

Découverte et initiation à Git et GitHub

Elise Martin & Blandine Serres



git



PRESENTATION

Elise MARTIN :



- **M2 Gphy**
- **Option Imagerie**

Blandine SERRES :



- **M2 Gphy**
- **Option Biotechnologie**

=> Alternantes chez AmeXio Group (Nantes & Paris)

SOMMAIRE

- 1. Présentation de Git**
- 2. Présentation de GitHub**
- 3. Gestion des conflits**
- 4. Bonnes pratiques sur Git**
- 5. GitHub et BlueJ**

Quiz - Google Form



<https://forms.gle/xkLkke5GKf4NhAfc7>

1. Présentation de Git

a. Qu'est-ce que Git ?

b. Concepts de base

c. Bilan sur Git



Qu'est-ce que Git ?

= Système de contrôle de version distribué

- Créé par Linus Torvalds en 2005.
- Essentiel pour suivre les modifications apportées aux fichiers au cours du temps (versioning)
 - => Qui ?
 - => Quoi ?
 - => Quand ?

Qu'est-ce que Git ?

Versioning



Dépôt (repository) = espace de stockage des fichiers versionnés

Qu'est-ce que Git ?

Pourquoi Git ?



Collaboration

Plusieurs développeurs
peuvent travailler
simultanément



Traçabilité

Historique des
modifications



Sécurité

Restauration des
versions précédentes



Compréhension

Documentation mise à
jour à chaque
modification envoyée

- ✓ **Meilleur suivi des versions**
- ✓ **Meilleure coordination**
- ✓ **Développement structuré**

Installation de Git



1. Windows :

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



2. Linux/Unix (ex : Debian/Ubuntu) :

`sudo apt-get install git`



3. macOS (Homebrew requis) :

`brew install git`

=> Pour installer Homebrew :

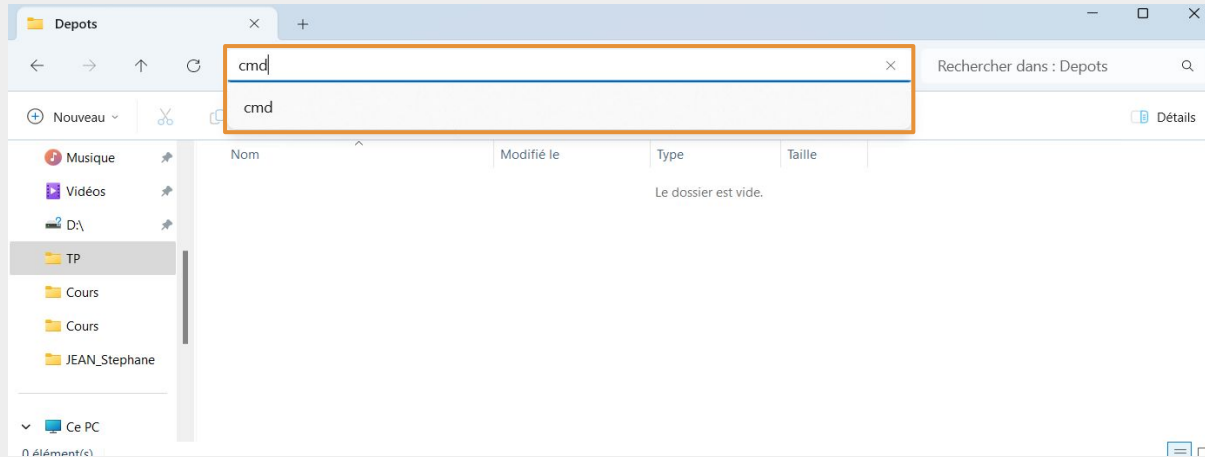
```
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

Mise en place d'un répertoire en local

Créer un répertoire

Stockage des
projets

1. Ouvrir un terminal



Mise en place d'un répertoire en local

Créer un répertoire

1. Ouvrir un terminal
2. Créer un dossier : `mkdir repLocal`
3. Se placer dans le répertoire : `cd repLocal`
4. Créer un dossier : `mkdir MonProjet`
5. Se placer dans le répertoire : `cd MonProjet`

Mise en place d'un répertoire en local

Configuration de Git

1. Configurer le nom d'utilisateur :

`git config --global user.name UserName01`

2. Configurer l'adresse mail :

`git config --global user.email user.name01@xx.com`

3. Configurer la branche (abordé plus tard) :

`git config --global init.defaultBranch master`

4. Vérifier la configuration : `git config --list`

```
user.name=PixElise
user.email=elise.martin@etu.univ-poitiers.fr
init.defaultbranch=master
```

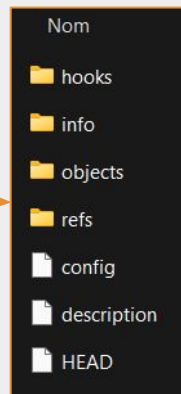
Mise en place d'un répertoire en local

Initialisation de Git

Initialiser un nouveau dépôt Git dans le dossier : **git init**

→ Créer un sous-dossier **.git**

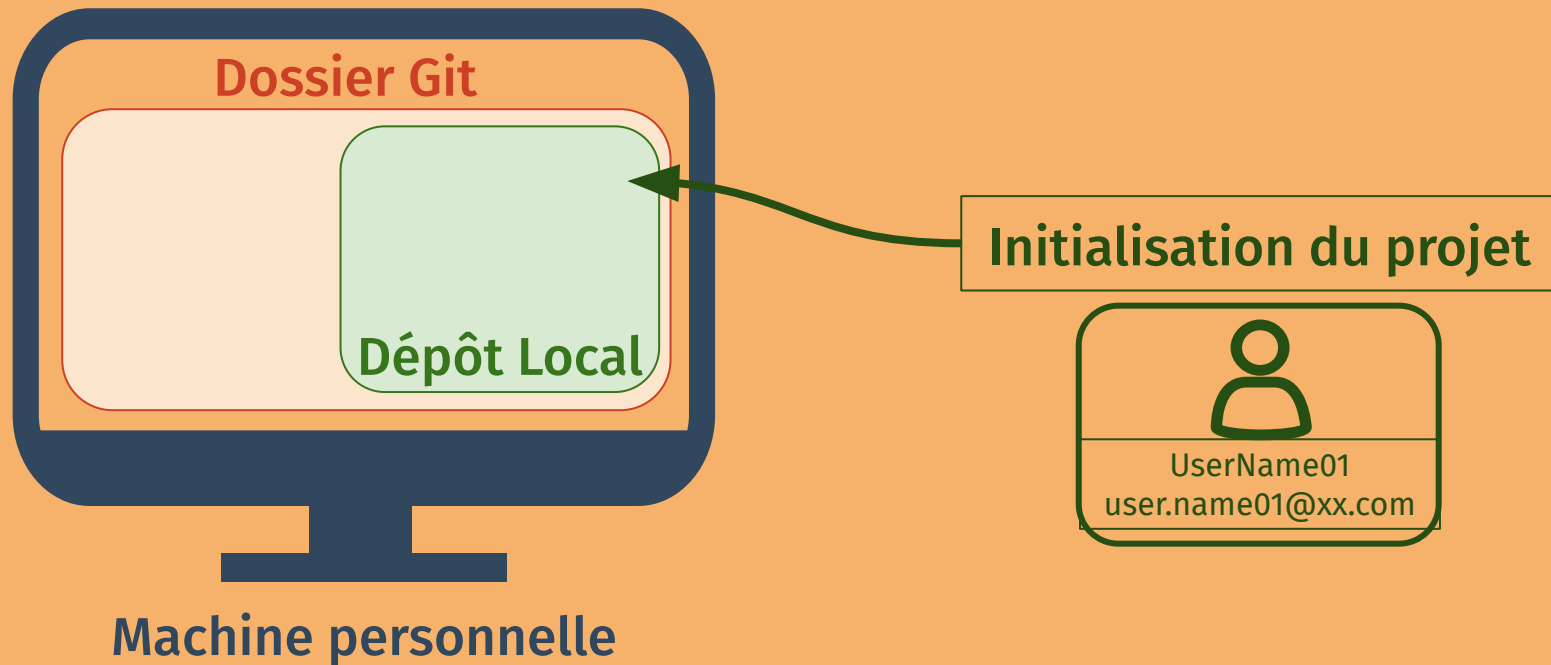
=> A faire à chaque nouveau projet Git



arborescence du dépôt Git

Mise en place d'un répertoire en local

Bilan



1. Présentation de Git

a. Qu'est-ce que Git ?

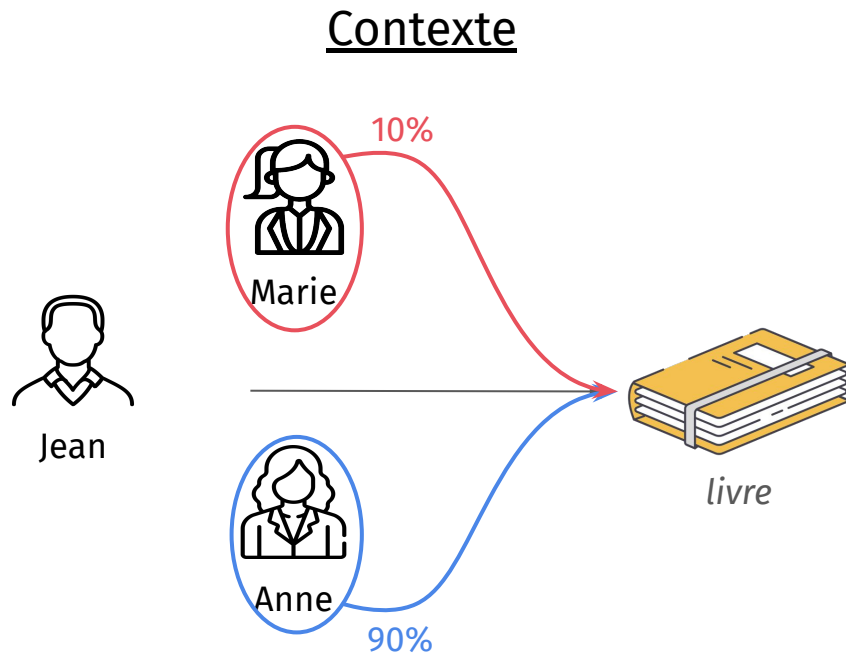
b. Concepts de base

c. Bilan sur Git



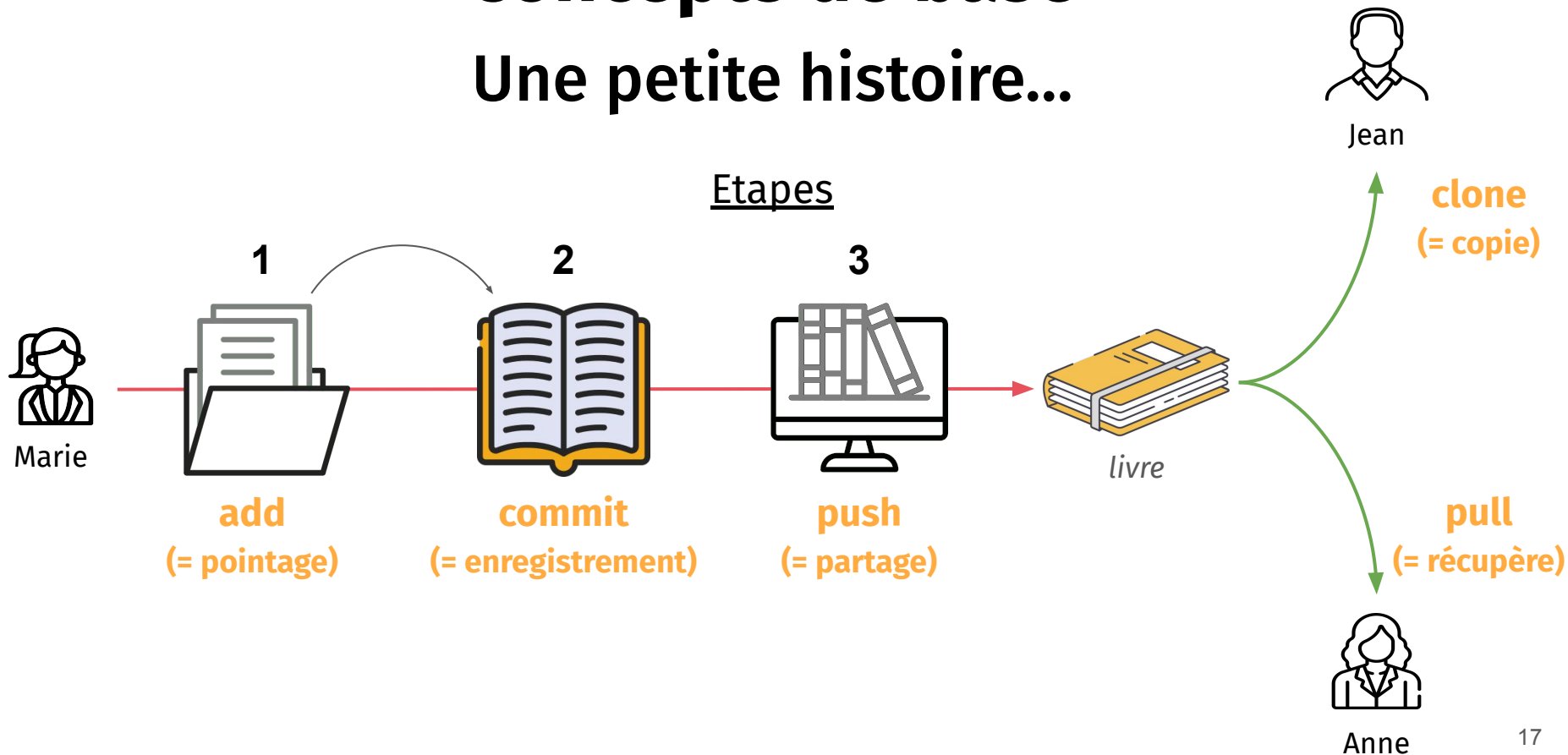
Concepts de base

Une petite histoire...



Concepts de base

Une petite histoire...



Concepts de base

Commandes essentielles

- Add : ajouter des fichiers au suivi de version
 - Status : permet de voir le flux de travail et les fichiers suivis

Avant Add :

Fichiers non suivis et modifiés

=> Vérifier les fichiers à
ajouter au suivi

Après Add :

Fichiers prêts à être enregistrés

=> Vérifier les fichiers
ajoutés au suivi

Concepts de base

Commandes essentielles

- Add : ajouter des fichiers au suivi de version
 - Status : permet de voir le flux de travail et les fichiers suivis
- Commit : enregistrer les modifications apportées au projet
 - Log : Visualiser l'historique des enregistrements
 - Checkout : Restaurer des fichiers à partir de l'historique de dépôt
 - Switch : Retourner à la dernière version
 - Reset : Annulation d'un commit

Concepts de base

Commandes essentielles

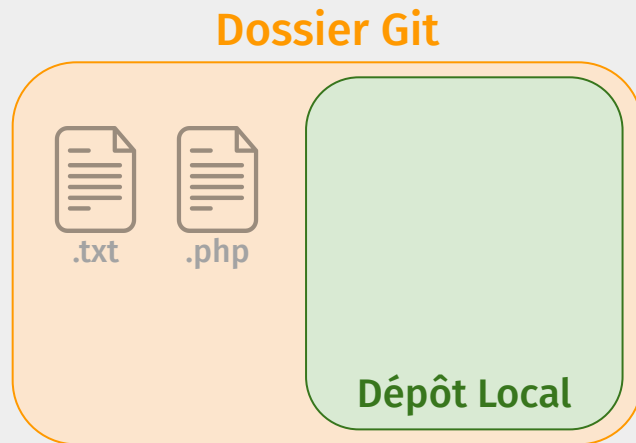
- Push : envoyer les commits vers un dépôt
- Clone : copier un dépôt intermédiaire dans un répertoire local
- Pull : fusion des modifications apportées

Utilisation de Git en local

Ajout de fichiers dans le répertoire

1. Créer un nouveau fichier dans le dossier :
newFichier.txt
2. Créer un autre fichier dans le dossier :
fichierVide.php
3. Regarder l'état actuel du suivi : **git status**

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



Utilisation de Git en local

Faire suivre les fichiers

1. Ajouter un fichier au suivi de version dans le dépôt local :

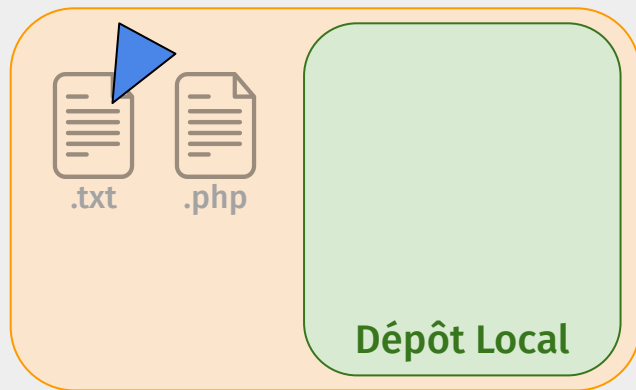
`git add newFichier.txt`

2. Regarder l'état actuel du suivi : `git status`

```
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   newFichier.txt

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

Dossier Git



=> Ajouter tous les fichiers en une seule commande : `git add .`

Utilisation de Git en local

Enregistrement local des fichiers

Commentaire décrivant les modifications

1. Enregistrer les fichiers suivis au dépôt local :

git commit -m "V1"

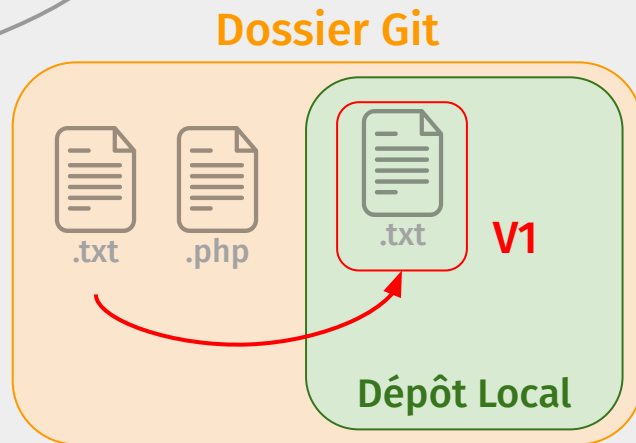
```
[master (root-commit) b33b0a8] V1
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 newFichier.txt
```

2. Vérifier l'historique des enregistrements :

git log

```
commit b33b0a8e082c6609183a30985a74d3a62b8e7d9e (HEAD -> master)
Author: PiixElise <elise.martin79@gmail.com>
Date:   Wed Oct 16 12:35:53 2024 +0200
```

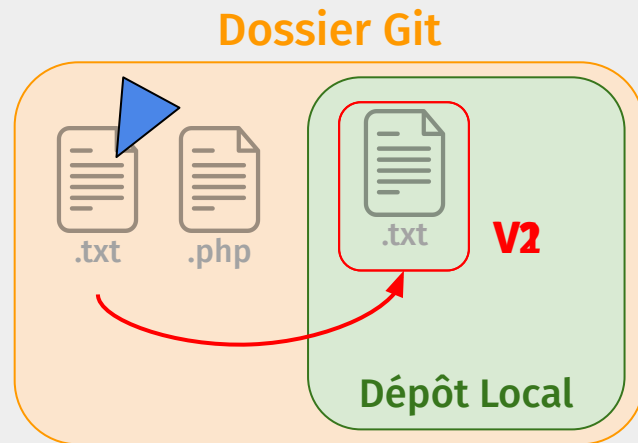
V1



Utilisation de Git en local

Enregistrement local des fichiers

1. Ouvrir et modifier le fichier **newFichier.txt**
2. Ajouter un fichier au suivi de version dans le dépôt local :
git add newFichier.txt
3. Enregistrer les fichiers suivis au dépôt local :
git commit -m "V2"



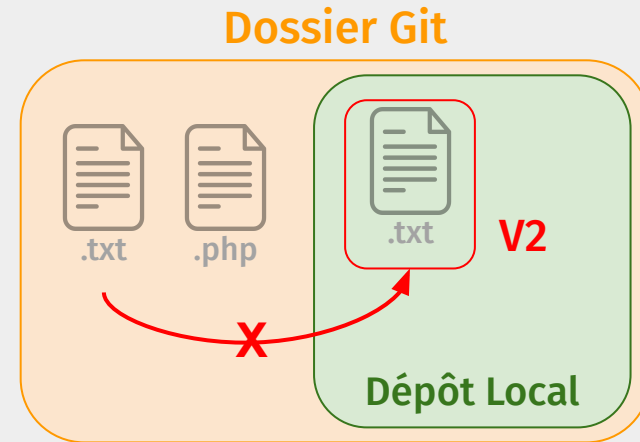
Utilisation de Git en local

Annuler un enregistrement local

1. Ouvrir et modifier le fichier **newFichier.txt**
2. Annuler le commit : **git reset id**

```
Unstaged changes after reset:  
M   newFichier.txt
```

3. Rouvrir le fichier **newFichier.txt**
4. Vérifier l'historique des enregistrements :
git log



Que constatez-vous ?

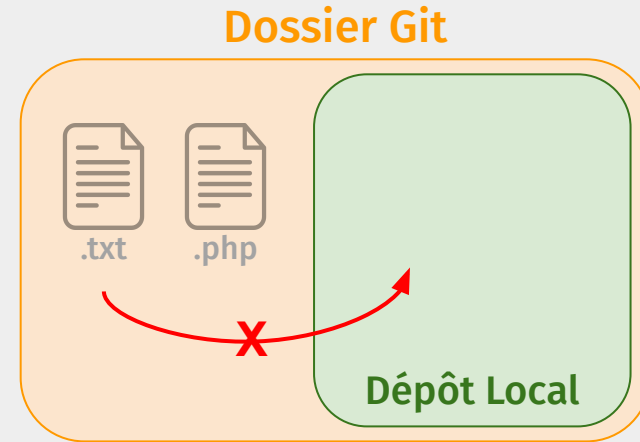
Utilisation de Git en local

Annuler un enregistrement local

1. Ouvrir et modifier le fichier **newFichier.txt**
2. Annuler le commit : **git reset id**

```
Unstaged changes after reset:  
M   newFichier.txt
```

3. Rouvrir le fichier **newFichier.txt**
4. Vérifier l'historique des enregistrements :
git log

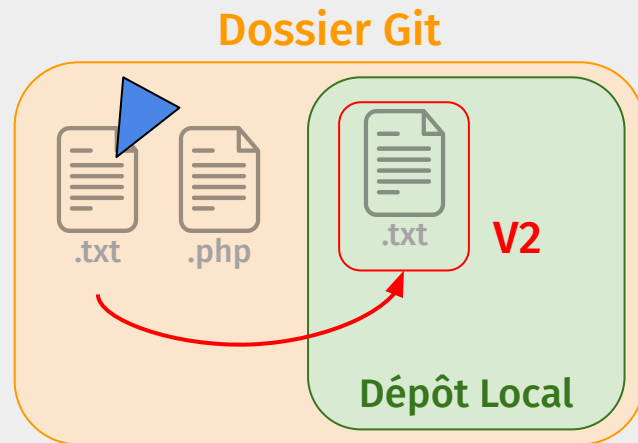


=> Suppression des fichiers enregistrés mais modifications conservées

Utilisation de Git en local

Enregistrement local des fichiers

1. Ouvrir et modifier le fichier **newFichier.txt**
2. Ajouter un fichier au suivi de version dans le dépôt local :
git add newFichier.txt
3. Enregistrer les fichiers suivis au dépôt local :
git commit -m "V2"



Utilisation de Git en local

Restaurer une version

1. Récupérer le numéro du commit V1 : **git log**

```
commit b33b0a8e082c6609183a30985a74d3a62b8e7d9e (HEAD -> master)
Author: PiixElise <elise.martin79@gmail.com>
Date:   Wed Oct 16 12:35:53 2024 +0200
```

V1

2. Restaurer la version V1 :

git checkout id

HEAD is now at b33b0a8 V1

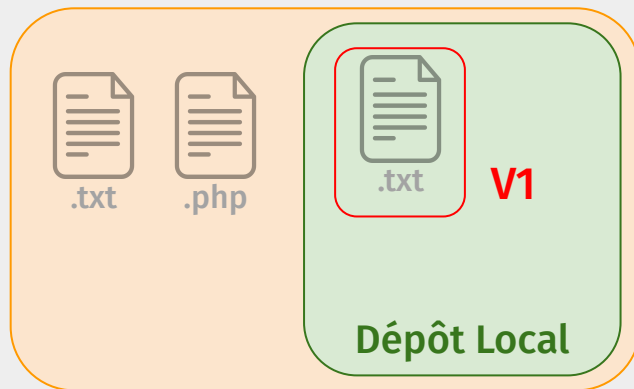
3. Vérifier l'historique des enregistrements :

git log

4. Rouvrir le fichier **newFichier.txt**

Que constatez-vous ?

Dossier Git



Utilisation de Git en local

Restaurer une version

1. Récupérer le numéro du commit V1 : **git log**

```
commit b33b0a8e082c6609183a30985a74d3a62b8e7d9e (HEAD -> master)
Author: PiixElise <elise.martin79@gmail.com>
Date:   Wed Oct 16 12:35:53 2024 +0200
```

V1

2. Restaurer la version V1 :

git checkout id

HEAD is now at b33b0a8 V1

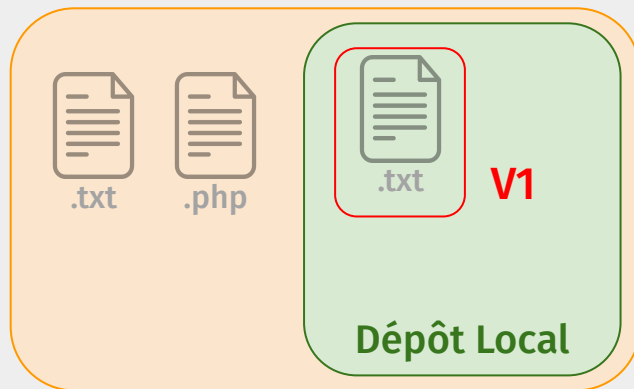
3. Vérifier l'historique des enregistrements :

git log

4. Rouvrir le fichier **newFichier.txt**

=> Retour aux anciennes modifications

Dossier Git



Utilisation de Git en local

Restaurer une version

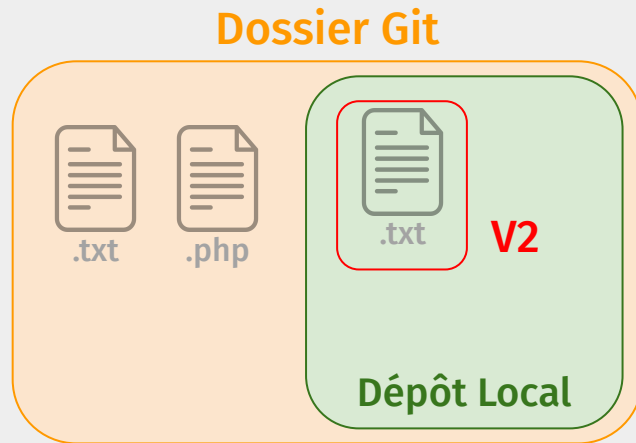
5. Retourner à la dernière version :

git switch -

```
Previous HEAD position was b33b0a8 V1 12h35  
Switched to branch 'main'  
Your branch is ahead of 'origin/main' by 2 commits.  
(use "git push" to publish your local commits)
```

6. Vérifier l'historique des enregistrements :

git log



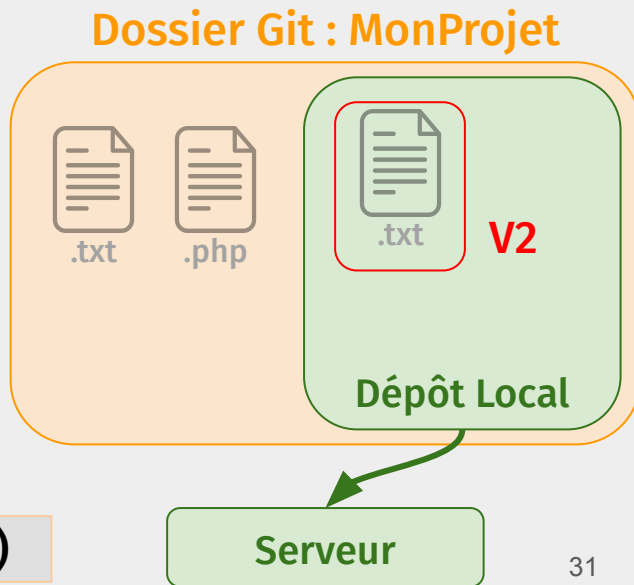
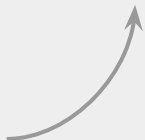
Comment envoyer ce projet vers un autre dépôt Git (partage) ?

Utilisation de Git en local

Créer un répertoire Git intermédiaire

1. Retourner dans le répertoire parent (“repLocal”) : `cd ..`
2. Créer un dépôt local intermédiaire (“Serveur”) :
`git clone --bare MonProjet Serveur`

Nom du dépôt intermédiaire

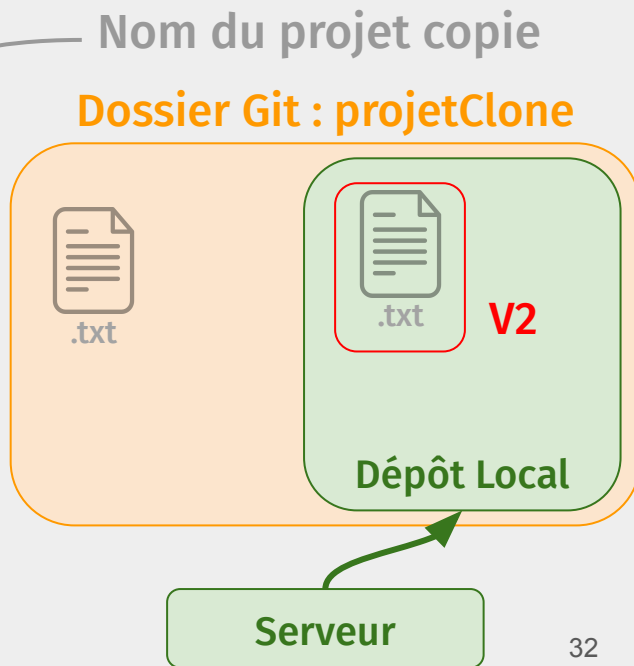


=> Configuration du dépôt local intermédiaire (Serveur)

Utilisation de Git en local

Créer un répertoire copie

1. Dans le dossier **repLocal**, cloner le projet :
git clone Serveur projetClone
2. Se placer dans le répertoire cloné :
cd projetClone
3. Vérifier la présence du fichier **newFichier.txt**



Utilisation de Git en local

Modifier le projet copie

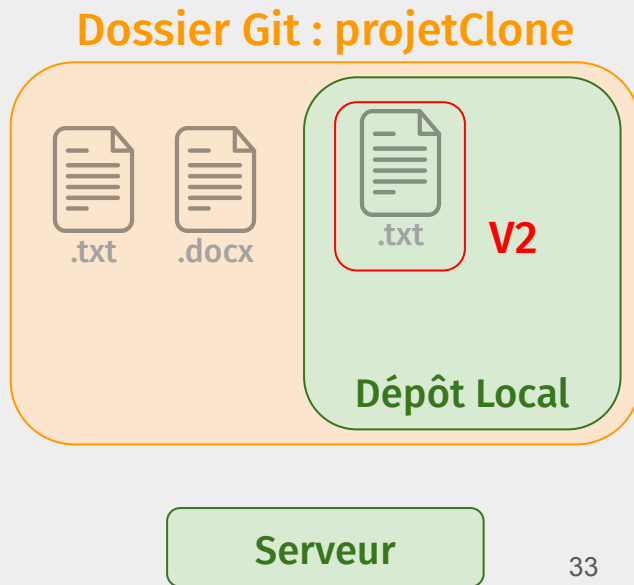
1. Vérifier l'historique des commits (V1 et V2) :

`git log`

2. Ouvrir et modifier le fichier `newFichier.txt`

3. Créer et modifier un nouveau fichier :

`notes.docx`



Utilisation de Git en local

Envoyer des modifications vers le répertoire initial

1. Ajouter un fichier au suivi de version dans le dépôt local :

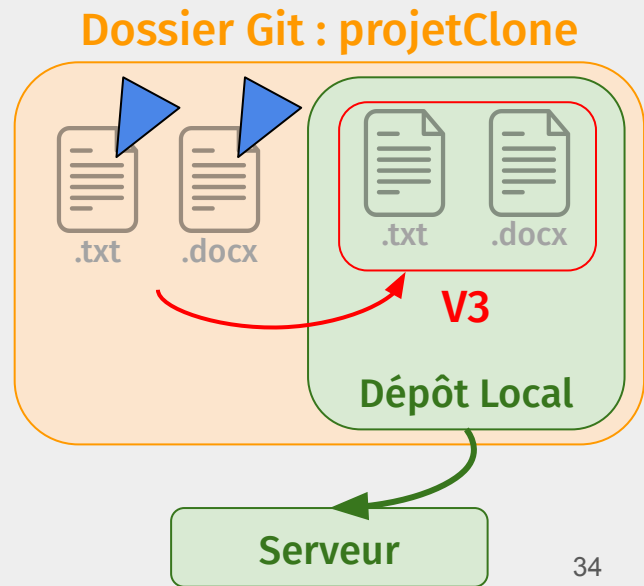
git add .

2. Enregistrer les fichiers suivis au dépôt local :

git commit -m "V3"

3. Envoyer les modifications (au "Serveur") :

git push origin master



Utilisation de Git en local

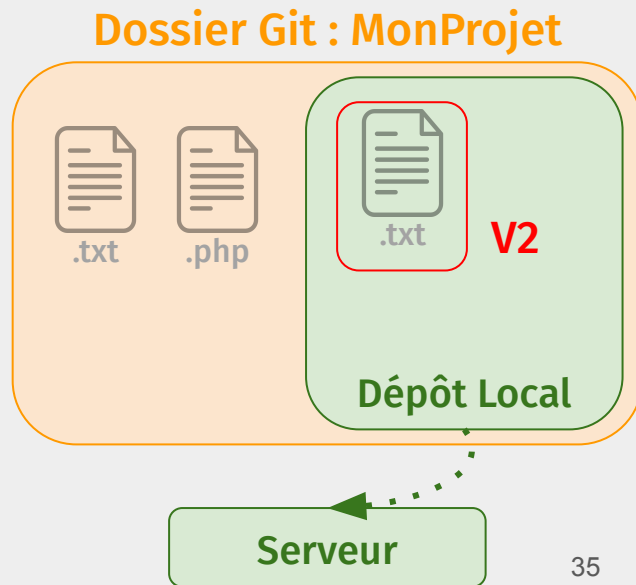
Récupérer les modifications du répertoire copie

1. Se déplacer dans le répertoire de son projet :

`cd ../MonProjet`

2. Lier les dépôts **MonProjet** et **Serveur** :

`git remote add origin Serveur`



Utilisation de Git en local

Récupérer les modifications du répertoire copie

1. Se déplacer dans le répertoire de son projet :

`cd ../MonProjet`

2. Lier les dépôts **MonProjet** et **Serveur** :

`git remote add origin Serveur`

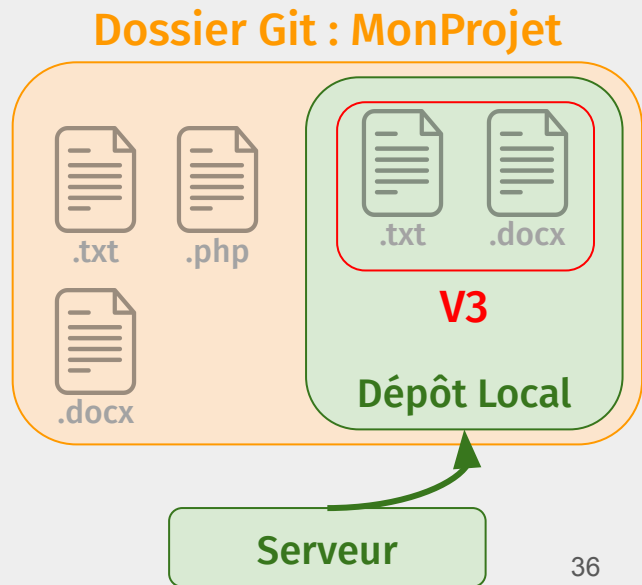
3. Récupérer les modifications :

`git pull origin master`

4. Vérifier la présence du fichier **notes.docx**

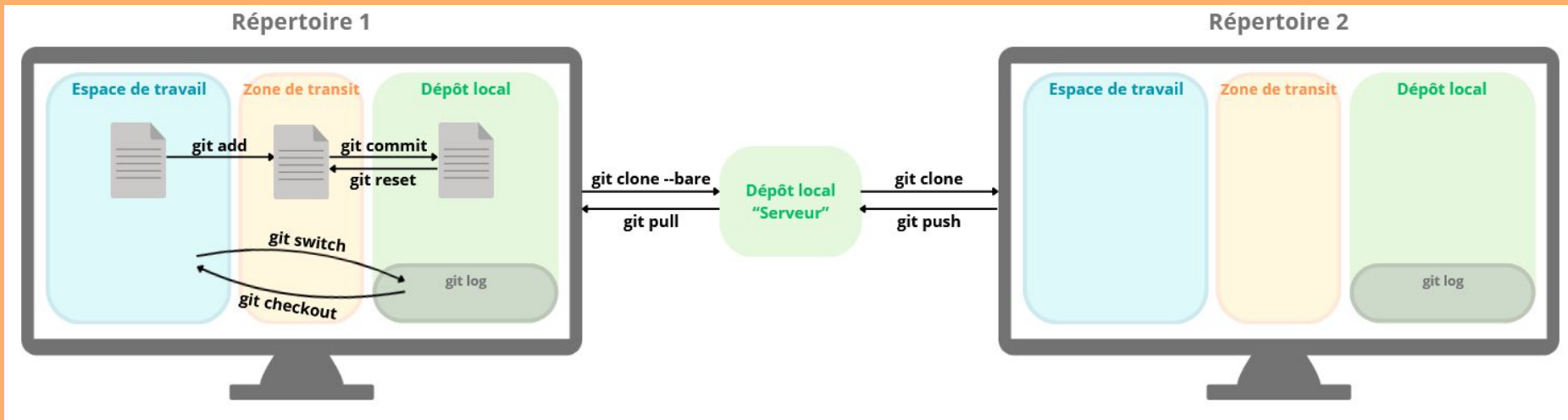
5. Vérifier les modifications des fichiers

newFichier.txt et **notes.docx**



Utilisation de Git en local

Bilan



Commandes Git essentielles

1. Présentation de Git

- a. Qu'est-ce que Git ?
- b. Concepts de base
- c. Bilan sur Git**



Bilan sur Git

Contrôle du Versioning

- **Maintenir l'historique des modifications :**
 - Objectif : conserver l'historique complet des modifications
 - Commande : `git commit -m "Version"`
- **Annuler un enregistrement :**
 - Objectif : éviter l'envoi de fichier par erreur
 - Commande : `git reset id`
- **Restaurer des versions :**
 - Objectif : récupérer des versions antérieures du projet
 - Commande : `git checkout id`

Bilan sur Git

Collaboration en Equipe

- **Documentation et communication :**
 - Objectif : informer les autres utilisateurs des modifications apportées
 - Comment :
 - écrire un message explicite lors des commit
 - rédiger un fichier README : donne les points clés du projet
- **Intégration avec des plateformes collaboratives :**
 - Objectif : gestion plus efficace des projets et partage des codes
 - Comment : utilisation plateformes comme GitHub, GitLab...



2. Présentation de GitHub

a. Qu'est-ce que GitHub ?

b. Pourquoi GitHub ?

c. Gestion d'un dépôt



Qu'est-ce que GitHub ?

= Plateforme de développement collaborative basée sur Git

- Hébergement gratuit de dépôts Git sur des serveurs distants
- Accessibilité mondiale et gestion centralisée des projets
- Offre des outils et services pour améliorer la collaboration et la gestion des projets

2. Présentation de GitHub

- a. Qu'est-ce que GitHub ?
- b. Pourquoi GitHub ?**
- c. Gestion d'un dépôt



Pourquoi GitHub ?

- **Hébergement de dépôts** : Stockage sécurisé sur le cloud
- **Collaboration** : Travail simultané avec des révisions, commentaires et approbations de modifications
- **Documentation** : Documentation est mise à jour lors des commits
 - + Inclusion d'un fichier explicatif (= README) facilitant la prise en main
- **Sécurité** : Restauration des versions précédentes et contrôle d'accès pour protéger les projets (privé ou public, en lecture, écriture ou administrateur)
- **Intégrations** : Compatibilité avec divers outils et services de développement (ex : BlueJ)

2. Présentation de GitHub

- a. Qu'est-ce que GitHub ?
- b. Pourquoi GitHub ?
- c. Gestion d'un dépôt**



Gestion d'un dépôt

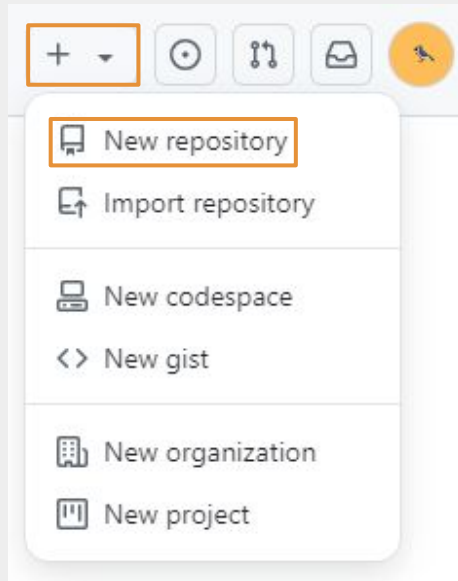
Créer un compte GitHub

1. Accéder au site GitHub
2. Cliquer sur **Sign up** pour créer un nouveau compte
3. Remplir les informations requises :
nom d'utilisateur, adresse e-mail, mot de passe
4. Suivre les instructions à l'écran pour vérifier votre compte (e-mail de confirmation)

Gestion d'un dépôt

Créer un projet & partage

1. Cliquer sur le bouton **New repository** pour créer un nouveau dépôt



Gestion d'un dépôt

Créer un projet & partage

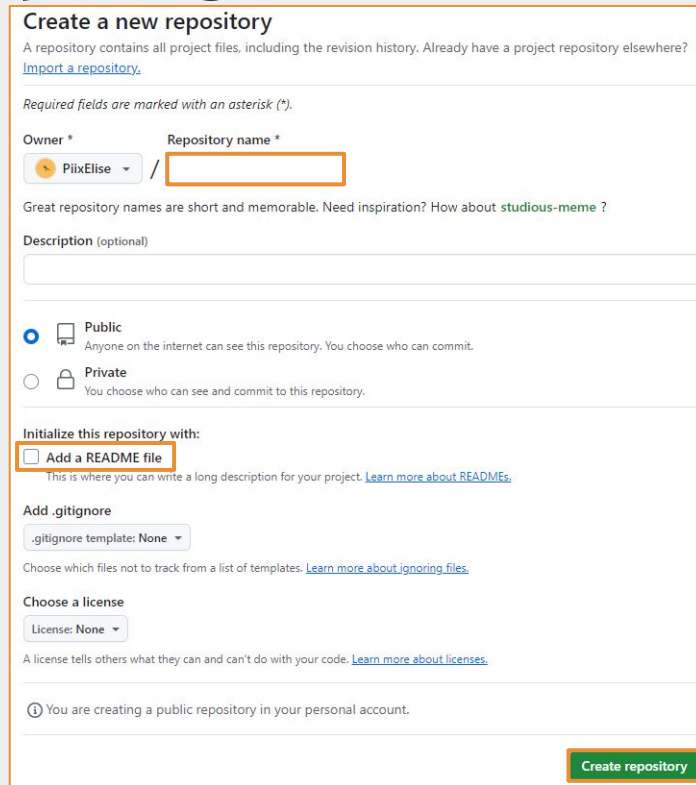
2. Remplir :

- **nom du dépôt** par **projetGH**,
- et son **accès** en **Public**

3. Cocher **Add a README file**

Abordé ultérieurement

4. Cliquer sur **Create repository**



Create a new repository

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

Required fields are marked with an asterisk (*).

Owner * PiixElise / Repository name *

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

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:

☒ Add a README file
This is where you can write a long description for your project. [Learn more about READMEs.](#)

Add .gitignore
.gitignore template: None

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

Choose a license
License: None

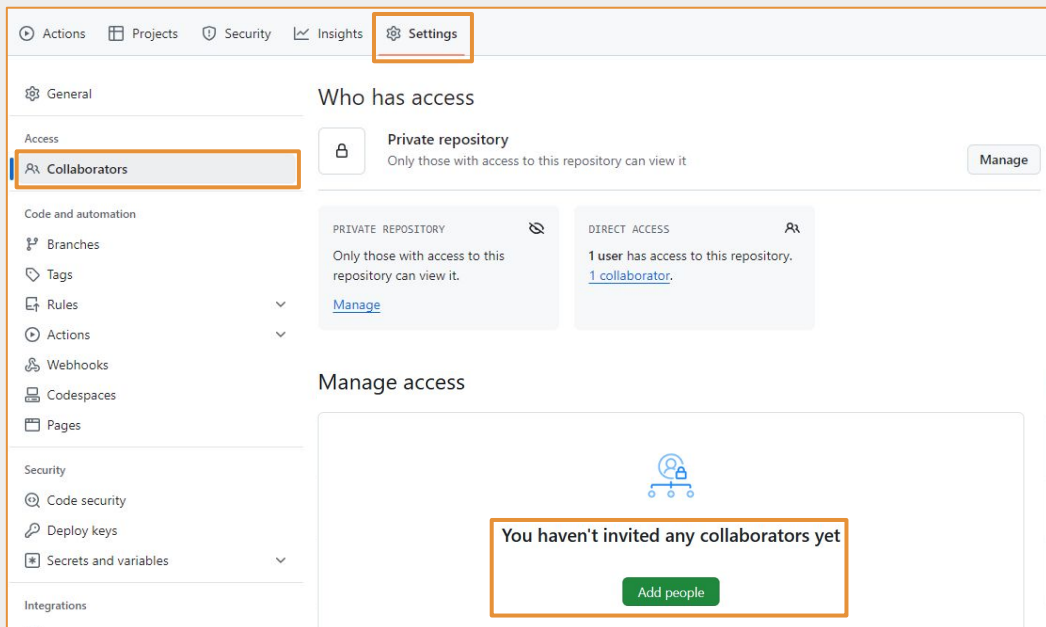
A license tells others what they can and can't do with your code. [Learn more about licenses.](#)

ⓘ You are creating a public repository in your personal account.

Create repository

Gestion d'un dépôt

Créer un projet & partage

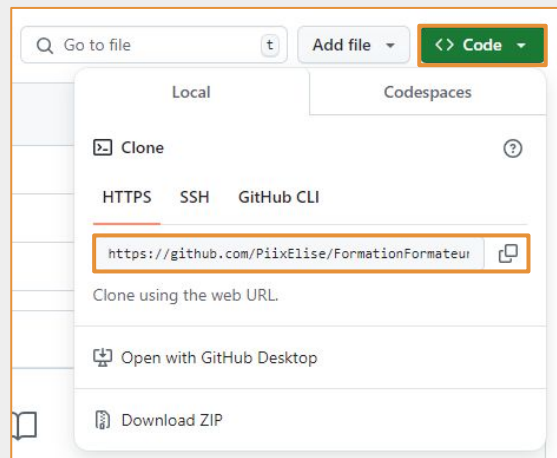


5. Dans le menu **Settings** :
- aller dans l'onglet **Collaborators**,
 - et ajouter son binôme

Gestion d'un dépôt

Clonage du dépôt GitHub

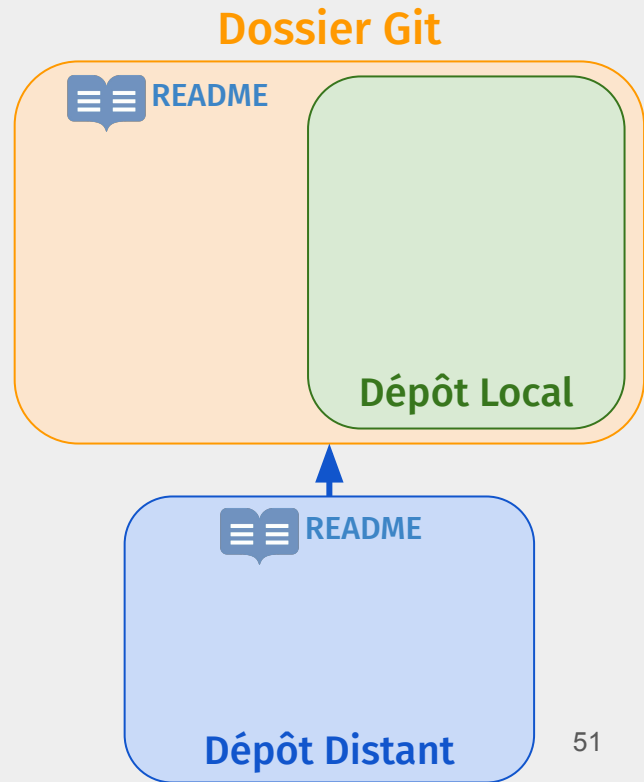
1. Cliquer sur le bouton **Code** et copier l'URL HTTPS



Gestion d'un dépôt

Clonage du dépôt GitHub

2. Créer un dossier : `mkdir repDistant`
3. Se placer dans le répertoire : `cd repDistant`
4. Cloner le projet :
`git clone URL DU DEPOT GIT`
5. Se placer dans le répertoire : `cd projetGH`



Gestion d'un dépôt

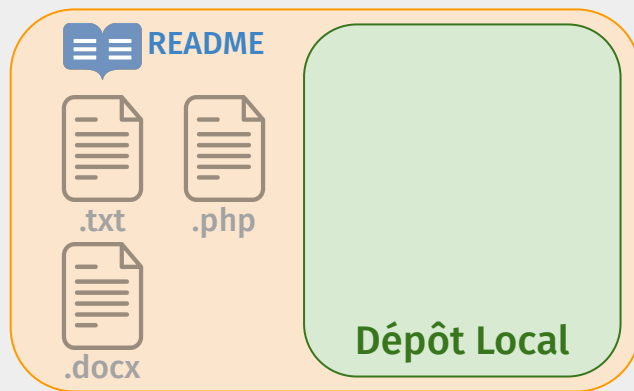
Ajout de fichiers dans le répertoire

1. Copier et coller les fichiers de l'exercice précédent :

- newFichier.txt,
- fichierVide.php,
- et notes.docx

2. Regarder l'état actuel du suivi : **git status**

Dossier Git

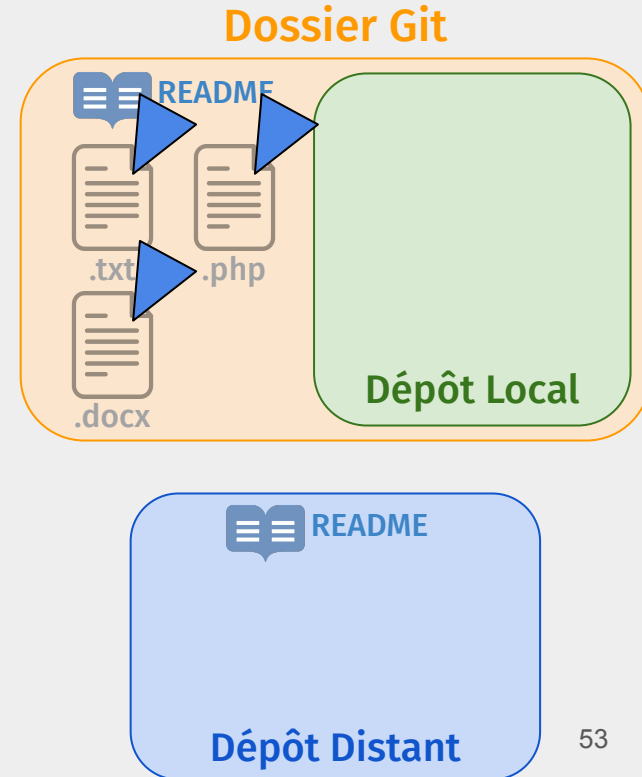


Gestion d'un dépôt

Envoyer des modifications au binôme

1. Ajouter un fichier au suivi de version en créant un lien entre les dépôts :

`git add .`



Gestion d'un dépôt

Envoyer des modifications au binôme

1. Ajouter un fichier au suivi de version en créant un lien entre les dépôts :

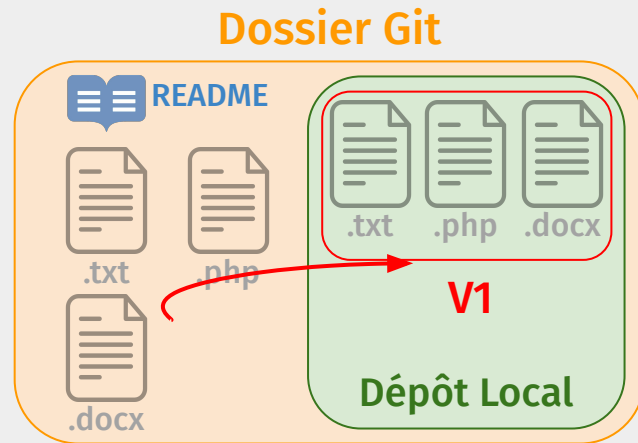
git add .

2. Regarder l'état actuel du suivi : **git status**

3. Enregistrer les fichiers suivis au dépôt local :

git commit -m "V1"

Allez sur GitHub, les fichiers sont-ils présents ?

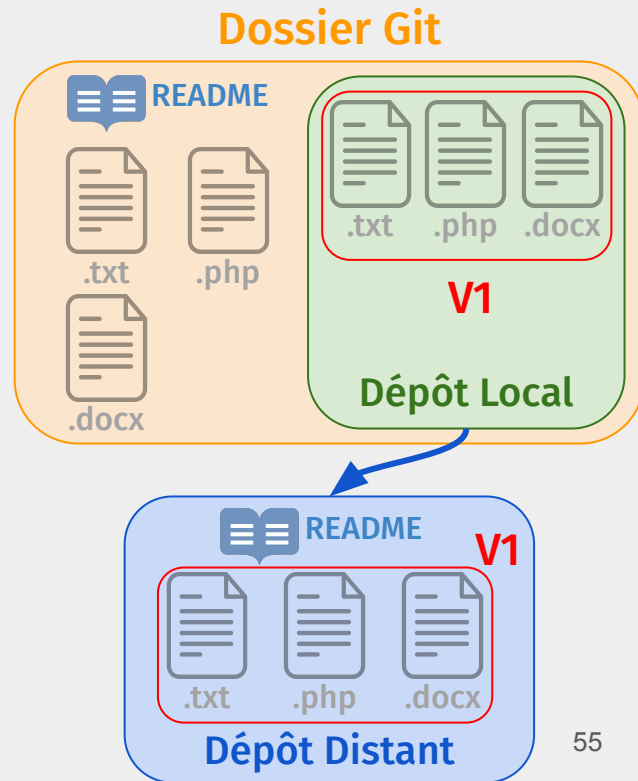


Gestion d'un dépôt

Envoyer des modifications au binôme

4. Envoyer les modifications :
`git push -u origin main`

=> Uniquement la première fois
=> Les fois suivantes, un simple : `git push`

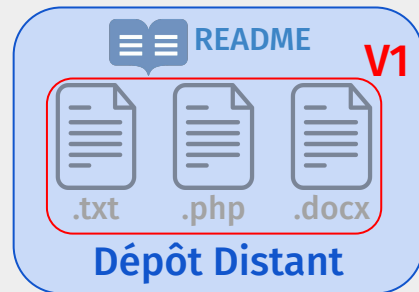
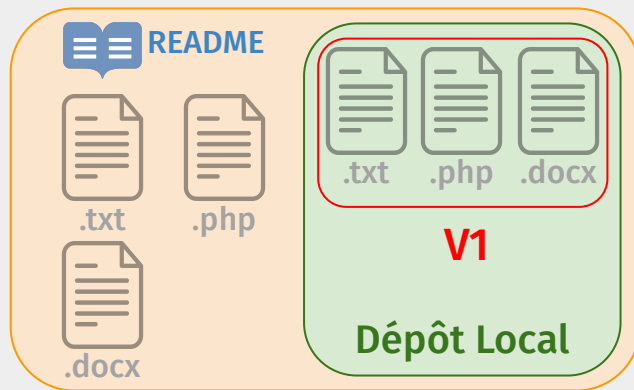


Gestion d'un dépôt

Envoyer des modifications au binôme

4. Envoyer les modifications :
`git push -u origin main`
5. Contrôler l'historique des commits (v1) :
`git log`

Dossier Git



Allez sur GitHub, les fichiers sont-ils présents ?

Gestion d'un dépôt

Exercice : Travail sur le projet binôme

Objectif :

1. Cloner le projet de son binôme dans un répertoire local
2. Supprimer les fichiers : **fichierVide.php** et **notes.docx**
3. Ajouter un fichier : **Binome.txt**
4. Faire suivre et enregistrer les modifications dans votre répertoire local
5. Envoyer les modifications dans le répertoire distant de votre binôme

Pour renommer un dépôt lors du clonage :
spécifier le nouveau nom après l'adresse du dépôt

Gestion d'un dépôt

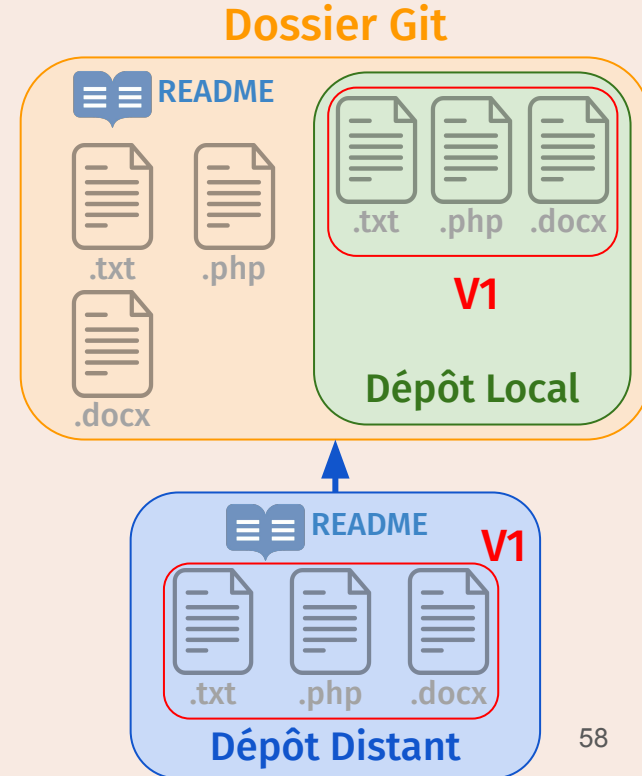
Correction

1. Cloner le projet de son binôme dans un répertoire local :

`git clone URL DU DEPOT GIT`

2. Se déplacer dans le répertoire cloné :

`cd NOM DU DEPOT GIT`



Gestion d'un dépôt

Correction

1. Cloner le projet de son binôme dans un répertoire local :

`git clone URL DU DEPOT GIT`

2. Se déplacer dans le répertoire cloné :

`cd NOM DU DEPOT GIT`

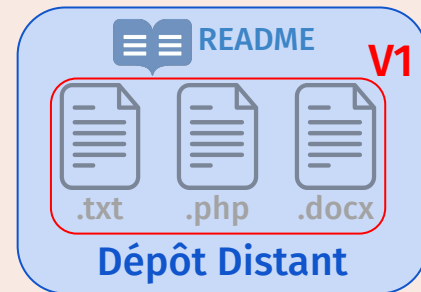
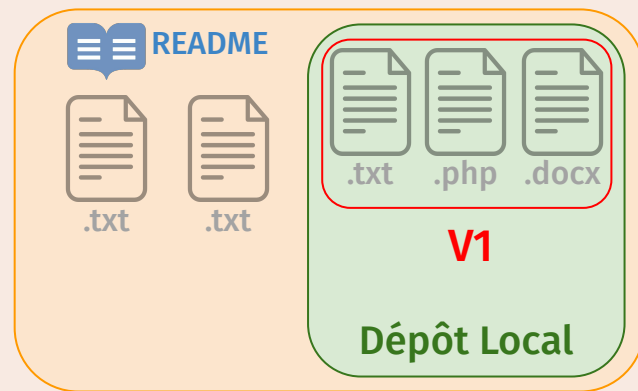
3. Supprimer les fichiers :

`rm fichierVide.php` et `rm notes.docx`

4. Ajouter un fichier dans le répertoire :

`Binome.txt`

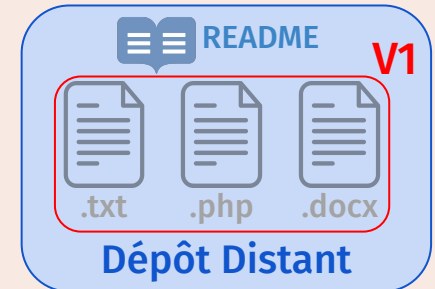
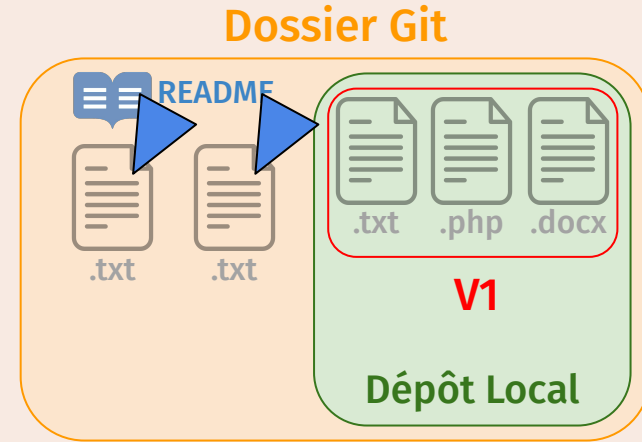
Dossier Git



Gestion d'un dépôt

Correction

5. Ajouter un fichier au suivi de version dans le dépôt local :
- git add .**



Gestion d'un dépôt

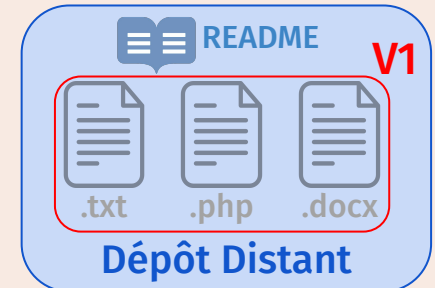
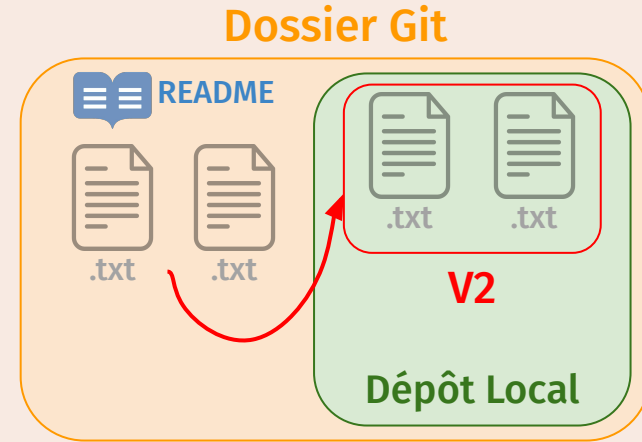
Correction

5. Ajouter un fichier au suivi de version dans le dépôt local :

`git add .`

6. Enregistrer les fichiers suivis au dépôt local :

`git commit -m "V2"`



Gestion d'un dépôt

Correction

5. Ajouter un fichier au suivi de version dans le dépôt local :

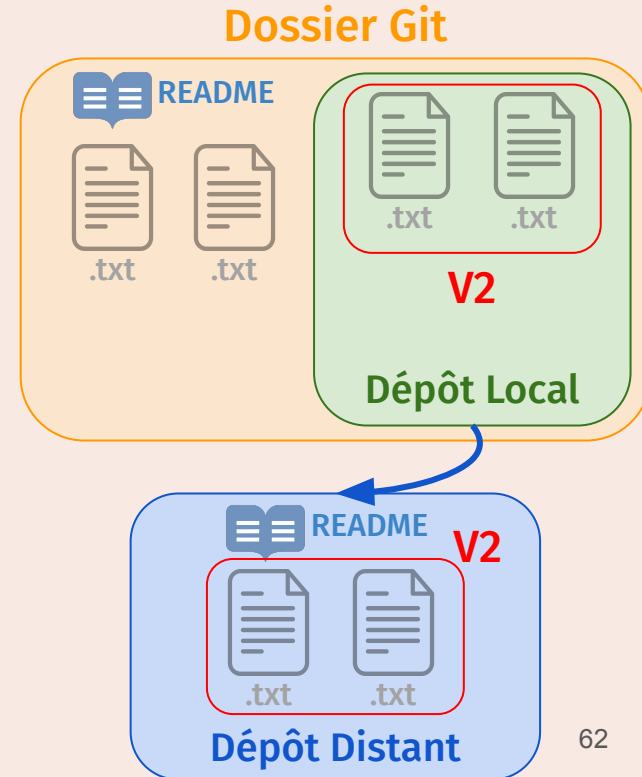
git add .

6. Enregistrer les fichiers suivis au dépôt local :

git commit -m "V2"

7. Envoyer les modifications :

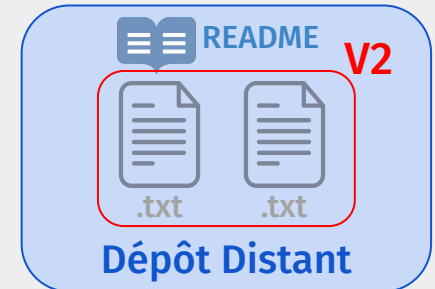
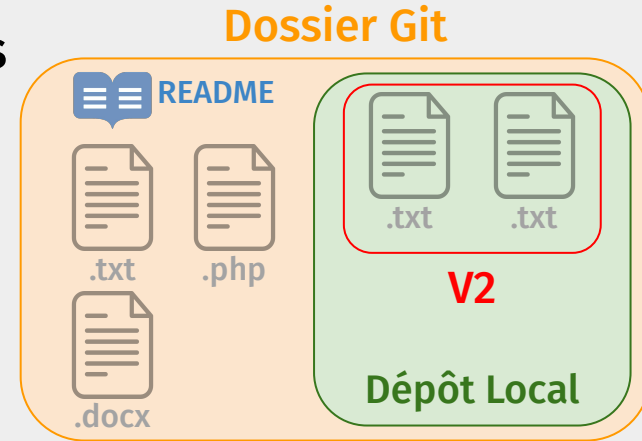
git push



Gestion d'un dépôt

Récupérer les modifications de son binôme

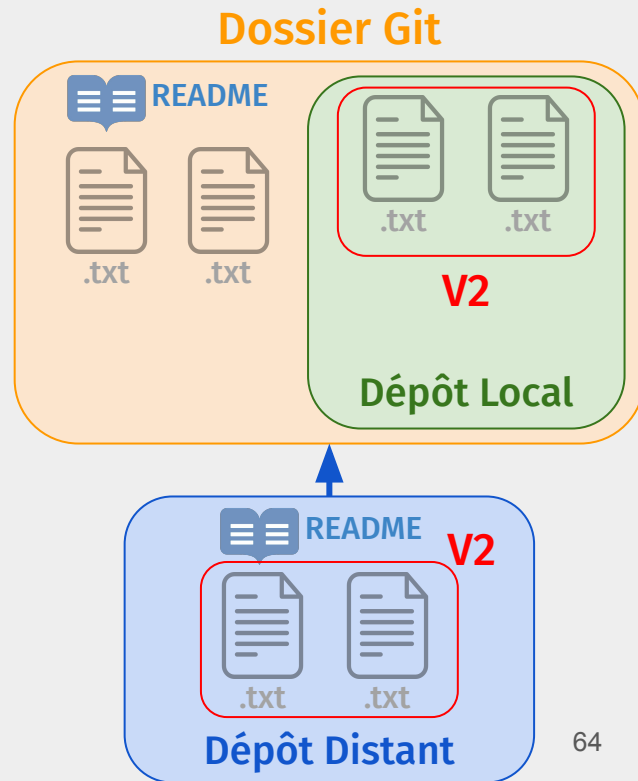
1. Sur GitHub, vérifier la présence des nouveautés



Gestion d'un dépôt

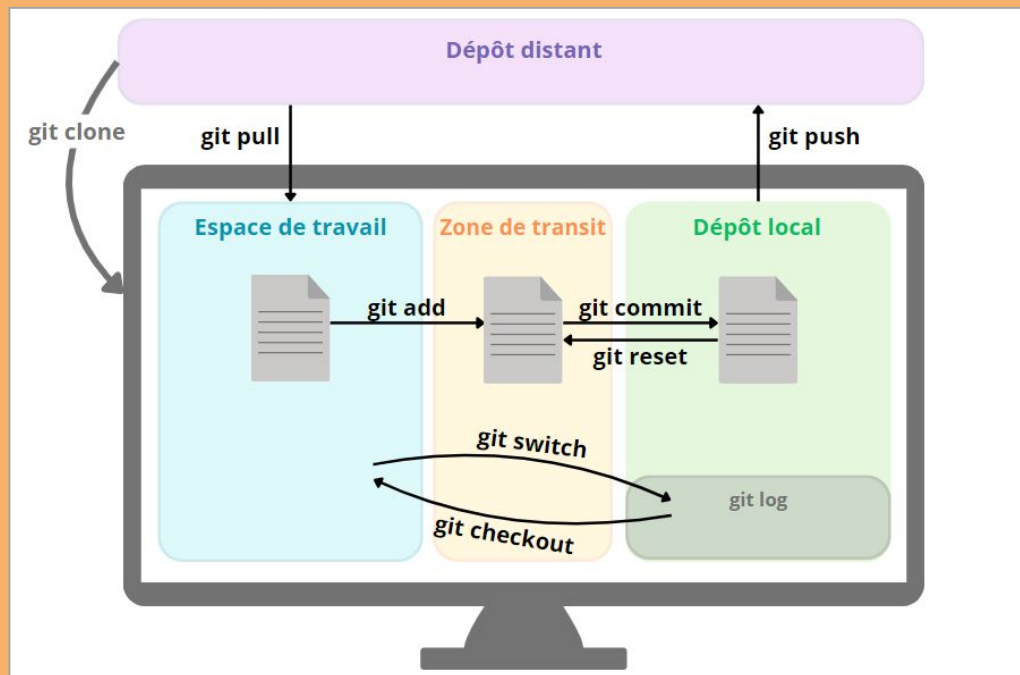
Récupérer les modifications de son binôme

1. Sur GitHub, vérifier la présence des nouveautés
2. Se déplacer dans le répertoire :
`cd projetGH`
3. Récupérer les modifications :
`git pull`
4. Vérifier :
 - l'absence des fichiers : `fichierVide.php` et `notes.docx`
 - la présence du fichier : `Binome.txt`



Gestion d'un dépôt

Bilan



Commandes GitHub essentielles

3. Gestion des conflits



Gestion des conflits

Origine

Si modifications simultanée d'un même code par plusieurs développeurs

- Exercices de la formation = simples => peu de risques
- Projets plus complexes (ex : COO-POO) => risques importants

Gestion des conflits = point important

P1 : personne 1
P2 : personne 2

Gestion des conflits

Exemple

Code initial (python)

```
def calc_array(arr):  
    total = 0  
    for i in range(len(arr)):  
        total += arr[i]  
    return total
```

Modifications de P2

```
def sum_array(arr):  
    total = sum(arr)  
    return total
```

```
def sum_array(arr):  
    total = 0  
    for i in range(len(arr)):  
        total += arr[i]  
    print("Total : ", total)  
    return total
```

Modifications de P1

```
def sum_array(arr):  
    <<<<<<< HEAD  
        total = 0  
        for i in range(len(arr)):  
            if isinstance(arr[i], (int,  
float)):  
                total += arr[i]  
        return total  
    =====  
        total = sum(arr)  
        return total  
>>>>>>>
```

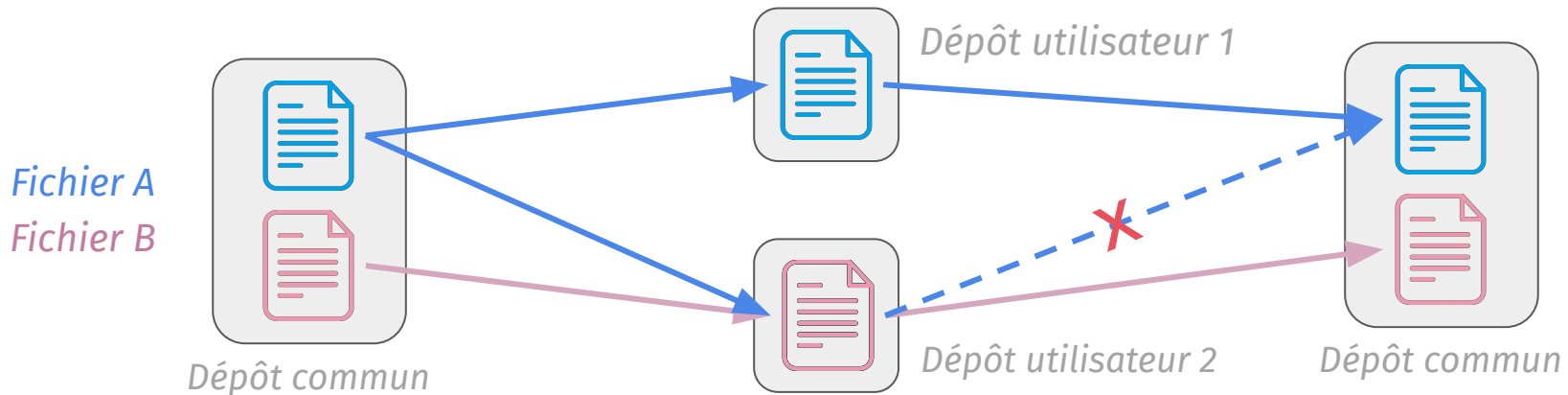
Conflit

Gestion des conflits

Comment les éviter ?

- **Répartir les tâches :**

→ Ne pas travailler simultanément sur les mêmes classes et fichiers



Gestion des conflits

Comment les éviter ?

- **Répartir les tâches :**
 - Ne pas travailler simultanément sur les mêmes classes et fichiers
1 tâche = 1 classe/fichier = 1 développeur
 - Ne pas modifier du code non lié à vos tâches
 - Planifier des réunions régulières pour discuter de l'avancement des tâches

Gestion des conflits

Comment les éviter ?

- **Mises à jour régulières :**
 - Faire un **git pull** avant de commencer une nouvelle tâche
=> récupérer les modifications des collaborateurs
 - Faire des **commits** fréquents et des **git push** réguliers
=> envoyer ses modifications fréquemment
 - Ne pas envoyer de classe/fichier contenant une erreur
- **Messages de commit clairs :**
 - Écrire son prénom dans le message de commit pour identifier l'auteur des modifications

4. Bonnes pratiques sur Git



Bonnes pratiques sur Git

Utiliser un fichier `.gitignore`

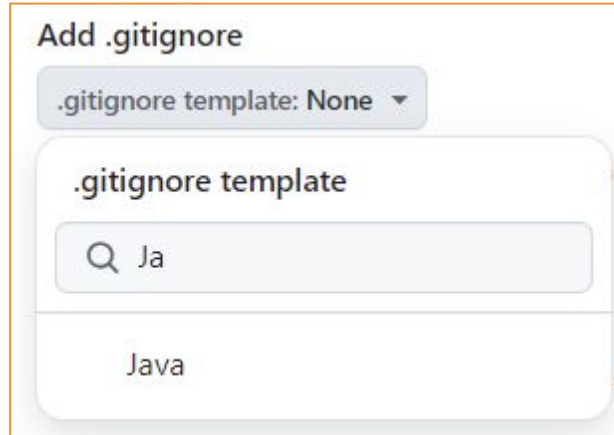
- But: ignorer des fichiers et répertoires spécifiés lors des `commits` et `push`
- Avantages:
 - évite les conflits avec les fichiers de compilation
(ex : `.class`)
 - garde le dépôt propre et sécurisé en excluant les fichiers temporaires et sensibles des dépôts locaux
(ex : `*.log`, `*.tmp`)

Bonnes pratiques sur Git

Utiliser un fichier `.gitignore`

→ Création :

- Lors de la création du projet sur GitHub
=> utilisation des templates de GitHub



Bonnes pratiques sur Git

Utiliser un fichier `.gitignore`

→ Création :

- Ajout manuel des extensions
 - => créer un fichier appelé `.gitignore`
 - => écrire les extensions, par exemple :

```
# Ignorer les fichiers de classe Java compilés
*.class

# Ignorer les fichiers journaux
*.log

# Ignorer les fichiers temporaires
*.tmp
```

Bonnes pratiques sur Git

Utiliser un fichier README.md

- Importance : premier fichier visible pour les contributeurs et utilisateurs
- Contenu : informations essentielles à la compréhension

The image shows a screenshot of a code editor displaying a README.md file. The file content is as follows:

```
1 # MonProjetJava
2
3 ## Description
4 Ce projet est une application Java développée sous BlueJ pour gérer les tâches quotidiennes.
5
6 ## Installation
7 1. Clonez le dépôt (sur un terminal): git clone https://github.com/votre-utilisateur/MonProjetJava.git
8 2. Accédez au répertoire du projet (sur un terminal) : cd MonProjetJava
9 3. Ouvrez le projet dans VSCode :
10    - Lancez VSCode.
11    - Ouvrez le projet en sélectionnant le répertoire cloné.
12
13 ## Utilisation
14 Pour exécuter l'application dans VSCode :
15 - Ouvrez la classe principale et cliquez sur "Start Debugging".
16 - Cliquez sur "Run without Debugging" pour exécuter l'application.
```

Annotations with arrows pointing to specific parts of the file:

- Titre**: Points to the first line `# MonProjetJava`.
- Description brève**: Points to the `## Description` section and its content.
- Instructions d'installer et configurer le projet**: Points to the `## Installation` section.
- Guide pour prendre en main le projet**: Points to the `## Utilisation` section.

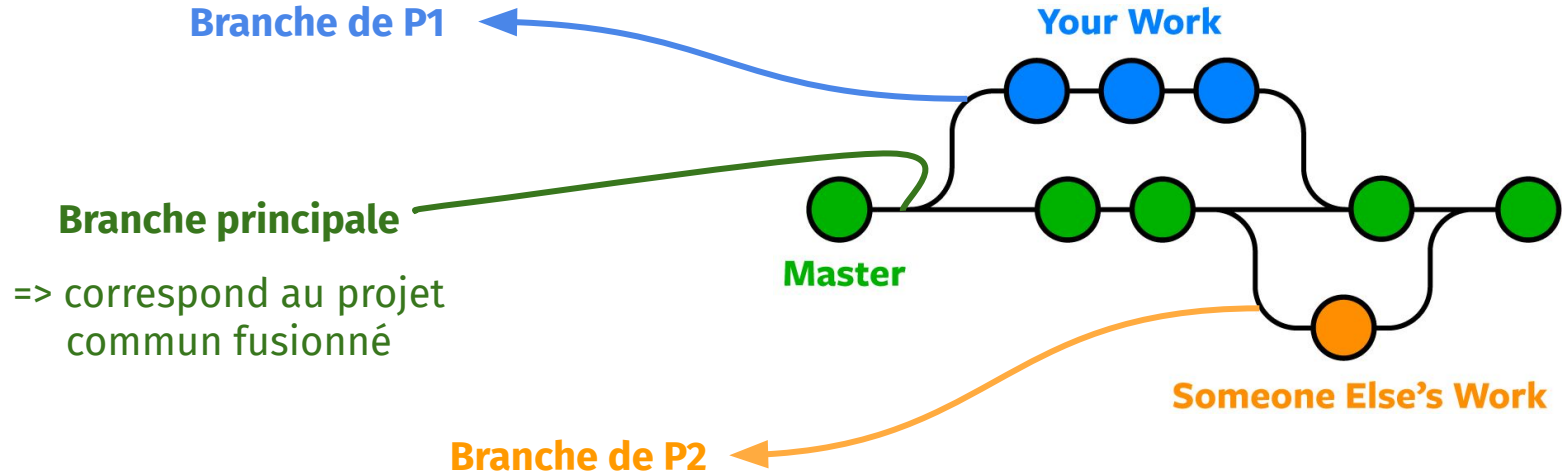
P1 : personne 1
P2 : personne 2

Notion supplémentaire

“Branches”

= Copie du projet

(ex : 1 personne travaille simultanément sur 2 parties différentes du projet)



5. GitHub et BlueJ



Gestion d'un projet BlueJ

Créer un Token

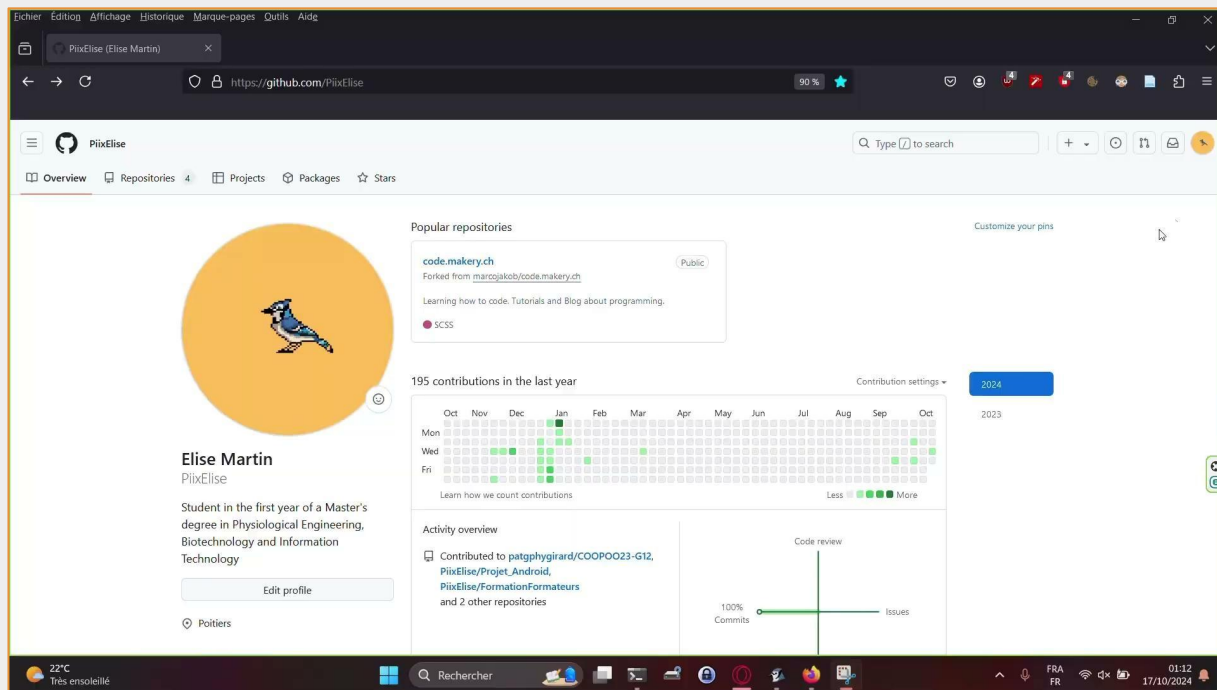
Clé d'accès

A. Aller dans :

1. **Settings,**
2. **Developer settings,**
3. **Personal access tokens,**
4. **Tokens (classic)**

B. Cliquer sur :

1. **Generate new token,**
2. **Generate new token (classic)**



Gestion d'un projet BlueJ

Créer un Token

Select scopes

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

<input checked="" type="checkbox"/> repo	Full control of private repositories
<input checked="" type="checkbox"/> repo:status	Access commit status
<input checked="" type="checkbox"/> repo:deployment	Access deployment status
<input checked="" type="checkbox"/> public_repo	Access public repositories
<input checked="" type="checkbox"/> repo:invite	Access repository invitations
<input checked="" type="checkbox"/> security_events	Read and write security events
<input type="checkbox"/> workflow	Update GitHub Action workflows
<input type="checkbox"/> write:packages	Upload packages to GitHub Package Registry
<input type="checkbox"/> read:packages	Download packages from GitHub Package Registry
<input type="checkbox"/> delete:packages	Delete packages from GitHub Package Registry
<input type="checkbox"/> admin:org	Full control of orgs and teams, read and write org projects
<input type="checkbox"/> write:org	Read and write org and team membership, read and write org projects
<input type="checkbox"/> read:org	Read org and team membership, read org projects
<input type="checkbox"/> manage_runners:org	Manage org runners and runner groups
<input type="checkbox"/> admin:public_key	Full control of user public keys
<input type="checkbox"/> write:public_key	Write user public keys
<input type="checkbox"/> read:public_key	Read user public keys
<input checked="" type="checkbox"/> admin:repo_hook	Full control of repository hooks
<input checked="" type="checkbox"/> write:repo_hook	Write repository hooks
<input checked="" type="checkbox"/> read:repo_hook	Read repository hooks
<input type="checkbox"/> admin:org_hook	Full control of organization hooks
<input type="checkbox"/> gist	Create gists

1. Donner un nom
2. Choisir le délai d'expiration de votre choix
3. Cocher les cases :
 - **repo**
 - **admin:repo_hook**
4. Cliquer sur **Generate token**

Gestion d'un projet BlueJ

Créer un Token



Sauvegardez votre **token** quelque part !

TokenTest_Formation — *admin:repo_hook, repo*

Last used within the last week

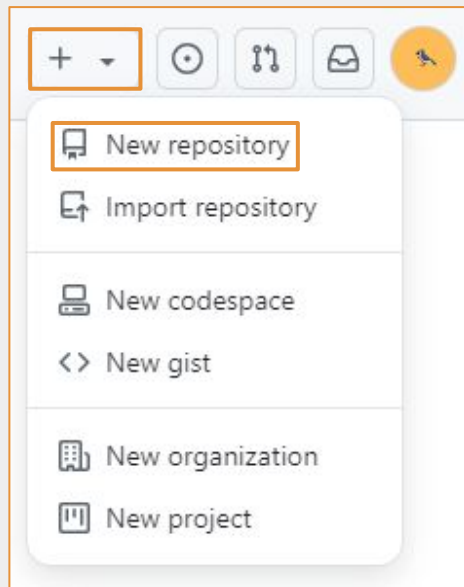
Delete

⚠ This token has no expiration date.

Gestion d'un projet BlueJ

P1 : Créer un projet & partage

1. Cliquer sur le bouton **New repository** pour créer un nouveau dépôt



Gestion d'un projet BlueJ

P1 : Créer un projet & partage

2. Remplir :

- **nom du dépôt,**
- **et son accès en Private**

3. Cocher **Add a README file**

4. Ajouter le **.gitignore** avec le template **Java**

5. Cliquer sur **Create repository**

The screenshot shows the GitHub 'Create a new repository' page. Several elements are highlighted with orange boxes and lines to correspond with the instructions:

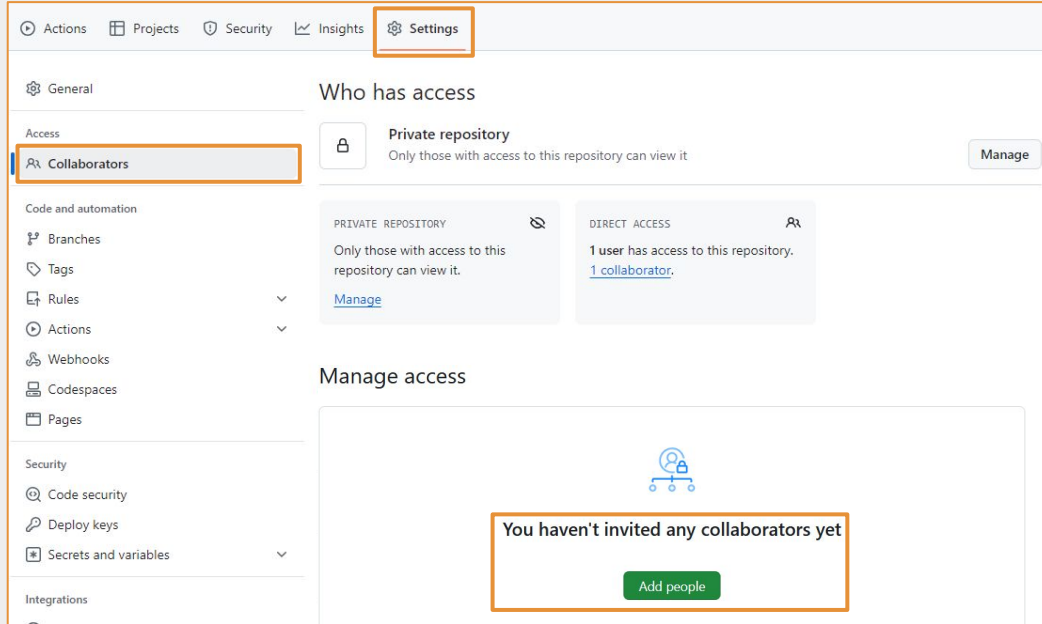
- Repository name ***: An empty text input field.
- Owner ***: A dropdown menu showing 'PitxElise'.
- Privacy**: The 'Private' radio button is selected.
- Initialize this repository with:** A section containing:
 - Add a README file**: A checkbox that is checked.
 - Add .gitignore**: A section with a dropdown menu set to '.gitignore template: None', a search input containing 'Ja', and a 'Java' template option highlighted below.
- Create repository**: A green button at the bottom right.

Other visible text includes: 'Create a new repository', 'A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)', 'Required fields are marked with an asterisk (*)', 'Great repository names are short and memorable. Need inspiration? How about **sturdy-spoon** ?', '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.', 'This is where you can write a long description for your project. [Learn more about READMEs.](#)', 'This template [relates to](#) [Learn more about ignoring files.](#)', and 'with your code. [Learn more about licenses.](#)'.

P1 : personne 1
P2 : personne 2

Gestion d'un projet BlueJ

P1 : Créer un projet & partage

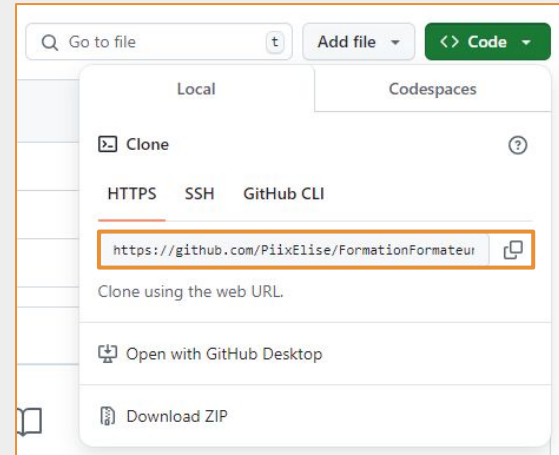


6. Dans le menu **Settings** :
- aller dans l'onglet **Collaborators**,
 - et ajouter son binôme (P2)

Gestion d'un projet BlueJ

P2 : Cloner le dépôt GitHub

1. Sur GitHub, cliquer sur le bouton **Code** et copier l'URL HTTPS

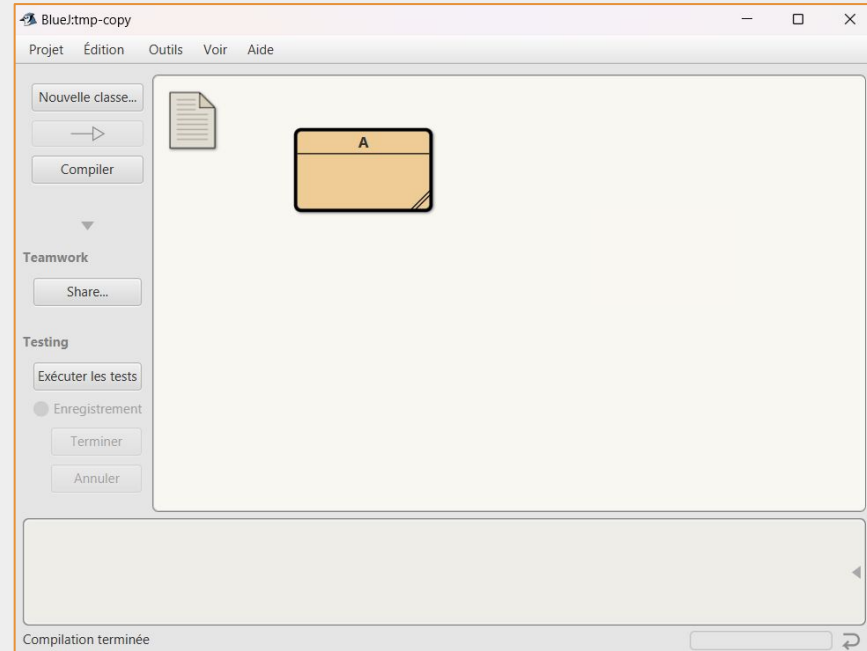


2. Cloner le projet de son binôme : **git clone** URL DU DEPOT GIT
3. Se déplacer dans le répertoire cloné :
cd NOM DU DEPOT GIT

Gestion d'un projet BlueJ

P2 : Créer un projet BlueJ

1. Créer un nouveau projet BlueJ
2. Enregistrer le projet dans le dépôt cloné
3. Créer la classe **A**
4. Conserver le remplissage de la classe par défaut
5. Enregistrer et fermer



Gestion d'un projet BlueJ

P2 : Envoyer les nouveautés sur le dépôt GitHub

1. Ajouter un fichier au suivi de version en créant un lien entre les dépôts : **git add .**
2. Enregistrer les fichiers suivis au dépôt local : **git commit -m "V1"**
3. Envoyer les modifications : **git push**

Pseudo GitHub →  ← Token généré

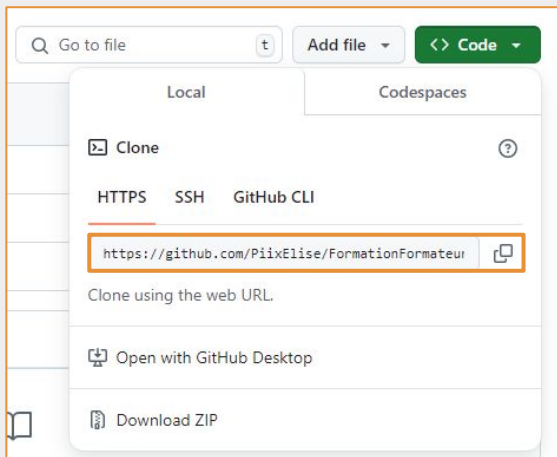
```
Username for 'https://github.com':  
Password for 'https://PiixElise@github.com':
```

4. Supprimer le clone local

Gestion d'un projet BlueJ

Ouvrir un projet GitHub via BlueJ

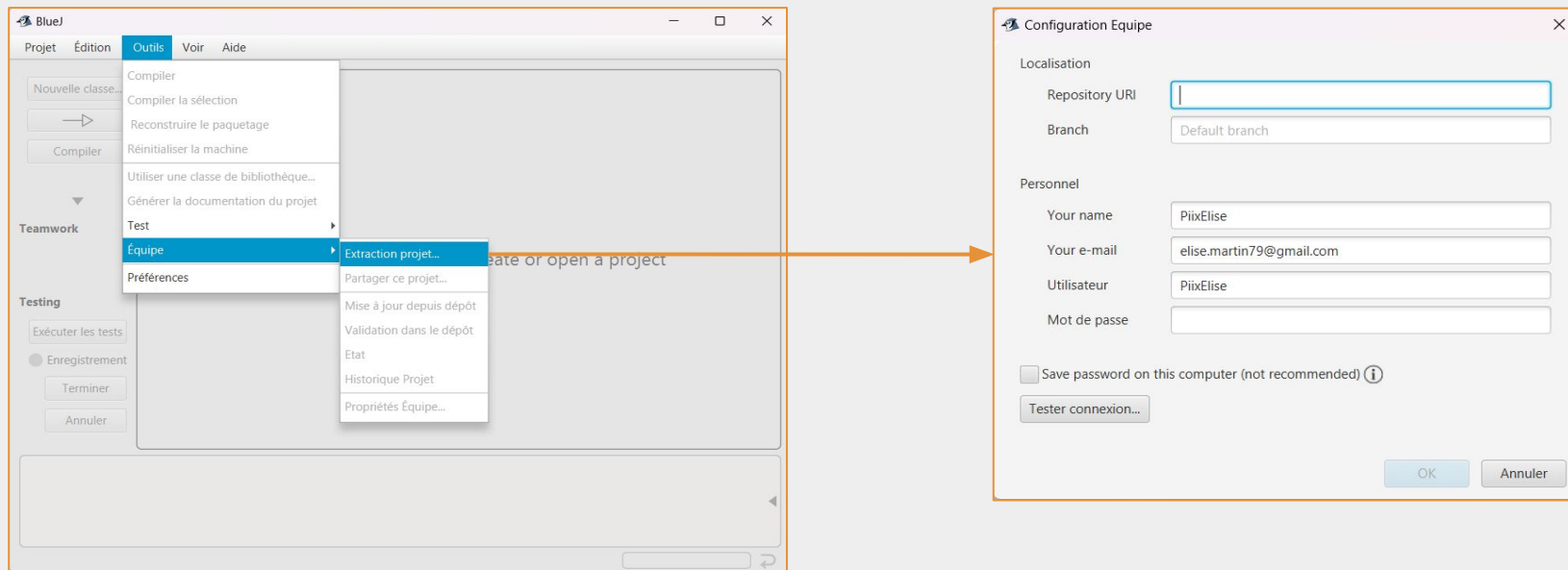
1. Sur GitHub, cliquer sur le bouton **Code** et copier l'URL HTTPS



Gestion d'un projet BlueJ

Ouvrir un projet GitHub via BlueJ

2. Sur BlueJ, allez dans **Outils > Equipes > Extraction Projet**



Gestion d'un projet BlueJ

Ouvrir un projet GitHub via BlueJ

3. Coller l'URL HTTPS dans la barre **Repository URI**

4. Remplir :

- **nom** (Your name),
- **e-mail** (Your e-mail),
- **pseudo** (Utilisateur)
- **token** (Mot de passe)

5. Cliquer sur **Tester connexion...** puis **OK**

Configuration Equipe

Localisation

Repository URI

Branch

Personnel

Your name

Your e-mail

Utilisateur

Mot de passe

☐ Save password on this computer (not recommended) ⓘ

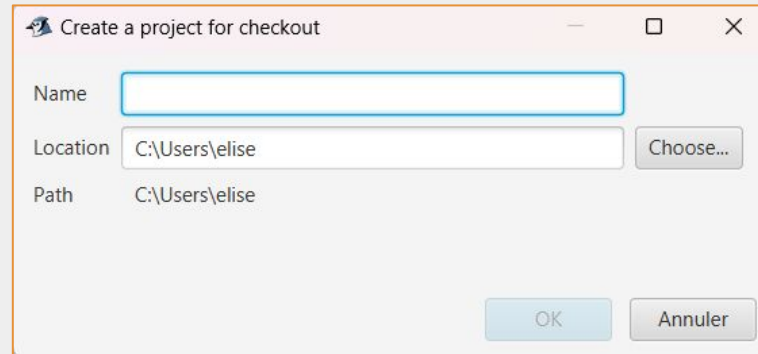
Tester connexion...

OK Annuler

Gestion d'un projet BlueJ

Ouvrir un projet GitHub via BlueJ

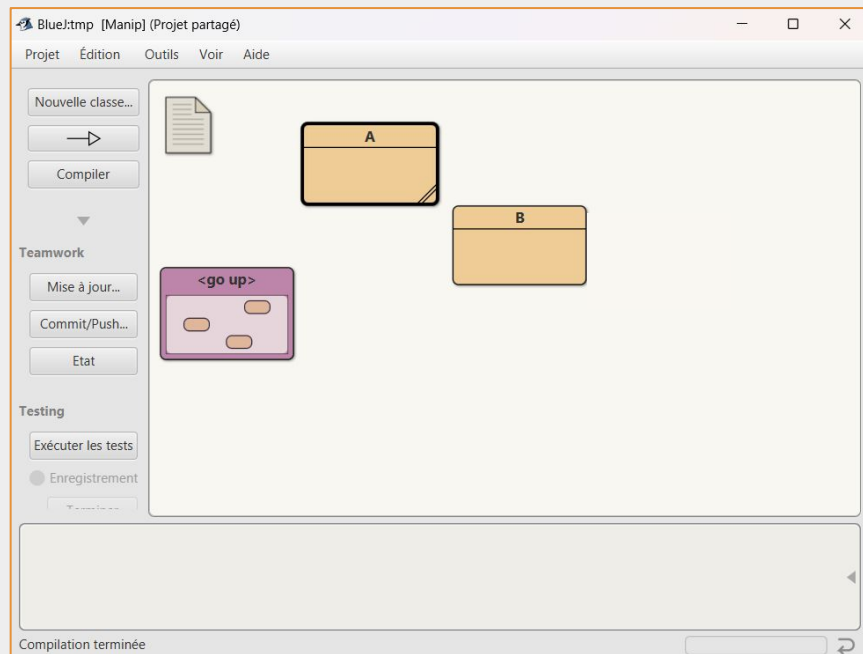
6. Entrer le nom du projet
7. Modifier la localisation pour le cloner le projet à l'endroit souhaité



Gestion d'un projet BlueJ

P1 : Modifier un projet sur BlueJ

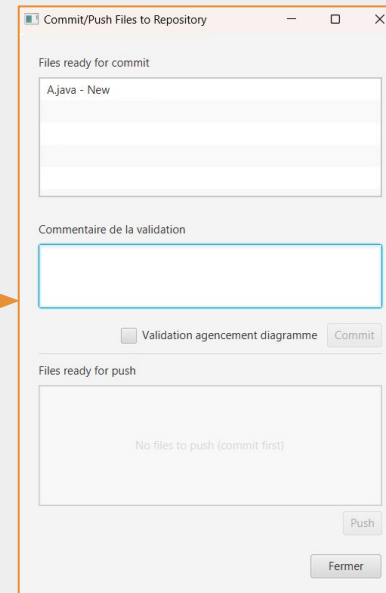
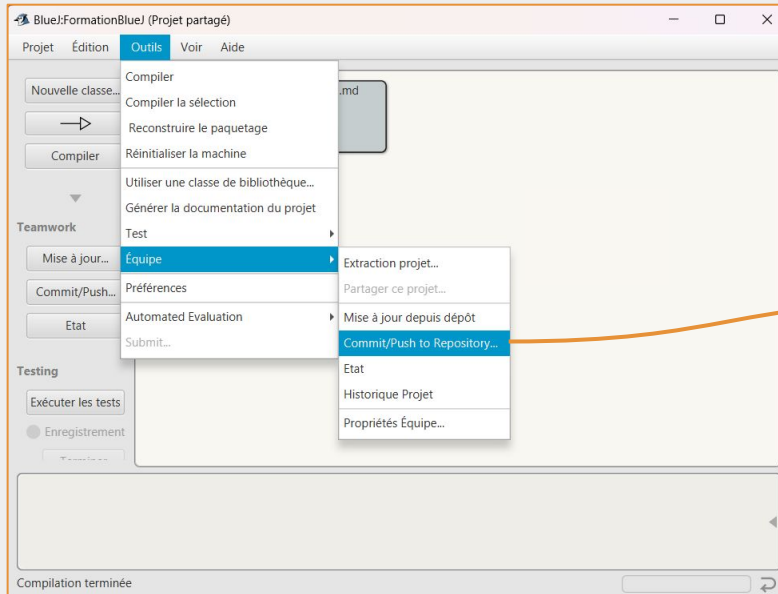
1. Créer la classe **B**
2. Conserver le remplissage de la classe par défaut



Gestion d'un projet BlueJ

P1 : Envoyer les modifications de BlueJ vers GitHub

1. Sur BlueJ, allez dans **Outils > Equipes > Commit/Push to Repository...**

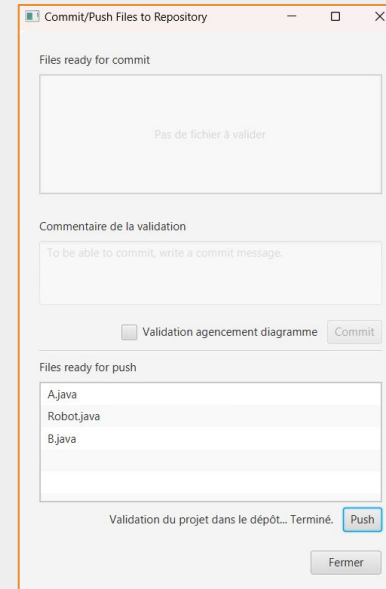


Vue de P1

Gestion d'un projet BlueJ

P1 : Envoyer les modifications de BlueJ vers GitHub

2. Ecrire un commentaire (équivalent du message du **git commit**)
3. Appuyer sur **Commit**
4. Appuyer sur **Push**
5. Fermer la page

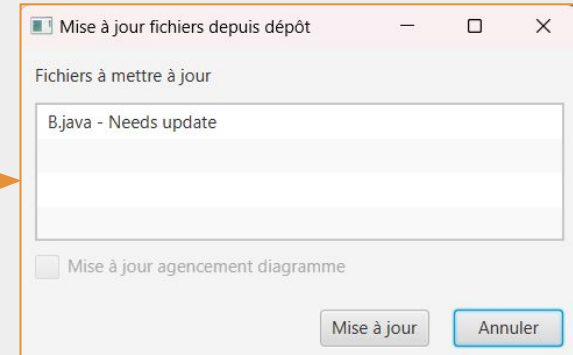
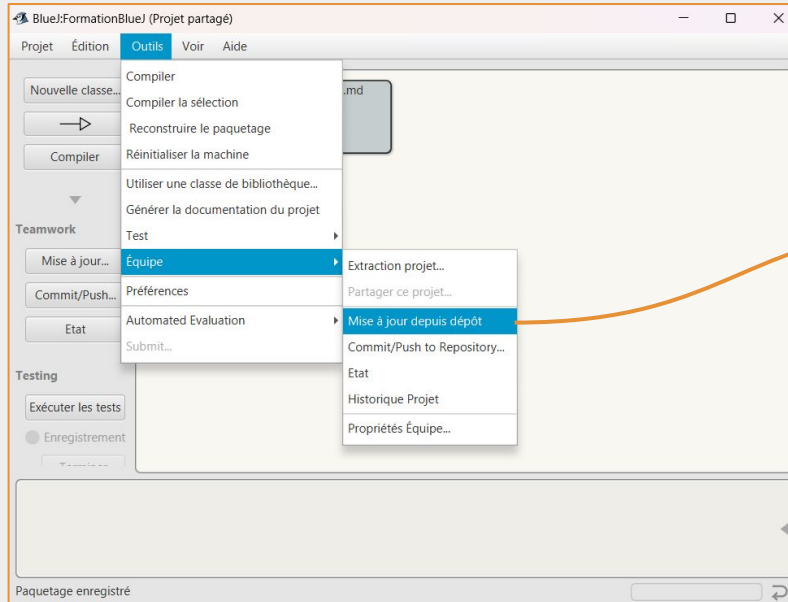


Vue de P1

Gestion d'un projet BlueJ

P2 : Récupérer des modifications

1. Sur BlueJ, allez dans **Outils > Equipes > Mise à jour depuis dépôt**

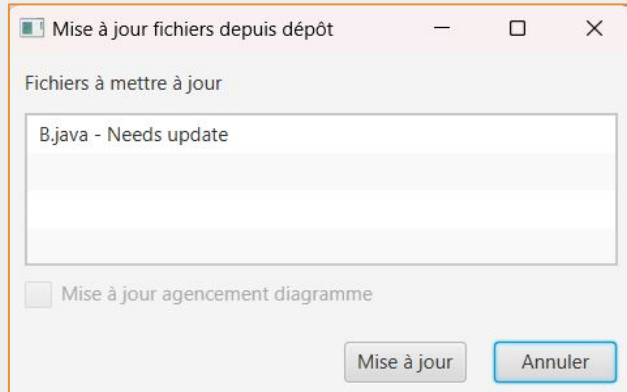


Vue de P2

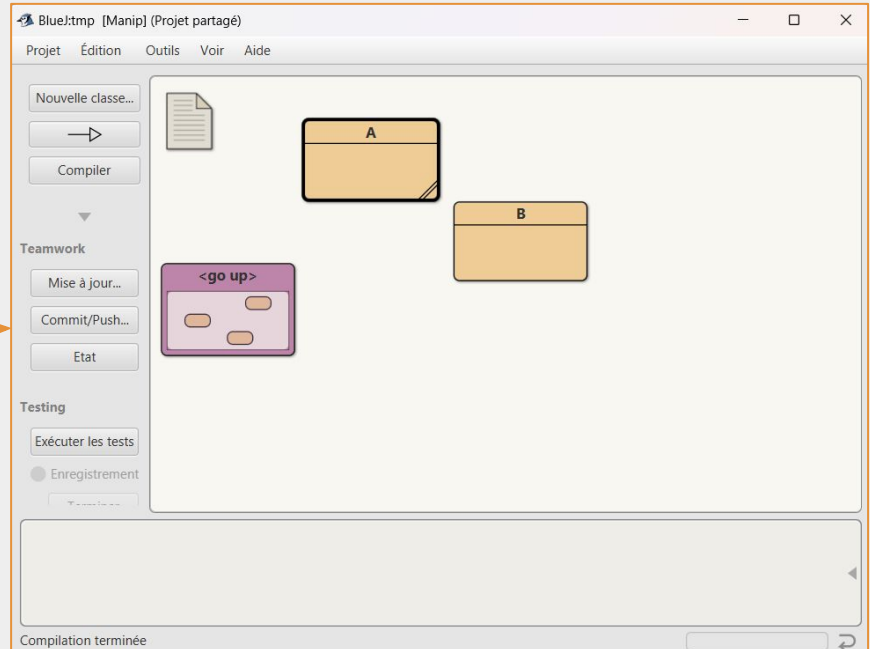
Gestion d'un projet BlueJ

P2 : Récupérer des modifications

2. Cliquer sur **Mise à jour** pour récupérer la classe créée par votre binôme



Vue de P2



P1 : personne 1
P2 : personne 2

Gestion d'un projet BlueJ

Génération d'un conflit

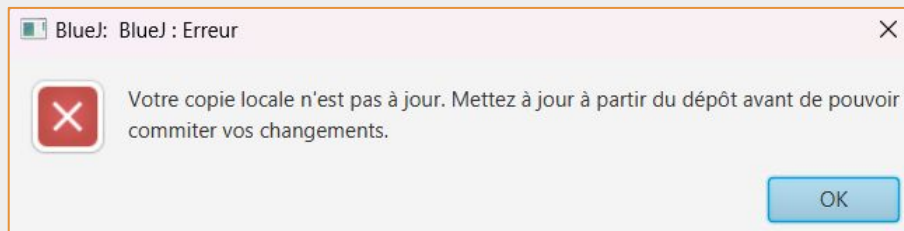
1. Ouvrir la classe A et modifier le nom de la variable :
 - P1 : renomme par **y**
 - P2 : renomme par **z**
2. Changer l'appel de la variable dans le reste de la classe
3. P1 envoie ses modifications :
Outils > Equipes > Commit/Push to Repository...

Gestion d'un projet BlueJ

Génération d'un conflit

4. P2 envoie ses modifications :

Outils > Equipes > Commit/Push to Repository...



Gestion d'un projet BlueJ

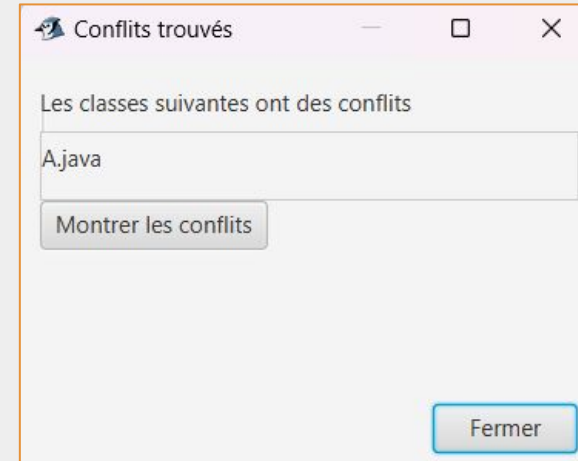
Génération d'un conflit

4. P2 envoie ses modifications :

Outils > Equipes > Commit/Push to Repository

5. P2 met à jour son dépôt local :

Outils > Equipes > Mise à jour depuis dépôt



Gestion d'un projet BlueJ

Génération d'un conflit

6. Observez les conflits sur l'ordinateur de P2

```
<<<<<< HEAD
  private int z;
=====
  private int y;
>>>>>> 1eb549055cfc34e1047a29f0b75531e8f2e843ce
```

→ Bloc de conflits 1

```
<<<<<< HEAD
  z = 0;
=====
  y = 0;
>>>>>> 1eb549055cfc34e1047a29f0b75531e8f2e843ce
```

→ Bloc de conflits 2

```
<<<<<< HEAD
  return z + y;
=====
  return y + y;
>>>>>> 1eb549055cfc34e1047a29f0b75531e8f2e843ce
```

→ Bloc de conflits 3

Gestion d'un projet BlueJ

P2 : Résolution d'un conflit

7. Corriger les erreurs (conserver une des deux variables)
8. Supprimer les marqueurs :
 - Marqueur de début de conflit : <<<<<< HEAD,
 - Séparateur de conflit : =====,
 - Marqueur de fin de conflit : >>>>>> nom branche
9. Envoyer les modifications :
Outils > Equipes > Commit/Push to Repository...

Gestion d'un projet BlueJ

Génération d'un conflit

7. P2 corrige les erreurs (conserve une des deux variables)

8. P2 supprime les marqueurs :



Toujours faire un **git pull avant de faire des modifications !**

- Séparateur de conflit : =====,

- Marqueur de début de conflit : >>>>>> nom branche

9. P2 envoie ses modifications :

Outils > Equipes > Commit/Push to Repository...

Quiz - Kahoot



Conclusion

- **Exploration des fondamentaux de Git et GitHub**
 - Initialisation et gestion des dépôts
 - Collaboration avec des binômes
 - Résolution des conflits
- **Intégration de ces outils avec BlueJ**
 - Facilitera la gestion du projet de COO-POO

Conclusion

Questions ?

Elise MARTIN :

elise.martin@etu.univ-poitiers.fr

Blandine SERRES :

blandine.serres@etu.univ-poitiers.fr

Bonus

Dépôt GitHub (URL : <https://github.com/PiixElise/FormationFormateurs.git>)

- Diaporama
- Cours rédigé récapitulatif
- Tableau récapitulatif des commandes