ASSURER LA MAINTENANCE CORRECTIVE OU ÉVOLUTIVE D'UNE SOLUTION APPLICATIVE 1/6

GIT ET LA GESTION DE VERSIONS

Auteur : Sébastien Inion

studi

Le 08/01/2025 de 17:00 à 18:00
Ajouter au calendrier

Travaux Dirigés

Description

"Objectif :
Comprendre les bases de Git pour la gestion de version.
Apprendre à utiliser GitHub pour héberger et collaborer sur des projets.

Prérequis :
Connaissances de base en ligne de commande (terminal).
Avoir Git installé sur la machine.

Contenu :
Introduction à Git
Introduction à GitHub
Flux de travail Git et GitHub

Exercices pratiques :
Initialiser un dépôt Git local et y ajouter des fichiers.
Connecter un projet Git à un dépôt GitHub et effectuer un push.
Créer une branche, y faire des modifications, puis la fusionner dans la branche principale

Problèmatique

Gestion d’un projet avec plusieurs développeurs devant intervenir sur plusieurs fichiers. Comment faire ?

- Comment se passer les informations, les changements ?
- Supports ? DD externe, clefs, cloud ?
- Pannes des supports
- On a fait une petite modification sur un gros fichiers de plusieurs Go. Cela ne passe pas par courrier …
- Qui a fait quoi ? Modification… Suppression.. Ajout…

Problèmatique

Gestion d’un projet faisant intervenir plusieurs développeurs intervenant sur plusieurs fichiers. Comment faire ?

- On veut retrouver le fichier tel qu’il était il y a une semaine
- 4 développeurs travaillent sur un même fichier. Comment fusionner ?

Logiciel de gestion de versions

- Répond à toutes ces questions
- Pour le travail à plusieurs sur des projets de plusieurs fichiers et dossiers
- il permet d’avoir un historique de vos fichiers
- Offre aussi des outils de communication

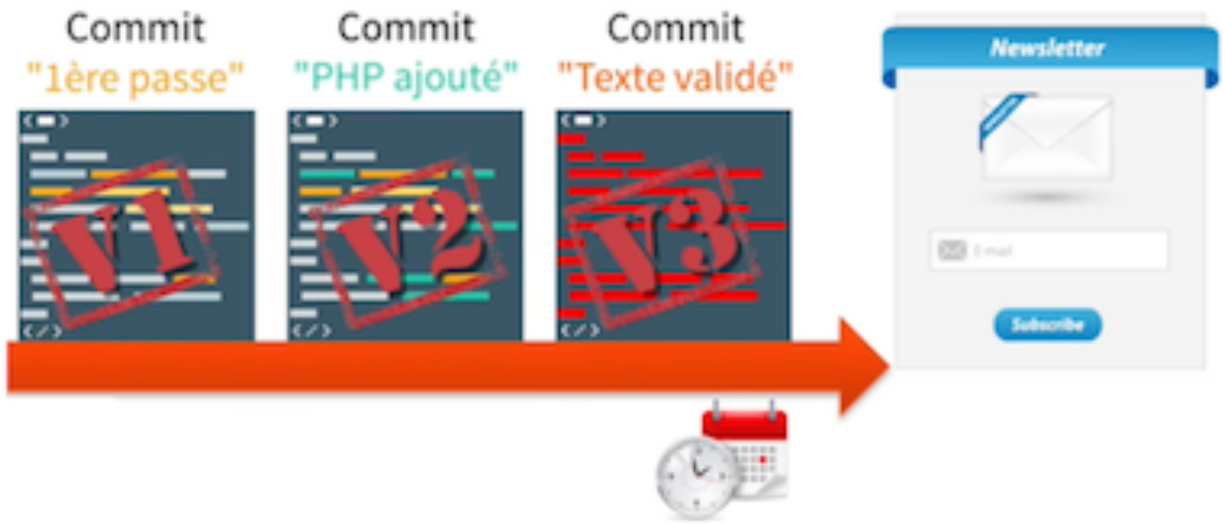
Pourquoi versionner votre code ?

- ☑ revenir en arrière sur un code valide

- ☑ pour valider une modification et une étape de votre code

Par exemple, si vous travaillez sur un formulaire de newsletter en ligne :

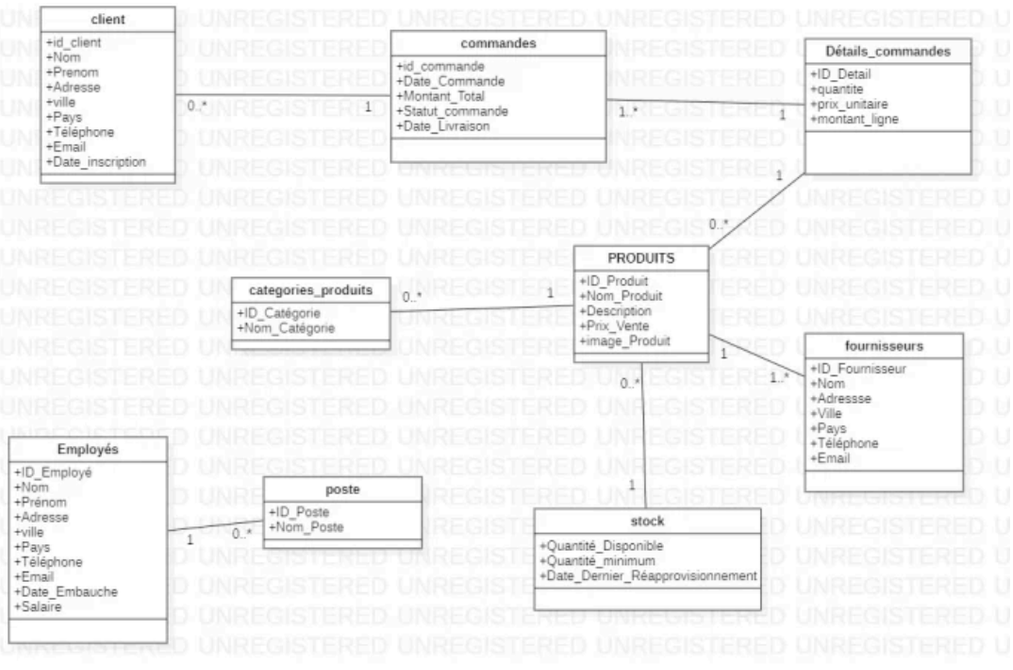
- Vous allez d'abord faire une 1re série de modifications pour créer la base HTML/CSS de votre formulaire.
  => Une fois ces modifications faites, vous pourrez faire un premier commit dans Git et le nommer "1re passe sur le formulaire de newsletter".
- Ensuite, vous rendrez peut-être votre formulaire dynamique avec du PHP.
  => 2e commit : "PHP ajouté".
- Enfin, un collègue vous demandera de changer le wording sur le champs du formulaire.
  => 3e commit : "Texte reformulé".



GESTION DE VERSION ?

- Bases de données relationnelles (SQL) : Utilisent un modèle structuré basé sur des tables (relations), où les données sont organisées en lignes et colonnes.

- Les relations entre les données sont définies par des clés étrangères, ce qui impose une structure rigide.

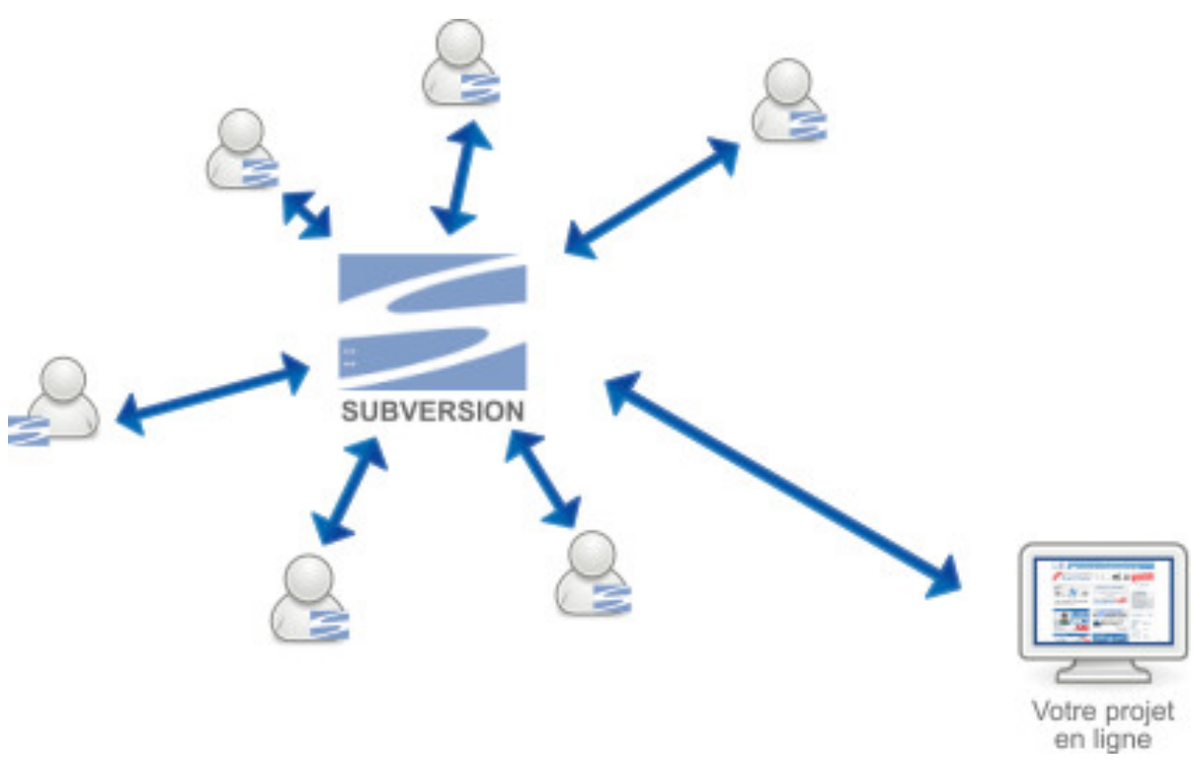


La gestion de versions : généralités, vocabulaire

- La gestion de versions consiste à maintenir l'ensemble des versions d'un projet : c'est l'historique.
- Chaque étape d'avancement d'un projet est appelée révision. (c'est le commit)
- Entre deux révisions on a des modifications —> Delta
- Ce delta est appelé « diff » ou « patch »
- On parle parfois de versions mais il ne faut pas confondre avec la version d'un logiciel qui est une étape de distribution sous forme « finie »
- On peut revenir à n'importe quel endroit de l'historique

Gestion de versions centralisée et décentralisée

- Gestion de versions centralisée : un dépôt fait référence
  - CVS
  - SVN -> Subversion



Gestion de versions centralisée et décentralisée

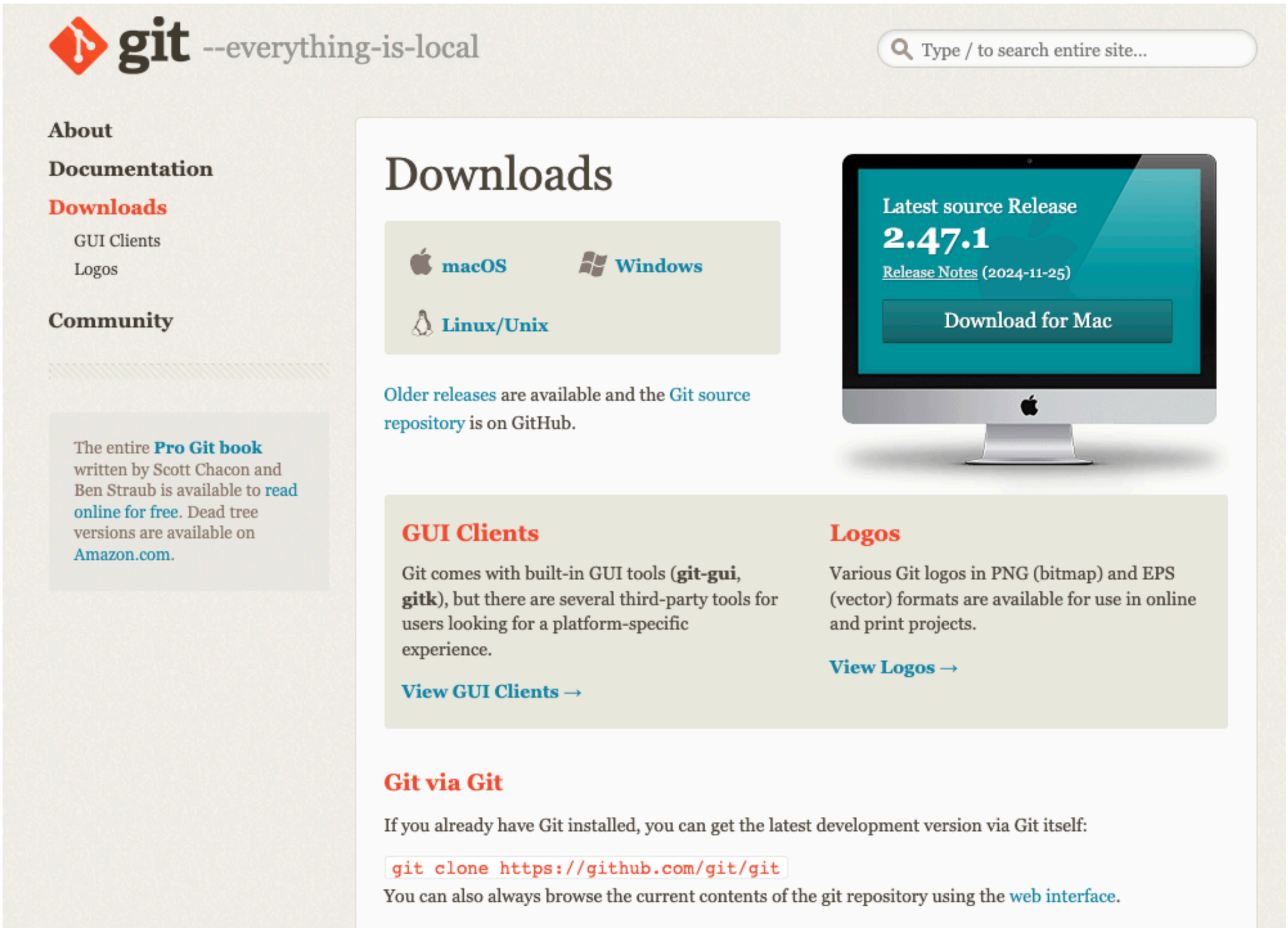
- Gestion de versions décentralisée
  - Git
  - Mercurial
  - Bazaar

GIT ET LA GESTION DE VERSIONS

Particularités de GIT

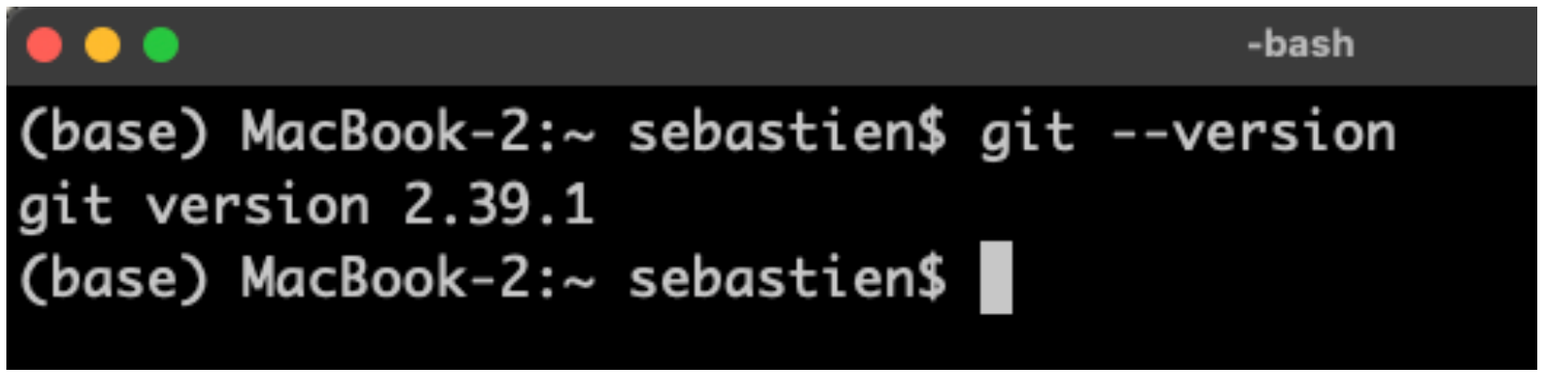
- Très populaire
- Logiciel Libre
- Git sera toujours gratuit
- Sa performance
- Sait travailler par branche de manière très flexible
- A l’origine prévu pour Linux (mais fonctionne sous Mac OS/X et Windows)

Installation de Git



Installer Git

Puis tester si git est bien sur votre système :



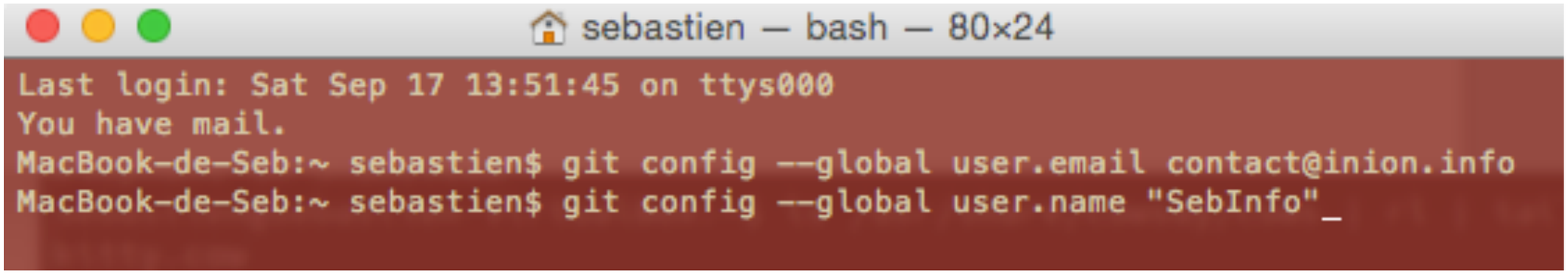
Avoir de l’aide Git

```
(base) MacBook-2:~ sebastien$ git help
usage : git [--version] [-h | --help] [-C <chemin>] [-c <nom>=<valeur>]
           [--exec-path[=<chemin>]] [--html-path] [--man-path] [--info-path]
           [-p | --paginate | -P | --no-pager] [--no-replace-objects] [--bare]
           [--git-dir=<chemin>] [--work-tree=<chemin>] [--namespace=<nom>]
           [--super-prefix=<chemin>] [--config-env=<nom>=<variable-d-environnement>]
           <commande> [<args>]

Ci-dessous les commandes Git habituelles dans diverses situations :

démarrer une zone de travail (voir aussi : git help tutorial)
   clone     Cloner un dépôt dans un nouveau répertoire
   init      Créer un dépôt Git vide ou réinitialiser un existant

travailler sur la modification actuelle (voir aussi : git help revisions)
   add       Ajouter le contenu de fichiers dans l'index
   mv        Déplacer ou renommer un fichier, un répertoire, ou un lien symbolique
   restore   Restaurer les fichiers l'arbre de travail
   rm        Supprimer des fichiers de la copie de travail et de l'index

examiner l'historique et l'état (voir aussi : git help revisions)
   bisect    Trouver par recherche binaire la modification qui a introduit un bogue
   diff      Afficher les changements entre les validations, entre validation et copie de travail, etc
   grep      Afficher les lignes correspondant à un motif
   log       Afficher l'historique des validations
   show      Afficher différents types d'objets
   status    Afficher l'état de la copie de travail
```

git add —help

```
GIT-ADD(1)                                Git Manual                                GIT-ADD(1)

NAME
       git-add - Add file contents to the index

SYNOPSIS
       git add [--verbose | -v] [--dry-run | -n] [--force | -f] [--interactive | -i] [--patch | -p]
                 [--edit | -e] [--[no-]all | --[no-]ignore-removal | [--update | -u]] [--sparse]
                 [--intent-to-add | -N] [--refresh] [--ignore-errors] [--ignore-missing] [--renormalize]
                 [--chmod=(+|-)x] [--pathspec-from-file=<file> [--pathspec-file-nul]]
                 [--] [<pathspec>...]

DESCRIPTION
       This command updates the index using the current content found in the working tree, to prepare the
       content staged for the next commit. It typically adds the current content of existing paths as a whole,
       but with some options it can also be used to add content with only part of the changes made to the
       working tree files applied, or remove paths that do not exist in the working tree anymore.

       The "index" holds a snapshot of the content of the working tree, and it is this snapshot that is taken as
       the contents of the next commit. Thus after making any changes to the working tree, and before running
       the commit command, you must use the add command to add any new or modified files to the index.

       This command can be performed multiple times before a commit. It only adds the content of the specified
       file(s) at the time the add command is run; if you want subsequent changes included in the next commit,
       then you must run git add again to add the new content to the index.

       The git status command can be used to obtain a summary of which files have changes that are staged for
       the next commit.

       The git add command will not add ignored files by default. If any ignored files were explicitly specified
```

Première Etape : indiquer votre identité



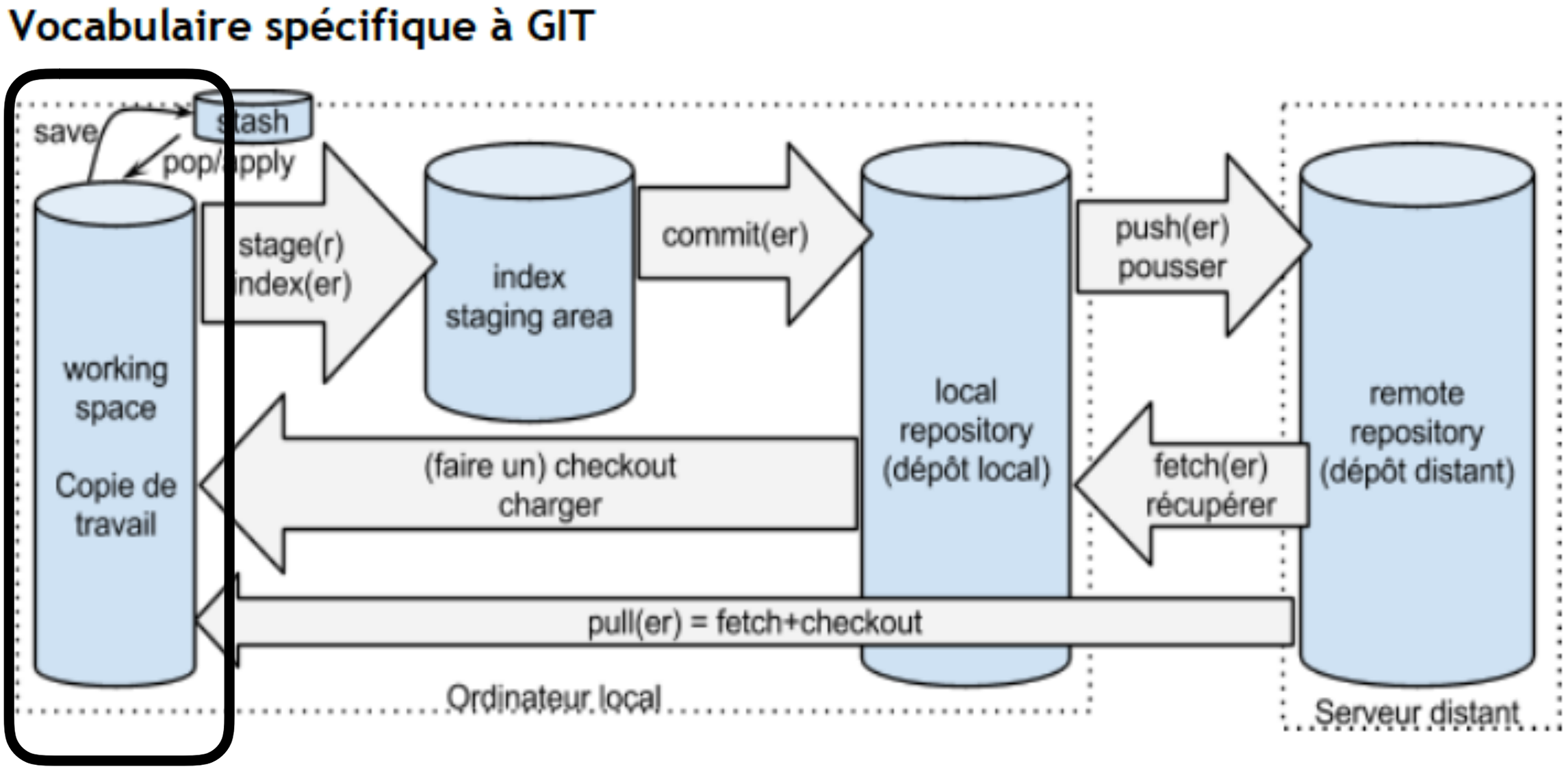
Création d'un dépôt

On se place dans le répertoire que l'on veut (ici je viens de le créer) :

- commande **git init**
- Ceci crée un répertoire **.git** avec les méta-informations nécessaire à Git
- Le répertoire de travail est **TestRepGit**
- Remarque : le dossier peut ne pas être vide. Placez vous à la racine de ce que vous voulez versionner.

```
pc6:Informatique sebastien$ ls
Android         TestRepGit
pc6:Informatique sebastien$ cd TestRepGit/
pc6:TestRepGit sebastien$ ls
pc6:TestRepGit sebastien$ git init
Initialized empty Git repository in /Users/sebastien/Documents/Informatique/Test
RepGit/.git/
pc6:TestRepGit sebastien$ ls
pc6:TestRepGit sebastien$ ls -l
pc6:TestRepGit sebastien$ ls -al
total 0
drwxr-xr-x   3 sebastien  staff  102 12 mar 15:07 .
drwxr-xr-x   4 sebastien  staff  136 12 mar 15:07 ..
drwxr-xr-x  10 sebastien  staff  340 12 mar 15:07 .git
pc6:TestRepGit sebastien$
```

Vocabulaire spécifique à Git

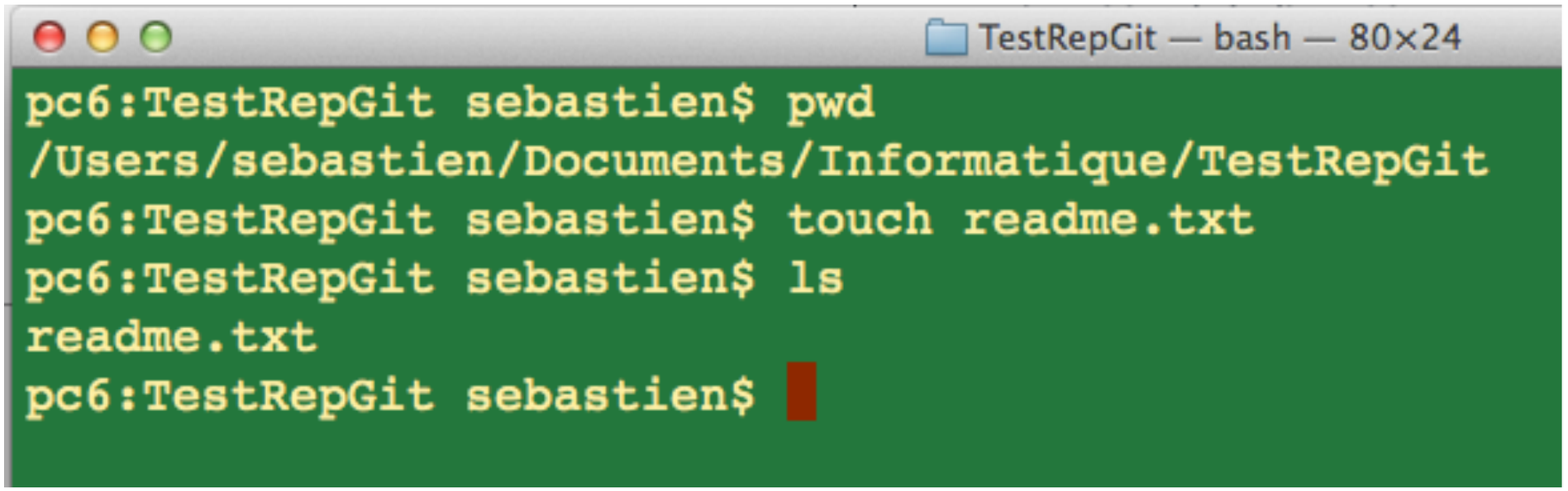


Troisième Etape : création d'un fichier

On se place dans le répertoire TestRepGit.

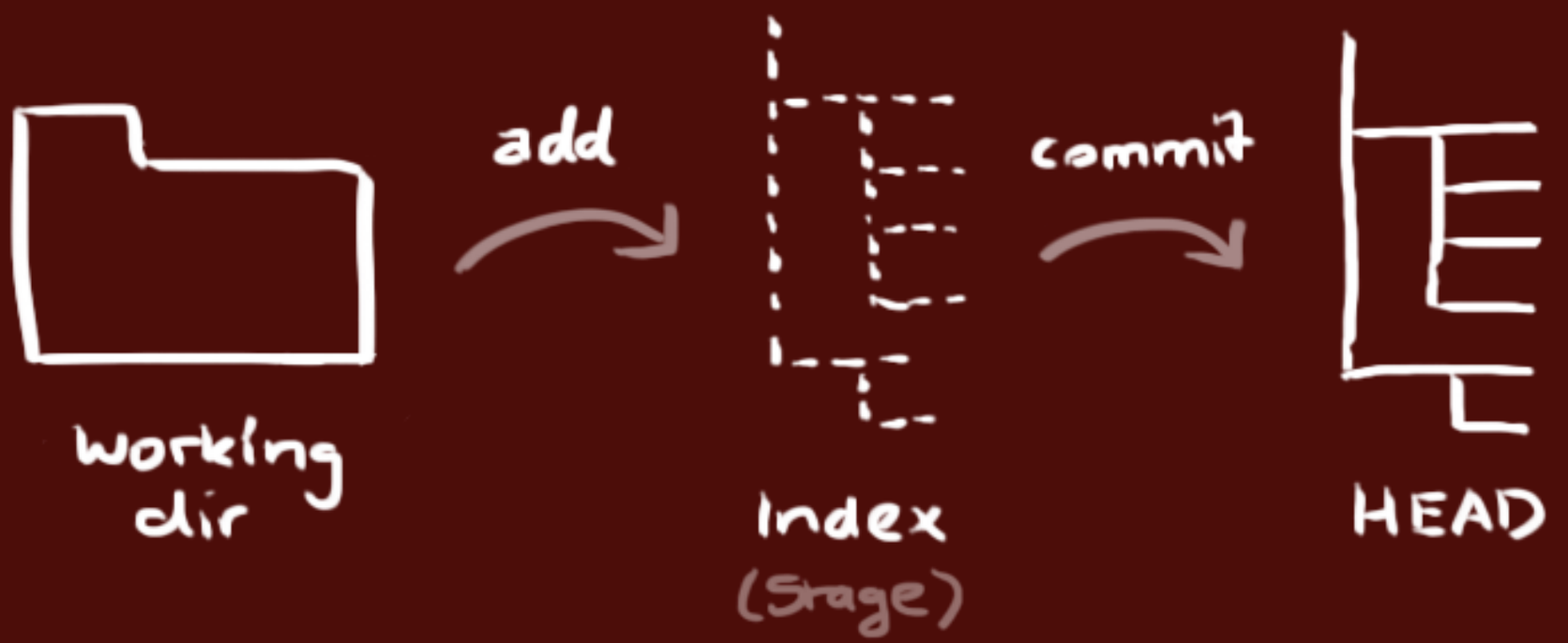
Cette zone correspond au répertoire du système de fichiers sur lequel travaille le développeur.

Remarque *: une bonne pratique est de créer un fichier README afin de préciser l'objectif du projet. Ce fichier contient plusieurs parties (présentation du projet, utilisation, liens ressources, informations pour les contributeurs, liste des auteurs, la ou les licences, moyens de contacter les mainteneurs du projet*.



arbres

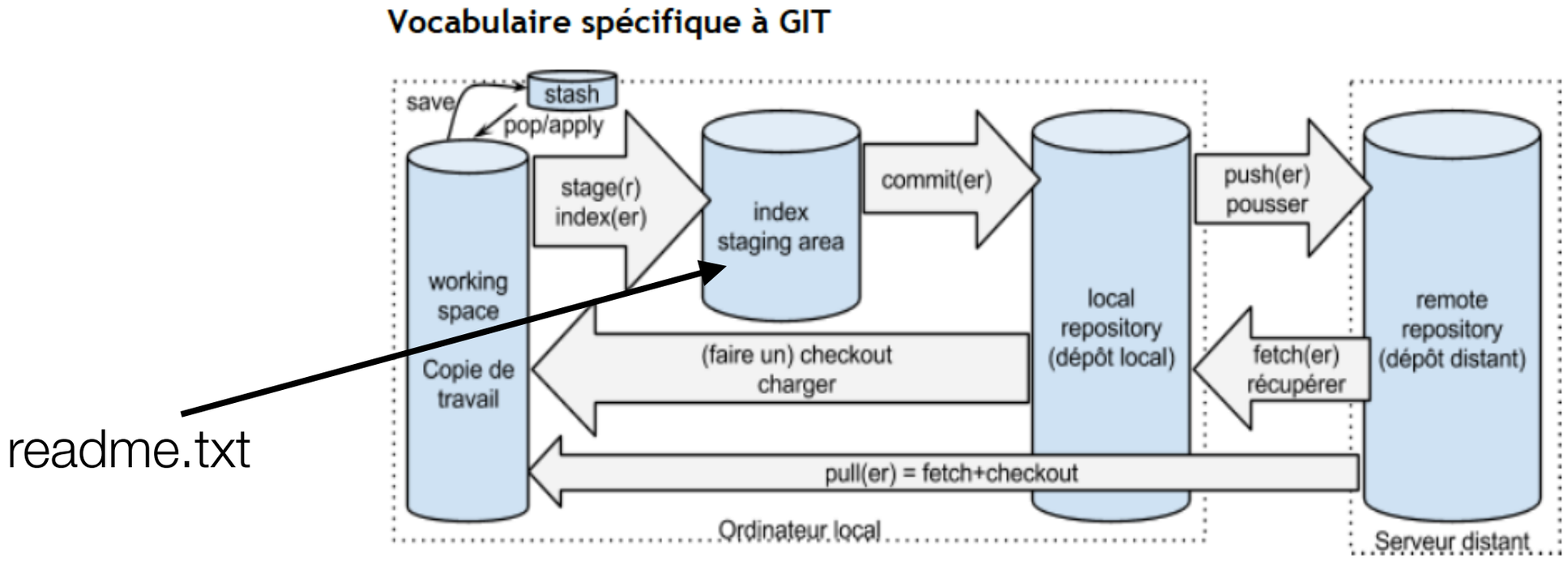
votre dépôt local est composé de trois "arbres" gérés par git. le premier est votre `espace de travail` qui contient réellement vos fichiers. le second est un `Index` qui joue un rôle d'espace de transit pour vos fichiers et enfin `HEAD` qui pointe vers la dernière validation que vous ayez fait.



Ajouter un fichier à l'Index

- Pour l'instant votre fichier readme.txt n'est pas connu de Git car il ne figure pas dans l'index. Il est dans le working space.
- git add fichier: ajoute le fichier à l'index

```
pc6:TestRepGit sebastien$ git add readme.txt
```

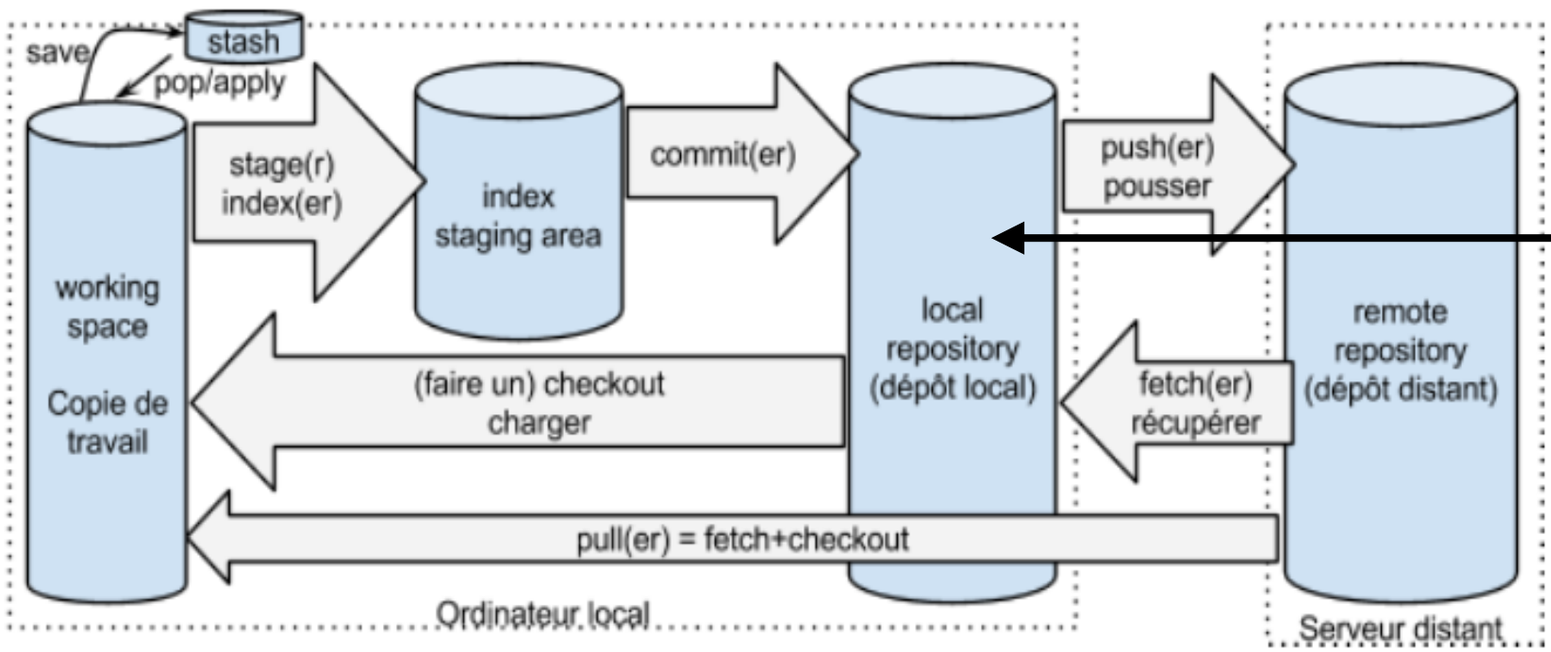


HEAD

- Votre fichier est présent dans l'index maintenant mais n'est pas encore dans le HEAD. Il n'a pas été comité !
- Pour le valider :

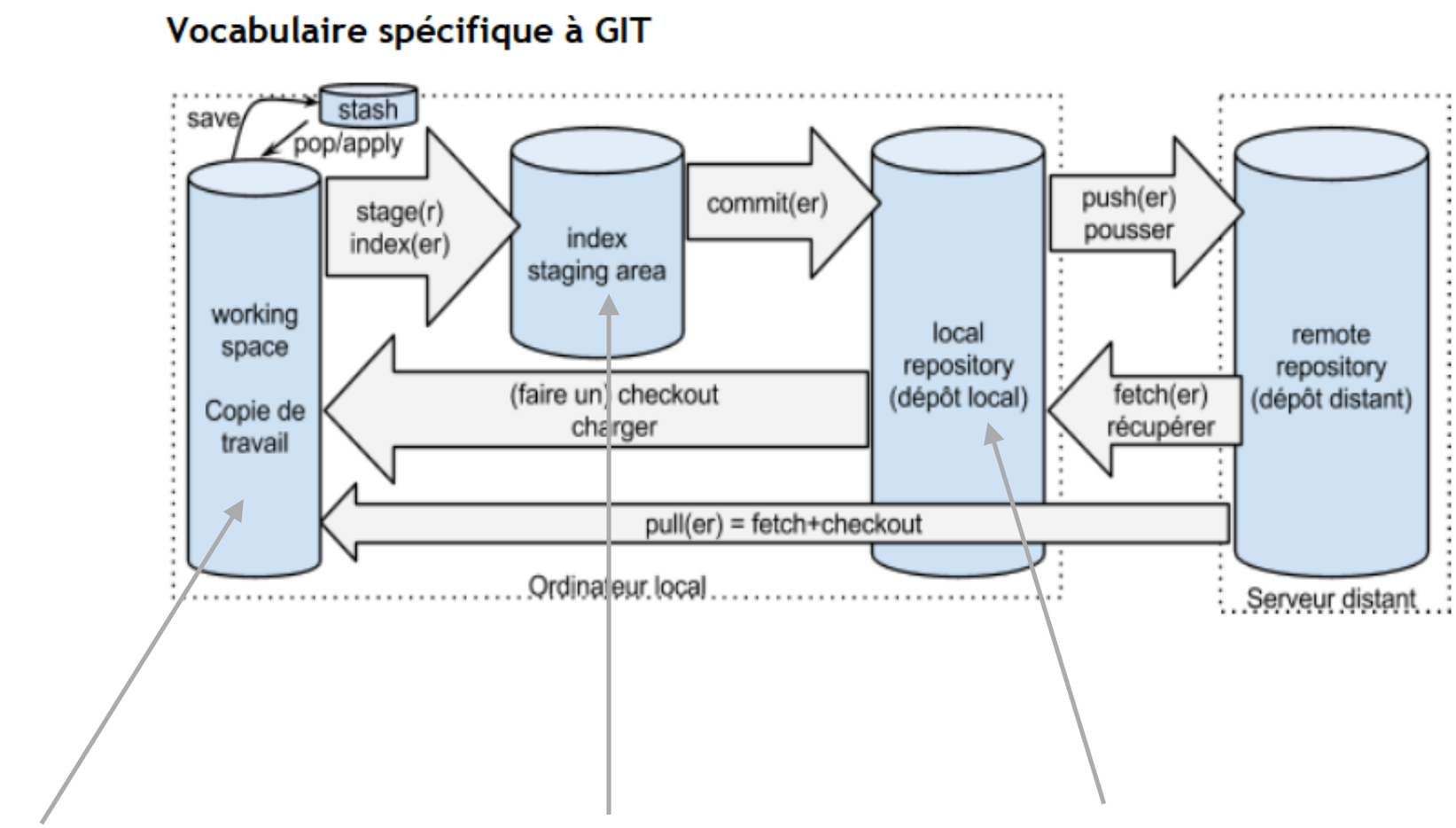
```
pc6:TestRepGit sebastien$ ls
readme.txt
pc6:TestRepGit sebastien$ git commit -m "Premier commit"
[master (root-commit) 79663ee] Premier commit
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 readme.txt
pc6:TestRepGit sebastien$
```

Vocabulaire spécifique à GIT



readme.txt

Lister les fichiers



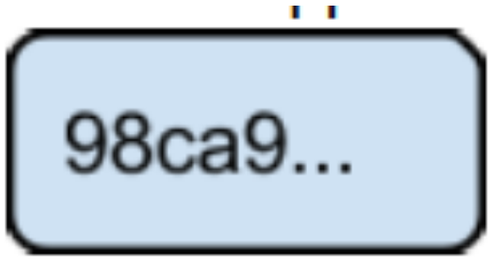
ls **git ls-files —stage** **git ls-tree -r HEAD**

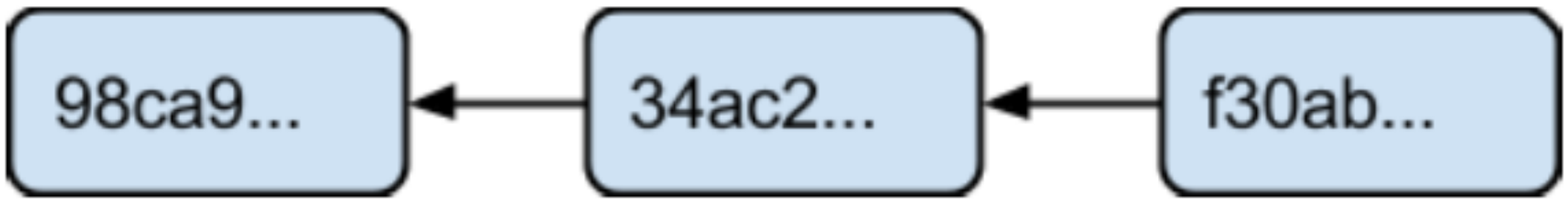
Modifier le fichier

- Je vais modifier le contenu du fichier.

```
pc6:TestRepGit sebastien$ cat readme.txt
Bonjour amis terriens !
pc6:TestRepGit sebastien$ git status
# On branch master
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#       modified:   readme.txt
#
no changes added to commit (use "git add" and/or "git commit -a")
pc6:TestRepGit sebastien$ git diff
diff --git a/readme.txt b/readme.txt
index e69de29..fdb33ae 100644
--- a/readme.txt
+++ b/readme.txt
@@ -0,0 +1 @@
+Bonjour amis terriens !
pc6:TestRepGit sebastien$
```

Git : Commit

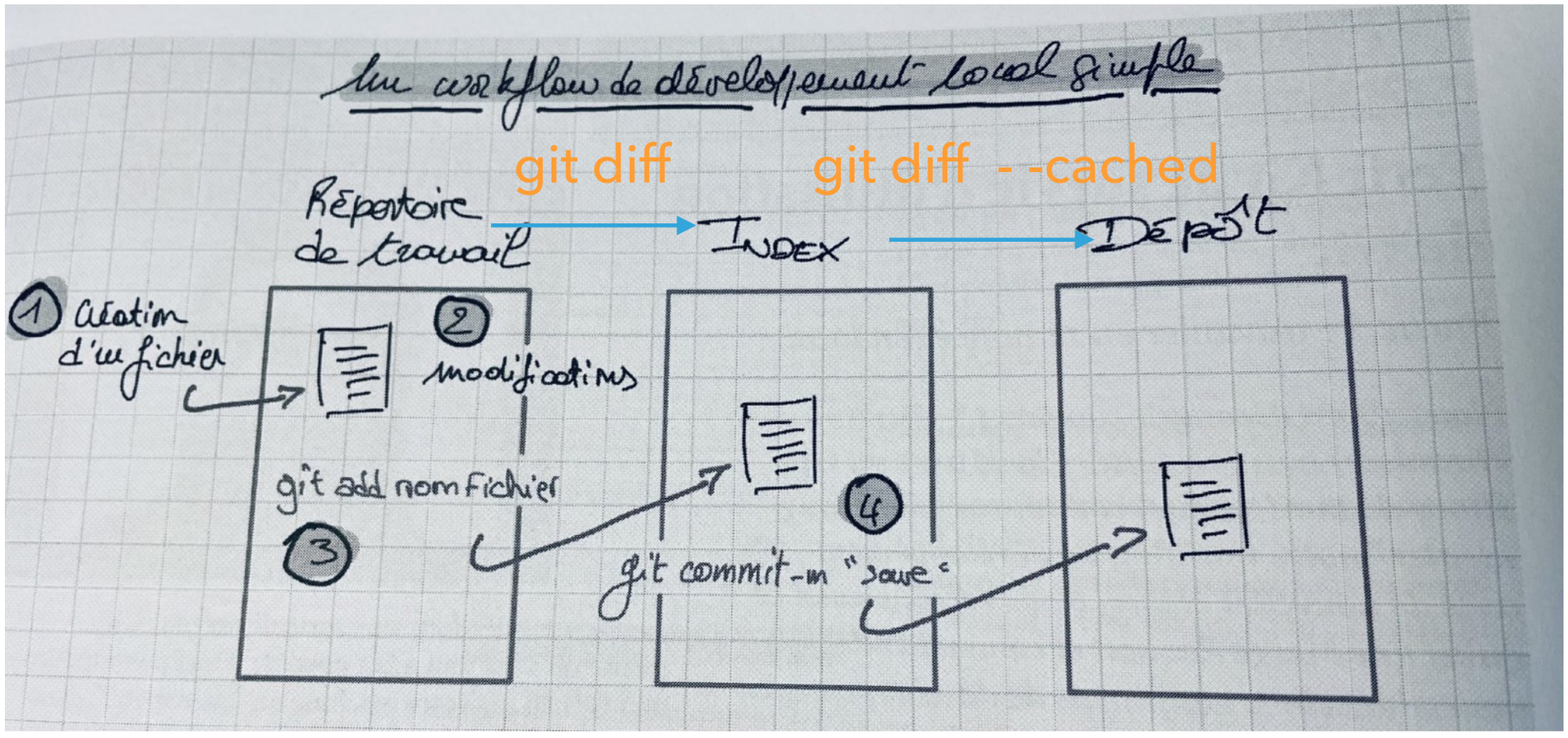




Un dépôt Git est finalement un graphe de commits

LE FLUX DE TRAVAIL

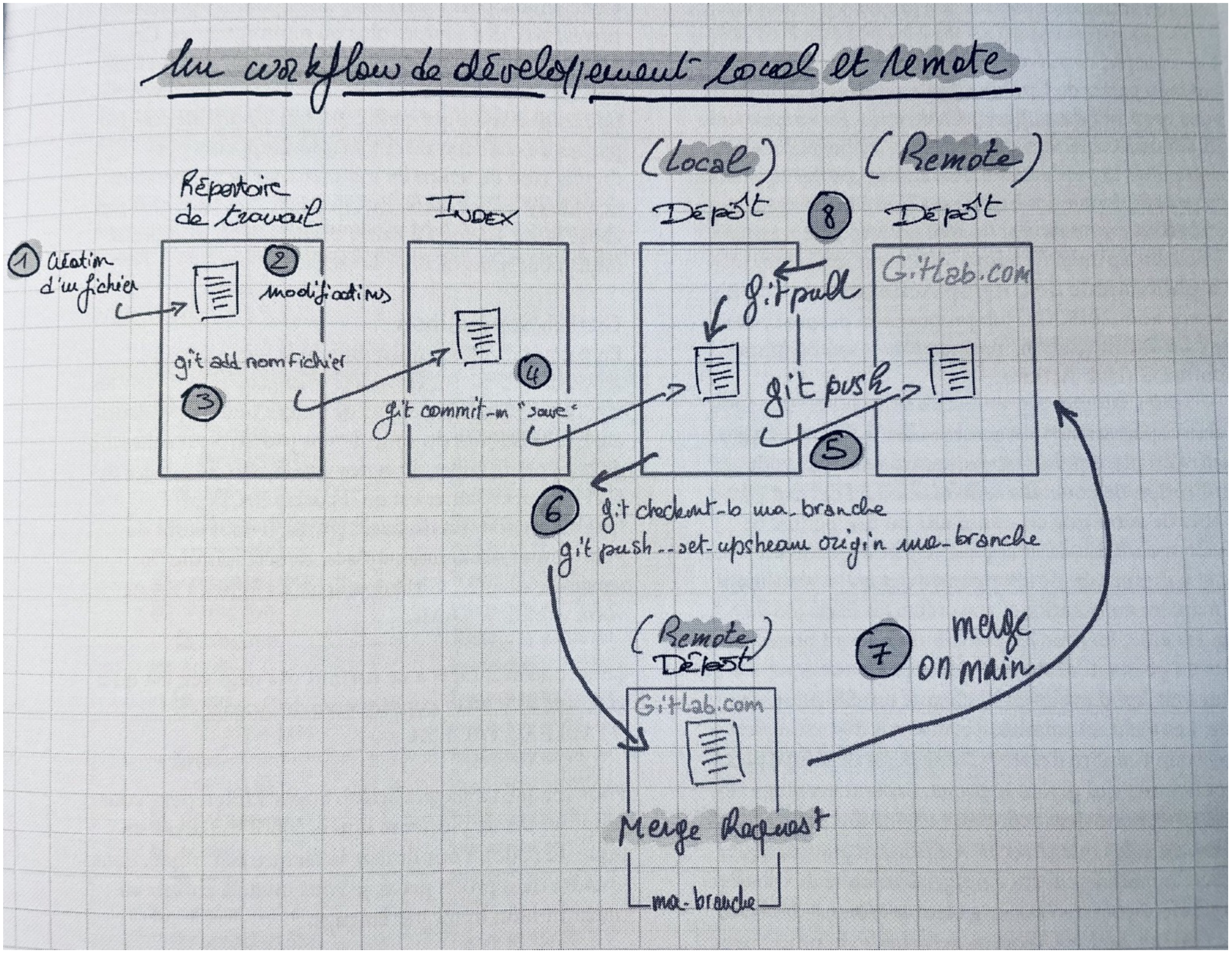
EN LOCAL



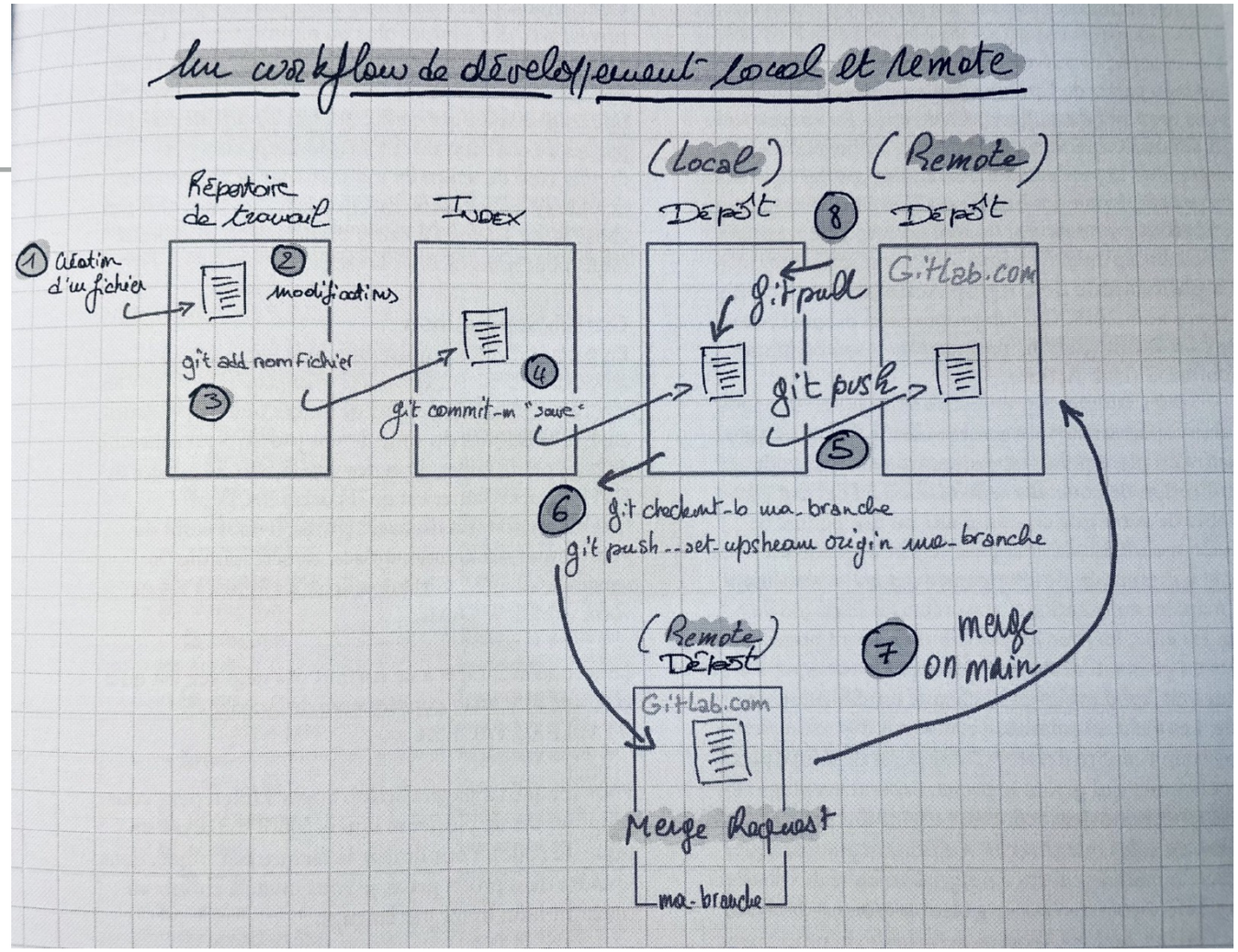
git status : a faire régulièrement pour savoir où vous en êtes.

Étape 1 : Création ou modification d'un fichier dans le répertoire de travail.
Étape 2 : Ajout des modifications au staging area (ou index) avec git add nom_fichier.
Étape 3 : Enregistrement des modifications dans le dépôt local avec git commit -m "message".
Étape 4 : Les modifications sont désormais sauvegardées dans l'historique du dépôt Git.

EN LOCAL


Un workflow de développement local et remote
(Local)
(Remote)
Répertoire de travail
INDEX
Dépôt
Dépôt
1 Création d'un fichier
2 modifications
git add nomFichier
3
4
git commit -m "save"
8
git pull
git push
5
GitHub.com
6 git checkout -b ma_branche
git push --set-upstream origin ma_branche
7 merge on main
(Remote)
Dépôt
GitLab.com
Merge Request
ma-branche

EN LOCAL

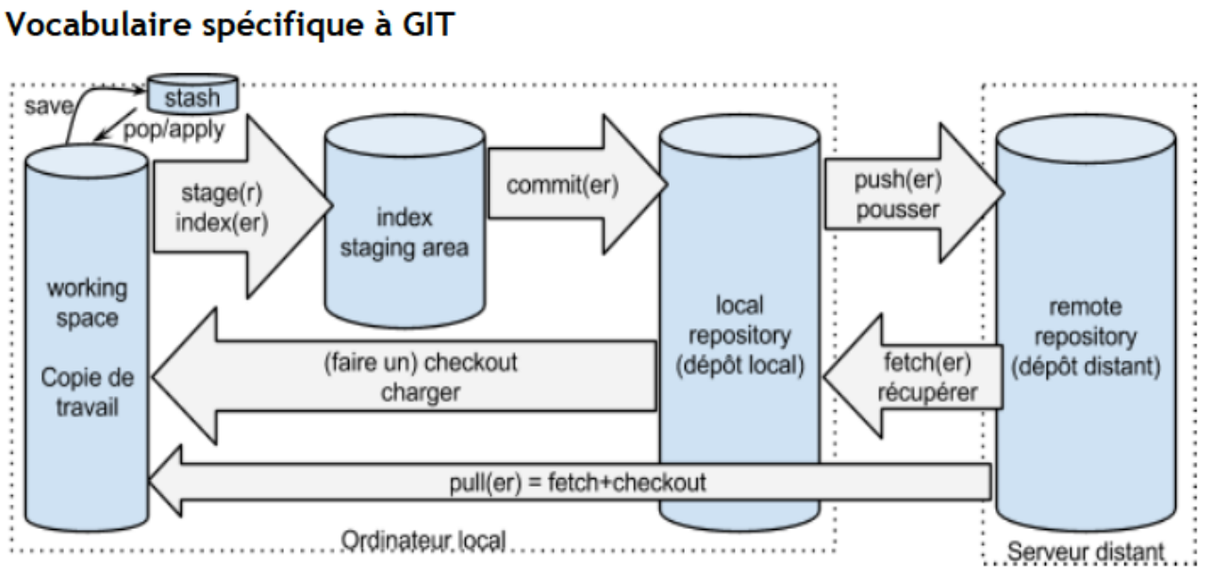


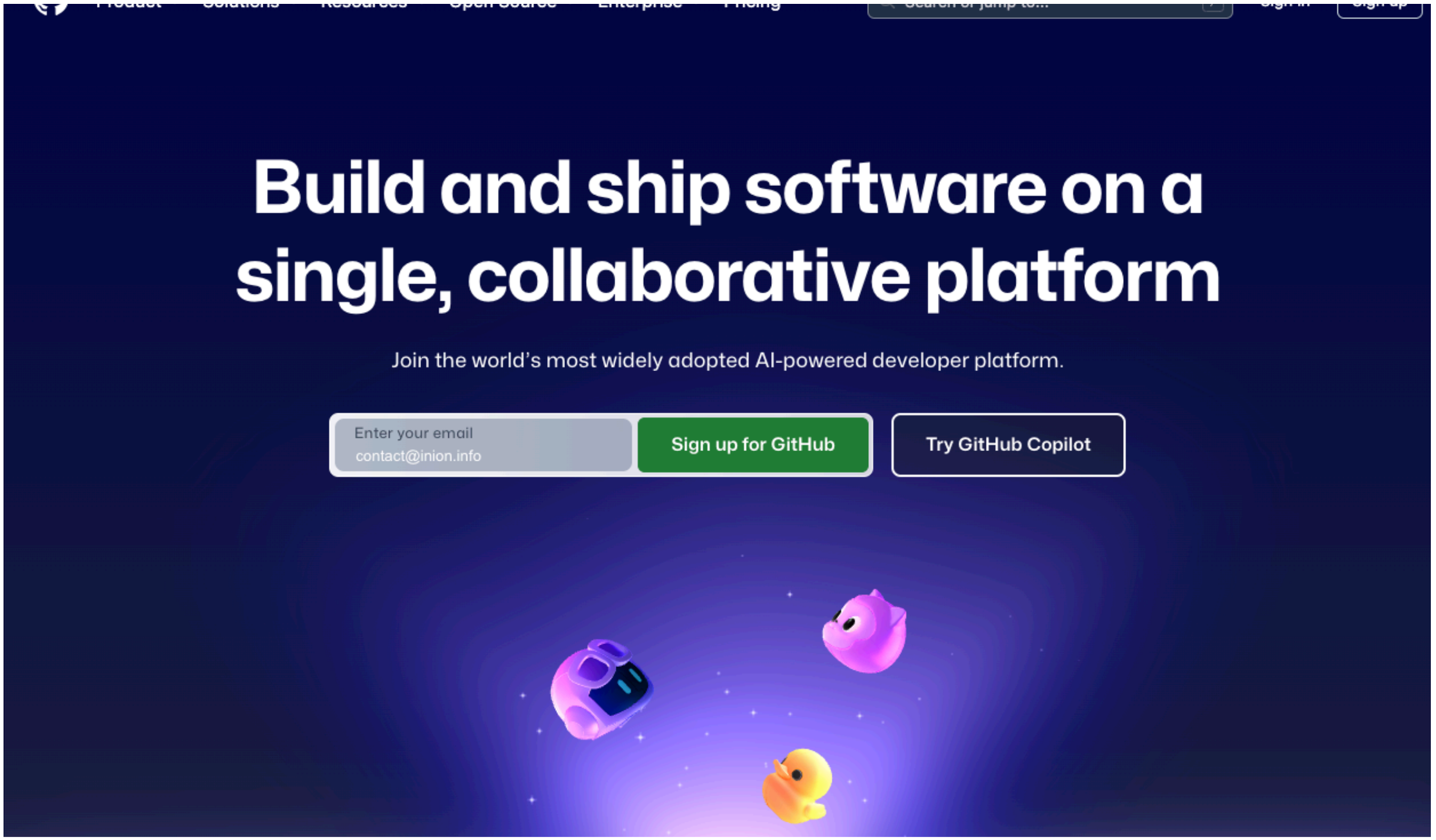
Étape 1 : Création ou modification d'un fichier dans le répertoire de travail.
Étape 2 : Ajout des modifications au staging area (ou index) avec la commande git add nom_fichier.
Étape 3 : Enregistrement des modifications dans le dépôt local avec git commit -m "message".
Étape 4 : Création d'une branche locale avec git checkout -b ma_branche et association au dépôt distant av
git push --set-upstream origin ma_branche.
Étape 5 : Envoi des modifications sur le dépôt distant (ex. GitLab ou GitHub) avec git push.
Étape 6 : Création d'une merge request dans l'interface du dépôt distant pour proposer la fusion des changements.
Étape 7 : Validation et fusion de la branche sur la branche principale (ex. main).

Création d’un dépôt distant avec Github

En effet pour l’instant le fichier est dans le HEAD de votre dépôt local.

Si vous voulez le partager avec d’autres développeurs il faut le transférer sur un dépôt distant.





GitHub

Création d'un compte

testStudi2025 Public

Pin | Unwatch 1 | Fork 0 | Star 0

Set up GitHub Copilot

Use GitHub's AI pair programmer to autocomplete suggestions as you code.

Get started with GitHub Copilot

Add collaborators to this repository

Search for people using their GitHub username or email address.

Invite collaborators

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

Set up in Desktop or HTTPS SSH `https://github.com/SebInfo/testStudi2025.git`

Get started by creating a new file or uploading an existing file. We recommend every repository include a README, LICENSE, and .gitignore.

...or create a new repository on the command line

```
echo "# testStudi2025" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/SebInfo/testStudi2025.git
git push -u origin main
```

...or push an existing repository from the command line

```
git remote add origin https://github.com/SebInfo/testStudi2025.git
git branch -M main
git push -u origin main
```

ProTip! Use the URL for this page when adding GitHub as a remote.

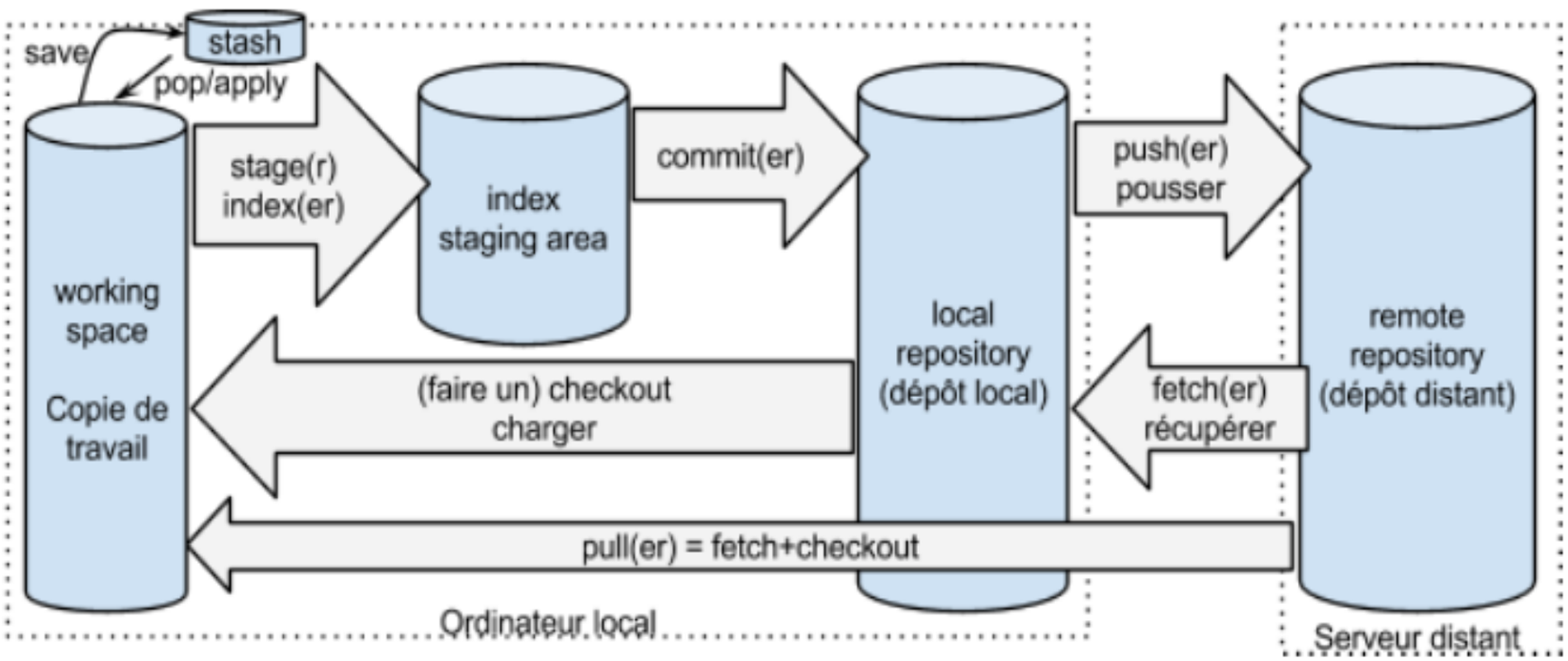
GitHub | Création d'un dépôt

On envoie le fichier avec le push

git remote add monDepot https://github.com/SebInfo/testStudi2025.git

```
(base) MacBook-2:testsGitStudi sebastien$ git push -u monDepot main
Énumération des objets: 8, fait.
Décompte des objets: 100% (8/8), fait.
Compression par delta en utilisant jusqu'à 12 fils d'exécution
Compression des objets: 100% (6/6), fait.
Écriture des objets: 100% (8/8), 612 octets | 612.00 Kio/s, fait.
Total 8 (delta 2), réutilisés 0 (delta 0), réutilisés du pack 0
remote: Resolving deltas: 100% (2/2), done.
To https://github.com/SebInfo/testStudi2025.git
 * [new branch]      main -> main
la branche 'main' est paramétrée pour suivre 'monDepot/main'.
(base) MacBook-2:testsGitStudi sebastien$
```

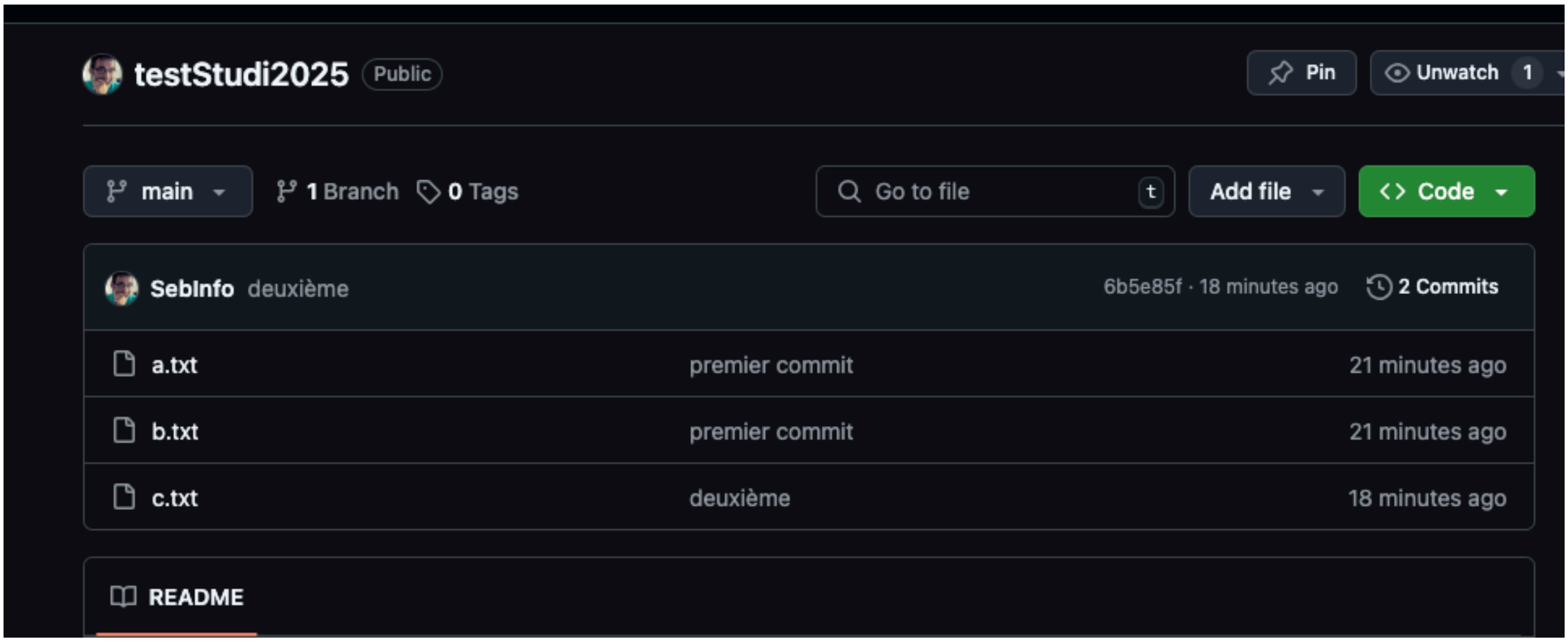
Vocabulaire spécifique à GIT



On envoie le fichier avec le push

---

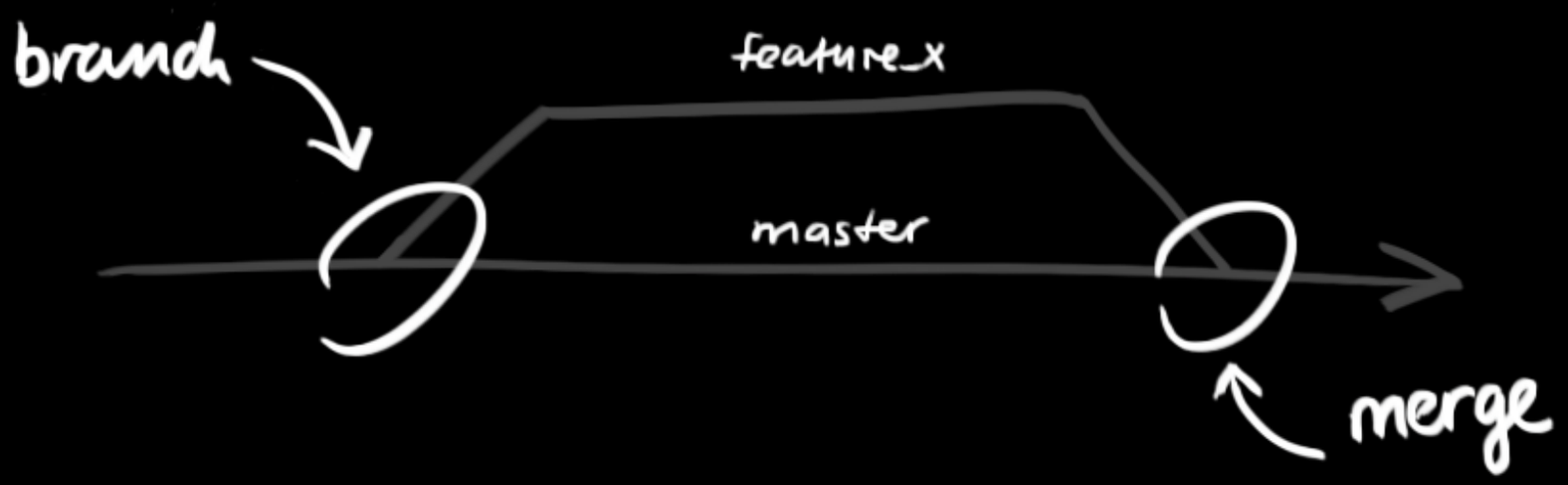
git remote add monDepot https://github.com/SebInfo/testStudi2025.git



Branches

branches

Les branches sont utilisées pour développer des fonctionnalités isolées des autres. La branche *master* est la branche par défaut quand vous créez un dépôt. Utilisez les autres branches pour le développement et fusionnez ensuite à la branche principale quand vous avez fini.



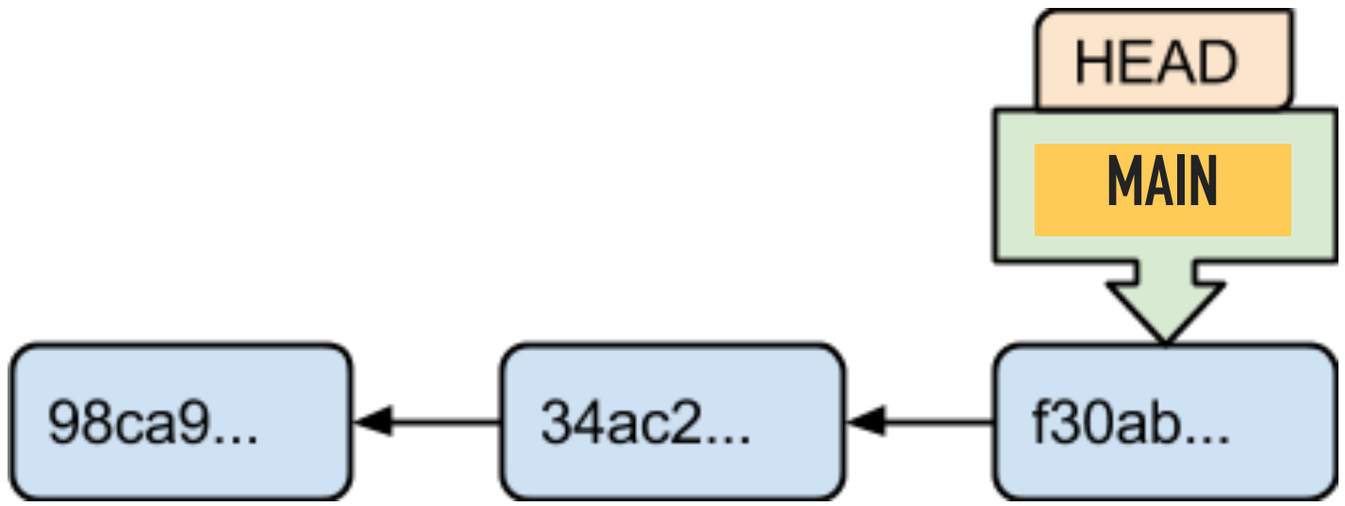
Git et les branches

Première branche : **main**

Une branche pointe sur un commit

Git et les branches

Un pointeur spécial nommé HEAD pointe vers la branche courante de travail !

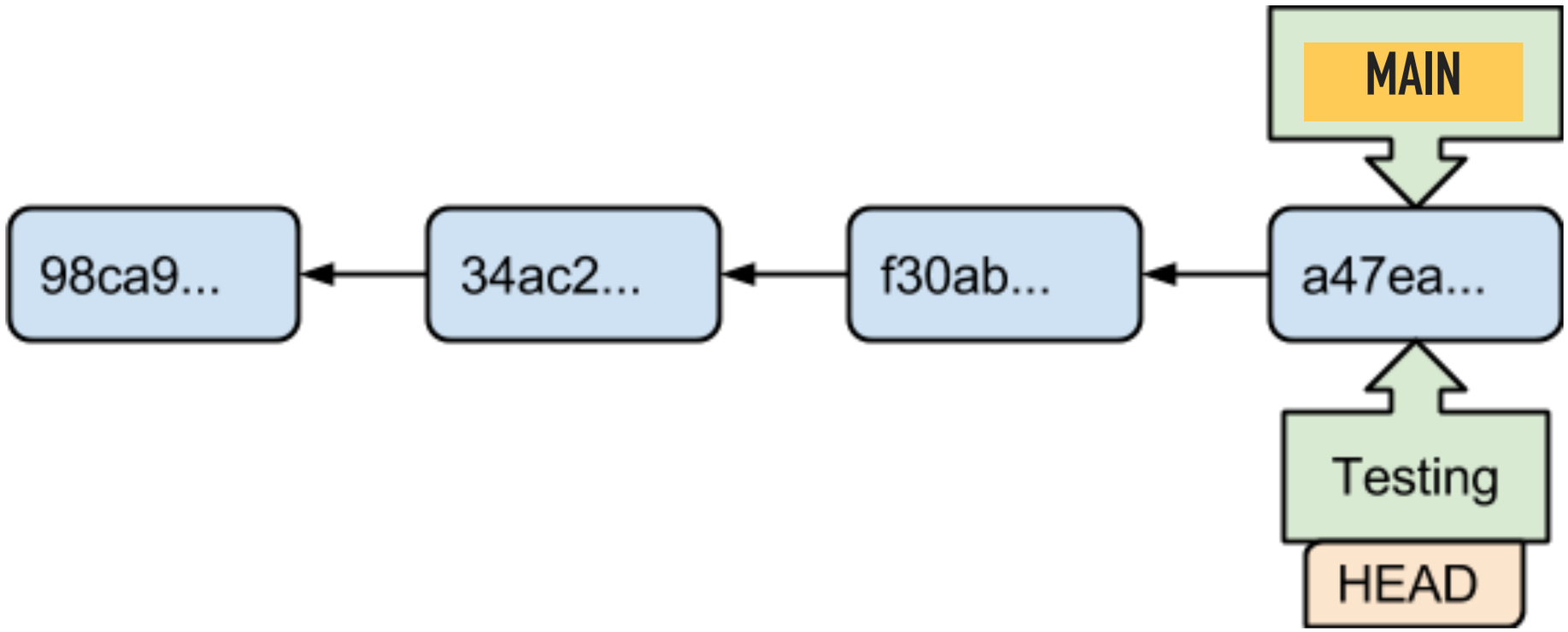


Git et les branches

Création d'une nouvelle branche

$git branch Testing

$git checkout Testing



Git et les bonnes pratiques

- Commiter des modifications en relation les unes des autres
  -> deux aspects = deux commit
- Tester le code avant de comiter
- Utiliser les branches
- Commiter souvent
- Ecrire de bonne description dans les commit
- Ne pas commiter un travail à moitié fait
- Git n’est pas un système de Sauvegarde/Backup

MERCI POUR VOTRE ATTENTION