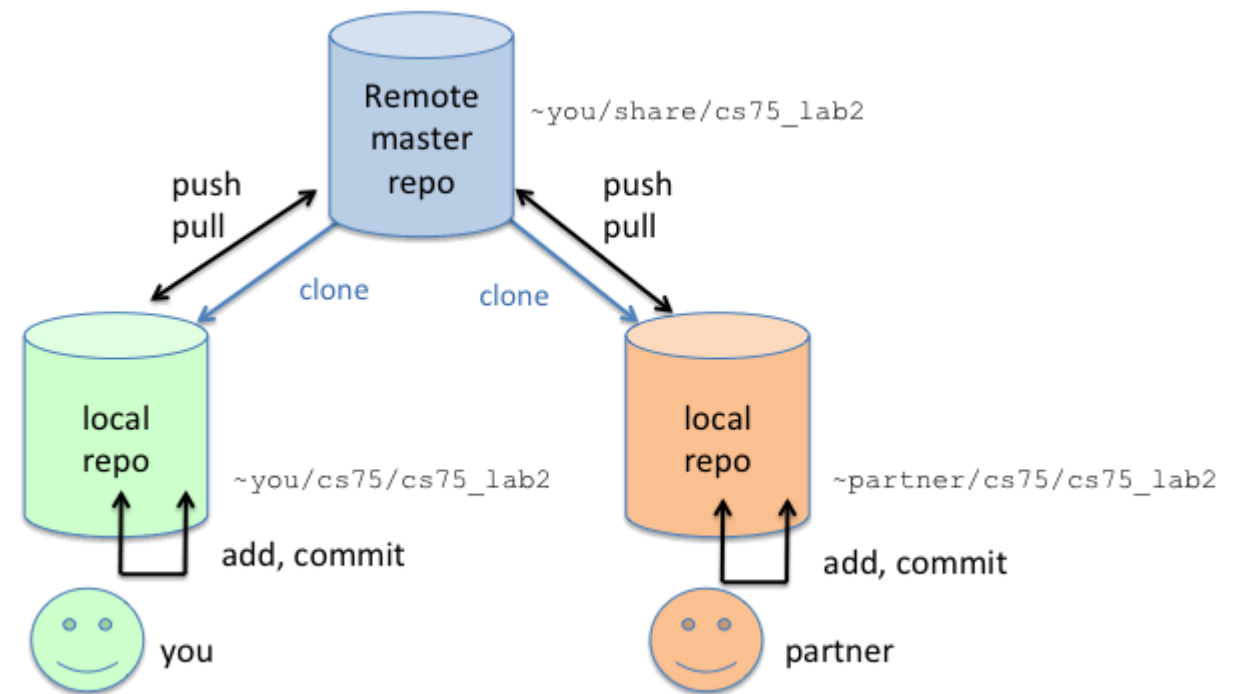


Figure 3 : Dépôt distant et dépôts locaux. Source : <https://web.cs.swarthmore.edu/~adanner/cs40/f14/git.php>

Quelles problématiques Git permet-il de résoudre ?

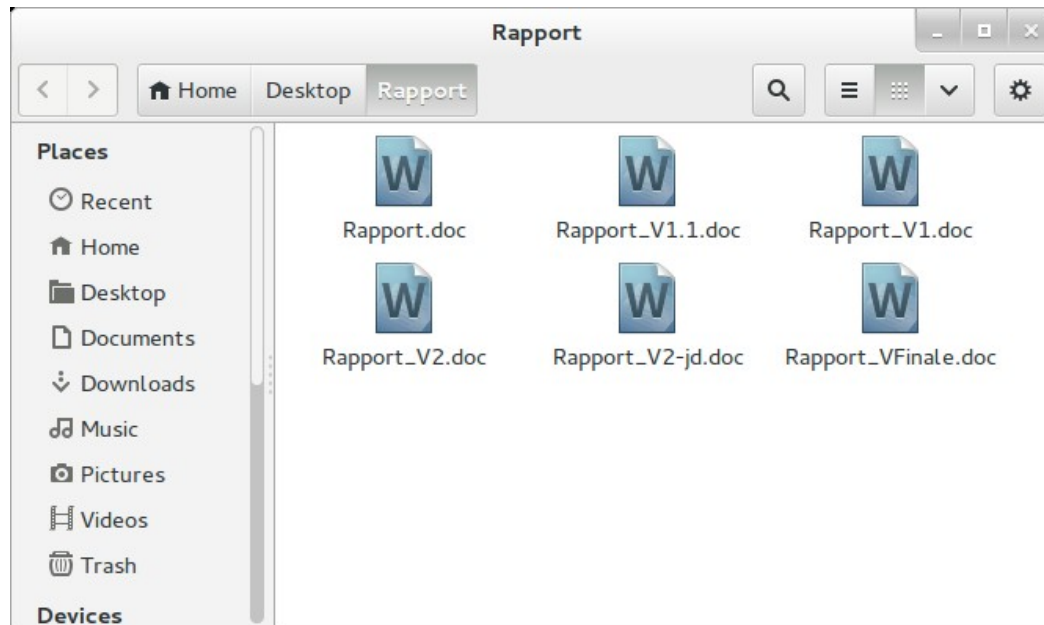


Figure 1 : Problème du versionnage. Source : <https://perso.liris.cnrs.fr/pierre-antoine.champin/enseignement/intro-git/>

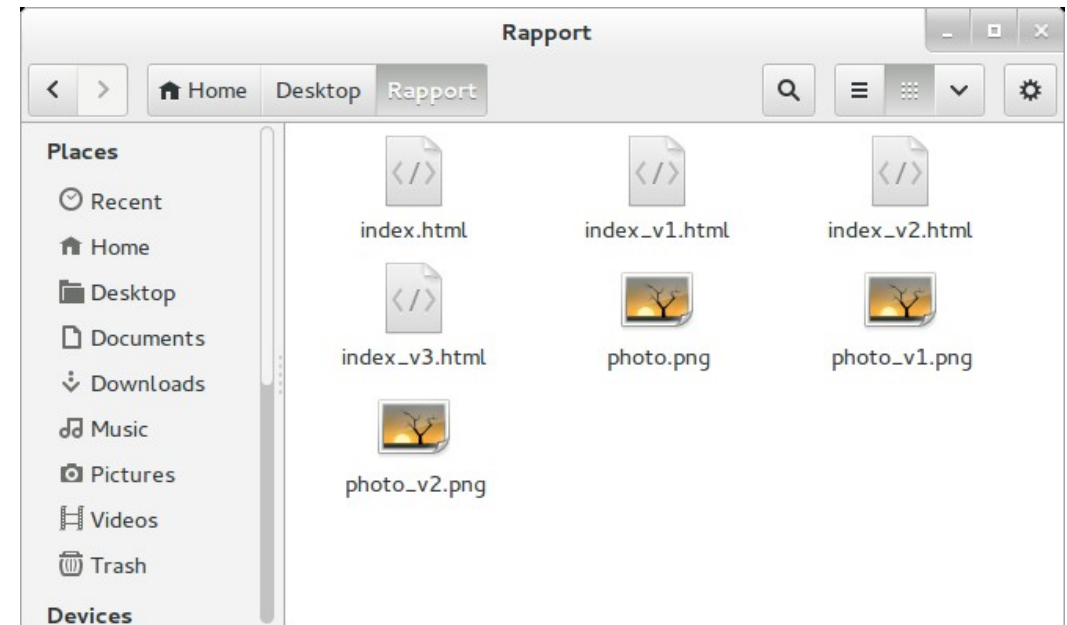


Figure 2 : Problème du versionnage. Source : <https://perso.liris.cnrs.fr/pierre-antoine.champin/enseignement/intro-git/>

Les trois tapes de traitement

Trois étapes de traitement sur le dépôt local :

- *Working directory* : copie de travail. On travaille sur nos fichiers.
- *Staging area* : index. On ajoute les fichiers souhaités à ce qui va être archivé.
- *Git repository* : dépôt. On archive.

On peut ensuite publier nos modifications effectuées en local sur le dépôt distant.

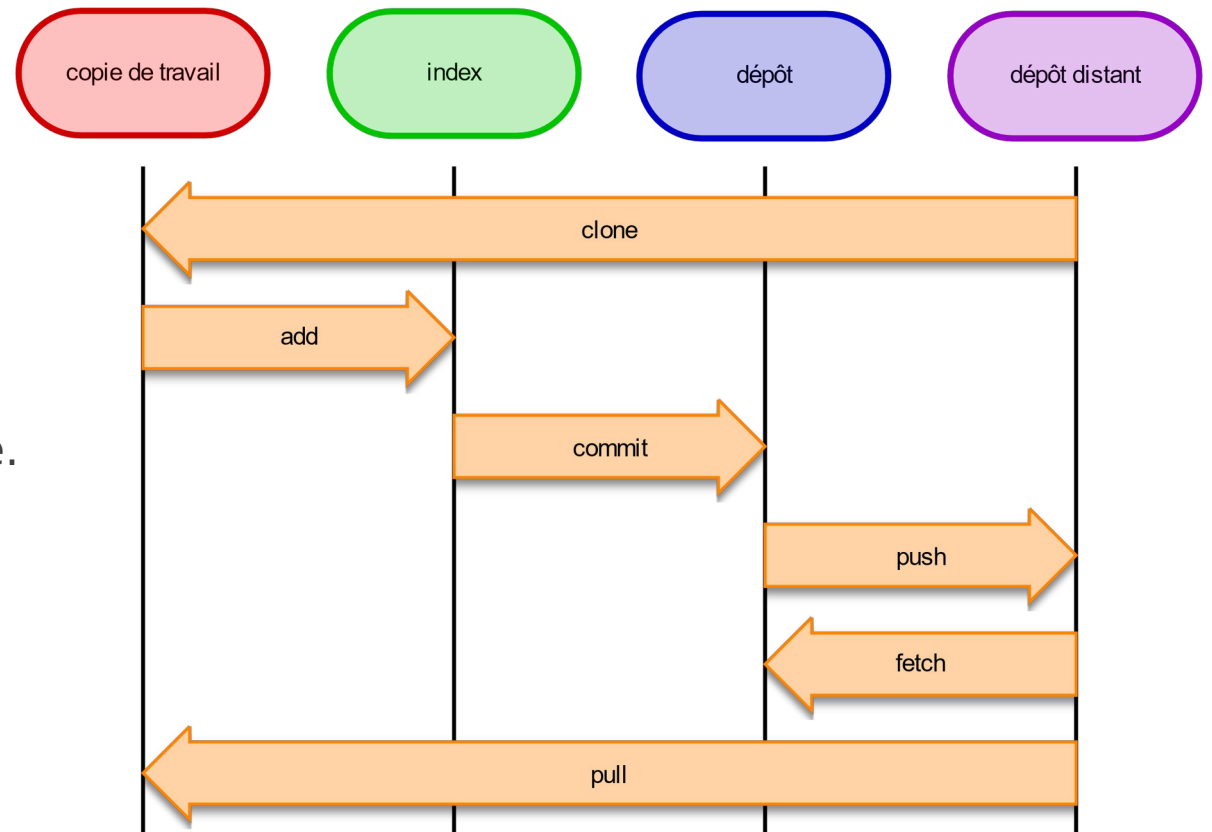


Figure 4 : Fonctionnement de Git. Source : <https://perso.liris.cnrs.fr/pierre-antoine.champin/enseignement/intro-git/>

Introduction : Github

Github est un service web d'hébergement et de développement de logiciels lancé en 2008.

Il permet de travailler à partir de plusieurs dépôts locaux et distants et sert à harmoniser les efforts des équipes de développement.

Github prend en charge l'aspect „dépôt distant“ de Git et propose de multiple outils qui facilitent le travail en groupe.

Il propose également un service de visualisation de données qui permet d'afficher en ligne des fichiers au formats csv, tsv, xml, py, ipynb, etc.

Il existe bien sûr d'autres services d'hébergement et de développement fonctionnant sous git : GitLab, Framagit, GNU Savannah, etc.



Figure 5 : logo Github
<https://fr.wikipedia.org/wiki/GitHub>

Le système de branches

Une branche est une image que l'on crée de la branche principale (*main* ou *master*). Elle en est déconnectée, mais peut à tout moment être versée dans cette branche ou dans toute autre branche parente (fonctionne en arborescence).

Deux niveaux de branches :

- Avec un dépôt ne nous appartenant pas: un *fork* permet de créer une image d'un dépôt pour travailler dessus.
- Avec un dépôt qui nous appartient : le *branch* permet de travailler sur notre projet ou sur un *fork*.

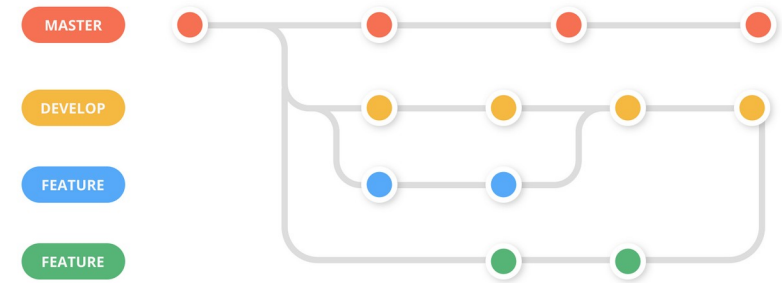


Figure 5 : Branches Git. Source : <https://zepel.io/blog/5-git-workflows-to-improve-development/>

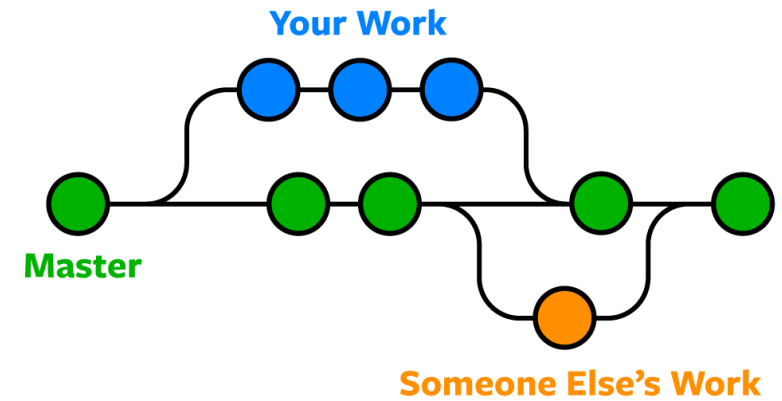


Figure 5 : Branches Git. Source : <https://www.nobledesktop.com/learn/git/git-branches/>

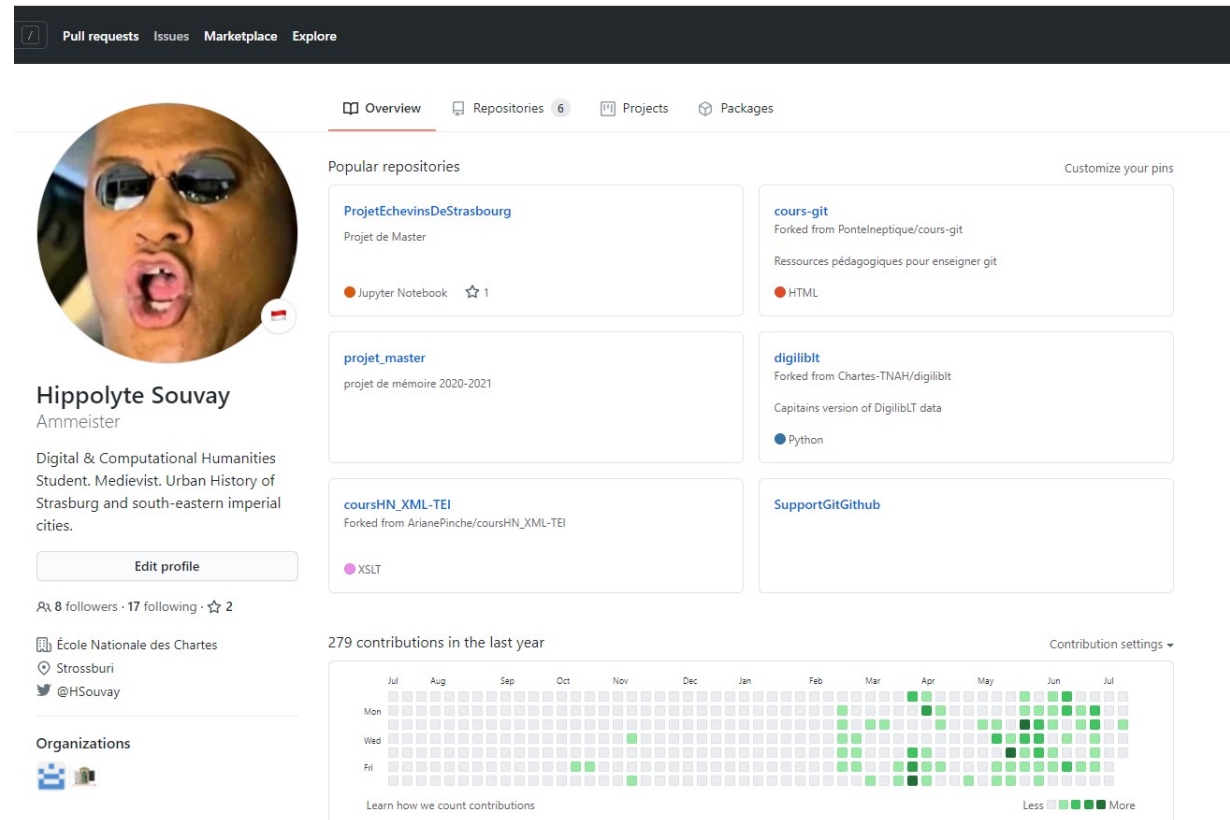
Initiation au développement collaboratif de projets avec Github

Nous allons passer à pratique et nous initier au versionnage. Il suffit pour cela :

[Optionnel] D'avoir installé git si l'on souhaite travailler depuis le terminal.

D'avoir téléchargé le client-bureau de Github (Github Desktop).

De s'être créé un compte Github.



The screenshot shows the GitHub profile of Hippolyte Souvay. The profile includes a circular profile picture, the name 'Hippolyte Souvay', and the title 'Ammelster'. Below this, it lists the user's bio: 'Digital & Computational Humanities Student. Medievalist. Urban History of Strasbourg and south-eastern imperial cities.' and a button to 'Edit profile'. The profile also shows '8 followers · 17 following · 2 stars'. Under 'Organizations', it lists 'École Nationale des Chartes', 'Strossburi', and '@HSouvay'. The 'Popular repositories' section displays five repositories: 'ProjetEchevinsDeStrasbourg', 'cours-git', 'projet_master', 'coursHN_XML-TEI', and 'SupportGitGithub'. The '279 contributions in the last year' section features a calendar heatmap showing activity from July to July, with a legend indicating 'Less' and 'More' contributions.

Créer un *repository*

Il est conseillé de toujours commencer par créer votre dépôt (*repository*) distant.

Cela simplifie le clonage du dépôt en local à l'aide de *Github Desktop*.

Il est également conseillé de toujours ajouter un fichier README (il est par défaut au format *markdown*). Cela évite les potentiels problèmes posés par un *repository* vide

Vous pouvez ajouter la licence sous laquelle se trouve votre projet.

Le gitignore sert à ajouter des fichiers du *repository* local qui ne doivent pas être pris en compte par les commandes Git (par exemple des fichiers trop volumineux).

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Owner *

 Ammeister ▾

Repository name *

/

Great repository names are short and memorable. Need inspiration? How about [redesigned-telegram](#)?

Description (optional)

☒  **Public**

Anyone on the internet can see this repository. You choose who can commit.

☐  **Private**

You choose who can see and commit to this repository.

Initialize this repository with:

Skip this step if you're importing an existing repository.

☒ **Add a README file**


This is where you can write a long description for your project. [Learn more.](#)

☐ **Add .gitignore**

Choose which files not to track from a list of templates. [Learn more.](#)

☐ **Choose a license**

A license tells others what they can and can't do with your code. [Learn more.](#)

This will set  **main** as the default branch. Change the default name in your [settings](#).

Create repository

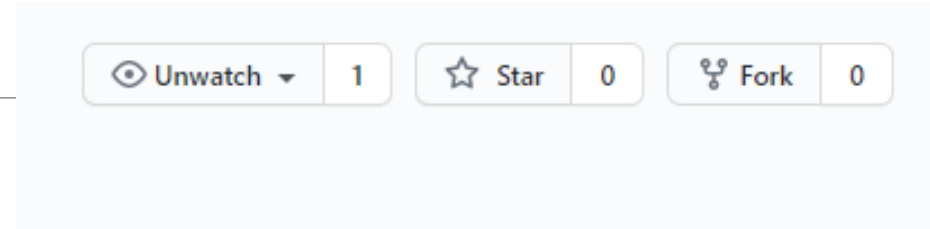
Travailler sur une branche

Rendez vous au lien suivant :
<https://github.com/Ammeister/SupportGitGithub>

Cliquez sur *Fork* pour créer un *fork* (une branche). Il apparaît désormais dans la liste de vos repository et vous pouvez travailler dessus comme n'importe quel autre dépôt distant.

Si votre dépôt distant est un *fork*, la branche *main* dont il est issu apparaîtra en-dessous de son nom.

À présent clonez votre dépôt distant (qui est une branche du mien) en local sur votre ordinateur.



S'authentifier sur GitHub Desktop

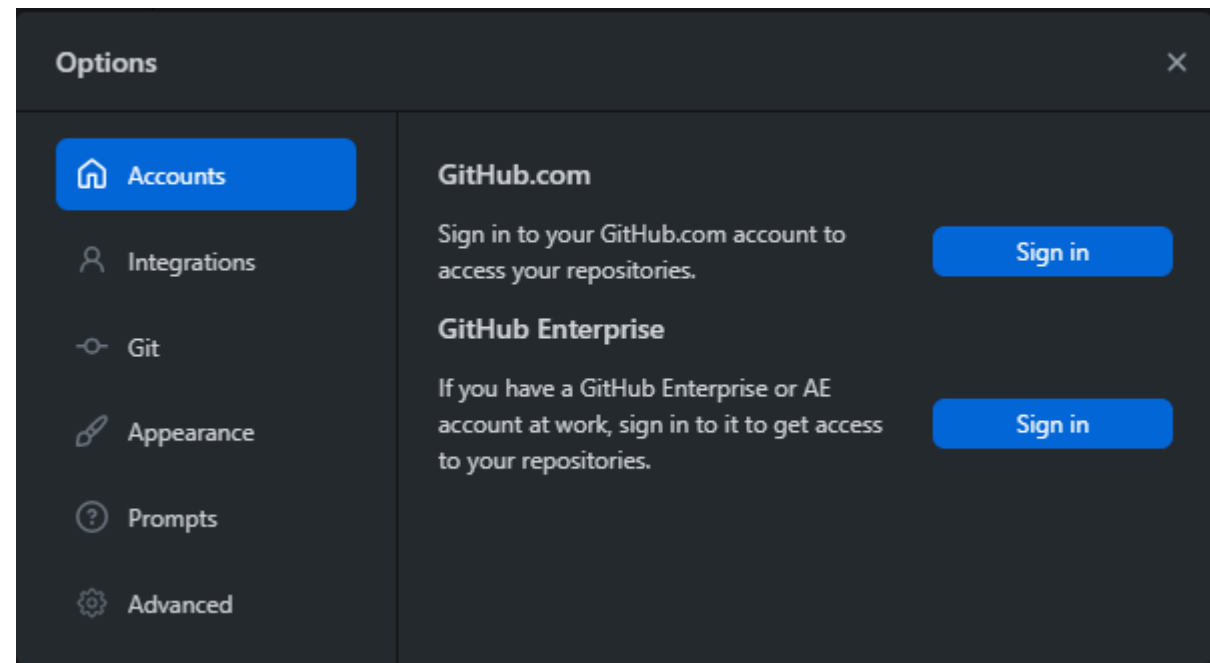
Pour vous authentifier sur GitHub Desktop :

File > Options > Accounts :

GitHub.com :

Sign in > Continue with browser

Cette manipulation permettra de vous authentifier automatiquement si vous êtes connectés à votre compte GitHub sur votre navigateur.



Créer un *token*

Cette manipulation est réservée aux utilisateurs qui emploient git en lignes de commande.

Le *token* (ou identificateur de session) n'est nécessaire que dans le cas où vous évoluez sans l'application GitHub Desktop.

Couplé à votre identifiant, il sert à vous authentifier lorsque vous envoyez les *commits* de votre dépôt local vers votre dépôt distant.

Les *tokens* ont remplacé définitivement les mots de passe il y a quelques mois sur GitHub.

Settings > Developer settings > Personal access tokens > Generate new token.

Cochez au moins la case repo.

The screenshot shows the GitHub 'Developer settings' page. On the left is a sidebar with the user 'Ammeister' and a list of settings: 'Your profile', 'Your repositories', 'Your codespaces', 'Your organizations', 'Your projects', 'Your stars', 'Your gists', 'Upgrade', 'Feature preview', 'Help', 'Settings', and 'Sign out'. The main content area is titled 'Personal access tokens' and includes a 'Generate new token' button. Below this, it shows a list of existing tokens, with one token named 'motdepasse' for 'repo, workflow' scope, expiring on 'Thu, Apr 7 2022'. A 'New personal access token' section follows, with a 'Note' field, an 'Expiration' dropdown set to '30 days', and a 'Select scopes' section. The 'repo' scope is selected, and its sub-options are listed: 'repo:status', 'repo_deployment', 'public_repo', 'repo:invite', and 'security_events'.

Settings / Developer settings

Signed in as **Ammeister**

petting a stork

Your profile
Your repositories
Your codespaces
Your organizations
Your projects
Your stars
Your gists

Upgrade
Feature preview
Help
Settings
Sign out

GitHub Apps
OAuth Apps
Personal access tokens

Personal access tokens Generate new token Revoke all

Tokens you have generated that can be used to access the [GitHub API](#).

motdepasse — repo, workflow Last used within the last week Delete

Expires on Thu, Apr 7 2022.

Personal access tokens function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to [authenticate to the API over Basic Authentication](#).

New personal access token

Personal access tokens function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to [authenticate to the API over Basic Authentication](#).

Note

What's this token for?

Expiration *

30 days The token will expire on Wed, Mar 9 2022

Select scopes

Scopes define the access for personal tokens. [Read more about OAuth scopes](#).

- ☒ **repo** Full control of private repositories
 - ☐ repo:status Access commit status
 - ☐ repo_deployment Access deployment status
 - ☐ public_repo Access public repositories
 - ☐ repo:invite Access repository invitations
 - ☐ security_events Read and write security events

Cloner un dépôt distant

Cela revient à créer un dépôt local.

Ouvrez la page d'accueil du *repository* que vous venez de créer.

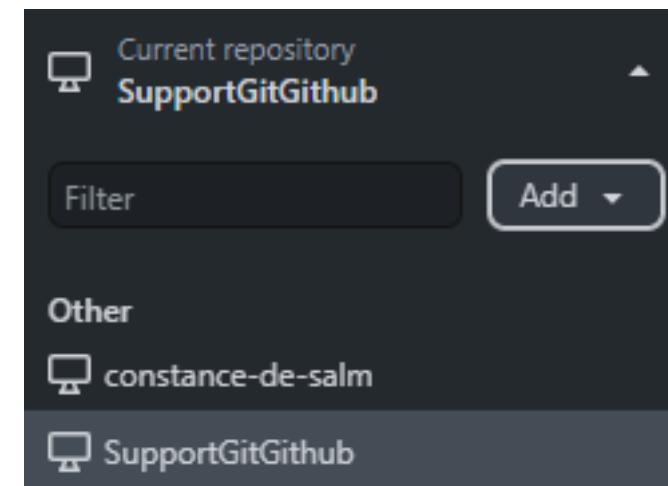
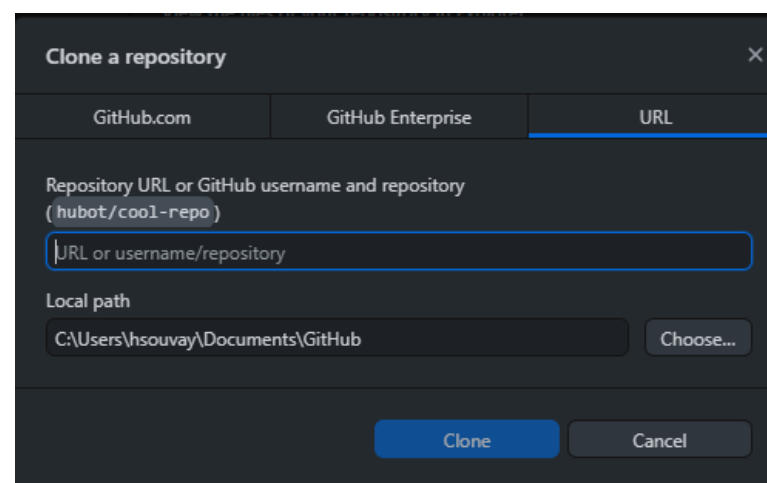
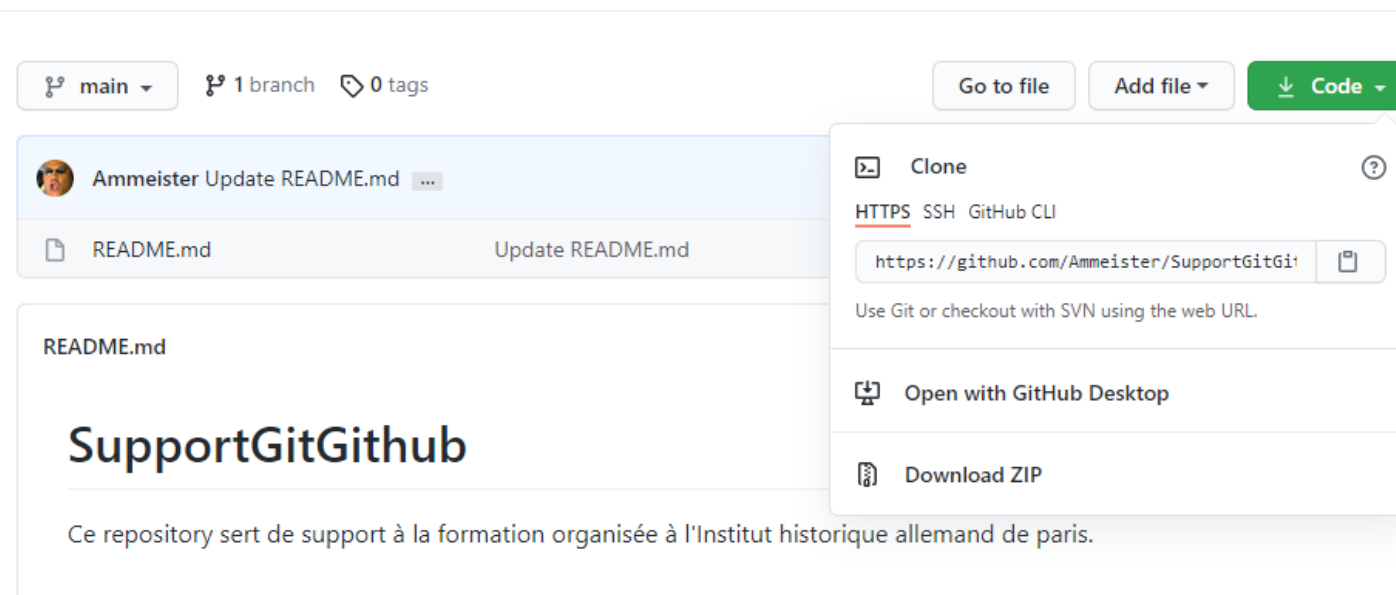
Cliquez sur le menu déroulant „Code“ puis copiez l'URL https.

Dans *Github Desktop* :

Ajoutez un nouveau Repository.

Choisissez le clonage par URL et collez le lien que vous avez copié sur Github.

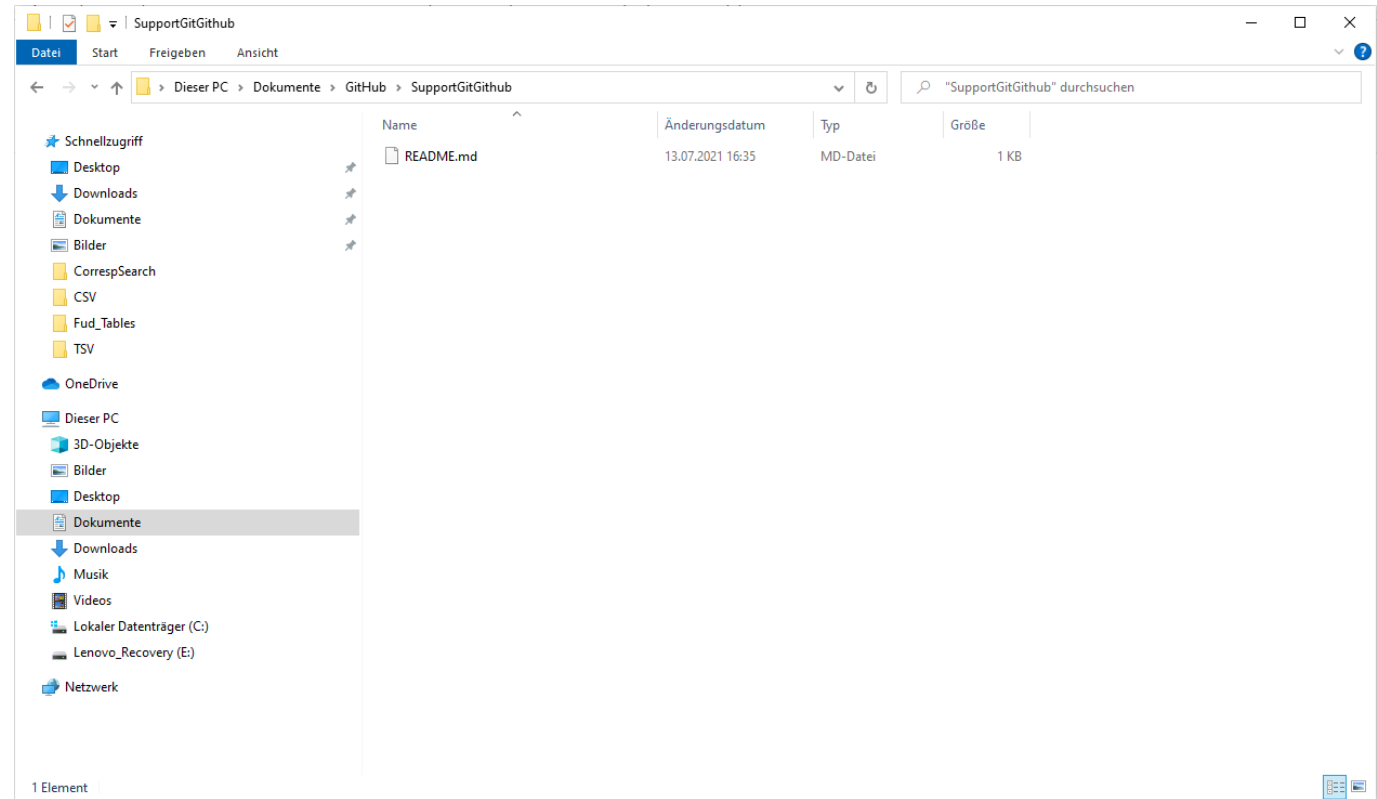
Vous pouvez personnaliser le chemin au bout duquel se trouvera le dépôt local Git (c'est-à-dire où le retrouver dans vos fichiers).



Travailler dans son dépôt local

Vous pouvez désormais travailler comme dans n'importe quel dossier de votre ordinateur.

Vous pouvez organiser votre *repository* comme vous le feriez pour n'importe quel projet.



Faire son premier *commit*

Ouvrir le bloc note.

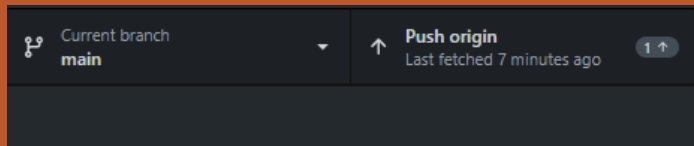
Écrire „hello world“.

Enregistrez le fichier au format .txt dans votre dépôt local.

Donner un titre au commit (faites si possible figurer le titre du fichier modifié).

Rédigez un résumé concis des modifications effectuées.

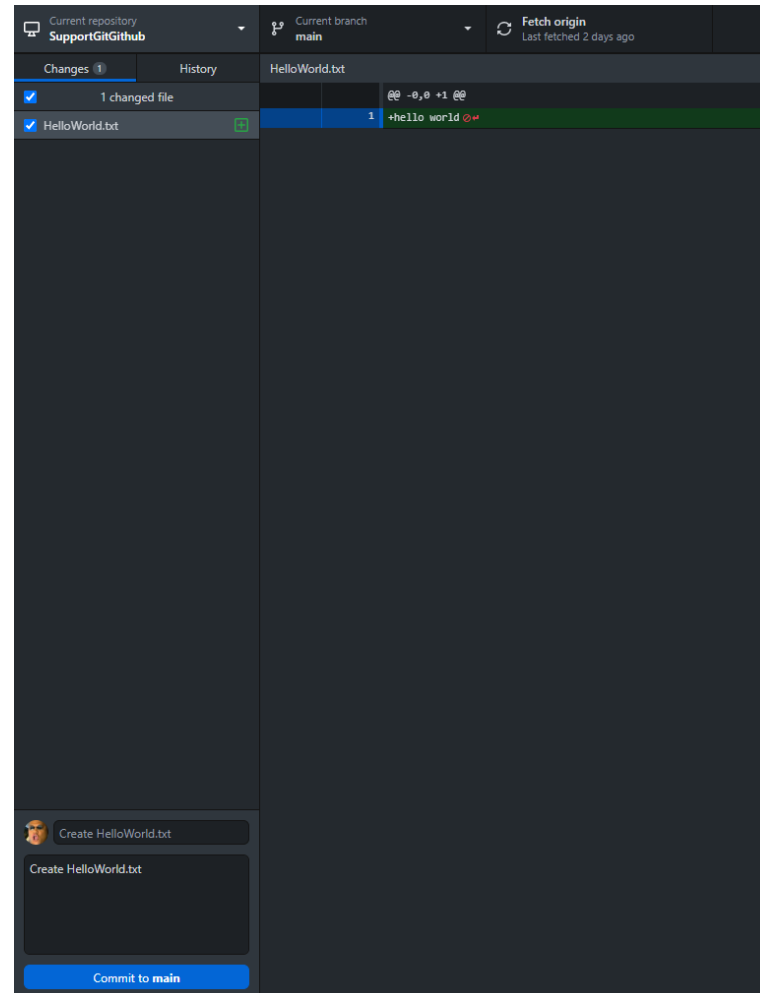
Commit vers la branche *main*.



En cliquant sur „Push origin“ on envoie nos *commits* effectués en local sur le dépôt distant.

À moins d'avoir réglé la confidentialité du dépôt distant sur „privé“ ou de figurer dans le fichier gitignore, tout le contenu envoyé sur le dépôt distant sera accessible en ligne, téléchargeable et réutilisable.

Les commandes pour faire un *commit* en ligne de commande :



git status : affiche les informations relatives aux statuts des documents du *repository*.

git add <nom du fichier> : ajouter un fichier à la *staging area*.

git commit : faire un commit de tout ce qui se trouve dans la *staging area*.

git push : envoyer les commits locaux vers le dépôt distant.

Travailler sur une branche locale

Créez une nouvelle branche (ici appelée *InitialsUpperCase*, mais elle peut porter le nom que vous souhaitez lui donner).

Les modifications apportées au fichier seront désormais prise en compte par la branche *InitialsUpperCase* et non plus par la branche *main*.

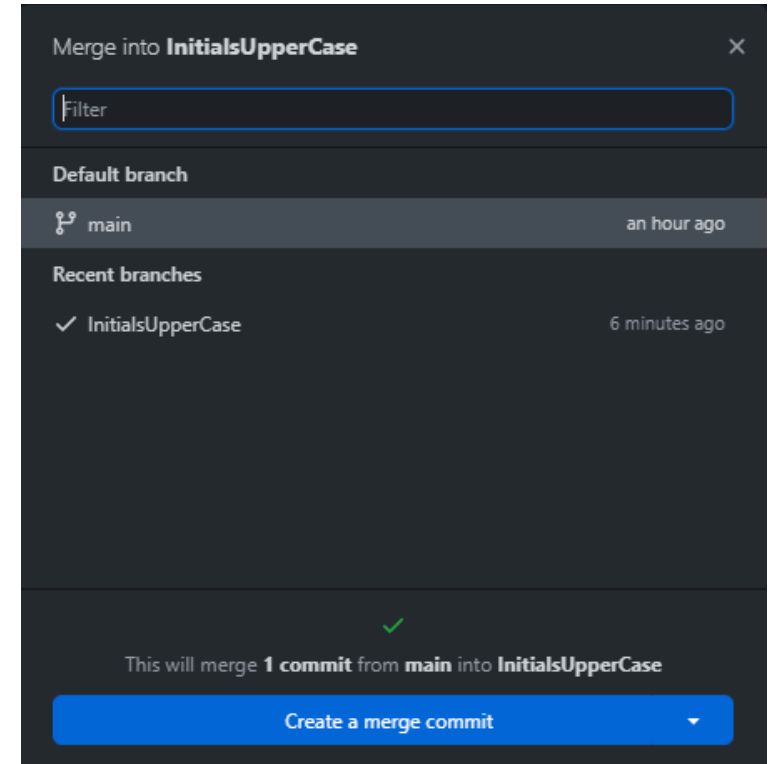
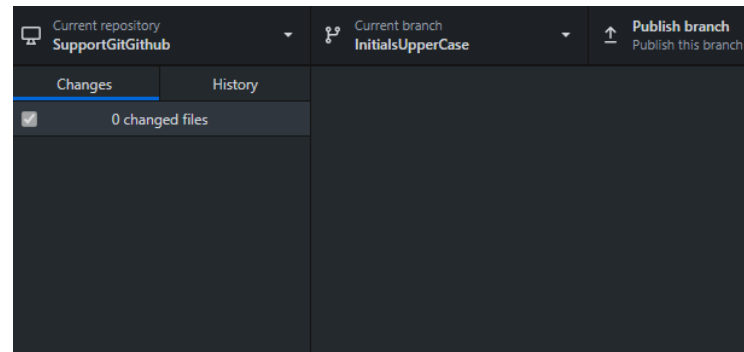
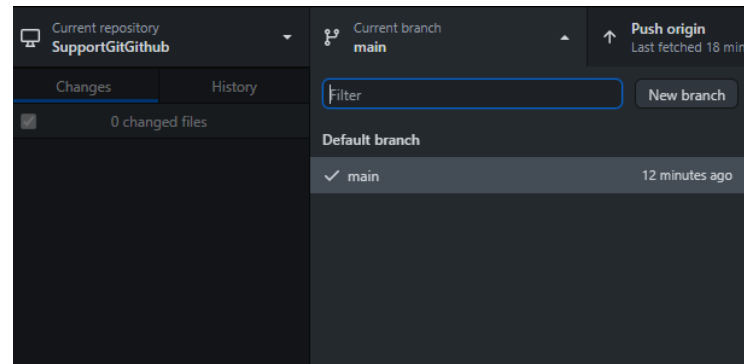
On peut changer de branche à tout moment.

Modifiez la casse des initiales dans votre fichier .txt, sauvegardez et fermez le. Rédigez un commit sur la même branche.

Publiez la branche. Cela envoie la branche nouvellement créée en local sur votre dépôt distant.

Si l'on souhaite conserver le contenu de la branche en la versant dans notre branche principale (*main*), il suffit de se mettre dans la branche *main* et d'y verser la branche *InitialsUpperCase* (Branch > Choose a branch to merge into main > Create a merge commit).

Travailler sur une branche permet de conserver plusieurs versions des fichiers d'un même dossier puisque les modifications effectuées sur une branche ne se retrouvent pas sur les autres.



Travailler sur une branche locale

Voici les commandes à utiliser dans la console pour travailler en lignes de commande :

git checkout -b <nom de la nouvelle branche> : créer une branche et s'y déplacer.

git push origin new-feature : envoyer les changements faits sur la branche « new-feature » sur le repository GitHub.

git branch -d <nom de la branche> : supprimer une branche.

Et pour rester à jour avec le repository principal, on va utiliser les commandes suivantes :

git pull upstream master

git push origin master

Pratique : correction du fichier Exemple.xml

Pour travailler de concert, il faut pouvoir s'organiser.

GitHub propose à cet effet la fonctionnalité des *issues*.

Il s'agit de problème à résoudre et ces problèmes peuvent être soulevés par tous les utilisateurs / collaborateurs.

Une bonne *issue* doit comporter :

- Le nom du document/fichier concerné
- Le problème détaillé.
- [Optionnel] Des propositions de solution.
- [Optionnel] Des tags pour catégoriser les *issues*.

Une fois le problème résolu par une *pull request*, un administrateur peut clore l'*issue*.

Mise en pratique ; chacun va se voir attribuer une *issue* et dans l'ordre :

- Créer une branche locale.
- Corriger l'erreur sur sa branche locale.
- Fusionner la branche locale avec la branche main.
- *Push* son dépôt local vers son dépôt distant.
- Faire une *pull request* pour intégrer son travail au dépôt source.

