

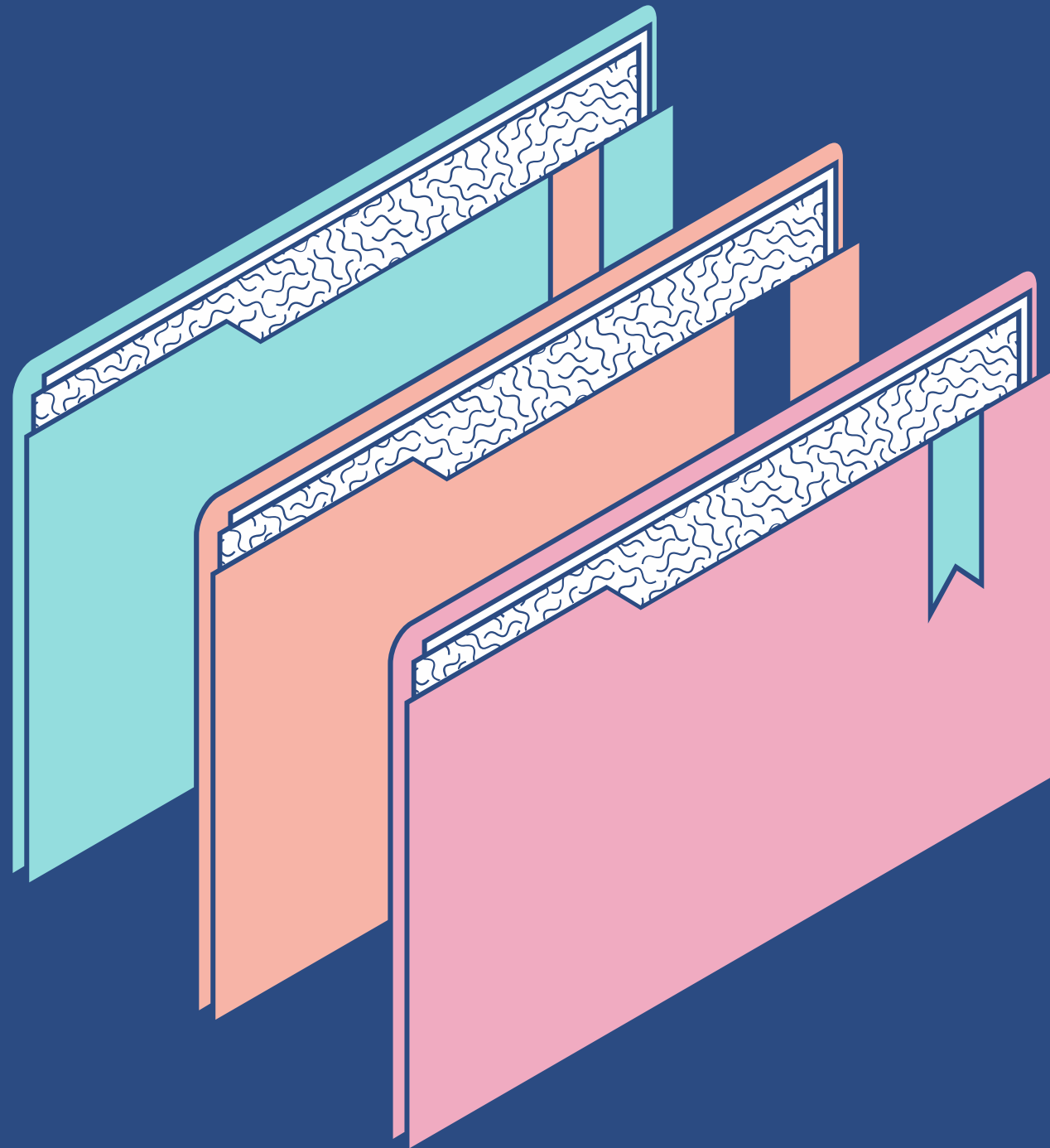


GESTION DES SOURCES ET VERSIONNING

GIT

SOMMAIRE

E



- Introduction aux lignes de commandes
 - Pourquoi utiliser les lignes de commandes ?
 - Présentation des commandes élémentaires
- Découverte du fonctionnement de GIT
 - Présentation et configuration
 - Commandes élémentaires
 - Système de branches
 - Correction de problèmes
- Présentation GIT GUI et GIT VSC
 - GIT GUI
 - GIT VSC



Pourquoi utiliser les lignes de commande ?

Quels sont les différents systèmes d'exploitation présents sur ordinateur, et leurs spécificités ?



Pourquoi utiliser les lignes de commande ?

Il existe 3 systèmes d'exploitation sur ordinateur :

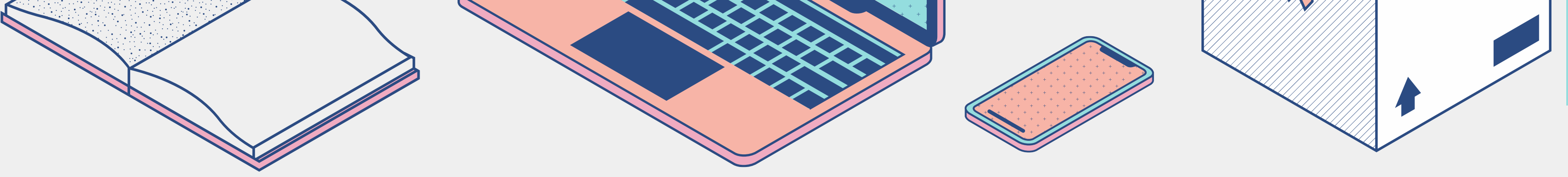
- Windows
 - Système le plus populaire, compatibilité presque totale
 - Nombreuses failles de sécurité, prix élevé
- Mac OS
 - Bonne sécurité
 - Faible compatibilité, nécessite un ordinateur MAC, prix élevé
- Linux
 - Gratuit, très sécurisé, open source
 - Peu répandu en dehors du monde informatique, faible compatibilité



Pourquoi utiliser les lignes de commande ?

Dans le monde informatique, un grand nombre d'applications et de serveurs sont basés sur Linux.

La complexité d'usage n'est pas un frein pour les informaticiens, qui tirent profit de la rapidité d'exécution et de la modularité de ce langage.



Pourquoi utiliser les lignes de commande ?

Qu'est ce qu'une ligne de commande, et pourquoi l'utiliser ?



Pourquoi utiliser les lignes de commande ?

Les lignes de commandes servent à interagir avec un système informatique

Historiquement, les lignes de commandes étaient la seule façon d'interagir avec les ordinateurs. Aujourd'hui, l'immense majorité des interactions homme-machine s'effectuent à l'aide d'interfaces graphiques.

Cependant, en informatique, un certain nombre de fonctionnalités ne sont toujours possibles qu'en lignes de commandes (administration serveurs linux, configurations réseaux ...)

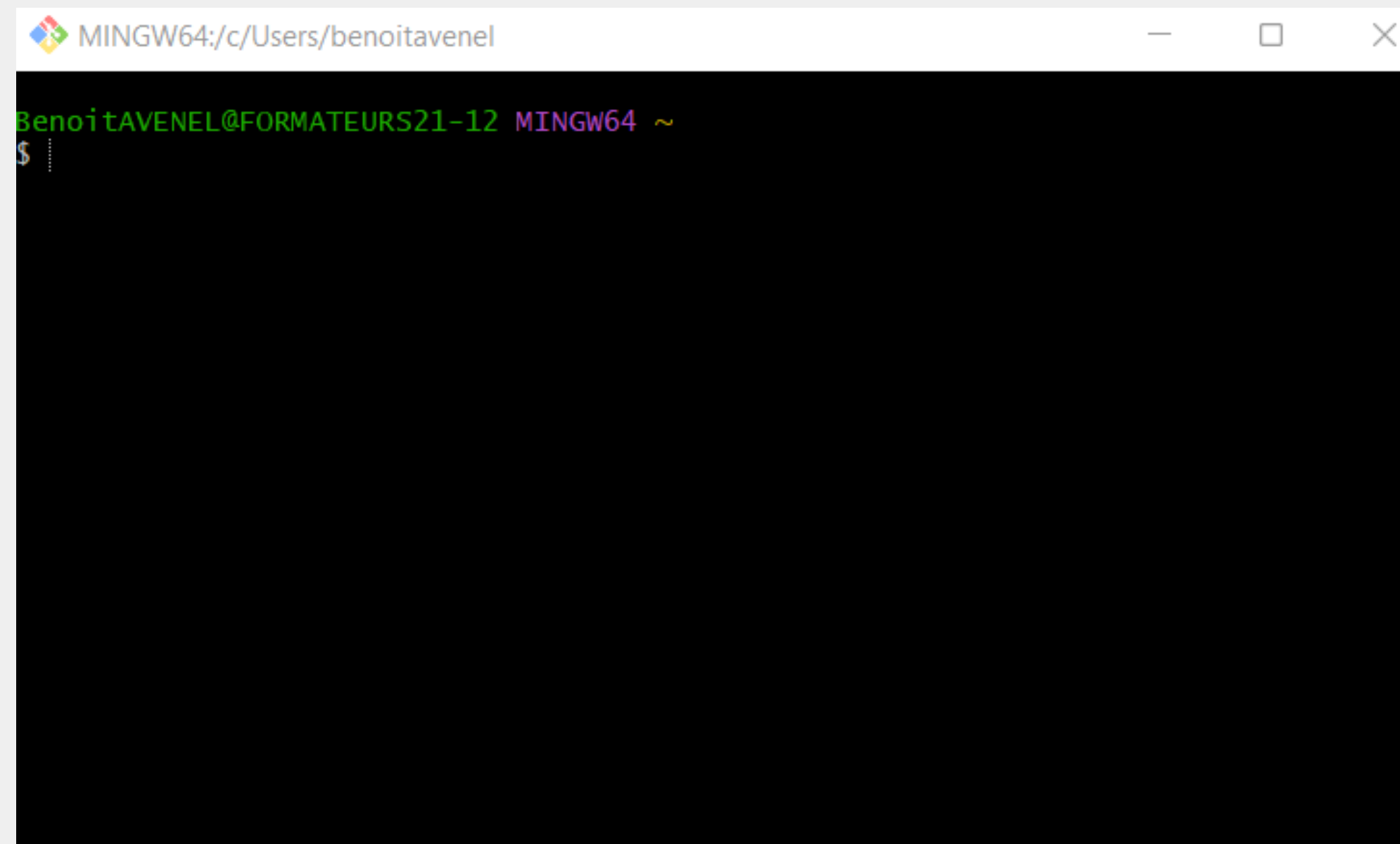


Pourquoi utiliser les lignes de commande ?



<https://git-scm.com/downloads>

Pourquoi utiliser les lignes de commande ?



```
MINGW64:/c/Users/benoitavenel

BenoitAVENEL@FORMATEURS21-12 MINGW64 ~
$
```



Présentation des commandes élémentaires :

Navigation dans les fichiers

ls : Liste le contenu d'un dossier

Options utiles : -a affiche les fichiers cachés, -R affiche les sous-dossiers

cd : Change de répertoire

- Soit en navigation absolue (ne dépend pas du dossier courant) : `cd /c/ProgramData`
- Soit en navigation relative (en fonction du dossier courant) : `cd ProgramData`
- Retour au dossier précédent : `cd ..`
- Accéder au dossier de l'utilisateur courant : `cd ~`

pwd : Affiche le chemin absolu du dossier courant

clear : Nettoie la fenêtre du terminal



Présentation des commandes élémentaires :

Opération sur les fichiers

mkdir nom_du_dossier : créé un nouveau dossier

touch nom_du_fichier : créé un nouveau fichier

cat nom_du_fichier : affiche le contenu d'un fichier texte

cp fichier_original copie_fichier : permet de copier un fichier

cp -R : permet de copier un dossier

nb : il est possible de copier directement les fichiers et dossiers dans un autre dossier.

La syntaxe sera : cp fichier_original nouveau_dossier/copie_du_fichier



Présentation des commandes élémentaires :

Opération sur les fichiers

rm nom_du_fichier : supprime un fichier
options : -f force la suppression
-i demande une confirmation

rm -R nom_du_dossier : supprime un dossier

mv nom_du_fichier : déplace un fichier

mv -R nom_du_dossier : déplace un dossier

rem : les commandes mv et cp fonctionnent de manière analogues !



Présentation des commandes élémentaires :

Ecrire dans un fichier

<i>vi nom_du_fichier</i>	: permet d'ouvrir un fichier pour éditer son contenu
<i>i</i>	: permet l'insertion de contenu au sein du fichier
<i><esc></i>	: permet l'utilisation de commandes au sein du fichier
<i>:wq!</i>	: permet de sauvegarder et quitter un fichier (en mode commande)
<i>:q!</i>	: permet de quitter un fichier SANS sauvegarder(en mode commande

rem : vi possède une multitude de commandes !

Vous pouvez en trouver une partie sur : https://doc.fedora-fr.org/wiki/Utilisation_de_vi



Présentation des commandes élémentaires :

Raccourcis clavier

- ctrl + s** : Permet de stopper l'affichage
- ctrl + q** : Permet de reprendre l'affichage
- ctrl + d** : Permet de déconnecter proprement une session
- ctrl + u** : Permet de couper la ligne courante
- ctrl + y** : Permet de coller le contenu coupé
- ctrl + c** : Permet de stopper l'exécution d'une commande
- ↑** : Permet de remonter l'historique des commandes
- <tab>** : Permet de compléter la fin d'une commande



Introduction aux lignes de commandes

Exercice pratique

TP1 GIT

Découverte du fonctionnement de GIT

Présentation de GIT



Qu'est ce que GIT ?

GIT est un outil de gestion de projets et de versions collaboratif.

C'est un logiciel libre de droit créé autour du noyau linux.

Avec plus de 36 millions d'utilisateurs dans le monde, GIT est le logiciel de gestion de versions le plus utilisé dans le monde.



GIT est un outil incontournable pour le développement en team au sein du monde professionnel

Configuration de GIT

18

git config --global user.name "nom_utilisateur" : Enregistrement du login

```
BenoitAVENEL@FORMATEURS21-12 MINGW64 ~  
$ git config --global user.name "Benoît Avenel"
```

git config --global user.email "adresse_mail" : Enregistrement du mail

```
BenoitAVENEL@FORMATEURS21-12 MINGW64 ~  
$ git config --global user.email "benoitavenel@adrrar-formation.com"
```

git config --global init.defaultBranch main : Change la branche par défaut

```
BenoitAVENEL@FORMATEURS21-12 MINGW64 /c/dev/git (main)  
$ git config --global init.defaultBranch main
```

git config --list : Permet de vérifier sa configuration

rem : les "" sont indispensables pour les saisies composées.

Création d'un projet GIT



git init : Création d'un nouveau projet GIT

Cette commande est à effectuer dans chaque nouveau dossier à connecter à GIT

```
BenoitAVENEL@FORMATEURS21-12 MINGW64 /c/dev/git  
$ git init
```

Cette commande générera un dossier caché .git au sein du dossier sélectionné

```
BenoitAVENEL@FORMATEURS21-12 MINGW64 /c/dev/git (master)  
$ ls -a  
./ ../ .git/
```

Configuration github

20

Se connecter sur GitHub :
<https://github.com>

nb : ne pas oublier de valider le compte par email !

Welcome to GitHub!
Let's begin the adventure

Enter your email

✓ benoitavenel@adrar-formation.com

Create a password

✓

Enter a username

✓ benoitavenel

Would you like to receive product updates and announcements via email?

Type "y" for yes or "n" for no

→ n

Continue

Création d'un repository GitHub

nb : toujours cocher l'option "Add a README file" !

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere?

[Import a repository.](#)

Owner *

 benoitavenel ▾

Repository name *


monPremierRepo ✓

Great repository names are short and memorable. Need inspiration? How about [silver-octo-disco?](#)

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.](#)

.gitignore template: None ▾

Choose a license

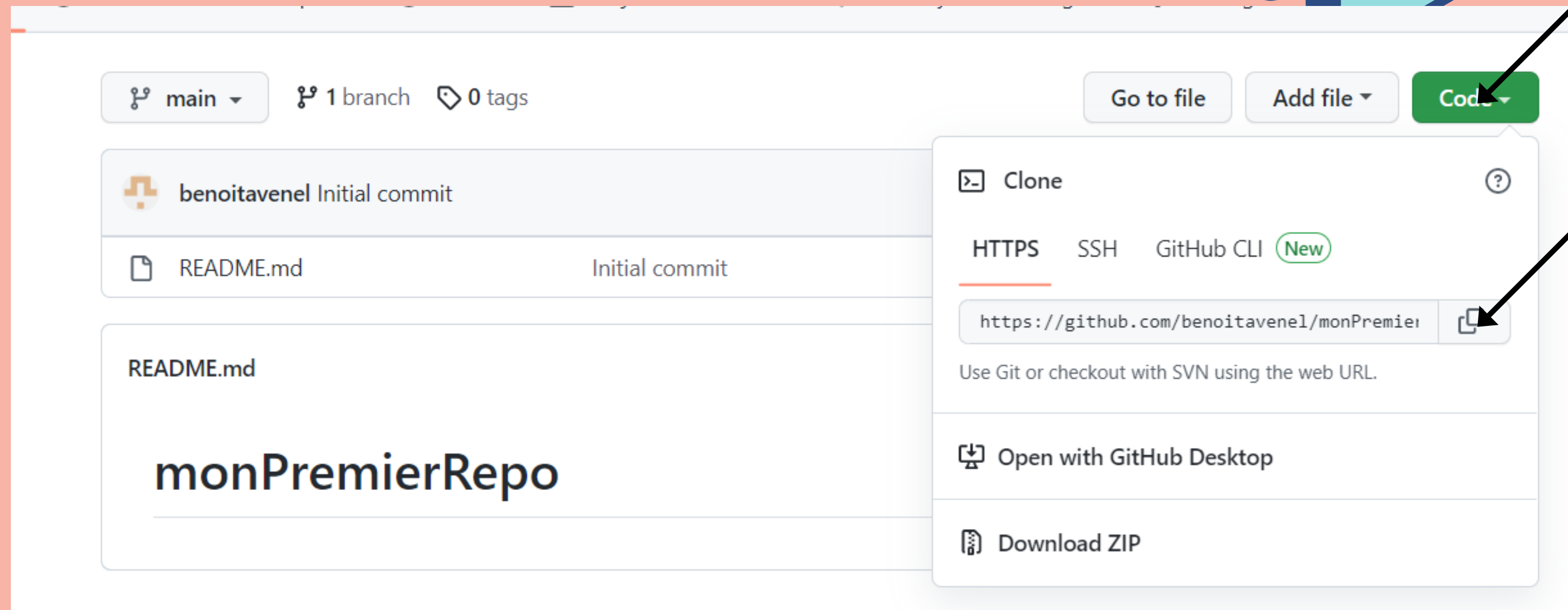
A license tells others what they can and can't do with your code. [Learn more.](#)

License: None ▾

21

cloner un dépôt GitHub

22



cloner un dépôt GitHub



23

git remote add origin lien_du_depot_git : liaison entre le dépôt GitHub et le dépôt Local

```
BenoitAVENEL@FORMATEURS21-12 MINGW64 /c/dev/git (master)
$ git remote add origin https://github.com/benoitavenel/monPremierRepo.git
```

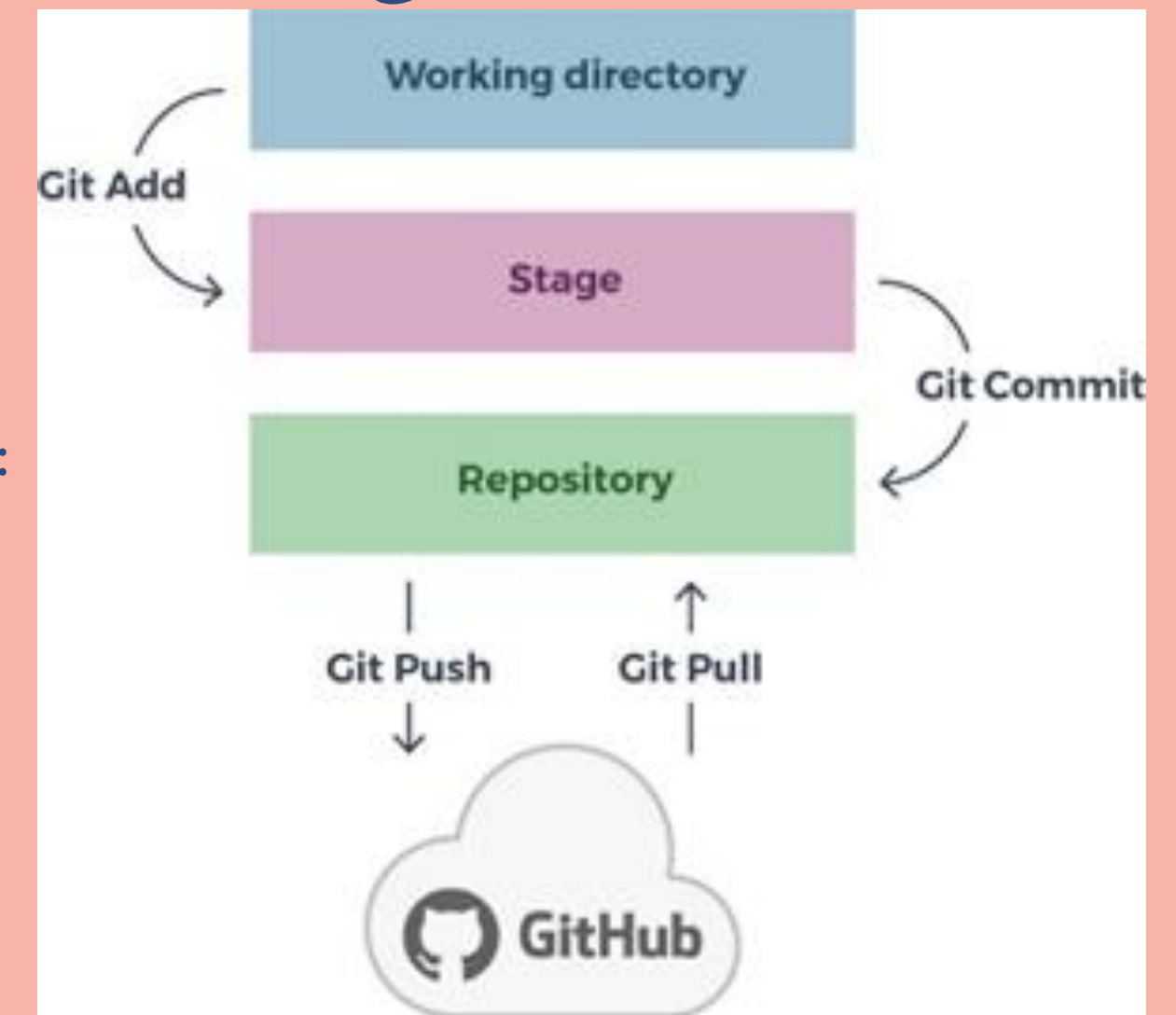
git pull lien_du_depot_git : clone les données GitHub sur le dépôt Local

```
BenoitAVENEL@FORMATEURS21-12 MINGW64 /c/dev/git (master)
$ git pull https://github.com/benoitavenel/monPremierRepo.git
```

nb : Attention à bien avoir effectué la commande git init au préalable !

Fonctionnement de la sauvegarde

24



Git gère les versions de nos travaux locaux à travers 3 zones majeures :

- Le répertoire de travail (working directory/WD) ;
- L'index, ou stage (File d'attente) ;
- Le dépôt local (Git directory/repository commit).

Fonctionnement de la sauvegarde

25

git add nom_du_fichier : ajout d'un fichier dans le stage

rem : les commandes `git add -A`, `git add .` et `git add *` permettent d'ajouter tous les fichiers dans le stage

```
BenoitAVENEL@FORMATEURS21-12 MINGW64 /c/dev/git (master)  
$ git add index.html
```

git commit -am "message de la modification" : sauvegarde des modifications

rem : il est possible d'utiliser uniquement la commande `git commit`, et d'éditer le message à l'aide de `vi`

```
BenoitAVENEL@FORMATEURS21-12 MINGW64 /c/dev/git (master)  
$ git commit -am "Ajout de l'index"
```

git push : envoi des modifications sur le serveur

```
BenoitAVENEL@FORMATEURS21-12 MINGW64 /c/dev/git (main)  
$ git push
```

rem : Lors du premier envoi, il sera nécessaire de valider les informations de connexion à GitHub

Découverte de GIT

26



Exercice pratique

TP2 GIT

Découverte de GIT



Mise en place du repo GIT

https://github.com/benoitavenel/structure_dev.git

Fonctionnement des branches

28

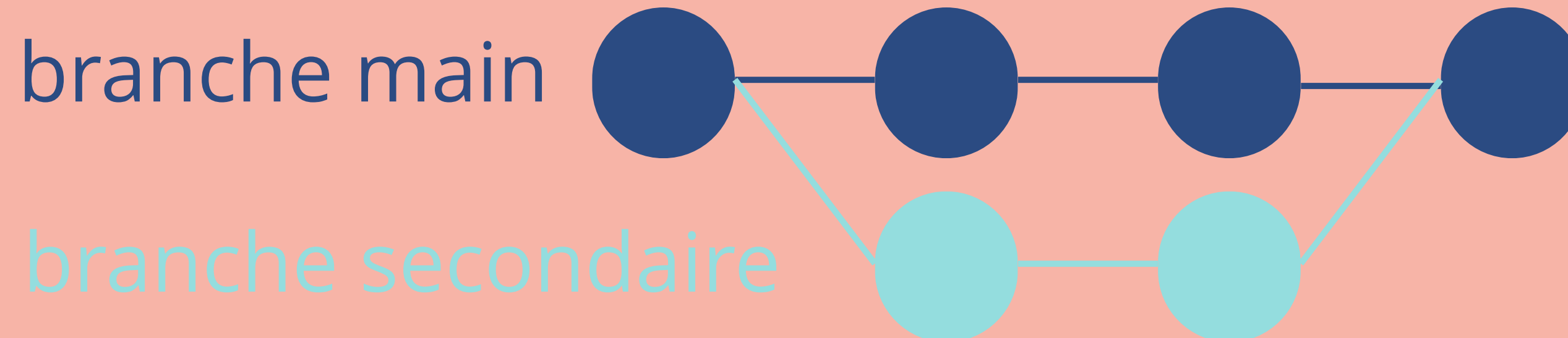


La principale force de GIT est son fonctionnement en branches.

Les branches permettent de stocker différentes versions d'une même application, de façon sécurisé et totalement opérationnel.

La branche principale de l'application est la branche "main", anciennement "master".

Au fûr et à mesure du développement, il sera possible de créer autant de branches que nécessaires, que nous fusionnerons ensuite à la branche master.



Fonctionnement des branches

29



git branch : affiche la liste des branches du projet

rem : la branche actuellement connectée est affichée entre parenthèses à côté de l'invité de commandes

```
BenoitAVENEL@FORMATEURS21-12 MINGW64 /c/dev/git (main)
$ git branch
* main
master
```

git branch nom_de_la_branche : crée une nouvelle branche

git checkout nom_de_la_branche : se connecte sur une nouvelle branche

git branch -d nom_de_la_branche : supprime une branche

Fusionner des branches

La fusion de branches permet de récupérer les fonctionnalités d'une branche secondaire sur une branche principale.

Il faut faire très attention lors de cette étape, pour ne pas inverser le processus, et revenir en arrière sur les modifications.

Assurez vous d'être sur la bonne branche, en effectuant un git status, et au besoin un git checkout.

Se positionner sur la branche à merge et merger la branche qui reçoit les modifications

git merge nom_branche_reçoit_merge : Merge la branche SUR LAQUELLE NOUS SOMMES CONNECTES sur la branche indiquée

```
BenoitAVENEL@FORMATEURS21-12 MINGW64 /c/dev/git (dev)  
$ git merge main
```

: merge branche dev sur la branche main

rem : Si un fichier a été modifié sur les 2 branches qui fusionnent, il faudra résoudre les conflits manuellement



Corriger une erreur



git revert HEAD^ : permet d'annuler le dernier push

*rem : un fichier vi s'ouvre vous proposant d'ajouter un message spécifique pour l'annulation du push.
Une fois le git revert HEAD effectué, il faut effectuer un push des modifications vers le serveur*

git log : permet de visualiser l'historique des push

id du commit

```
commit bb3dbfbbec9988a292acdf9cd760a30717aa16a9  
Author: benoitavenel <113111323+benoitavenel@users.noreply.github.com>  
Date: Thu Sep 8 15:19:42 2022 +0200
```

git revert id_commit : permet de supprimer les modifications après l'id du commit saisi

rem : Une fois le git revert effectué, il faut effectuer un push des modifications vers le serveur

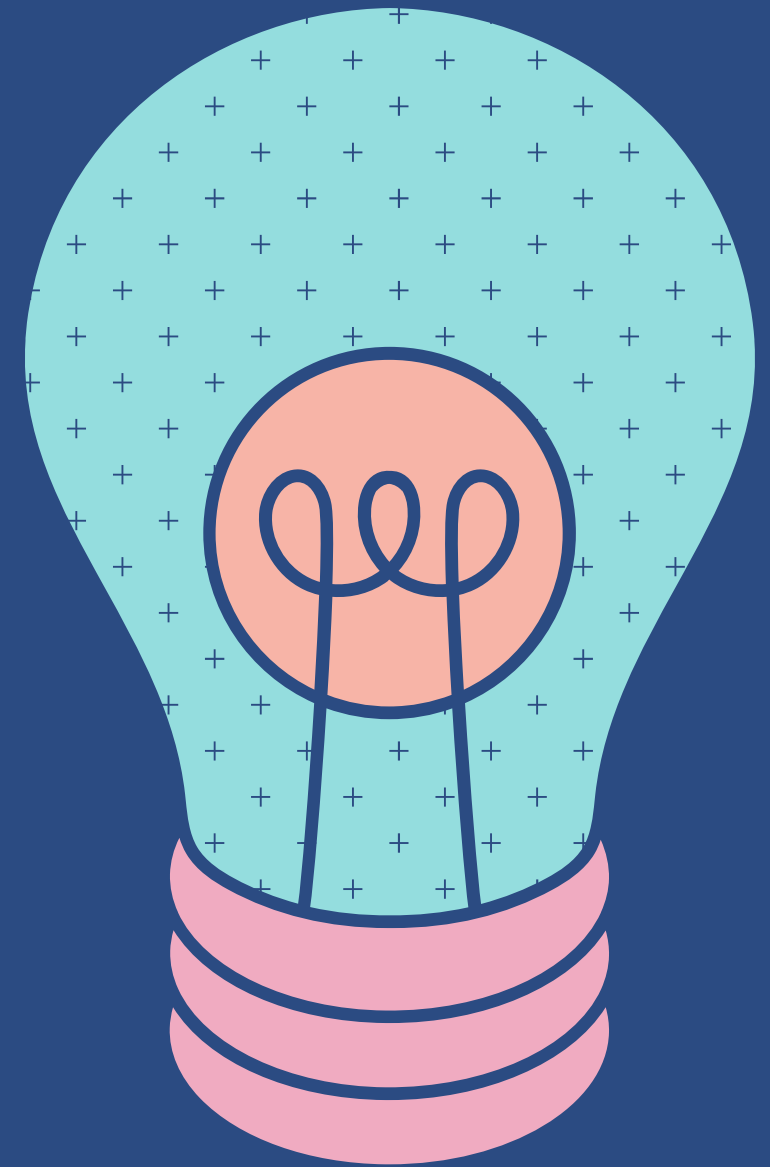
Découverte de GIT



Exercice pratique

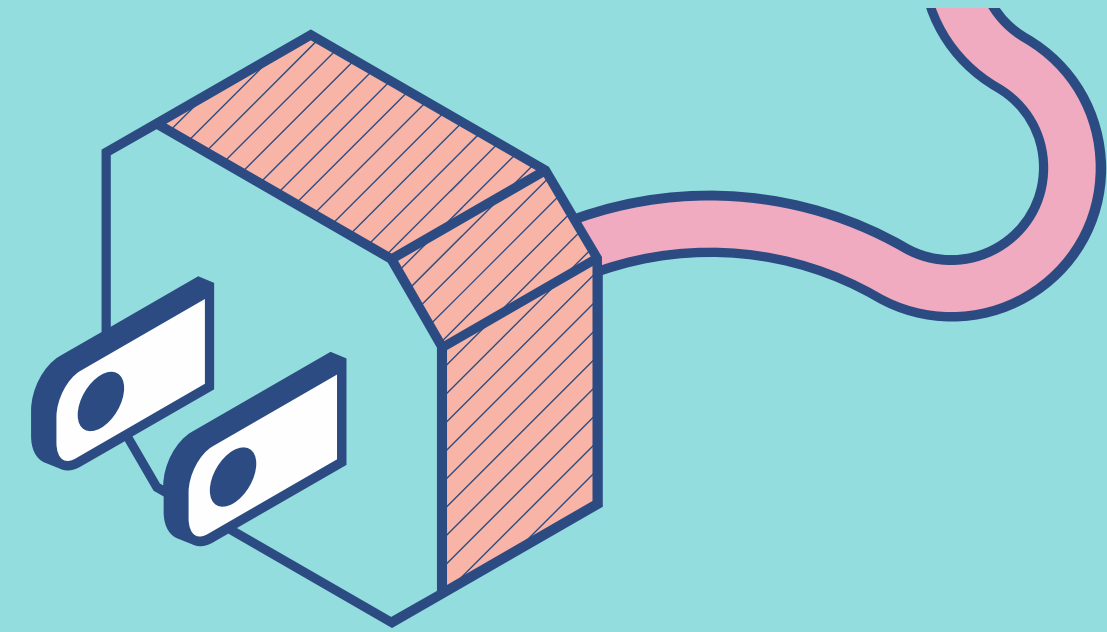
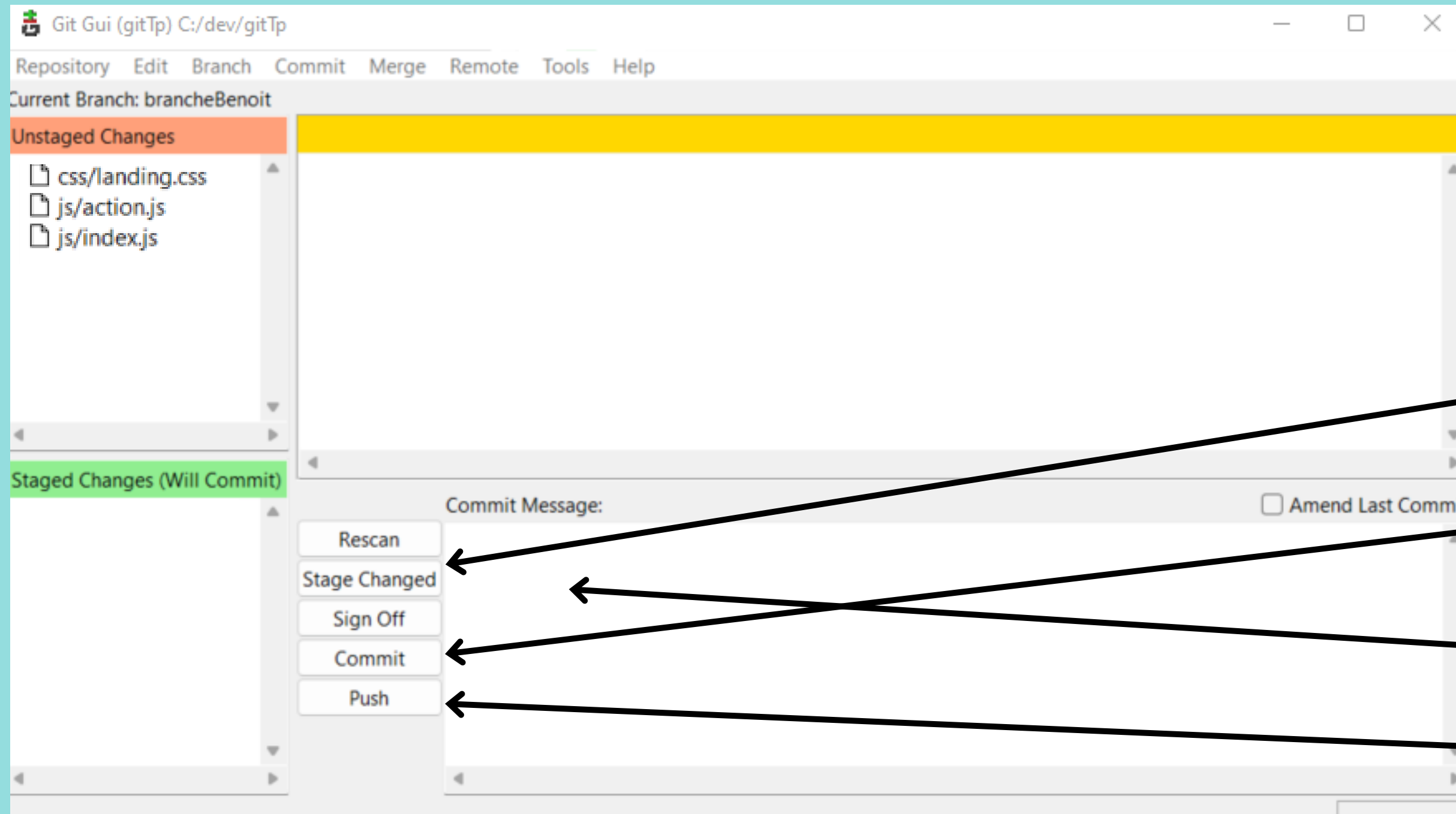
TP3 GIT

Evaluation GLT



Solutions alternatives

GIT GUI



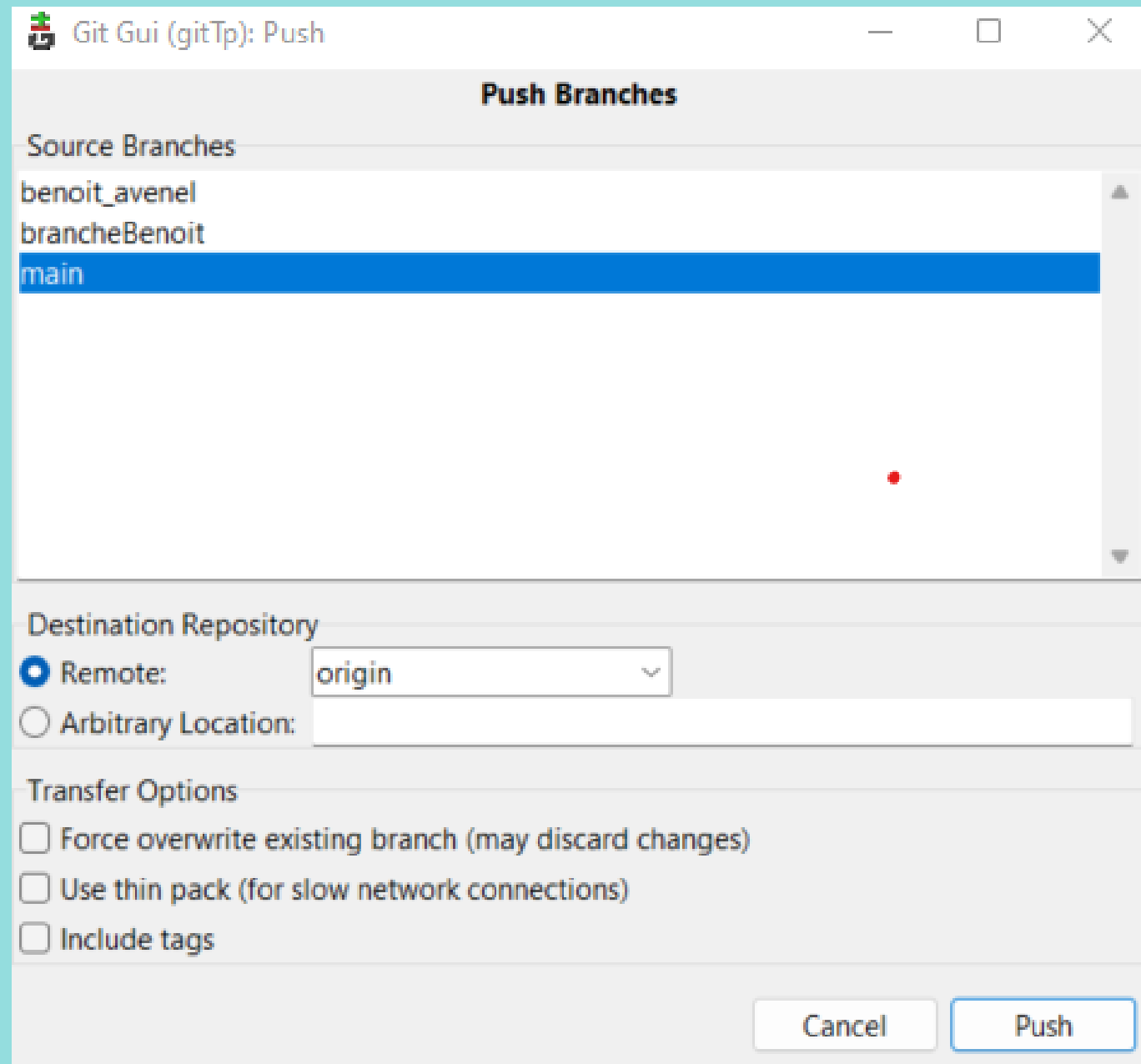
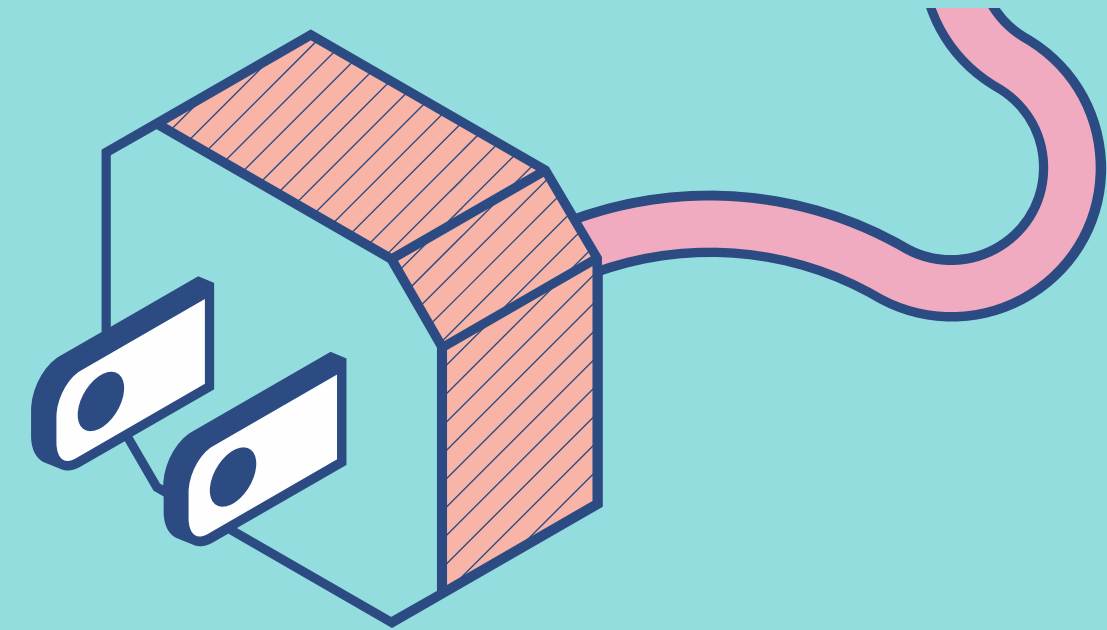
git add .

git commit

-m ""

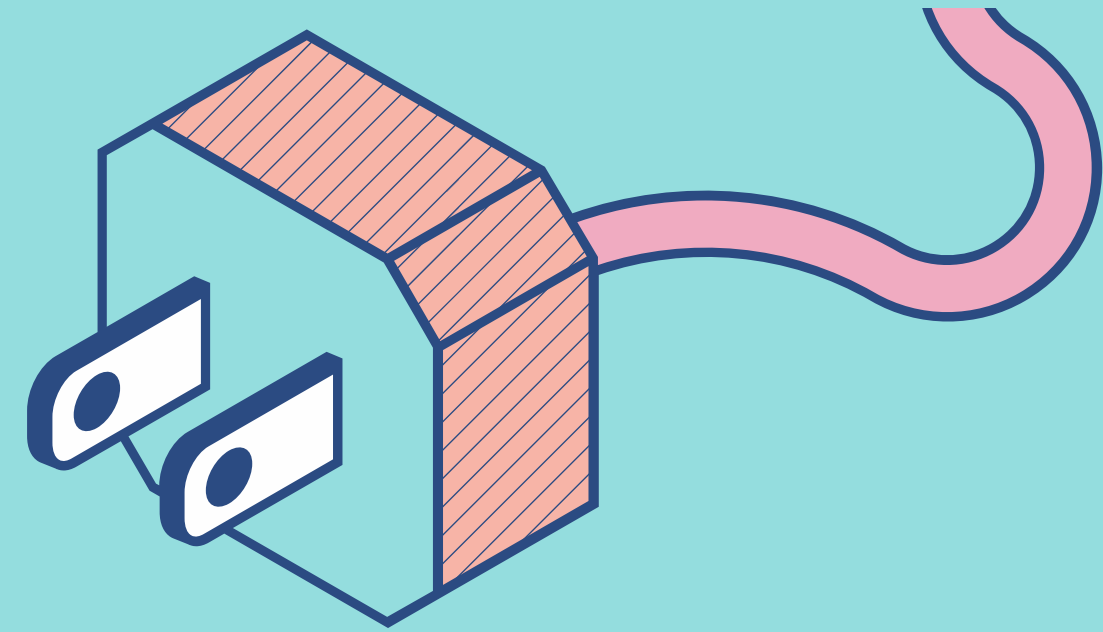
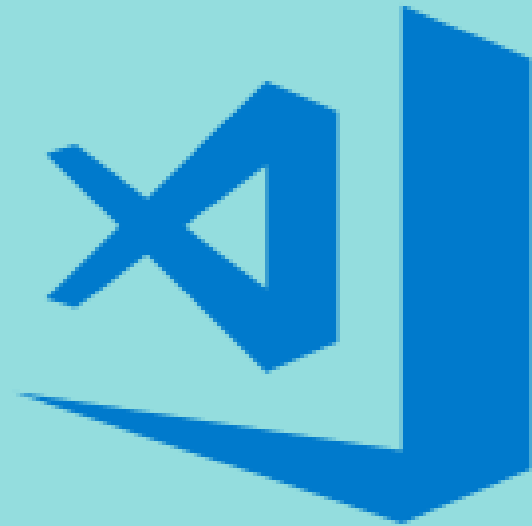
git push

GIT GUI



Pensez à modifier la branche sur laquelle vous souhaitez effectuer votre push avant validation !

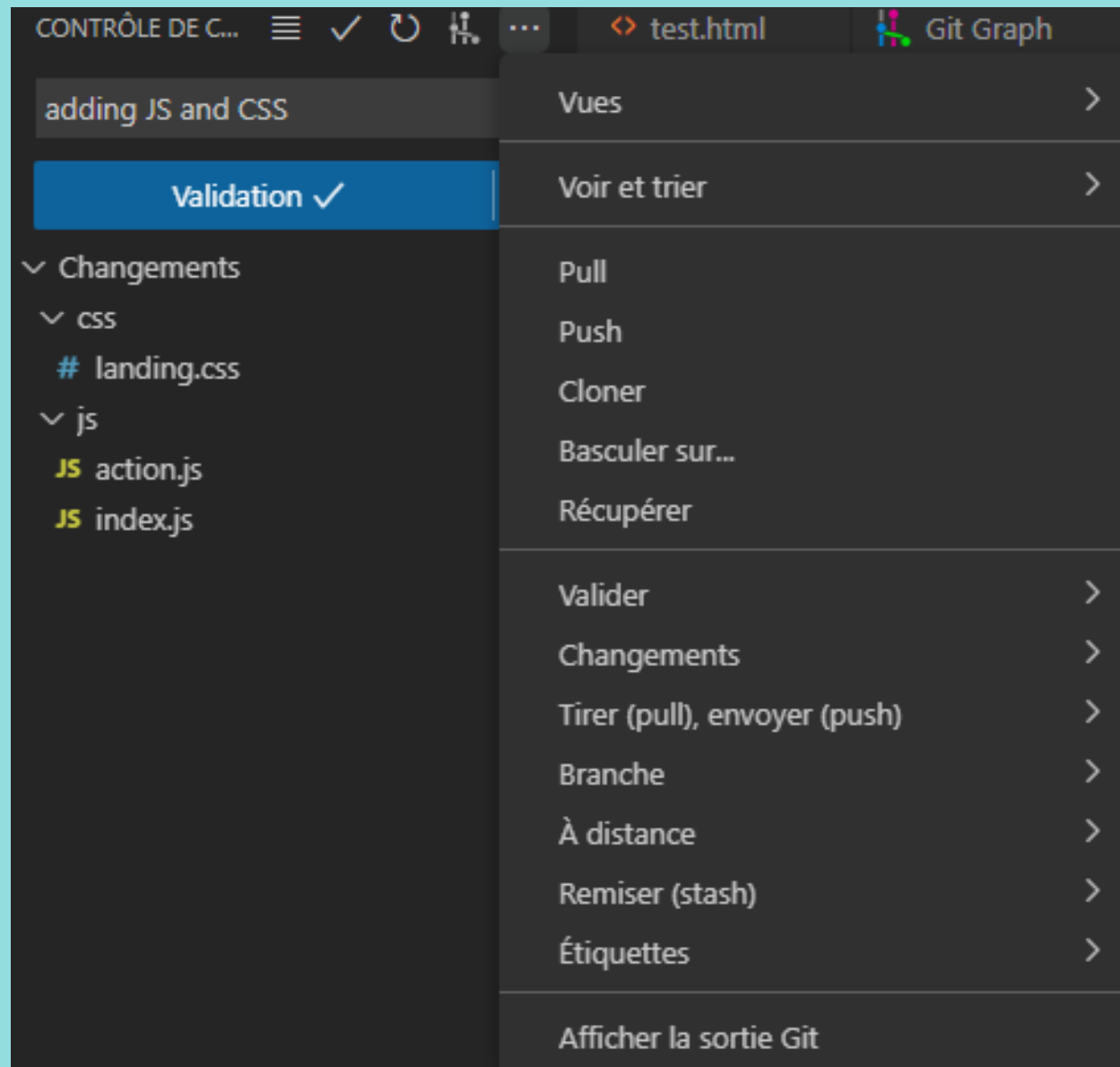
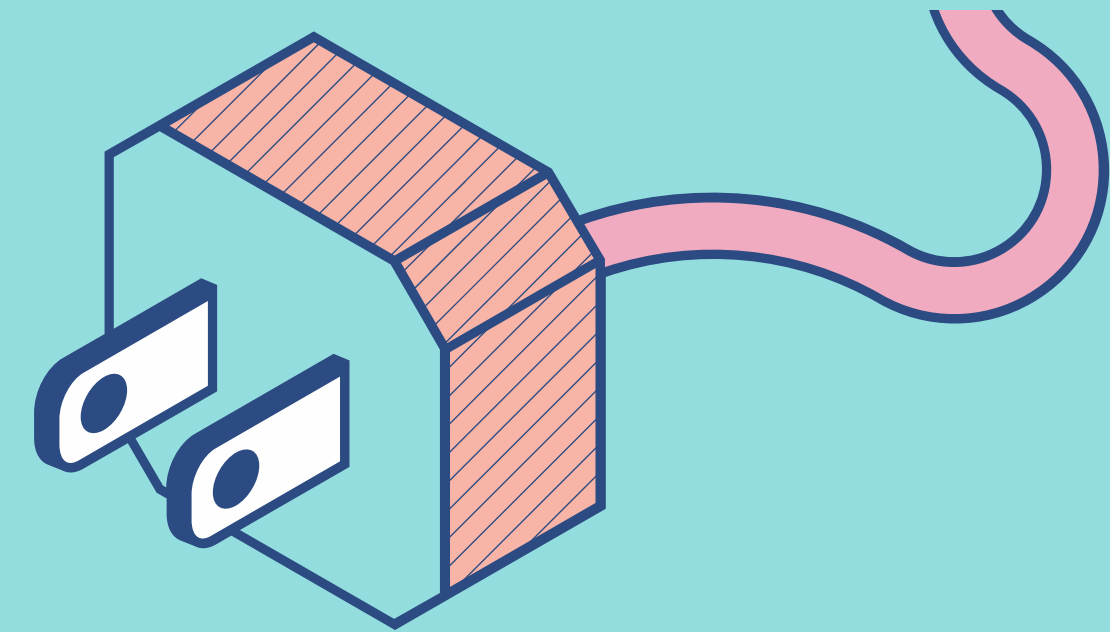
GIT VSC



Visual Studio Code

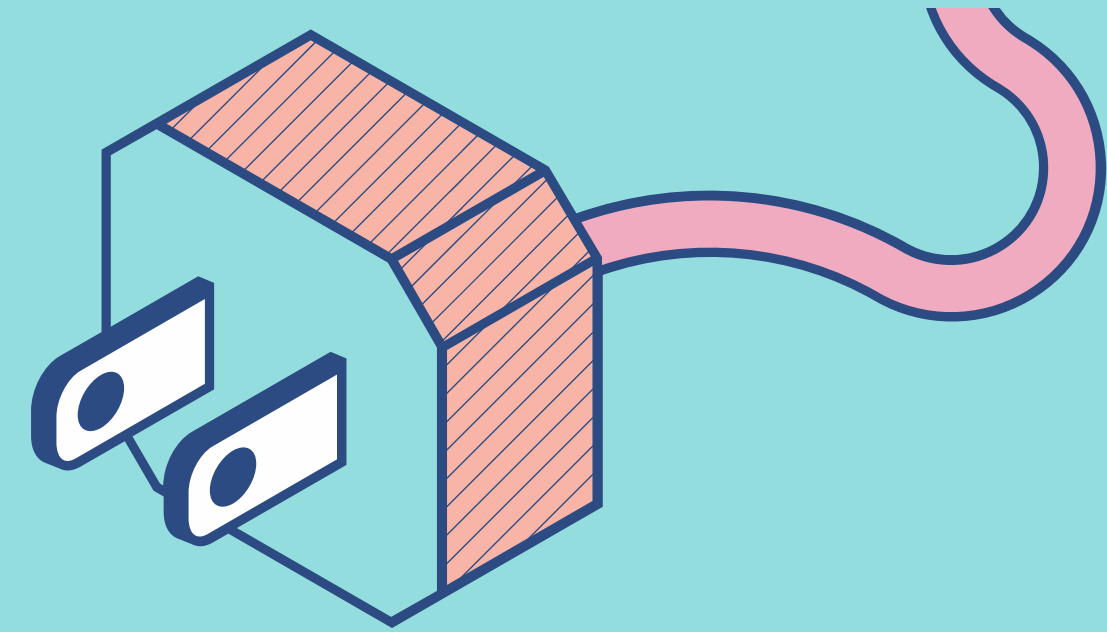
<https://code.visualstudio.com/download>

GIT VSC



Visual Studio Code (VSC) permet d'effectuer l'ensemble des opérations via une interface graphique directement depuis l'éditeur

Git Graph



Git Graph - gitTp - Visual Studio Code

Branches: Show All ☒ Show Remote Branches

Graph	Description	Date	Author	Commit
Uncommitted Changes (3)		12 Sep 2022 12:22	*	*
○ brancheBenoit origin	ajout index css	12 Sep 2022 12:16	benoitavenel	fb94cca8
■ benoit_avenel origin	test html	12 Sep 2022 12:15	benoitavenel	a8bf0628
	commit h2	12 Sep 2022 12:06	benoitavenel	4cd779d7
	ajout de paragraphe	12 Sep 2022 12:04	benoitavenel	486822bd
	ajout de l'index	12 Sep 2022 12:03	benoitavenel	0d399f22
■ main origin	Initial commit	12 Sep 2022 12:01	benoitavenel	3dee22b9