



**ENSAE-ENSAI**  
Formation continue

# Initiation au système de gestion de version (GIT)

Certificat Data Scientist BDF 23/24

Riad Ladjel  
Data Scientist – Quadratic  
[riad.ladjel@quadratic-labs.com](mailto:riad.ladjel@quadratic-labs.com)

# Plan

---

1. Contexte
2. Théorie et notions de base de GIT
3. Fonctionnalités avancées de GIT
4. Travailler à plusieurs avec GIT



# Contexte

Pourquoi faut-il utiliser un système de gestion de versions

---

- Chaque projet informatique évolue dans le temps
  - Des bugs sont découverts et doivent être corrigés
  - De nouvelles fonctionnalités sont développées
- Comment faire pour ajouter apporter des modifications ?
  - Travailler directement sur le projet ?
    - **risque de perte de version stable**
  - Avoir plusieurs dossiers pour chaque modification ?
    - **risque de s'y perdre très rapidement**



# Contexte

Pourquoi faut-il utiliser un système de gestion de versions

---

- Un projet informatique nécessite de pouvoir travailler en équipe
- Comment collaborer sur un même projet ?
  - Les fichiers sont envoyés par mail ?
  - Travailler sur un serveur ? Que se passe-t-il si on modifie le même fichier ?

**SOLUTION** : systèmes de gestion de versions



# Systèmes de gestion de versions

## Historique

---

- La gestion de versions permet de gérer plusieurs versions d'un document, d'un programme
  - Permet de garder un historique chronologique des modifications
  - Et de passer d'une version à une autre facilement
- Les systèmes de gestion de versions les plus connus :



- Le plus répandu est GIT

# Pourquoi GIT ?

## Rapide introduction

---

- Git est un logiciel de gestion de versions
  - **Décentralisé**, permet de travailler hors ligne sans serveur distant
  - **libre** d'utilisation
  - et **gratuit**
- Créé en 2005 par l'auteur de linux (Linus Torvalds), pas convaincue par l'existant (cvs et svn)
  - Ne passent pas à l'échelle
  - Pas gratuits
- Git offre de nombreux avantages (en plus d'être décentralisé, libre et gratuit):
  - Simple d'utilisation (notamment avec les interfaces graphiques)
  - Rapide



# Théorie et notions de base de GIT

# La théorie de GIT

## Comprendre la structure d'un projet Git

---

Zone de travail  
(Working directory)

Zone de transit  
(Index ou staging area)

Dépôt local  
(Local repository)

Concrètement un projet sous git se compose de trois zones:

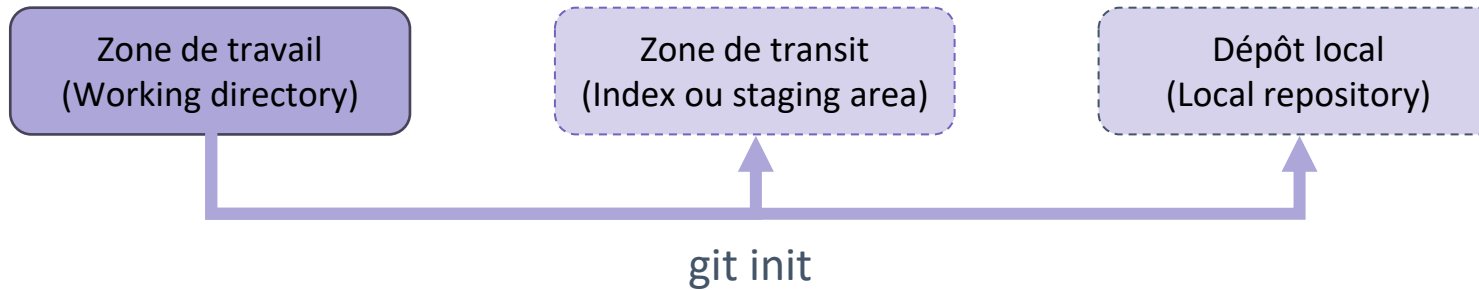
- Zone de Travail
  - Visible naturellement, c'est le répertoire où vous éditez vos fichiers
  - Les modifications dans ce répertoire ne sont pas encore suivies par Git
- Dépôt Local
  - Où Git stocke définitivement l'historique des versions du projet
  - Non visible directement
- Zone de Transit
  - Zone intermédiaire
  - Prépare la prochaine version/commit, pouvant être enregistrée dans le dépôt local
  - Non visible directement





# Fonctionnement basic de GIT

## Initialisation d'un dépôt



- **git init** permet d'initialiser le dépôt
  - C'est-à-dire créer la zone de transit et le dépôt local
- C'est la première commande à exécuter habituellement
  - Sauf si on clone un projet existant (voir plus loin)

# Comprendre les états des fichiers dans Git

## État de la zone de travail

- Après modification de la zone de travail (ajout, modification suppression), on peut voir l'état de notre zone de travail avec **git status**

```
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   premier_fichier.txt

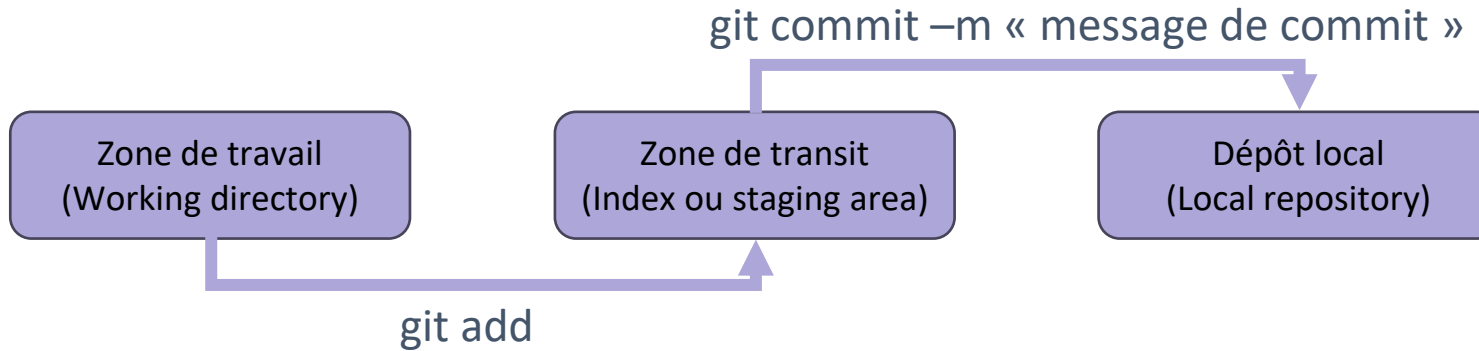
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   troisieme_fichier.txt

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

- Trois états possibles :
  - 1-modifiés et sera pris en compte dans la prochaine version
  - 2-modifiés mais ne sera pas pris en compte pour la prochaine version
  - 3-non suivi par GIT

# Premier commit

## Processus d'ajout et de validation dans Git



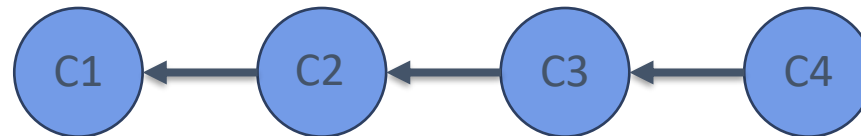
- Pour ajouter un fichier dans la zone de transit:
  - **git add chemin\_du\_fichier**
  - Ou bien **git add** . Pour ajouter tous les fichiers
- Ensuite, pour pousser le fichier dans le dépôt local:
  - **git commit -m 'message de commit'**
  - Le message doit être court < 65 caractères et décrit bien les modifications apportées

# Les commits

## Le rôle essentiel des commits dans Git

---

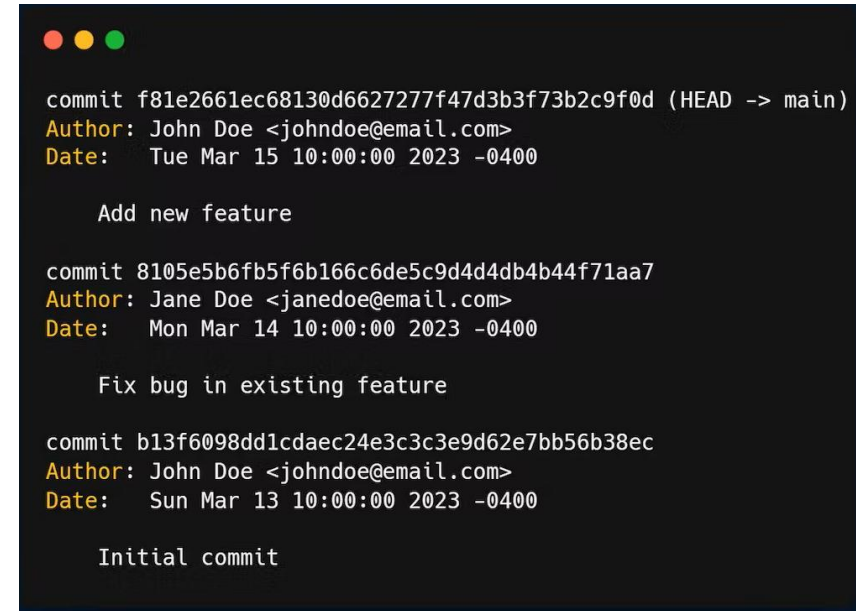
- C'est quoi un commit ?
  - Brique centrale de GIT
  - Chaque version sauvegardée dans GIT est appelée un commit.
  - Contient les changements effectués dans les fichiers du projet (ajouts, suppressions, modifications)
- Chaque commit est identifié par une chaîne de caractère unique
  - C'est le hash de l'ensemble des informations contenues dans ce commit.
- Les commits sont organisés sous forme d'une chaîne
  - chaque commit pointe sur le ou les commit(s) parent(s) dont il est issu.



# Les commits

## Exploration de l'historique des commits

- Pour afficher la liste de tous les commit, on utilise **git log**
- Git log affiche:
  - L'auteur du commit
  - La Date du commit
  - Le message de commit
  - Et l'identifiant

A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. It displays the output of the 'git log' command, showing three commits in reverse chronological order. Each commit entry includes the commit hash, author name and email, date, and a descriptive message.

```
commit f81e2661ec68130d6627277f47d3b3f73b2c9f0d (HEAD -> main)
Author: John Doe <johndoe@email.com>
Date: Tue Mar 15 10:00:00 2023 -0400

    Add new feature

commit 8105e5b6fb5f6b166c6de5c9d4d4db4b44f71aa7
Author: Jane Doe <janedoe@email.com>
Date: Mon Mar 14 10:00:00 2023 -0400

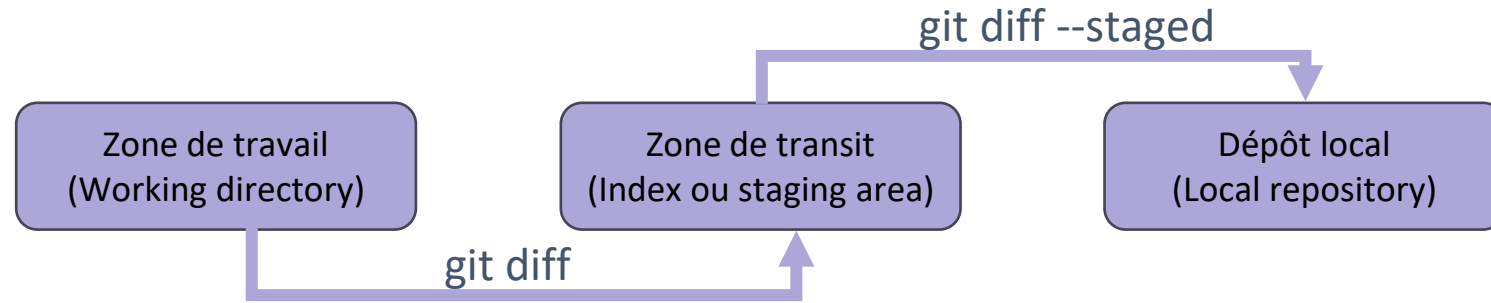
    Fix bug in existing feature

commit b13f6098dd1cdaec24e3c3c3e9d62e7bb56b38ec
Author: John Doe <johndoe@email.com>
Date: Sun Mar 13 10:00:00 2023 -0400

    Initial commit
```

# Affichage des différences dans Git

## Visualisation des modifications



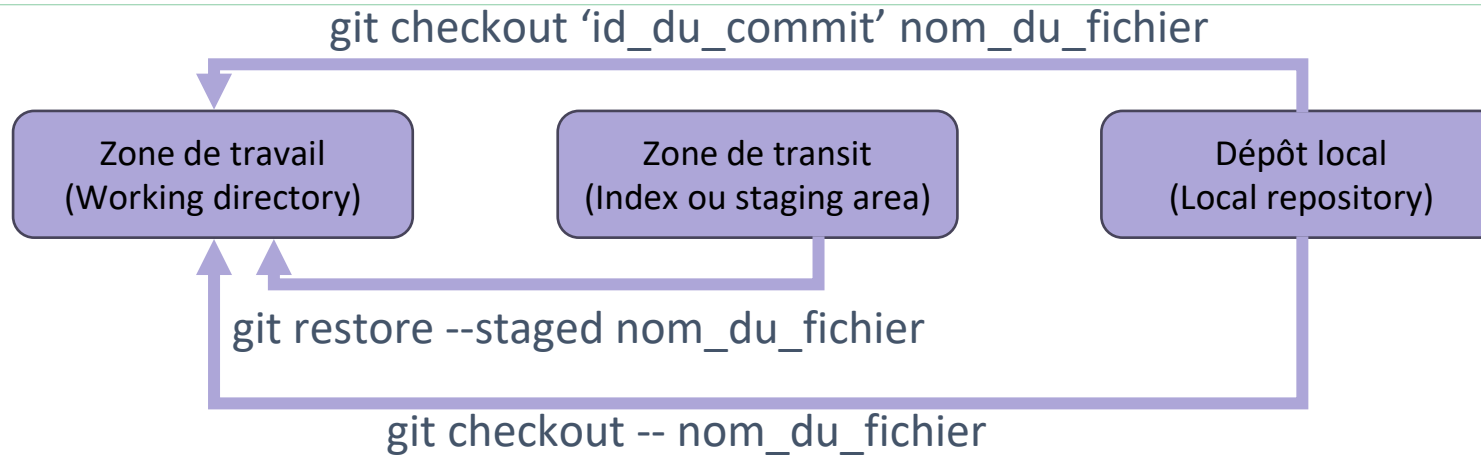
- On peut aussi voir la différence de contenu modifié
  - Entre zone de travail et zone de transit : **git diff**
  - Entre zone de transit et dépôt local : **git diff --staged**

```
>> git diff
diff --git a/premier_fichier.txt b/premier_fichier.txt
index 2dedbc1..f5ef272 100644
--- a/premier_fichier.txt
+++ b/premier_fichier.txt
@@ -1,3 +1,3 @@
 1ere ligne pour tester
 une autre ligne
-3eme ligne...
\ No newline at end of file
+Ajout d'une nouvelle ligne
\ No newline at end of file
```

# Fonctionnalités avancées de GIT

# Gestion des changements avec Git

## Voyager dans le temps



- GIT permet d'appliquer facilement les changements inverses entre zones
- Pour retirer un fichier de la zone de transit
  - **git restore --staged nom\_du\_fichier**
- Pour remettre un fichier dans l'état du dernier commit
  - **git checkout -- nom\_du\_fichier**
- Pour remettre un fichier dans l'état d'un ancien commit
  - **git checkout 'id\_du\_commit' nom\_du\_fichier**

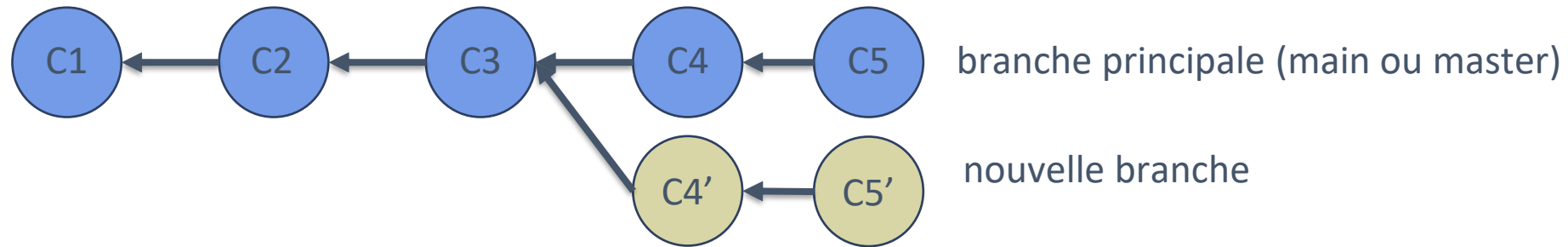


# Les branches

## Gestion des branches

---

- GIT offre la possibilité de créer des branches sur notre projet
- Une branche est une version parallèle du projet créé à partir d'un commit



- Utilité d'une branche
  - Apporter une évolution sans impacter le projet
  - Tester différentes implémentations d'une même fonctionnalité de manière indépendante

# Les branches

## Flexibilité des branches dans Git



- Il est possible d'avoir autant de branches qu'on veut
- Les modifications sur une branche n'affectent pas les autres branches
- Les branches peuvent être fusionnées
  - **git merge nom\_branche\_à\_merger**

# Gestion des conflits de fusion dans Git

## Ce qui fait peur dans GIT

- Un conflit se produit lorsque vous modifiez différemment la même partie du même fichier
  - Fusion de deux branches
  - Ou lors d'un pull sur un dépôt distant
- Git n'est pas capable de fusionner automatiquement les modifications
- Vous devez alors régler ces conflits manuellement
  - **git status** donne les fichiers n'ayant pas pu être fusionnés ("unmerged").
  - Modifié les fichiers manuellement
    - <<<<<< Indique les changements locaux
    - >>>>>> Indique les changements qui arrivent
  - Refaire un git add, git commit

```
>> git status
On branch main
Your branch and 'origin/main' have diverged,
and have 1 and 1 different commits each, respectively.
(use "git pull" if you want to integrate the remote branch with yours)

You have unmerged paths.
(fix conflicts and run "git commit")
(use "git merge --abort" to abort the merge)

Unmerged paths:
(use "git add <file>..." to mark resolution)
    both modified:   a.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

```
<<<<<< HEAD
    <li><a href="index.html">Home</a></li>
    <li><a href="about.html">About Us</a></li>
    <li><a href="product.html">Product</a></li>
    <li><a href="imprint.html">Imprint</a></li>
=====
    <li><a href="returns.html">Returns</a></li>
    <li><a href="faq.html">FAQ</a></li>
>>>>>> develop
    </ul>
```

# Travailler à plusieurs avec GIT

# Collaboration à travers les dépôts distants

L'avantage apporté par les serveurs distants

---

Zone de travail  
(Working directory)

Zone de transit  
(Index ou staging area)

Dépôt local  
(Local repository)

Dépôt distant  
(Remote repository)

- Le dépôt distant : est une version partagée du projet, généralement hébergée sur un serveur distant



- Un dépôt distant permet de stocker tous les commits et branches du projet
- Et permet à différents collaborateurs de contribuer au même projet

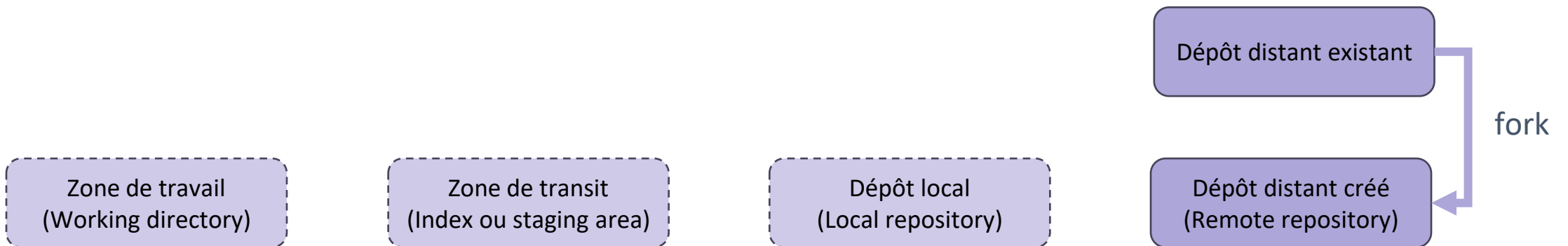
# Utilisation des dépôts distants

## Création de dépôts distants dans Git

- Deux manières pour créer un dépôt distant:
  - Création en ligne directement sur le site (interface graphique)



- En faisant un « fork » (i.e. copie) d'un autre dépôt existant (également en ligne)



# Utilisation des dépôts distants

## Sécurisation de la communication avec le dépôt distant

- Pour pouvoir communiquer avec le dépôt distant, il faut établir un canal sécurisé avec le serveur.
- Un canal sécurisé est une façon de transmettre des données entre deux machines (en utilisant un protocole cryptographique « ssh »)

1. Création d'une paire de clés  
publique/privée (avec ssh)



2. La clé publique est  
enregistrée sur le serveur



3. Les messages transmis  
sont maintenant chiffrés

ccadd99b16cd3d200c2

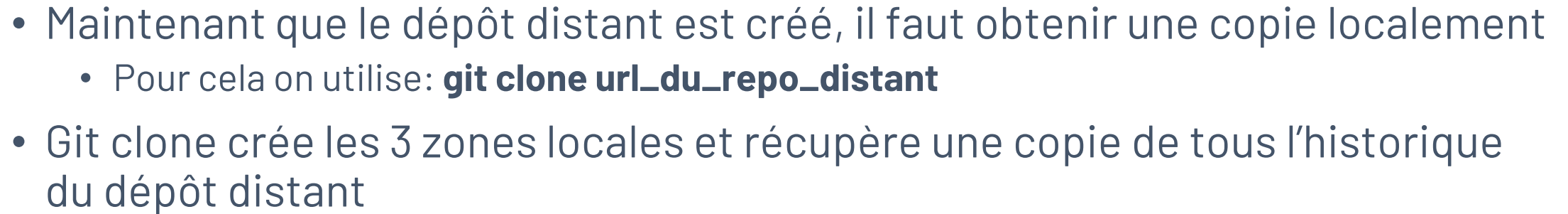


3a62ea1facd8584e44e



**ATTENTION** : le protocole ssh n'étant pas l'objet de ce cours, son fonctionnement a été **LARGEMENT** simplifié

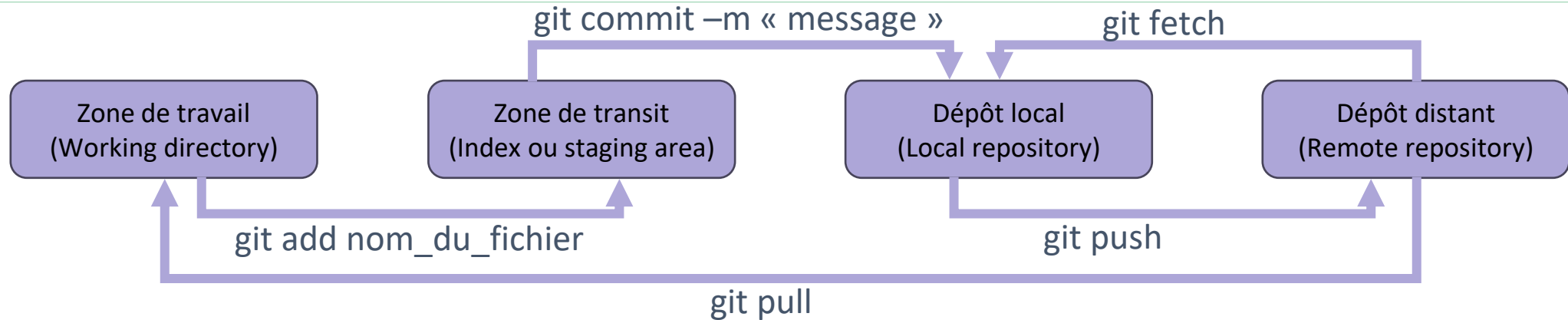
## Lier un dépôt distant à un dépôt local





# Utilisation des dépôts distants

## Gestion des échanges avec le dépôt distant



- Les premières étapes du flux de travail sont identiques à la version locale :
  - modification de fichiers, `git add`, `git commit`, ..etc
- Ensuite, pour envoyer une copie des commits locaux au dépôt distant :
  - `git push`
- Pour récupérer une copie des commits du dépôt distant :
  - `git fetch`
- Pour récupérer les commits du dépôt distant et mettre à jour la zone de travail :
  - `git pull`

# Utilisation efficace de Git

## Quelques bonnes pratiques en vrac

---

- GIT est adapté à la gestion de texte mais pas mais pas trop aux autres types de fichiers
- Ne jamais commiter les fichiers de configuration locaux
  - Surtout ceux qui continent des mots de passe
- Faire des commits et sauvegarde sur le dépôt distant régulièrement
- Utiliser une nouvelle branche pour chaque nouvelle fonctionnalité
- Ne jamais travailler sur la branche main/master directement
- Faire des pulls avant de commencer à travailler
  - Pour avoir la copie du dépôt la plus récente



# Conclusion

---

- Git est un outil puissant et simple d'utilisation qui simplifie la gestion de versions et le travail en équipe
- Il existe plusieurs outils permettant d'avoir une interface graphique pour utiliser GIT:
  - GitHub Desktop
  - Extensions VsCode
  - Rstudio
- Aller plus loin:
  - **git stash**
  - **git rebase**
  - .gitignore
  - ...

