

Cursus développeur JAVA full stack

M2I - Formation - 2024

Donjon Audrey

Git - GitHub

Versionner son code et travailler de manière collaborative



Donjon Audrey
Formatrice Frontend

Contact :



www.linkedin.com/in/audrey-djn



[audrey_](#)



[audrey-donjon](#)

Sommaire général

1. Introduction à Git
2. Installation de Git & premiers pas
3. Introduction à GitHub
4. Collaboration via GitHub
5. Ressources
6. Pour aller plus loin

Objectifs :

- ☐ Comprendre le versionning
- ☐ Utiliser Git pour gérer un projet
- ☐ Collaborer via GitHub



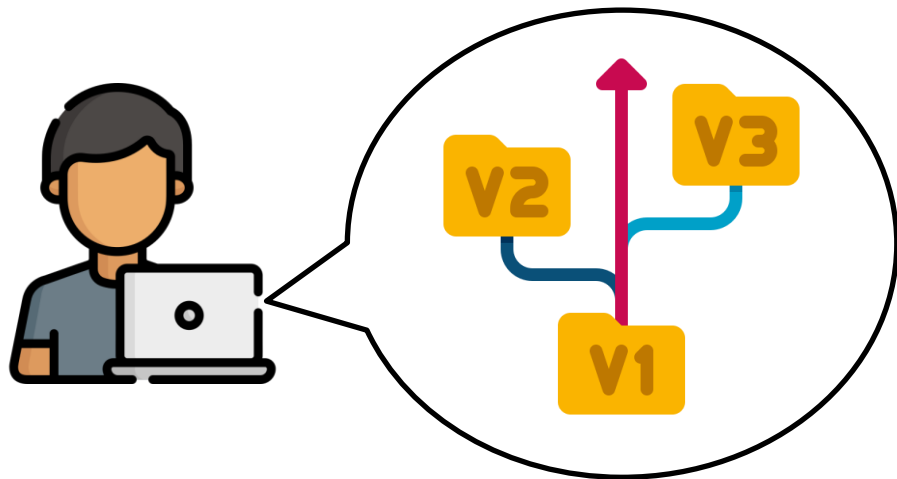
Introduction à Git

Sommaire : Introduction à Git

1. Introduction à Git
2. Histoire de Git
3. Quelques concepts clés de Git
4. Exercice 01

Qu'est ce que le versionning ?

Versionning ou gestion de versions, est une méthode utilisée pour suivre et gérer les modifications apportées aux fichiers au fil du temps.



Imaginez que vous travaillez sur un projet et que vous apportez des changements chaque jour.

Le **versionning** vous permet de sauvegarder chaque version de votre travail, de manière à pouvoir revenir en arrière ou voir les changements spécifiques effectués à tout moment.

Qu'est ce que GIT ?

Git est un outil de versionning qui permet de gérer les versions de vos fichiers de manière efficace et collaborative. Grâce à cet outil il sera possible en outre de :

- ❑ **Gérer le travail à plusieurs** en travaillant sur le même projet sans écraser les modifications des autres
- ❑ **Revenir en arrière** et restaurer une version antérieure du projet
- ❑ **Voir toutes les modifications** apportées au projet

Documentation officielle de Git : <https://git-scm.com/docs>

Cours officiel complet sur Git : <https://git-scm.com/book/fr/v2>

Les origines de GIT?

Git sera créée en avril 2005 par Linus Torvalds (le créateur du noyau linux) à la suite d'une problématique de son créateur qui à cette époque utilisait BitKeeper (un logiciel que linux distribuait avec ses patches et qui était gratuit pour cet usage) cependant après un désaccord en 2005, BitKeeper stoppât leur partenariat et c'est à ce moment que Linus Torvalds créera ce qui deviendra le logiciel de versionning le plus puissant et le plus utilisé au monde dans la très grande majorité des branches de l'informatique.

2005 : Première version de GIT conçu pour être rapide, avec une forte intégrité des données et un support puissant pour les branches et les merges.

2005-2006 : Git est rapidement adopté par les développeurs du noyau Linux et gagne en popularité, Junio Hamano rejoint le projet et devient le mainteneur principal, apportant de nombreuses améliorations.

Aujourd'hui : Git est devenu le système de contrôle de version le plus utilisé au monde.

1 - Dépôt (Repository) :

Un dépôt est comme un dossier spécial qui contient tous les fichiers de votre projet, ainsi que l'historique des modifications apportées à ces fichiers. Il peut être **local** (sur votre ordinateur) ou **distant** (sur un service comme GitHub).

2 - Commit :

Un commit est comme une « photo » de vos fichiers à un instant donné. Chaque fois que vous effectuez un commit, vous enregistrez un état spécifique de vos fichiers. C'est une étape importante qui permet de suivre ce qui a été modifié.

3 - Branche (Branch) :

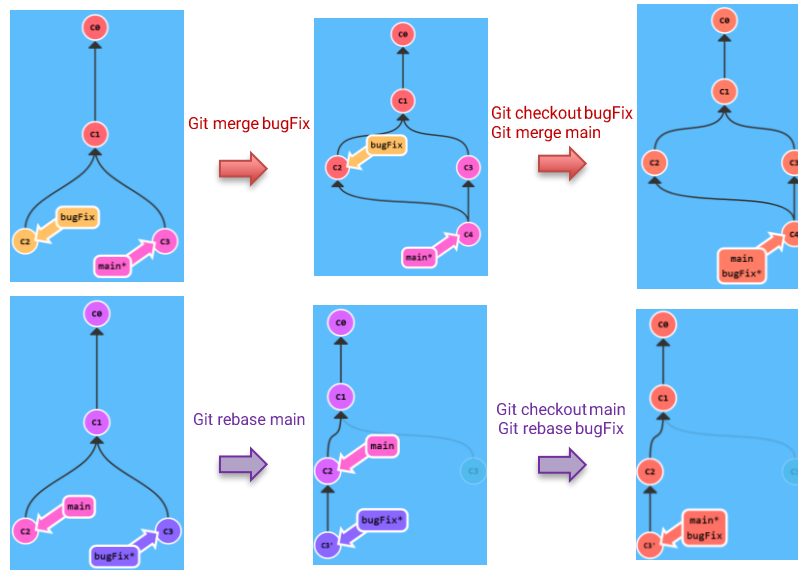
Une branche est une **version parallèle de votre projet**. Par défaut, Git crée une branche principale appelée 'main' ou 'master'. Vous pouvez créer des branches pour travailler sur différentes fonctionnalités ou corrections de bugs sans affecter la branche principale.

4 - Merge et Rebase :

Lorsque vous avez fini de travailler sur une branche et que vous souhaitez intégrer ces changements dans une autre branche (souvent la branche principale), vous effectuez un **merge**. Cela combine les modifications des deux branches.

Le **rebase** quand à lui est une opération qui permet de repositionner les commits d'une branche sur une autre branche. Cela aide à maintenir un historique de commits plus propre et linéaire. Le rebase réécrit l'historique des commits pour que tous les commits de la branche en cours apparaissent comme ayant été faits directement sur la branche cible (par exemple, main). Cela évite les commits de fusion (merge commits) et rend l'historique plus facile à lire.

Remarque : Supprimer les branches après avoir terminé un rebase ou un merge est une bonne pratique pour garder votre dépôt propre et organisé. Les étapes sont simples : fusionner la branche, supprimer la branche locale et, si nécessaire, supprimer la branche distante. Cela aide à maintenir la clarté et à éviter la confusion dans votre gestion des branches.



5 - Push et Pull :

'Push' -> Permet d'envoyer vos commits locaux à un dépôt distant pour les partager avec d'autres.

'Pull' -> Permet de récupérer les dernières modifications d'un dépôt distant pour les intégrer à votre copie locale.

Mon site web

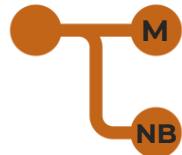


1

Mon projet



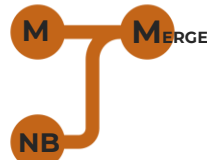
2



Branch **M**ain

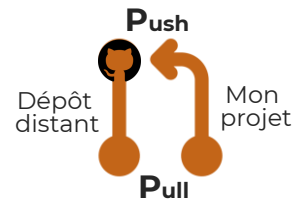
Nouvelle **B**ranche

3



Fusion de branche

4



5

- 1 – Je travaille sur un site web et je vais créer une nouvelle fonctionnalité
- 2 – Je crée un dépôt pour initialiser Git dans le dossier de mon projet
- 3 – Je crée une branche, pour travailler sur une branche séparée qui contiendra ma nouvelle fonctionnalité
- 4 – Je fusionne une fois que je suis satisfait de mon travail, en fusionnant ma branche avec la branche principale
- 5 – J'utilise 'pull' si besoin pour obtenir les dernières mises à jour de mes collègues et ensuite je partage mon travail en utilisant 'push' sur GitHub

Mon travail devient alors bien organisé, traçable et collaboratif.

Exercice 01 : entraînement des commandes de base

Entraînement :

- ☐ Sélectionner l'onglet « Principal »
- ☐ Faire la partie « Une introduction en douceur à la majorité des commandes Git »
- ☐ Compléter le niveau 1,2,3 et 4





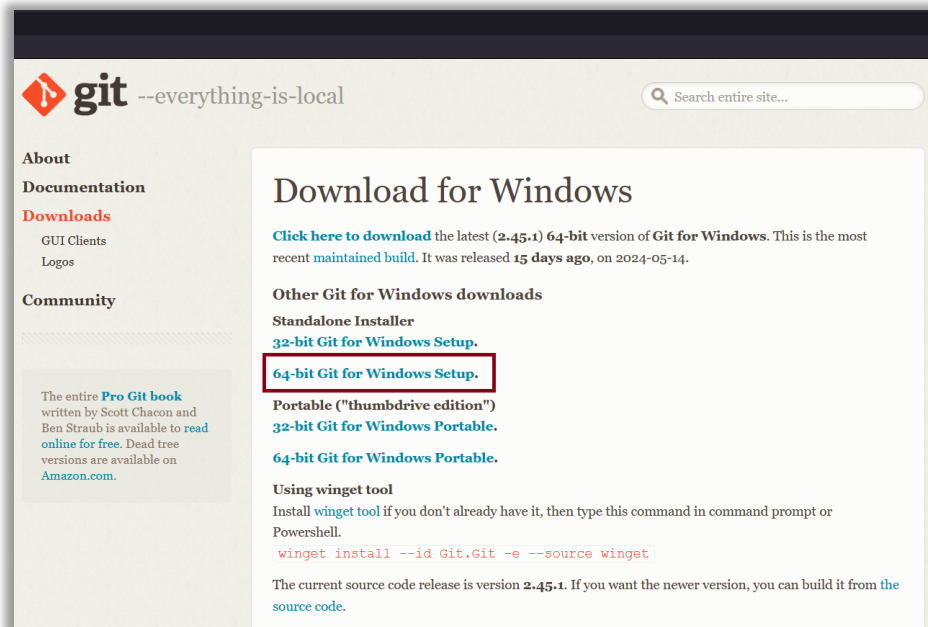
Installation de Git & premiers pas

Sommaire : Installation de Git & premiers pas

1. Installation de Git sous Windows
2. Vérifier l'installation
3. Quelques lignes de commandes avec Git Bash
4. Premiers tests avec un projet Git
5. Théorie sur le fonctionnement
6. Mieux comprendre les étapes et flux de travail
7. Résolution de conflits
8. Exercice 02

1 - Téléchargement sur le site officiel : <https://git-scm.com/downloads>

2 - Installation



The screenshot shows the Git website's 'Download for Windows' page. The page title is 'Download for Windows'. It provides information about the latest version (2.45.1) and offers download links for the 32-bit and 64-bit versions. The 64-bit version is highlighted with a red box. The page also includes a section for 'Other Git for Windows downloads' and a 'Using winget tool' section with a command to install Git.

Download for Windows

Click [here to download](#) the latest (2.45.1) 64-bit version of Git for Windows. This is the most recent [maintained build](#). It was released 15 days ago, on 2024-05-14.

Other Git for Windows downloads

Standalone Installer

32-bit Git for Windows Setup.

64-bit Git for Windows Setup.

Portable ("thumbdrive edition")

32-bit Git for Windows Portable.

64-bit Git for Windows Portable.

Using winget tool

Install [winget tool](#) if you don't already have it, then type this command in command prompt or Powershell.

```
winget install --id Git.Git -e --source winget
```

The current source code release is version 2.45.1. If you want the newer version, you can build it from [the source code](#).



The screenshot shows the 'Git 2.45.1 Setup' window. It displays the 'Information' tab, which includes the GNU General Public License (Version 2, June 1991). The license text is shown in a scrollable area. At the bottom, there are buttons for 'Next' and 'Cancel', and a checkbox for 'Only show new options'.

Git 2.45.1 Setup

Information

Please read the following important information before continuing.

When you are ready to continue with Setup, click Next.

GNU General Public License

Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc.
59 Temple Place - Suite 330, Boston, MA 02111-1307, USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change

<https://gitforwindows.org/>

☐ Only show new options

Next **Cancel**

3 – Installation paramétrage (12 étapes)

Git 2.45.1 Setup

Select Components
Which components should be installed?

Select the components you want to install; clear the components you do not want to install. Click Next when you are ready to continue.

- ☒ Additional icons
 - ☒ On the Desktop
- ☒ Windows Explorer integration
- ☒ Open Git Bash here
 - ☐ Open Git GUI here
- ☒ Git LFS (Large File Support)
- ☒ Associate .git* configuration files with the default text editor
- ☒ Associate .sh files to be run with Bash
- ☒ Check daily for Git for Windows updates
- ☐ (NEW!) Add a Git Bash Profile to Windows Terminal

Current selection requires at least 339,0 MB of disk space.

<https://gitforwindows.org/>

☐ Only show new options

Git 2.45.1 Setup

Choosing the default editor used by Git
Which editor would you like Git to use?

Use Visual Studio Code as Git's default editor

Visual Studio Code is an Open Source, lightweight and powerful editor running as a desktop application. It comes with built-in support for JavaScript, TypeScript and Node.js and has a rich ecosystem of extensions for other languages (such as C++, C#, Java, Python, PHP, Go) and runtimes (such as .NET and Unity).

Use this option to let Git use Visual Studio Code as its default editor.

<https://gitforwindows.org/>

☐ Only show new options

Git 2.45.1 Setup

Adjusting the name of the initial branch in new repositories
What would you like Git to name the initial branch after "git init"?

☐ Let Git decide

Let Git use its default branch name (currently: "master") for the initial branch in newly created repositories. The Git project [intends](#) to change this default to a more inclusive name in the near future.

☒ Override the default branch name for new repositories

NEW! Many teams already renamed their default branches; common choices are "main", "trunk" and "development". Specify the name "git init" should use for the initial branch:

main

This setting does not affect existing repositories.

<https://gitforwindows.org/>

☐ Only show new options

Git 2.45.1 Setup

Adjusting your PATH environment
How would you like to use Git from the command line?

☐ Use Git from Git Bash only

This is the most cautious choice as your PATH will not be modified at all. You will only be able to use the Git command line tools from Git Bash.

☒ Git from the command line and also from 3rd-party software

(Recommended) This option adds only some minimal Git wrappers to your PATH to avoid cluttering your environment with optional Unix tools. You will be able to use Git from Git Bash, the Command Prompt and the Windows PowerShell as well as any third-party software looking for Git in PATH.

☐ Use Git and optional Unix tools from the Command Prompt

Both Git and the optional Unix tools will be added to your PATH.
Warning: This will override Windows tools like "find" and "sort". Only use this option if you understand the implications.

<https://gitforwindows.org/>

☐ Only show new options

Git 2.45.1 Setup

Choosing the SSH executable
Which Secure Shell client program would you like Git to use?

☒ Use bundled OpenSSH

This uses ssh.exe that comes with Git.

☐ Use external OpenSSH

NEW! This uses an external ssh.exe. Git will not install its own OpenSSH (and related) binaries but use them as found on the PATH.

<https://gitforwindows.org/>

☐ Only show new options

Git 2.45.1 Setup

Choosing HTTPS transport backend
Which SSL/TLS library would you like Git to use for HTTPS connections?

☒ Use the OpenSSL library

Server certificates will be validated using the ca-bundle.crt file.

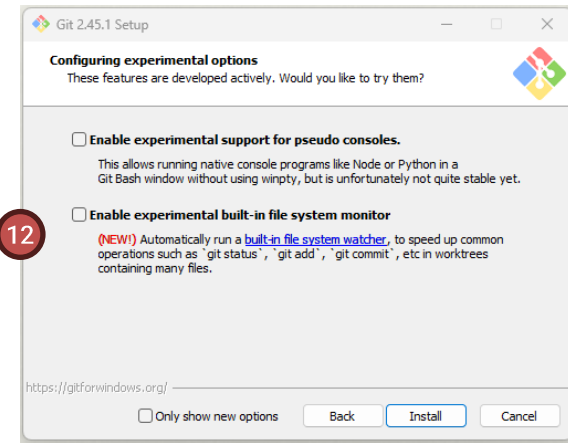
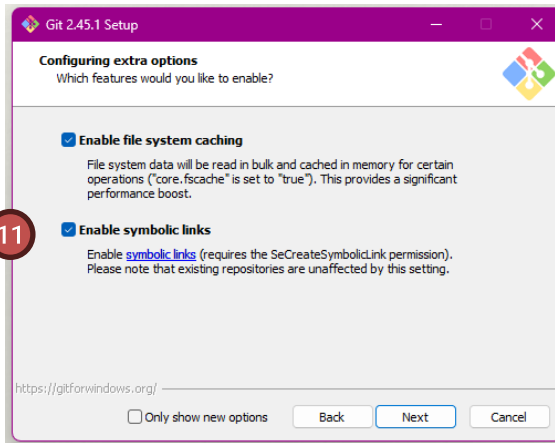
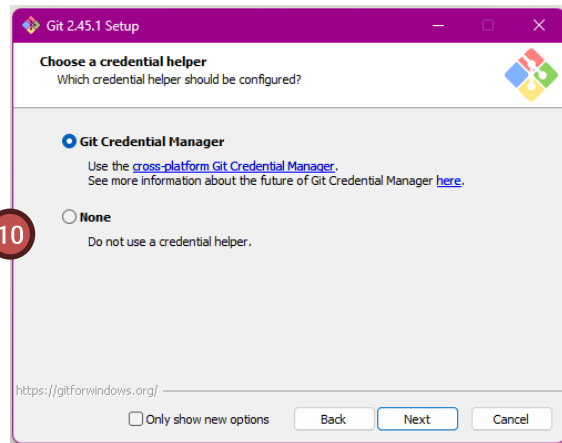
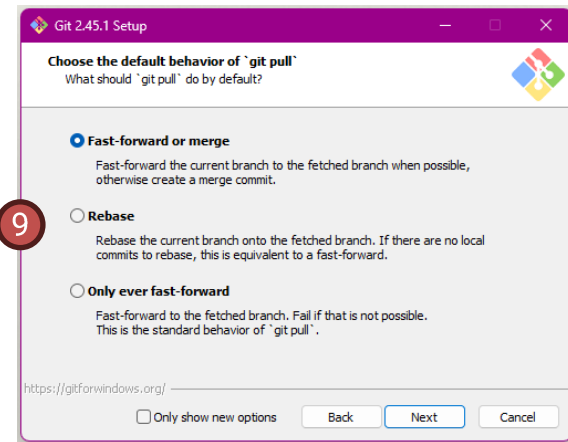
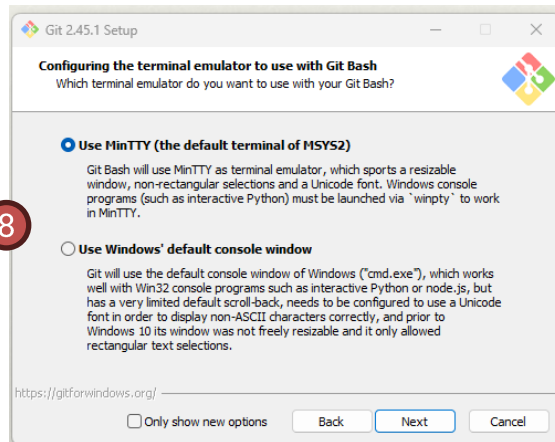
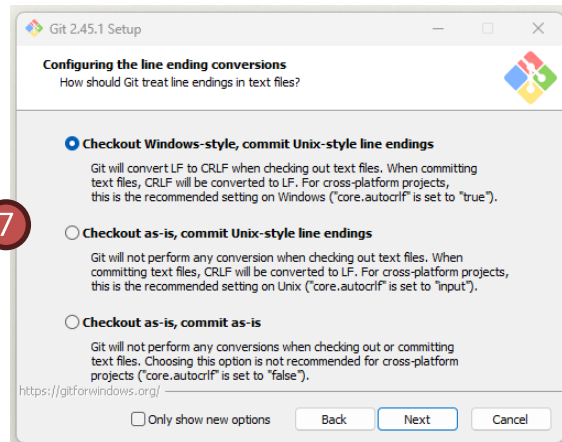
☐ Use the native Windows Secure Channel library

Server certificates will be validated using Windows Certificate Stores. This option also allows you to use your company's internal Root CA certificates distributed e.g. via Active Directory Domain Services.

<https://gitforwindows.org/>

☐ Only show new options

3 – Installation paramétrage (12 étapes)



Vérifier l'installation et configurer son identité

1 – Vérification de l'installation :

Afin de vérifier que Git soit bien installé, il suffit d'ouvrir un terminal et taper la commande suivante :

```
bash

git --version
```



```
bash

git --version
git version 2.45.1.windows.1
```

2 – Configurer son identité :

Git utilise le nom et l'adresse mail pour identifier vos commits, pour les configurer taper les commandes suivantes :

Attention ce nom et cette adresse email serviront uniquement pour "signer" numériquement vos futurs commits, ils n'ont rien à voir avec vos nom et email sur Github !

D'ailleurs vos commits pouvant être publiques, évitez de mettre votre adresse email principale ou des infos sensibles en nom (nom de famille, etc...) En général, on crée une adresse email dédiée exprès pour la signature des commits pour éviter de mettre son adresse qu'on ne souhaite pas divulguer publiquement. Pour le nom, vous pouvez mettre un pseudonyme, votre prénom ou n'importe quoi d'autre du moment où vous êtes conscient que ces infos seront visibles publiquement.

```
bash

git config --global user.name "Votre Nom"
git config --global user.email "votre.email@example.com"
```

3 – Voir toutes les configurations en place :

Pour voir toute les configurations présentent vous pouvez taper ceci : (votre nom et mail devraient apparaître)

```
bash

git config --list
```

1 – Naviguer dans le système de fichiers :

- ❑ Afficher le répertoire actuel :

```
bash
```

```
pwd
```

- ❑ Lister les fichiers et répertoires :

```
bash
```

```
ls
```

- ❑ Changer de répertoire :

```
bash
```

```
cd nom ou "Nom du répertoire"
```

```
cd ..      # Remonter d'un niveau
```

```
cd ~       # Aller au répertoire personnel
```

```
cd /       # Aller à la racine du système de fichiers
```

2 – Manipulation de fichiers et répertoires :

- ❑ Créer un répertoire

```
bash
```

```
mkdir nom ou "Nom du répertoire"
```

- ❑ Créer un fichier

```
bash
```

```
touch <nom_du_fichier>
```

- ❑ Copier un fichier

```
bash
```

```
cp <source> <destination>
```

- ❑ Déplacer/Renommer un fichier

```
bash
```

```
mv <source> <destination ou nouveau_nom>
```

- ❑ Supprimer un fichier

```
bash
```

```
rm <nom_du_fichier>
```

- ❑ Supprimer un répertoire

```
bash
```

```
rm -r <nom_du_fichier>
```

- ❑ Ajouter du contenu à un fichier

```
bash
```

```
echo "# Mon contenu" > <nom_du_fichier>
```

Premiers tests avec un projet Git



1 – Initialiser un dépôt Git :

- ☐ Créez un nouveau dossier pour votre projet, puis naviguez à l'intérieur de ce dossier avec le terminal ou cliquez droit directement sur le dossier et « open git bash here », pensez à vérifier que vous soyez bien dans votre dossier pour la suite des commandes.
- ☐ Tapez la commande suivante pour initialiser un dépôt Git :

```
bash
```

```
git init
```

2 – Ajouter des fichiers au dépôt :

- ☐ Créez un fichier (par exemple, "README.md") dans votre dossier
- ☐ Pour ajouter ce fichier à Git utilisez la commande suivante :

```
bash
```

```
#ajoute un fichier modifiés  
git add README.md  
#ajoute tous les fichiers modifiés  
git add .
```

```
Bash
```

```
# inverse de add, permet d'annuler l'ajout d'un fichier à l'index  
git restore --staged README.md
```

3 – Faire un commit :

- ☐ Un commit enregistre vos changements. Utilisez la commande suivante :

```
bash
```

```
git commit -m "Création du fichier README.md"
```

- ☐ Le message entre guillemets devrait décrire les changements que vous avez fait.

Remarque : si vous ne mettez pas -m et le message, git ouvrira l'éditeur défini lors de l'installation pour vous permettre de l'écrire dedans, il faudra alors l'écrire puis fermer l'onglet pour qu'il puisse être pris en compte.

5 – Organisation de branche :

- ☐ Créer une nouvelle branche :

```
bash
git branch feature
```

- ☐ Changer de branche

```
bash
git checkout feature
```

- ☐ Créer et basculer vers une nouvelle branche

```
bash
git checkout -b feature2
```

- ☐ Fusionner des branches

```
bash
git merge <nom_de_la_branche>
```

- ☐ Supprimer une branche locale (dans le cas où on a déjà fusionné nos branches et qu'on en a plus besoin, on bascule sur la branche main et on supprime les branches souhaités)

```
bash
git branch -d <nom_de_la_branche>
```

- ☐ Changer le nom d'une branche

```
bash
git branch -m <nouveau_nom>
```

5 – Vérifier l'état du dépôt :

- ❑ Pour voir l'état de votre dépôt, utilisez la commande :

```
bash
git status
```

réponse →

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

6 – Voir l'historique des commits :

```
bash
#historique de tous les commits
git log
#version condensée des commits
git log --oneline
```

```
bash
#version graphique de git log
gitk
```

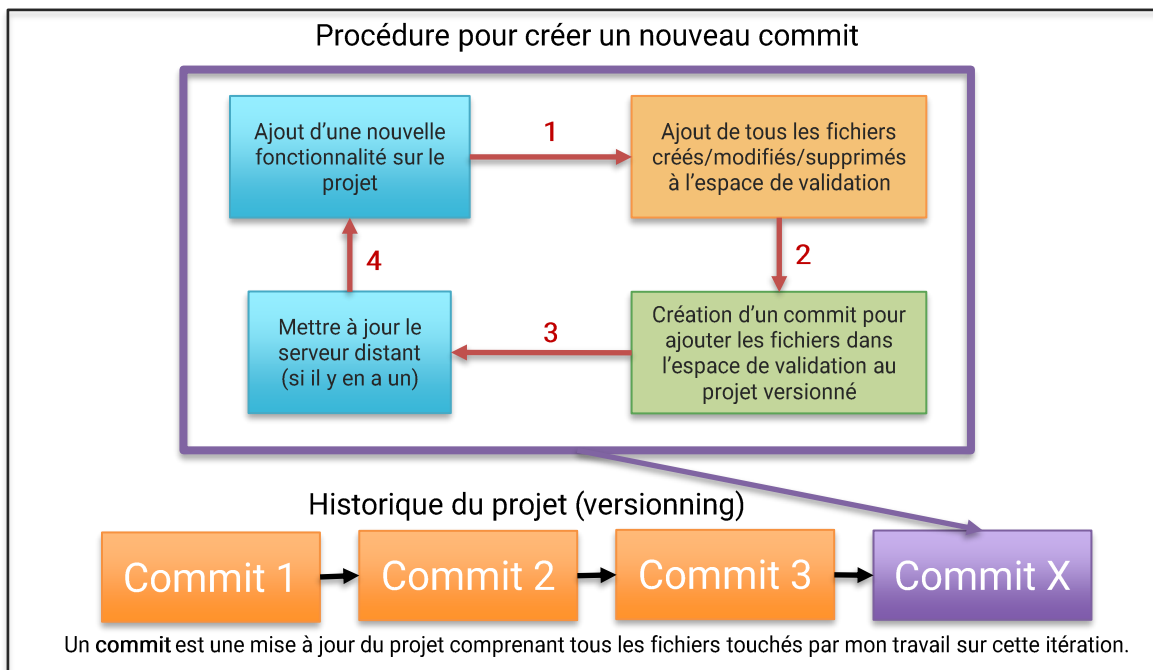
Réponse
approximative →

```
bash
$ git log
commit a986b27ef768776da69zzedsdfr568df489f83af (HEAD -> main)
Author: Nom <mail>
Date: DD MM NB HH:MIN:SEC ANNE+0200

Création du fichier README.md
```

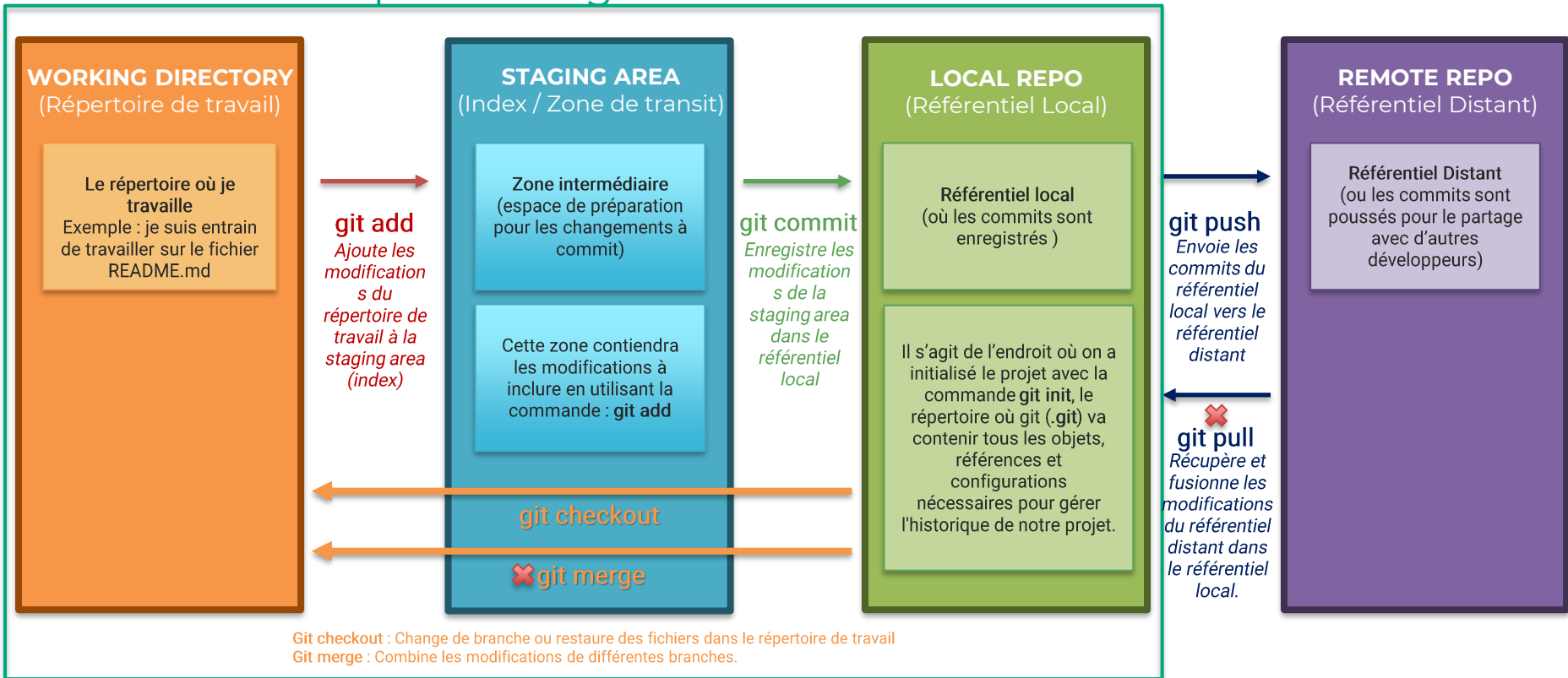
Quand on travail sur un projet avec GIT, on le fait en boucle en suivant une certaine procédure.

1. Je travail sur les fichiers du projet (ajout d'un formulaire sur mon projet web par exemple), dans le dossier du projet (appelé **espace de travail**)
2. J'ai terminé mon travail, j'ai vérifié que tout fonctionne bien, je décide donc d'ajouter mon travail terminé au **projet versionné**.
3. J'ajoute tous les nouveaux fichiers que j'ai créé/modifié/supprimé à l'**espace de validation** du prochain commit.
4. Je crée mon **commit** en lui donnant une description (le **commit** ajoutera au projet versionné tous les fichiers qui sont dans l'**espace de validation**)
5. J'ai créé mon **commit**, je peux maintenant recommencer à l'étape 1, en oubliant pas au passage de mettre à jour le projet sur GitHub par exemple.



Manipulations git avec dépôt distant

Manipulations git localement





Git merge et git pull :

Lors d'un **git merge** ou d'un **git pull**, si Git trouve des modifications conflictuelles dans le même fichier, il génère un conflit.

1 - Vérifier les conflits :

Utiliser **git status** pour voir quels fichiers sont en conflit.

2 - Résoudre les conflits manuellement :

Ouvrir les fichiers en conflit dans notre éditeur de texte où il faudra alors choisir quelle partie des modifications conserver ou combiner les deux. Puis sauvegarder le fichier.

3 - Marquer les Conflits Comme Résolus :

Après avoir résolu les conflits dans un fichier, ajouter le à l'index avec la commande **git add <fichier_en_conflit>**

4 – Continuer le processus de fusion :

Continuer l'opération avec la commande **git merge --continue**

Exercice 02 : QCM

Lien vers QCM donné par le formateur

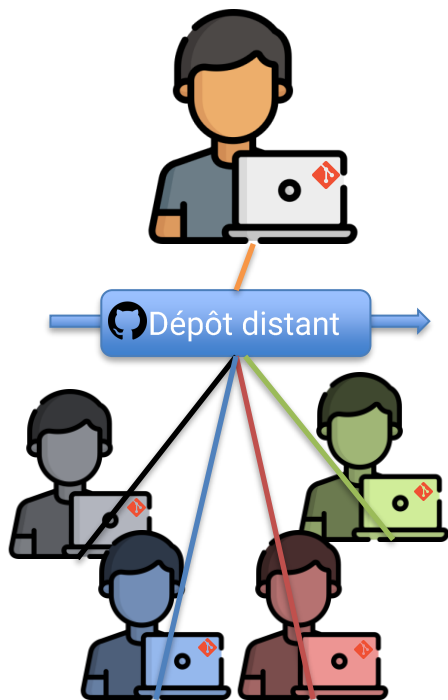




Introduction à GitHub

Qu'est ce que GitHub ?

GitHub est une plateforme en ligne qui permet aux développeurs de stocker, gérer, et partager leur code. C'est un endroit où les projets logiciels peuvent être hébergés et collaborés. Elle a été créée par Tom Preston-Werner, Chris Wanstrath, PJ Hyett et Scott Chacon. Il existe également d'autres concurrents tel que Gitlab, Bitbucket etc...



Contexte et création :

Avril 2008 : Lancement de la plateforme dont l'objectif est de faciliter la collaboration entre développeurs et héberger des projets utilisant le système de contrôle de version de Git.

Septembre 2021 : Github annonce qu'il y avait plus de 2,1 millions d'utilisateurs et plus de 3,7 millions de dépôts.

Juin 2018 : Acquisition par Microsoft pour 7,5 milliards de dollars. Par contre GitHub continue de fonctionner comme une entité indépendante et garde son propre branding.

Janvier 2023 : Dépassement de la barre des 100 millions d'utilisateurs.

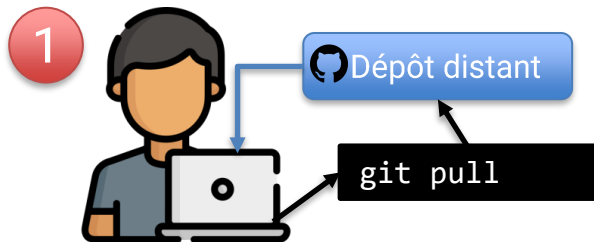


Collaboration via GitHub

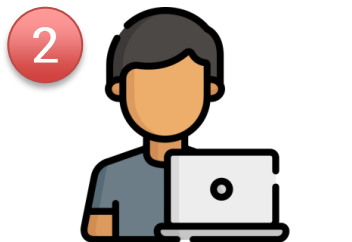
Sommaire : Collaboration via GitHub

1. Étapes théoriques de collaboration sur GitHub
2. Création de compte sur GitHub
3. Exercice 03 : Création de dépôt GitHub
4. Exercice 03 : Travail sur le dépôt local
5. Exercice 03 : Relier le dépôt local au dépôt GitHub
6. Exercice 03 : Collaboration
7. Exercice 03 : Création d'un Token
8. Exercice 03 : Collaboration (suite)

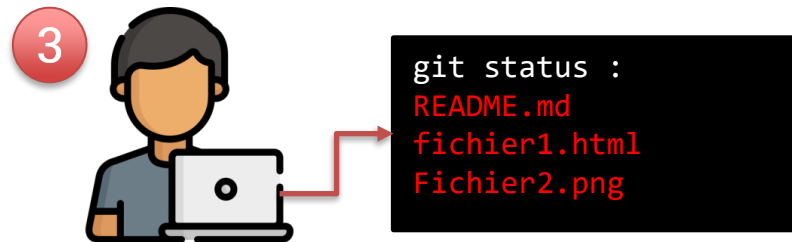
Étapes théoriques de collaboration sur GitHub



Je récupère le travail (les commits manquants) depuis GitHub (cette étape est inutile si on est absolument sûr que personne n'a fait de nouveau commit depuis la dernière fois !)



Je travaille sur le projet en local (ajout, modif, suppression de fichiers)



J'ai terminé mon travail, je vérifie que tout va bien avec git status :



J'ajoute les fichiers qui feront partie de mon futur commit :

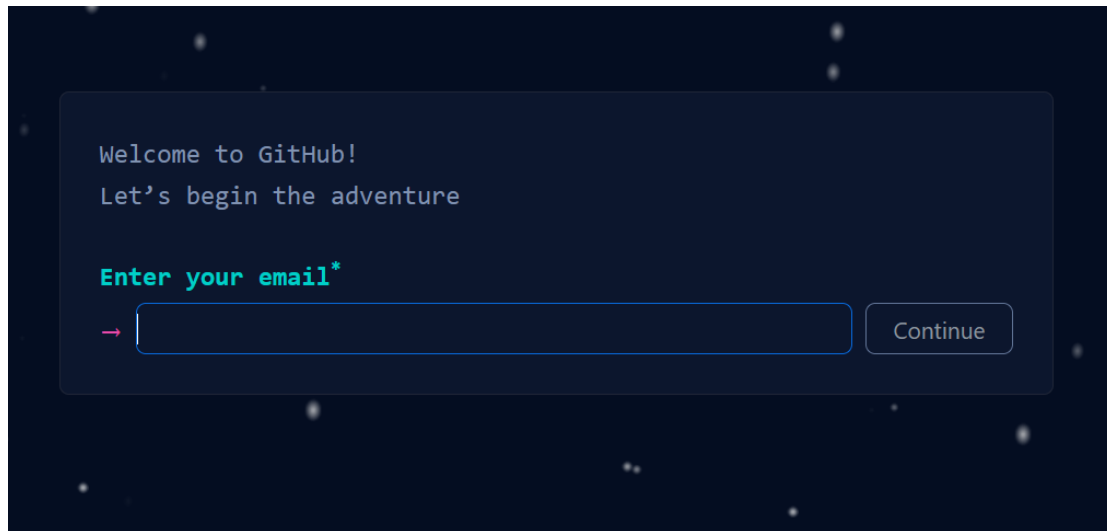


Je crée mon nouveau commit et je l'envoie sur le dépôt GitHub pour le mettre à jour :

Création de compte sur Github

Allez sur le lien pour créer votre compte GitHub :

<https://github.com/signup>



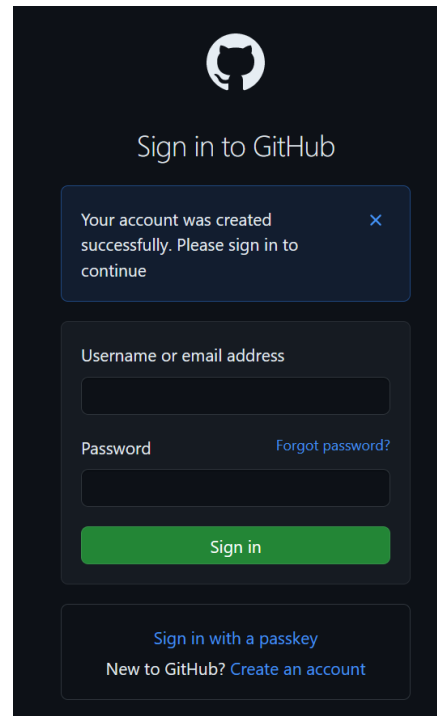
Welcome to GitHub!
Let's begin the adventure

Enter your email*

→

Continue

Suivez les étapes d'inscription jusqu'à la fenêtre vous permettant de vous connecté à votre compte. →



Sign in to GitHub

Your account was created successfully. Please sign in to continue

Username or email address

Password [Forgot password?](#)

Sign in

[Sign in with a passkey](#)
New to GitHub? [Create an account](#)

Exercice 03 : Création d'un dépôt distant

Après avoir créé votre compte sur Github, quand vous êtes connecté suivez ces étapes afin de réussir à créer votre premier dépôt distant :

- 1 – Sur GitHub, cliquer sur le bouton 'New' ou 'Create repository'
- 2 – Nommer votre dépôt 'mon-premier-depot'
- 3 – Ajouter une description si vous le désirez
- 4 – Choisissez si vous souhaitez le mettre en public ou privé
- 5 – Laissez la case décochée de add a readme file et passé directement au bouton 'create repository'
- 6 – Votre dépôt est à présent créé



Exercice 03 suite : Travail sur le Dépôt local

07 – Testons maintenant de faire notre dépôt local dans un nouveau dossier (exemple : exercice03)

08 – Dans ce dossier ouvrir git bash puis suivre la procédure pour initialiser le dépôt git local

09 – Après avoir initialiser le dépôt, créer un fichier README.md (aidez vous du cours si besoin)

10 – Faites votre premier commit




Exercice 03 suite : Relier notre dépôt local avec le dépôt distant créé

Attention quand on veut relier un dépôt local dans lequel on a travaillé sur un dépôt distant, il faut que le dépôt distant soit vide sur Github, il faudra bien avoir fait au moins un commit comme fait précédemment puis passer aux étapes suivantes :

11 – git remote add origin <https://github.com/xxxxxxxxx/test.git>
(le lien représente le lien du dépôt distant sur github à copier)

Quick setup — if you've done this kind of thing before

 Set up in Desktop

or

HTTPS

SSH

<https://github.com/> /mon-premier-depot.git

Get started by [creating a new file](#) or [uploading an existing file](#). We recommend every repository include a [README](#), [LICENSE](#), and

12 – git push -u origin main

Si vous avez fait les étapes correctement, vous devriez voir apparaître en rechargeant la page sur GitHub votre fichier README.md

Vous savez à présent comment faire pour pousser votre travail sur un dépôt distant vide



Exercice 03 suite : Collaboration

À présent changeons de point de vue, je suis à présent le collègue qui va vouloir récupérer une copie du projet à jour venant du dépôt distant, pour cela :

- 1 – je crée un nouveau dossier « exercice03-college »
- 2 – J'ouvre git bash dans ce dossier
- 3 – je tape git clone <https://github.com/xxxxxxxxx/test.git>
(le lien représente l'adresse du dépôt distant sur github à copier)

Quick setup — if you've done this kind of thing before

 Set up in Desktop or HTTPS SSH <https://github.com/mon-premier-depot.git>

Get started by [creating a new file](#) or [uploading an existing file](#). We recommend every repository include a [README](#), [LICENSE](#), and

- 4 – Il est possible qu'au moment du clone une fenêtre s'ouvre pour vous demandé de vous authentifier à votre compte gitHub
- 5- Nous allons suivre une des méthodes possibles qui est de généré un token, suivez les étapes qui suivent dans les slides suivantes pour cela



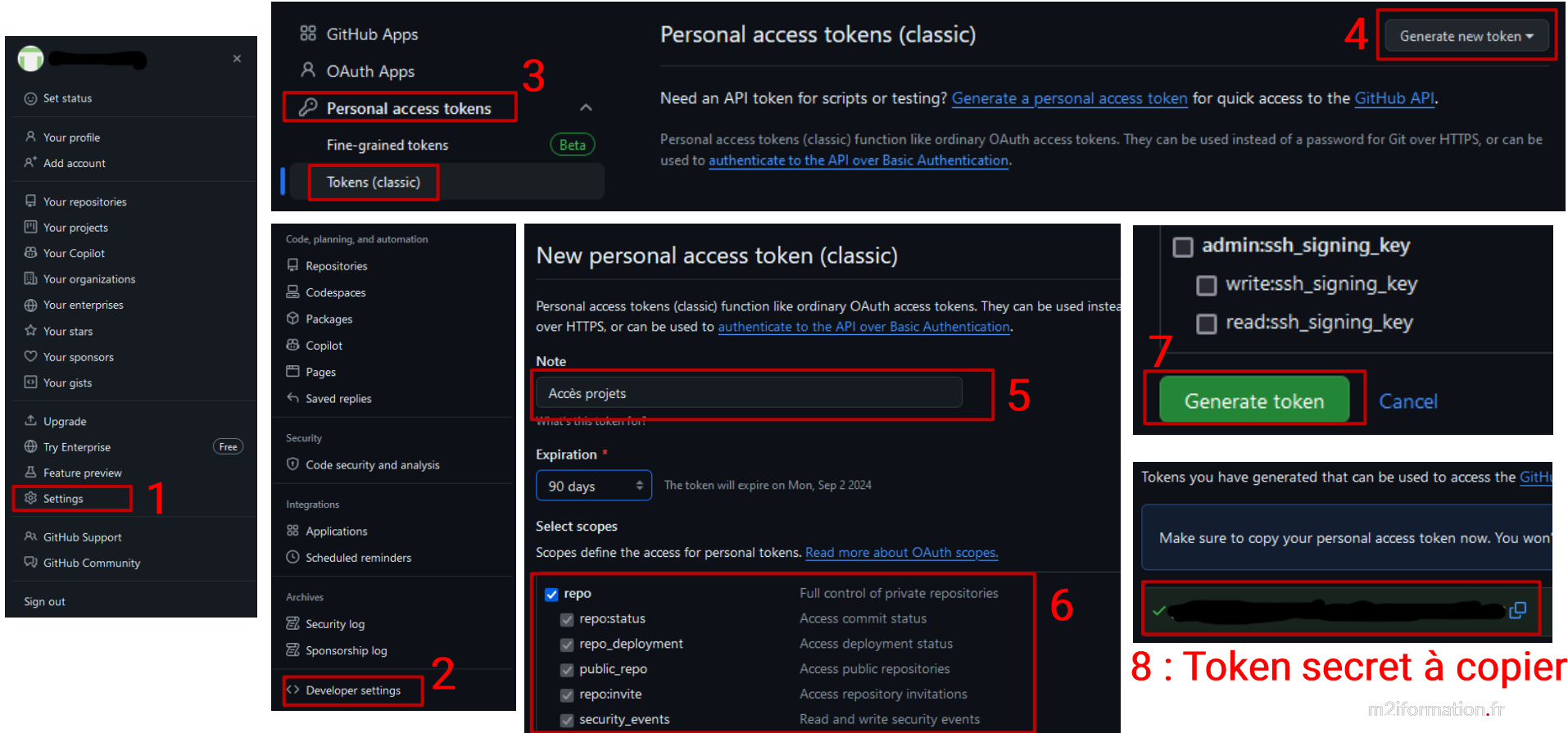
Exercice 03 suite : Création d'un Token pour l'authentification

Il y a plusieurs façon de procéder, nous allons choisir par le token que l'on va générer sur GitHub et copier dans le champ attendu.

Pour générer un nouveau Token suivre les étapes, après les étapes, copier votre Token dans la fenêtre qui vous le demandais lors de votre git clone. (slide suivante vous avez les screens des étapes à suivre)



Exercice 03 suite : Création de Token



1 Settings

2 Developer settings

3 Personal access tokens

4 Generate new token

5 Accès projets

6 repo

7 Generate token

8 : Token secret à copier

Code, planning, and automation

- Repositories
- Codespaces
- Packages
- Copilot
- Pages
- Saved replies

Security

- Code security and analysis

Integrations

- Applications
- Scheduled reminders

Archives

- Security log
- Sponsorship log
- Developer settings

GitHub Apps

OAuth Apps

Personal access tokens (classic)

Need an API token for scripts or testing? [Generate a personal access token](#) for quick access to the [GitHub API](#).

Personal access tokens (classic) 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](#).

Fine-grained tokens Beta

Tokens (classic)

New personal access token (classic)

Personal access tokens (classic) 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

Accès projets

What's this token for?

Expiration *

90 days The token will expire on Mon, Sep 2 2024

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

☐ admin:ssh_signing_key

☐ write:ssh_signing_key

☐ read:ssh_signing_key

Generate token Cancel

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

Make sure to copy your personal access token now. You won't be able to do this later.

Token secret à copier

Exercice 03 suite : Collaboration

À présent une fois que le clone s'est effectué, je vais pouvoir constater que j'aurais dans mon dossier la copie du projet distant avec le fichier README.md, maintenant je vais créer un fichier.

6 – Dans le dossier de mon projet local « exercice03>mon-premier-depot », je crée un fichier nommé « index.html » dans lequel je vais écrire une ligne « préparation du fichier index.html »

7 – Je répète les étapes que j'ai déjà vu pour ajouter, commit et pousser mon travail sur le dépôt distant

8 – Si j'ai fait les étapes correctement je devrais voir apparaître sur le dépôt distant mon fichier index.html

Vous avez donc vu comment copier un projet distant en local, travailler dessus et le mettre à jour sur le dépôt distant





Ressources

Quelques ressources

Site de Git : <https://git-scm.com/>

Documentation de Git fr : <https://git-scm.com/book/fr/v2>

Site de GitHub : <https://github.com/>

Documentation de GitHub fr : <https://docs.github.com/fr>

Mise en forme sur GitHub : <https://docs.github.com/fr/get-started/writing-on-github>

Autres commandes Git (bonus pour aller plus loin)



Annuler des changements si besoin :

- ❑ **Git reset** : Annule les changements en revenant à un état antérieur en modifiant l'historique des commits locaux, avant de les avoir poussés vers un dépôt distant. C'est donc comme si le(s) commit(s) n'avai(en)t jamais eu lieu.

```
bash
```

```
git reset <mode> <id_de_commit> #3 modes possible : --soft, --mixed et --hard
```

- ❑ **Git revert** : Annule les modifications apportées par un commit spécifique, mais de manière sûre pour les autres développeurs. Au lieu de supprimer ou de modifier l'historique des commits, git revert crée un nouveau commit qui inverse les changements du commit spécifié (exemple : si j'ai ajouté une ligne à un fichier, le revert va la supprimer).

Petit + : Le commit de réversion est visible dans l'historique, ce qui montre clairement quand et pourquoi les changements ont été annulés contrairement à reset qui réécrit l'historique.

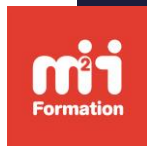
```
bash
```

```
git revert <id_de_commit>
```

- ❑ **Git stash** : Met de côté (stocke) les modifications locales non commit pour revenir à un état de travail propre. Vous pouvez réappliquer ces modifications plus tard. Cela peut arriver quand par exemple on a besoin de changer de branche sans perdre les modifications non commits en cours.

```
bash
```

```
git stash
git stash apply #pour réappliquer les modifications stashed
git stash list  #pour voir les stashes disponibles
git stash pop   #pour appliquer et supprimer le stash
```



Fin du module



m2information.fr