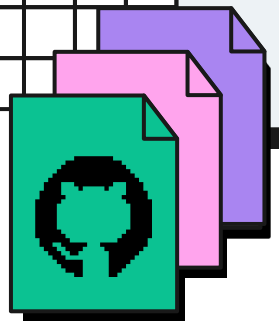
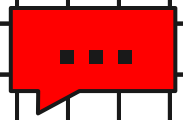
BONNES PRATIQUES DE PROGRAMMATION



Collaborer, documenter, & tester son code

Du code efficace, robuste & élégant



tinyurl.com/GIT-M2TNAH

Comment suivre ce cours ?

Mode écoute

Terminal fermé,
Oreilles attentives

C'est la fin de la journée, on se laisser bercer

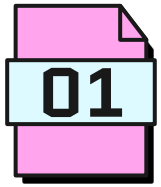
Mode lecture

Stylo posé,
Markdown qui défile

On se détend, tout ce qui doit être noté est déjà dans la transcription du cours

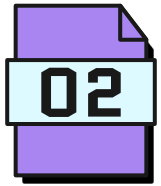
Mode action

DÉROULÉ DE LA SÉANCE



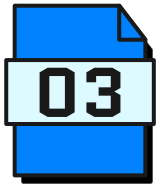
Git

Comment & pourquoi utiliser le *versioning*



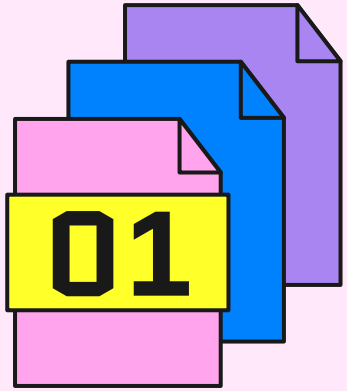
IDE

Prise en main d'un logiciel de développement



GitHub

Hébergement & collaboration



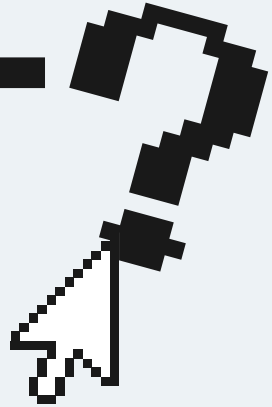
GIT

La magie de la gestion de version

GIT ?

git est un outil de gestion de versions décentralisé créé par Linus Torvald (le même que Linux) en 2005 sous licence *open-source* GPL

À quoi ça sert GIT ?



01 Sauvegarder des versions

Fini les fichiers aux nommages pourris…

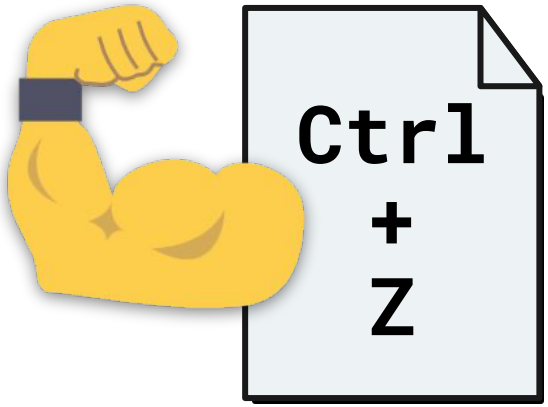
01 Sauvegarder des versions

Git prend des
instantanés de vos dossiers

repository = dossier versionné

commit = sauvegarde du *repository*

02 Conserver un historique



Un Ctrl+Z qui peut revenir 2 ans en arrière,
savoir qui a fait la modif et pourquoi

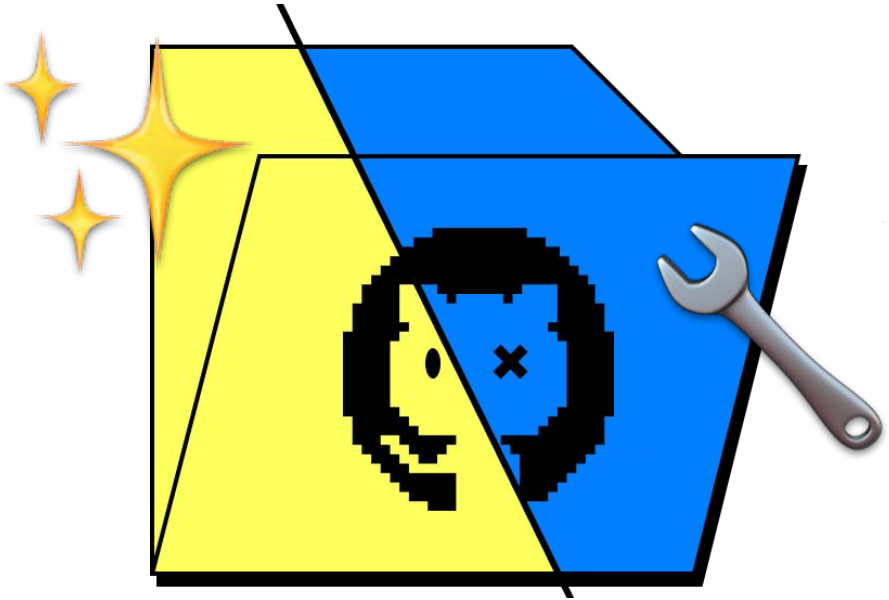
02 Conserver un historique

Chaque ***commit*** conserve en mémoire :

- sa date de création et son auteur
- les lignes modifiées (diff)
- un message expliquant le but des modifs

Permet de suivre précisément
l'évolution d'un *repository* et de
revenir à un état antérieur

03 Créer des états parallèles



La version propre des fichiers cohabite avec le *work-in-progress*

03 Créer des états parallèles

Les ***branches*** permettent de faire coexister différentes versions des fichiers du *repository*

Elle permettent :

- Expérimenter sans casser l'existant
- Faire cohabiter plusieurs états d'avancement

La branche par défaut s’appelle
main ou ***master***

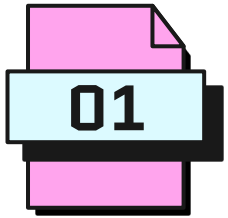
04 Collaborer sur un projet

04 Collaborer sur un projet

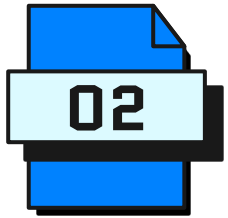
Git est un système **décentralisé** :

- chacun travaille sur sa machine et sa branche
- les historiques de modifications sont réconciliés automatiquement
- les ***branches*** peuvent être fusionnées (***merge*** ou ***rebase***)

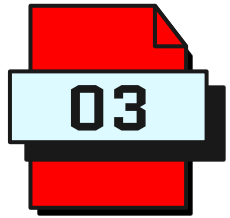
À quoi ça sert le *versioning* ?



SAUVEGARDER DES VERSIONS



CONSERVER UN HISTORIQUE



CRÉER DES ÉTATS PARALLÈLES

COLLABORER SUR UN PROJET

Exercice 1

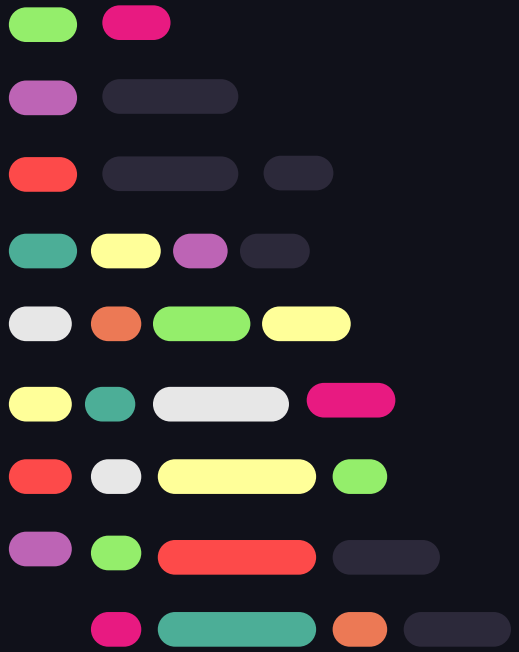
01 { ..

Initialiser git

< le moment où on ouvre son terminal >

} ..

Installation de git

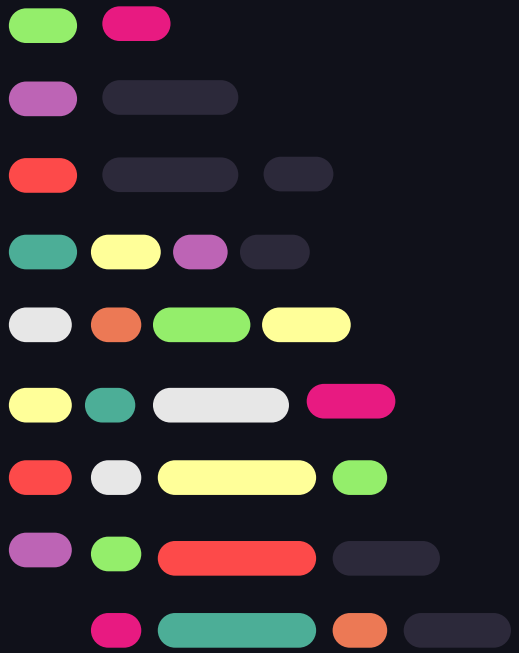


```
# Ouvrir son terminal (Ctrl+Alt+T sur Ubuntu)

# Mise à jour des dépendances système
sudo apt-get update

# Installation de git
sudo apt install git
```

Configuration de l'utilisateur

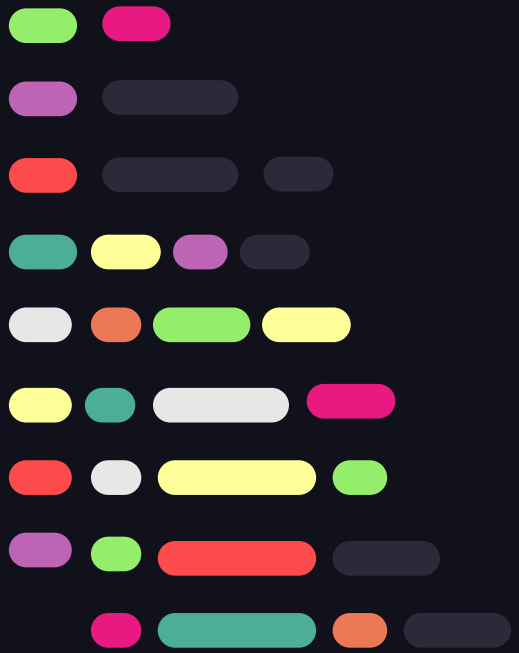
```
# Définition de l'utilisateur (⚠️ même que sur GitHub)
git config --global user.name "<github-user>"
git config --global user.email "<github-email>"

# Par exemple (⚠️ retirer les chevrons)
git config --global user.name "Segolene-Albouy"
git config --global user.email "segolene.abouy@gmail.com"

# Nous verrons plus tard comment lier son compte local
# avec son compte GitHub

# ⚠️ À REFAIRE À POUR CHAQUE NOUVEL ORDINATEUR
```

Configuration de git



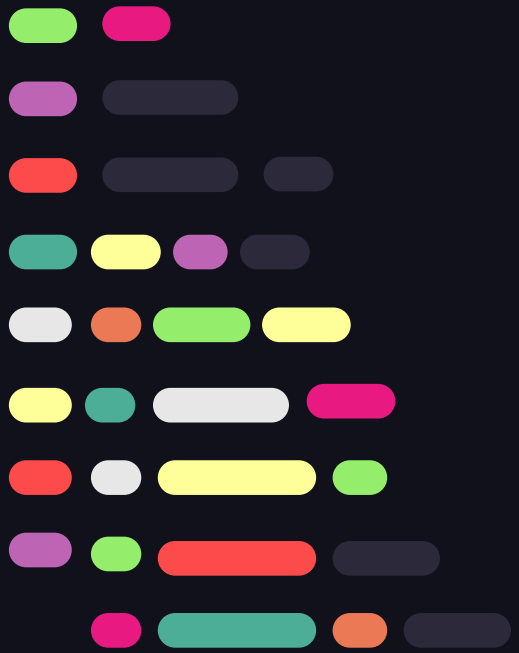
```
# Définition de l'éditeur de par défaut
git config --global core.editor "nano"

# Définition du nom de la branche de par défaut
git config --global init.defaultBranch main

# Configuration de la coloration par défaut
git config --global color.ui auto

# Choix de merge comme mode de fusion lors d'un pull
git config --global pull.rebase false

# Afficher vos configurations
git config --list
```

Modifier le .bashrc



```
sudo apt install curl

# ajout de commandes et alias à sa config bash
curl -s
https://raw.githubusercontent.com/Segolene-Albouy/
GIT-M2TNAH/refs/heads/main/01-Git_basics/templates
/.bashrc >> ~/.bashrc

# recharger la config pour voir les effets
source ~/.bashrc
```

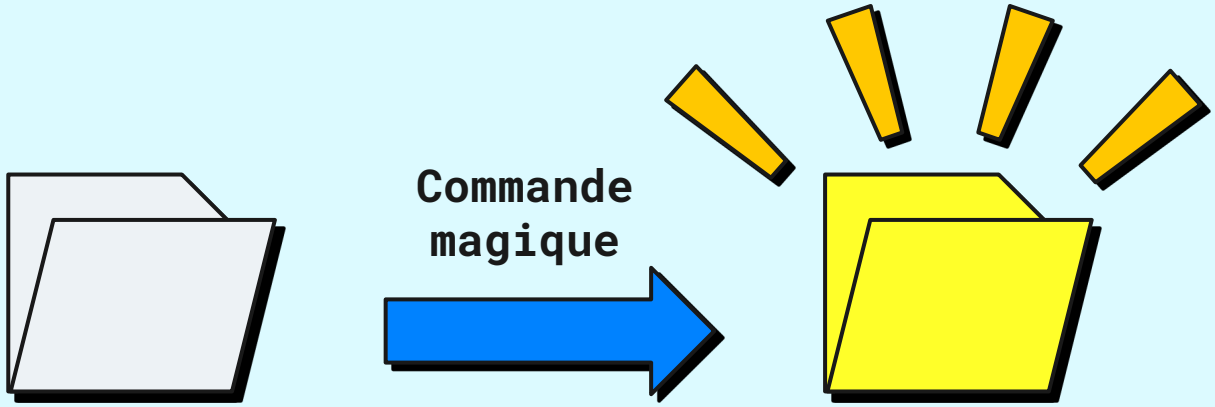
Maintenant vous pouvez 🪄



```
# Voir immédiatement la branche courante
user:/path/to/repo [branch] $

# Changer votre .bashrc facilement avec la commande
# Notamment pour changer les couleurs 🎨
bashrc

# Visualiser la diff entre la branche locale/remote
gdiff

# Afficher les logs bien formatés
glog
```


BYE!

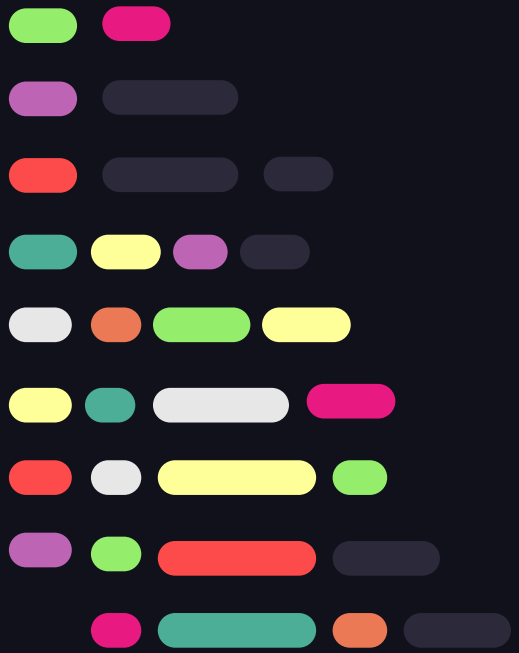
INITIALISATION D'UN *REPOSITORY*



Création normale
d'un dossier

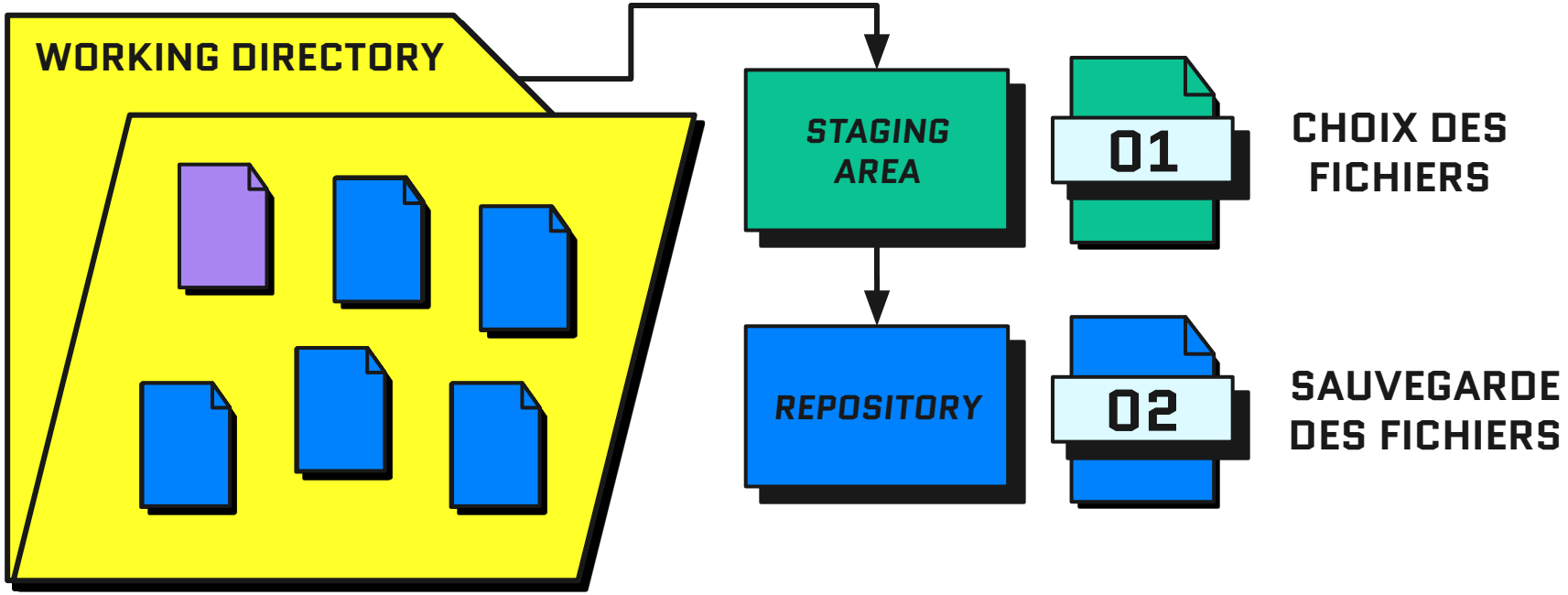
Transformation en
un ***repository***

Initialisation d’un *repository*



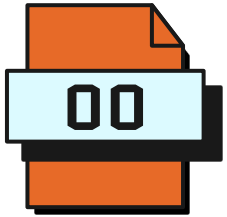
```
# Se déplacer dans le dossier où créer le repo
cd <directory-name>

# Initialiser le repository
git init

# Afficher le contenu du dossier
ls -al

# → un dossier .git/ caché a été créé
```

Processus de commit



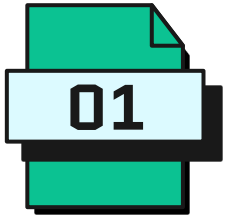
Sauvegarder un état du *repo*

La création d'un **commit**, se fait en plusieurs étapes :



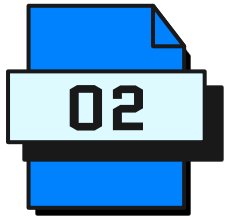
MODIFICATION DES FICHIERS

Ajout/suppression de fichiers, changement du contenu, renommage, etc.



CHOIX DES FICHIERS

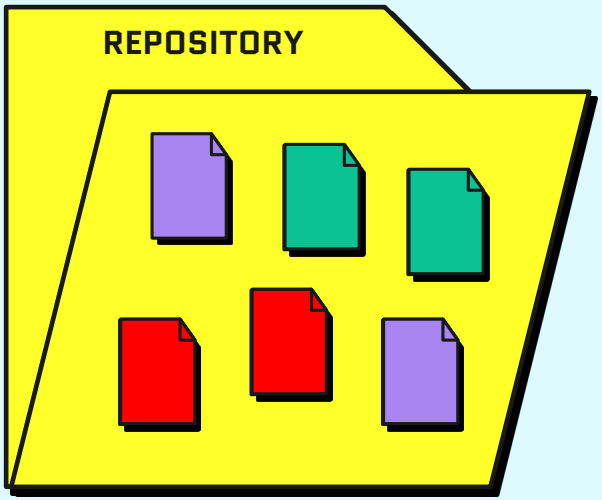
Sélection des fichiers à inclure dans le commit : ajout à la *staging area*



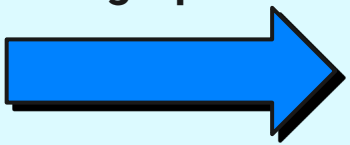
CRÉATION DE LA SAUVEGARDE

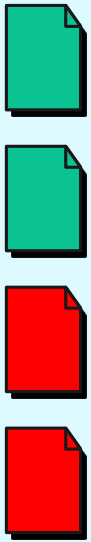
Enregistrement effectif de l'état des fichiers dans le commit

CONNAÎTRE L'ÉTAT DU *REPOSITORY*



Commande magique





Fichiers modifiés **dans** la *staging area*

Fichiers modifiés **hors** de la *staging area*

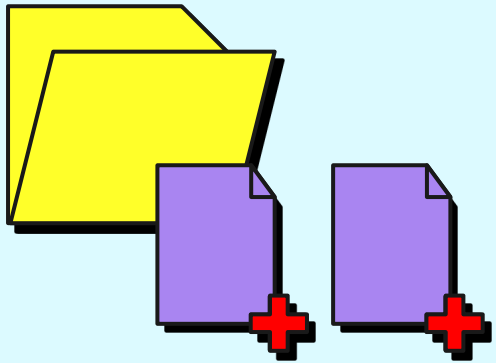
Lister les fichiers modifiés

```
$ git status
Sur la branche main

Modifications qui seront validées :
  (utilisez "git rm --cached <fichier>..." pour désindexer)
    nouveau fichier : fichier1.txt
    nouveau fichier : fichier2.txt

Fichiers non suivis:
  (utilisez "git add <fichier>..." pour inclure dans ce qui sera validé)
    fichier3.txt
    fichier4.txt
```

Fichiers **indexés** ["*added*"]

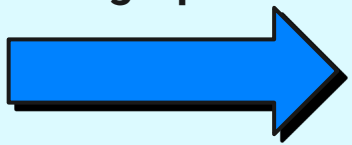
Fichiers non-indexés

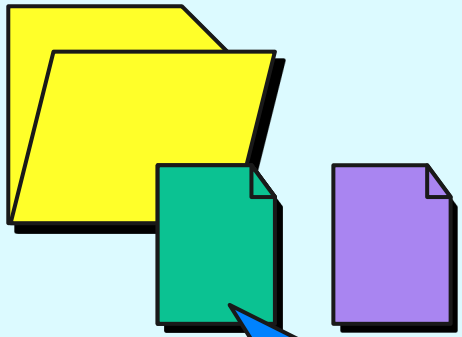
AJOUT DE FICHIERS À LA *STAGING AREA*



Modification de fichier(s) dans le repository

Commande magique



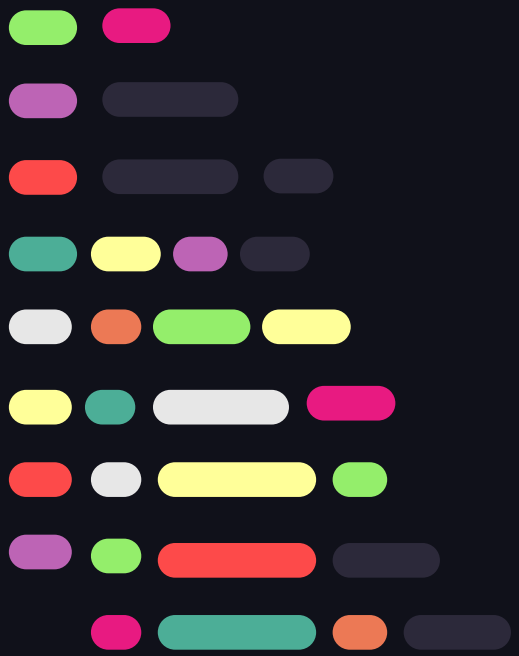


Ajout dans

git fonctionne mieux avec des fichiers texte brut, on évite `.docx`, `.odt`, etc.

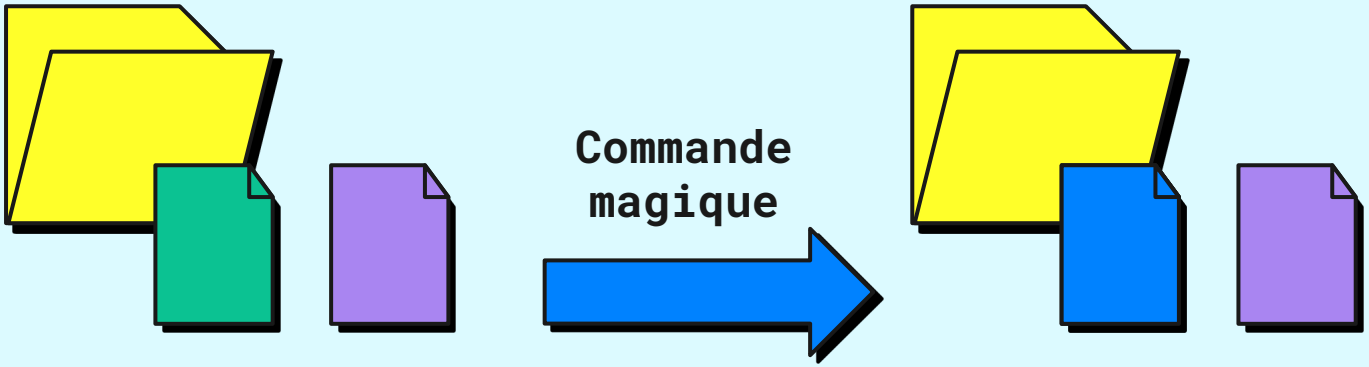
[On parle aussi d'**indexation** des fichiers]

Ajout à la *staging area*



```
# lister tous les fichiers modifiés
# depuis le dernier commit
git status

# ajout d'un fichier précis
git add <path/to/filename>
# ajout de plusieurs fichiers
git add <path/to/filename> <path/to/filename>
# ajout de tous les fichiers d'un dossier
git add <path/to/directory>

# ajout de tous les fichiers du repo
git add -A
```

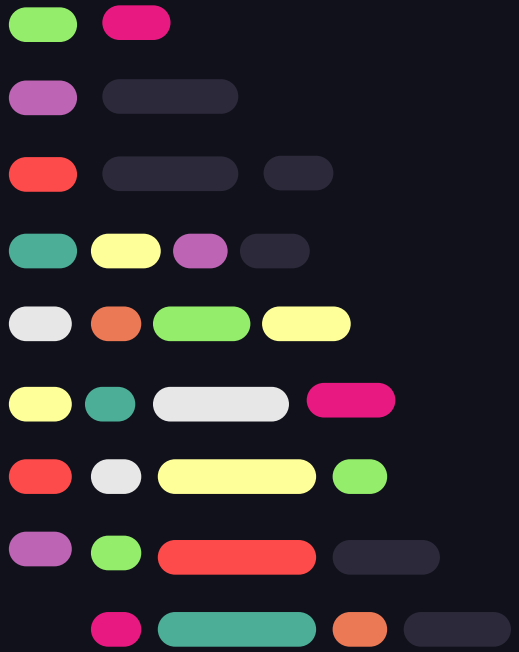
CRÉATION D'UN COMMIT



Certains fichiers ont été **indexés** (“**addés**”)

Seulement les fichiers “**addés**” sont “**commités**”

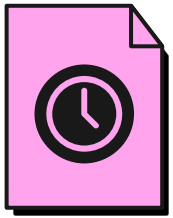
Création d'un *commit*



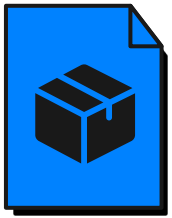
```
# commit des fichiers de la staging area
git commit

# commit avec message
git commit -m "<commit-message>"
```

C'est quoi un bon commit ?

Réussir ses commits



LE BON MOMENT



LE BON CONTENU

LE BON MESSAGE

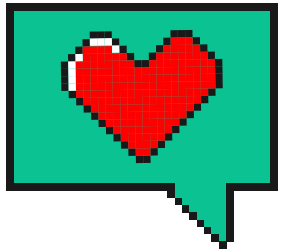
Le message de commit parfait

[type](scope)description

L'idée c'est de pouvoir identifier rapidement l'objet du commit :

- **type**: la catégorie de modification (*fix*, *feat*, *docs*, *refacto*, *test*, etc.)
- **scope**: quelle partie du projet est concerné (*search*, *database*, *login*, etc.)
- **description**: courte explication de ce qui a été réalisé (max 1 phrase)

https://gitmoji.dev

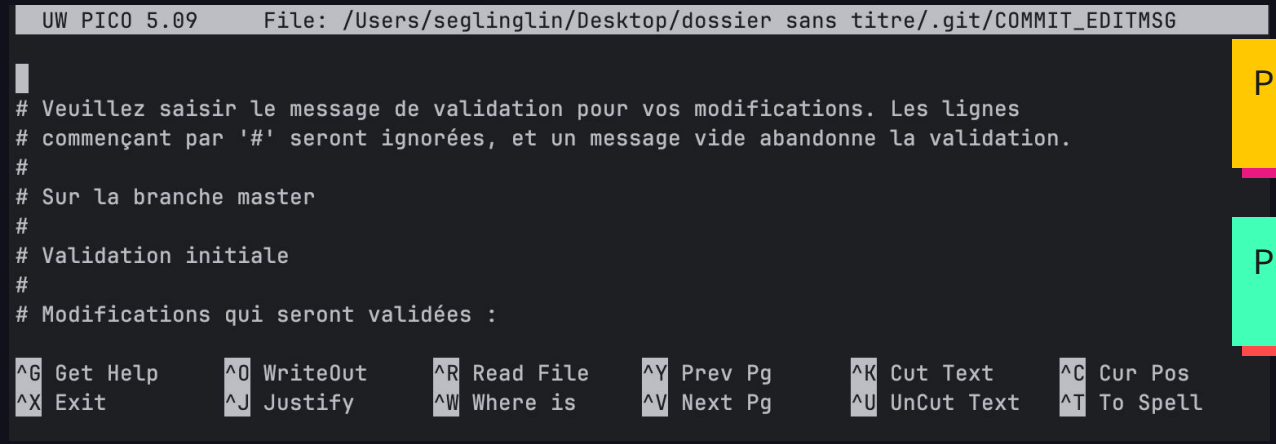


J’ai oublié le *-m* 😱 !!

 # Lorsque vous ne fournissez pas de message directement après ***git commit***
 # vous pénétrez dans l’éditeur de texte du terminal

Soit sur nano

Souvenez-vous, c’est celui qu’on a configuré tout à l’heure



Pour sauver
Ctrl + S

Pour sortir
Ctrl + X

J’ai oublié le *-m* 😱 !!

 # Lorsque vous ne fournissez pas de message directement après ***git commit***
 # vous pénétrez dans l’éditeur de texte du terminal

Soit sur vim

Edit mode **i**
Command mode **Esc**

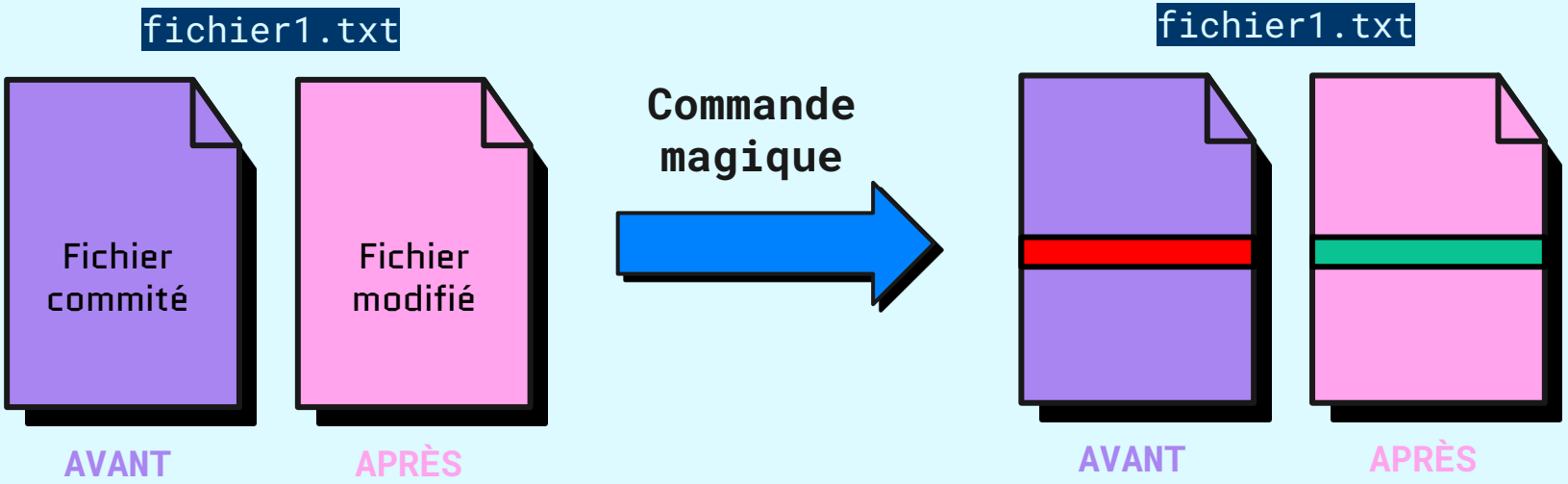
C’est normal de paniquer la première fois

```
# Veuillez saisir le message de validation pour vos modifications. Les lignes
# commençant par '#' seront ignorées, et un message vide abandonne la validation.
#
# Sur la branche vectorization
# Votre branche est à jour avec 'origin/vectorization'.
#
# Modifications qui seront validées :
#       modifié :         app/vectorization/routes.py
#
~
~
~
~
```

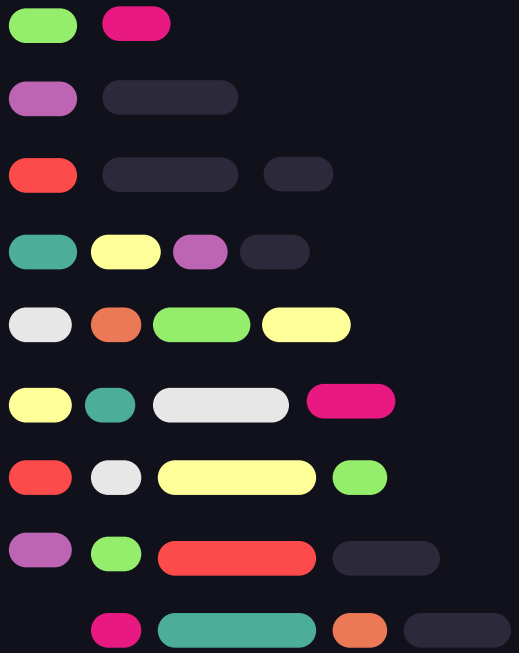
Pour sauver
:x!

Pour sortir
:q!

VOIR LES MODIFICATION DEPUIS LE DERNIER COMMIT



Visualiser les modifications



```
# voir le contenu des modifications
# depuis le dernier commit
git diff
# dans un fichier particulier
git diff <path/to/file>

# avec une autre branche
git diff <branch-name>
# entre deux branches
git diff <branch-name> <branch-name>
```


Visualiser le contenu modifié
$ git diff
Développement des fichiers comparés
diff --git a/fichier1.txt b/fichier1.txt
hash des versions
index 04af84e..c006980 100644
Fichier avant modification
Fichier après modification
--- a/fichier1.txt
+++ b/fichier1.txt
Ligne affectée par le changement
@@ -1 +1 @@
Ligne avant modification
Ligne après modification
-Bonjour, vous allez-bien ?
+Salut, tu vas bien ?

Exercice 2

02 { ..

Créer un *repository*
Ajouter des fichiers
Effectuer 3 *commits*

} ..

Dossier

Créer un dossier et l'ouvrir dans un terminal

Init

Initialiser un *repository* dans le dossier

Fichier

Créer plusieurs fichiers avec du contenu

Status

Lister les fichiers modifiés avec git status

Add

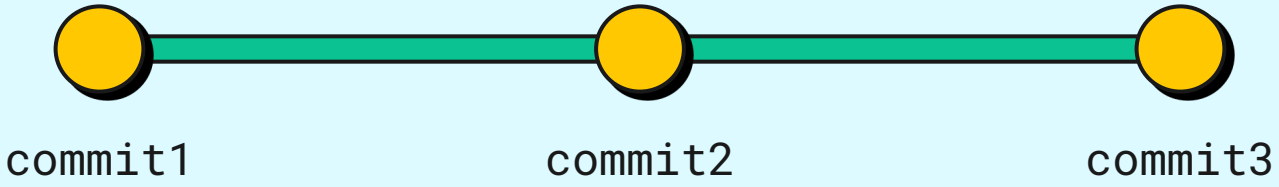
Ajouter un/des fichiers à la *staging area*

Commit

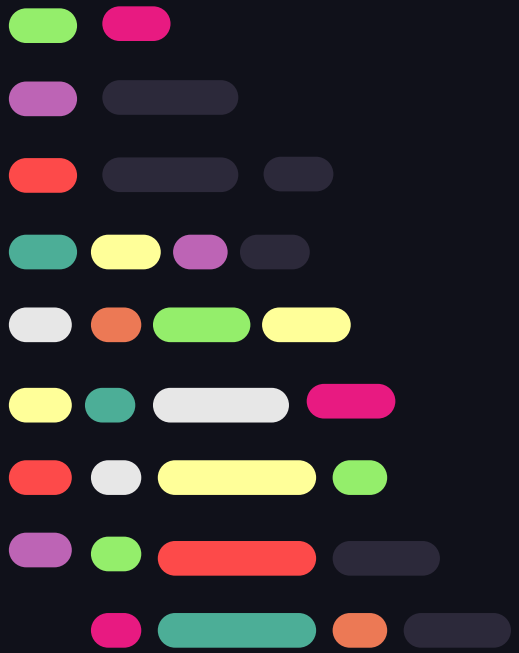
Valider la sauvegarde des fichiers addés


BYE!

HISTORIQUE DES COMMITS

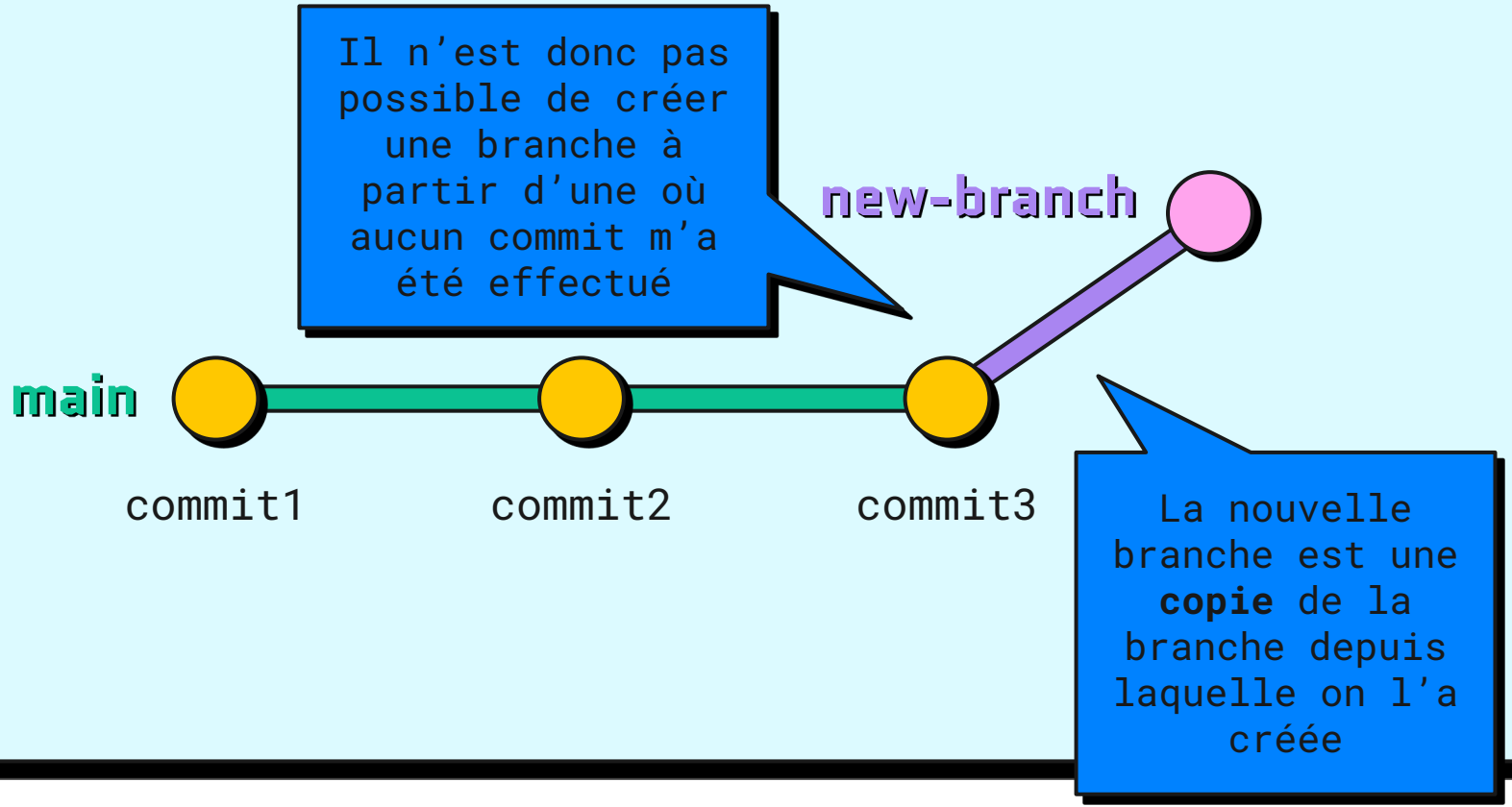


Visualiser l'historique

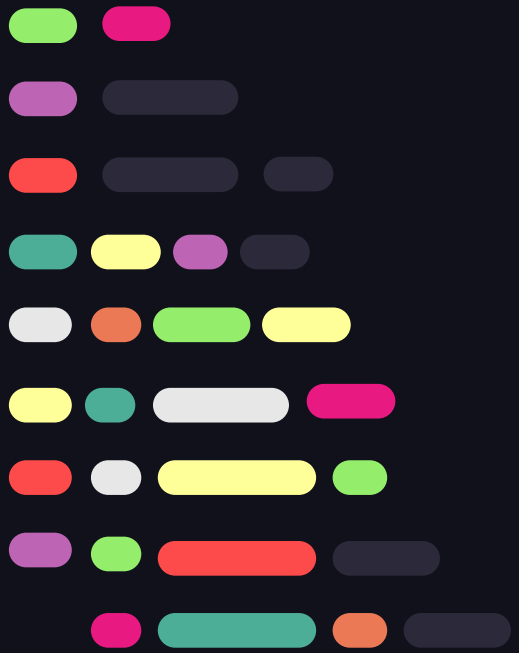


```
# affichage de l'historique in extenso
git log

# historique compact (récent en haut)
git log --oneline

# graphe de l'historique
git log -oneline --graph


# quitter avec q
```

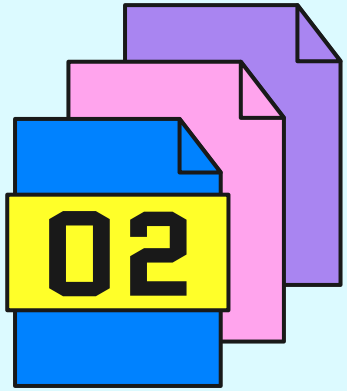
CRÉATION DE BRANCHE



Création d'une branche



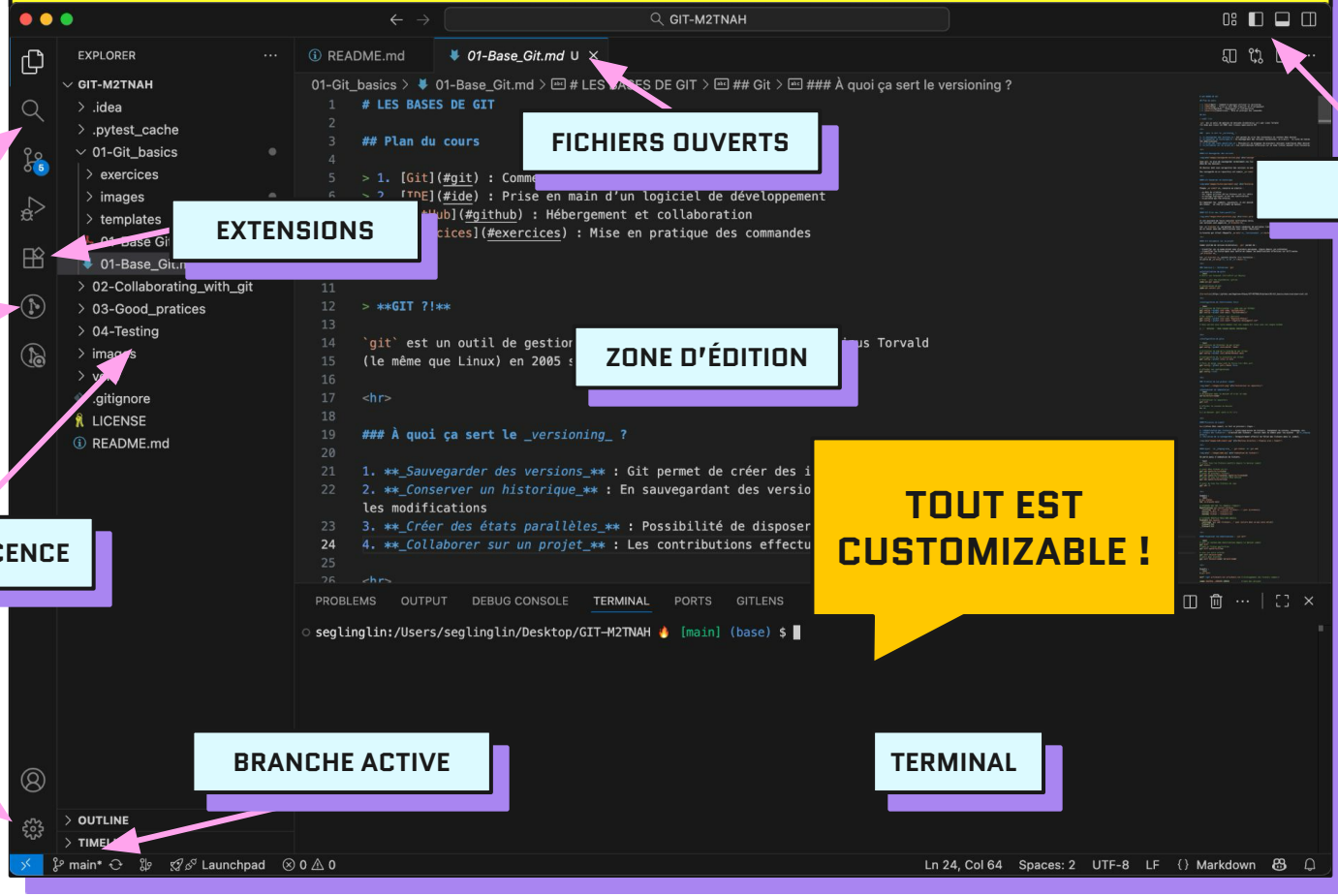
```
# affiche les branches existantes
git branch
# crée une branche nommée <branch-name>
git branch <branch-name>
# change la branche courante
git switch <branch-name>

# crée + change de branche
git switch -c <branch-name>

# ancienne façon de faire
git checkout -b <branch-name>
```



UTILISER UN IDE

Prendre en main un environnement de dev

DÉCOUVRIR SON IDE



RECHERCHER PARTOUT

EXTENSIONS

FICHIERS OUVERTS

LAYOUT

GITLENS

ZONE D'ÉDITION

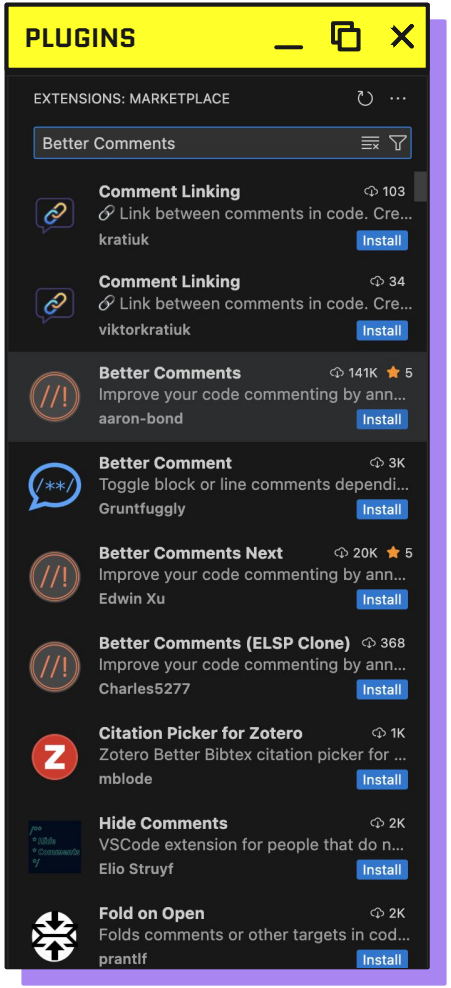
ARBORESCENCE

TOUT EST CUSTOMIZABLE !

PARAMÈTRES

BRANCHE ACTIVE

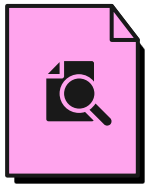
TERMINAL



Extensions

- Pyright
- Jupyter
- Git History
- Better comments

Keyboard shortcut



Ctrl+D

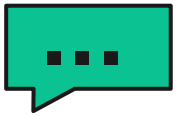
Sélectionner la prochaine occurrence

Ctrl+?

Supprimer une ligne entière

Ctrl+”

Commenter le texte sélectionné



La fusion de branches : 2 méthodes

Merge

Fusion **sans réécriture de l'historique**

Résolution de l'**ensemble des conflits** avant la fusion

Génère **un commit de merge** qui rassemble les modifs

VS

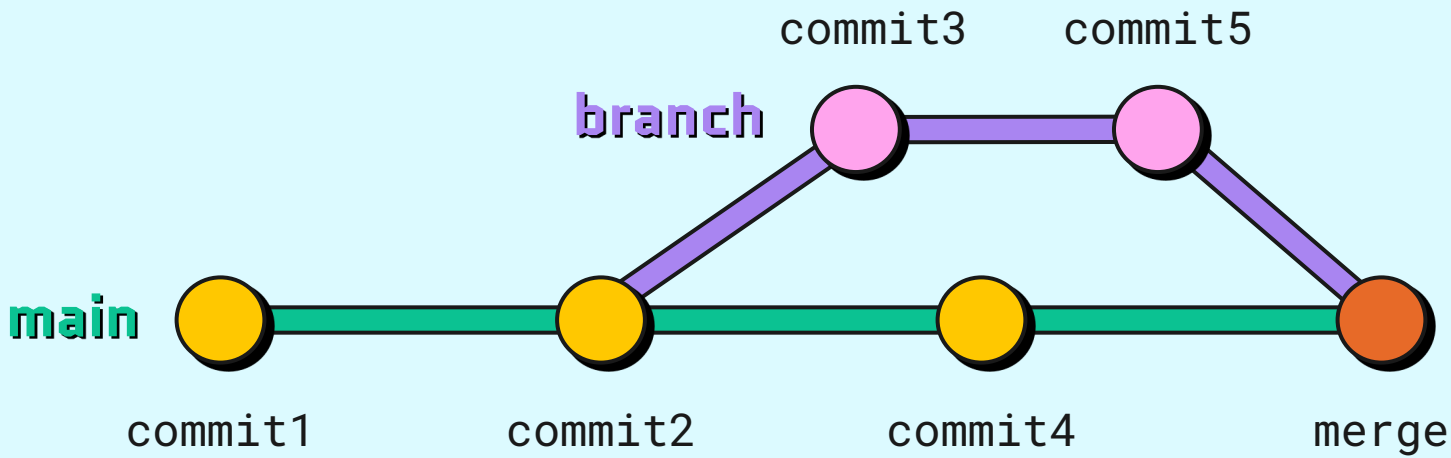
Rebase

Fusion en plaçant **les commits sur la branche rebasée**

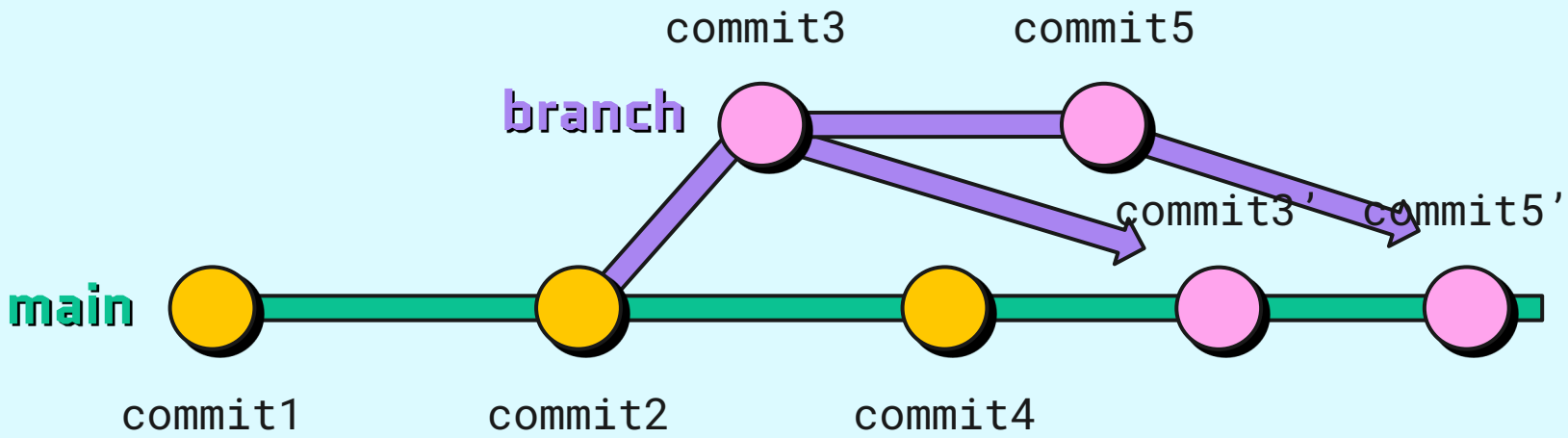
Résolution des conflits **commit après commit**

L'historique des commits est **linéaire**

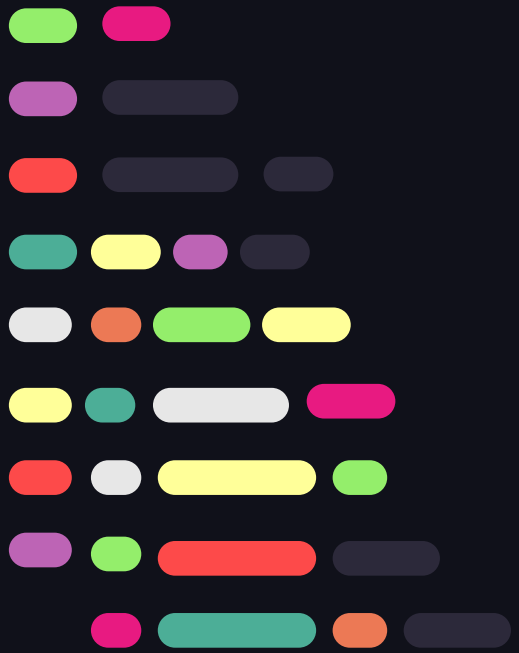
MERGE DE BRANCHES



REBASE DE BRANCHES



Fusionner des branches



```
# merge de main dans my-branch
[my-branch] git merge main

# rebase de main dans my-branch
[my-branch] git rebase main

# de manière générale, privilégiez merge
```

‼️ À retenir

Avant de fusionner, on met à jour :

1. **D'abord** : on récupère les dernières modifs

2. **Ensuite** : on partage ses modifications

Cas classique

```
# je récupère les changements de main
/my/repo [my-branch] git merge main
# ou
/my/repo [my-branch] git rebase main
# → Ma branche est à jour avec main

# je switch sur main
/my/repo [my-branch] git switch main

# je MERGE ma branche
/my/repo [main] git merge my-branch
# → main contient maintenant mes modifs


# ⚠️ on rebase JAMAIS depuis une branche
# ⚠️ où on travaille à plusieurs
```


⚠️
synchroniser
avant de
merger

Exercice 3

03 { ..

Ouvrir son *repository*
dans son IDE
et merger sa branche

} ..

IDE

Ouvrir son *repository* dans VSCode

Branche

Créer une branche et s'y déplacer

Modification

Modifier le contenu de certains fichiers

Commit

Depuis le terminal intégré, commiter ses modifications

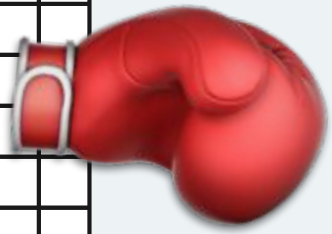
Fusion

Fusionner main dans sa branche

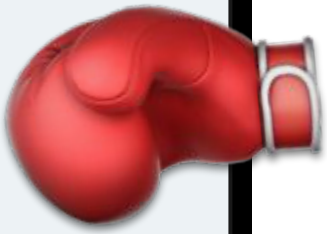
Merge

Depuis main, merger sa branche


BYE!



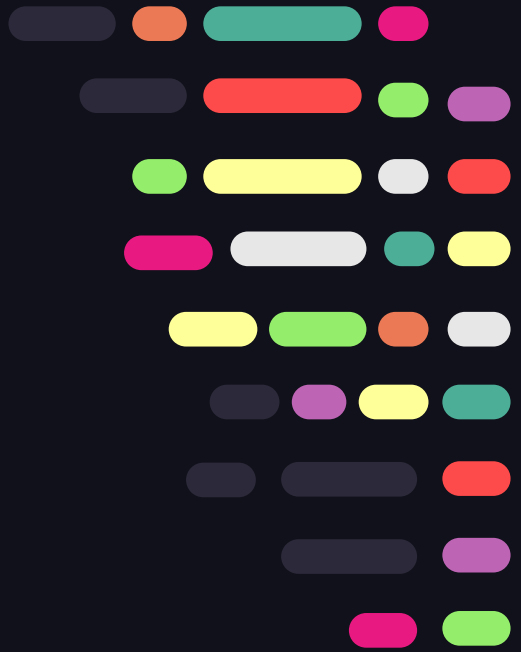
Conflits



Lorsque les deux branches à fusionner comportent des **modifications sur les mêmes lignes de code**, il est nécessaire de **choisir manuellement** quelle modification doit être conservée

Créer un conflit

```
# Création d'un fichier et commit
[main] echo "Hello world!" > hello.txt
[main] git commit -am “Hello file creation”

# Modification du fichier dans une branche
[main] git switch -c branch
[branch] echo "Hello from branch!" > hello.txt
[branch] git commit -am “Hello file modif”

# Modification du fichier dans une autre branche
[branch] git switch -c main
[main] echo "Hello from main!" > hello.txt
[main] git commit -am “Hello file update”

[main] git merge branch # CONFLIT ⛔
```



Trouver les conflits

Recherche

Rechercher
“>>>>>”
dans tout le code

```
# HEAD correspond à la branche où vous êtes

def hello():
<<<<<<< HEAD
    print("Hello from my-branch!")
=======
    print("Hello from other-branch!")
>>>>>>> other-branch
```

Résoudre un conflit

Comment savoir quoi garder ?

1. Garder les modifications pertinentes
2. Assurer le bon fonctionnement du code
3. Combiner à la main les portions de chaque branche

```
# Résoudre un conflit revient à ne laisser qu'une version
def hello():
    print("Hello from my-branch and other-branch!")
```

Finaliser le *merge*

```
Merge branch 'other-branch' into 'my-branch'

# Conflicts:
#     hello.py
# It looks like you may be committing a merge.
# If this is not correct, please remove the file
# .git/MERGE_HEAD and try again.
# Please enter the commit message for your changes.
# Lines starting with '#' will be ignored
```

1. **`git add`**

Après avoir résolu les conflits, on ajoute les fichiers résolus à *staging area*

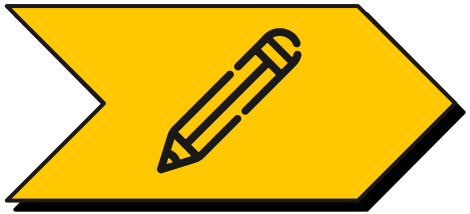
2. **`git commit`**

git génère alors un message de commit automatique.

Il peut être modifié ou conservé tel quel avec `Ctrl+S` et `Ctrl+X` (nano)

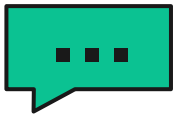
Régler un conflit de *merge*

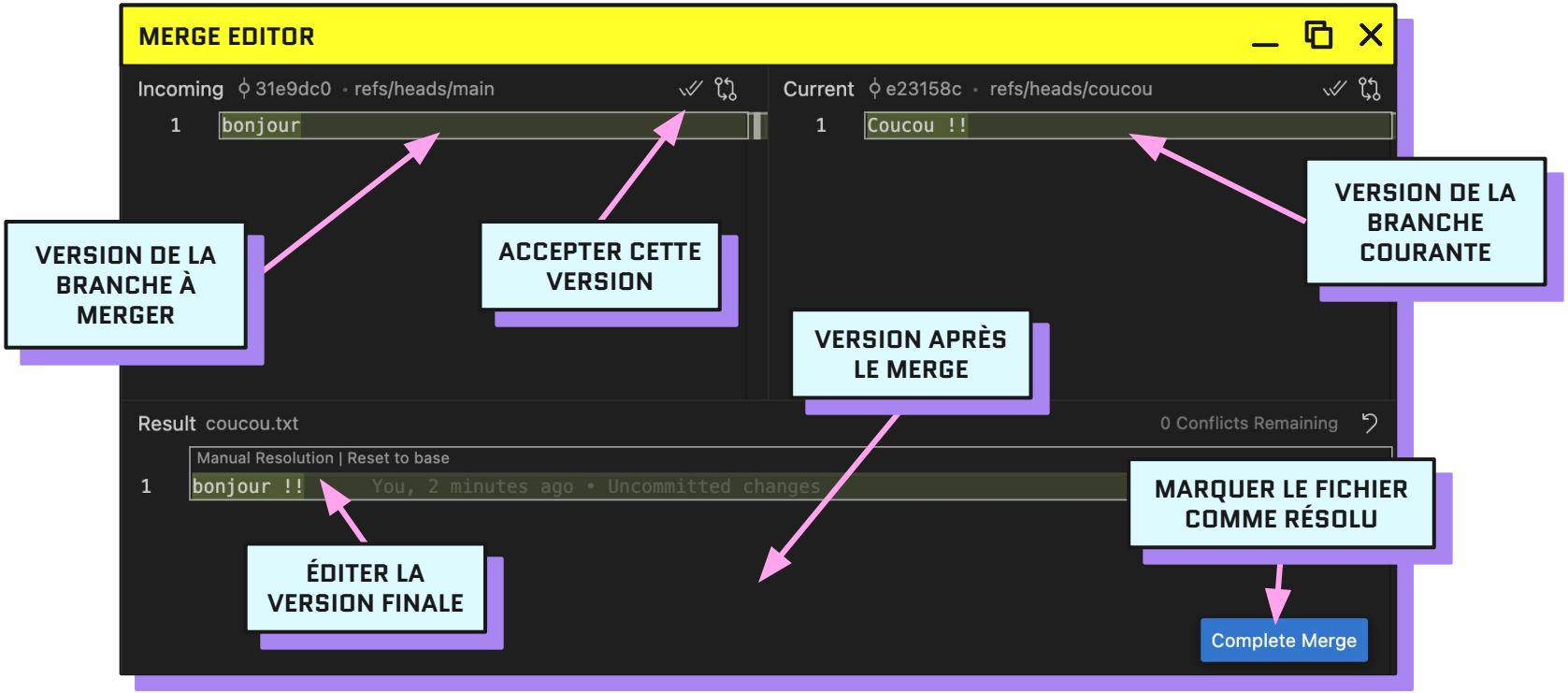
Trouver les endroits où sont situés les conflits



Modifier les fichiers pour ne conserver qu'une seule version

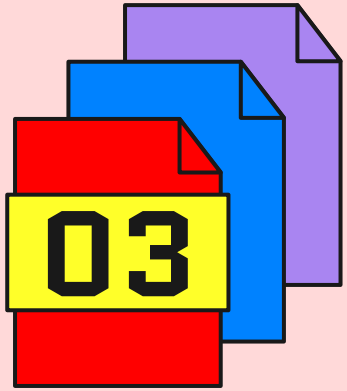
Ajouter les fichiers à la *staging area* et commiter





Dans le menu ***Source control*** :

1. Pour chaque fichier en conflit sous "*Merge Changes*"
    a. Résoudre les conflits
    b. Cliquer sur "*Complete Merge*" (`add` le fichier)
2. Finaliser le merge avec "*Continue*" (`commit`)



GITHUB

La puissance de git en ligne

GitHub?

GitHub est un service en ligne pour héberger son *repository*

Plateformes en ligne

GitHub

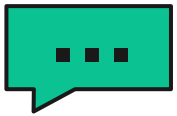
La plus répandue, sert de portfolio aux développeurs

GitLab

Principale alternative, peut être hébergé sur des serveurs privés

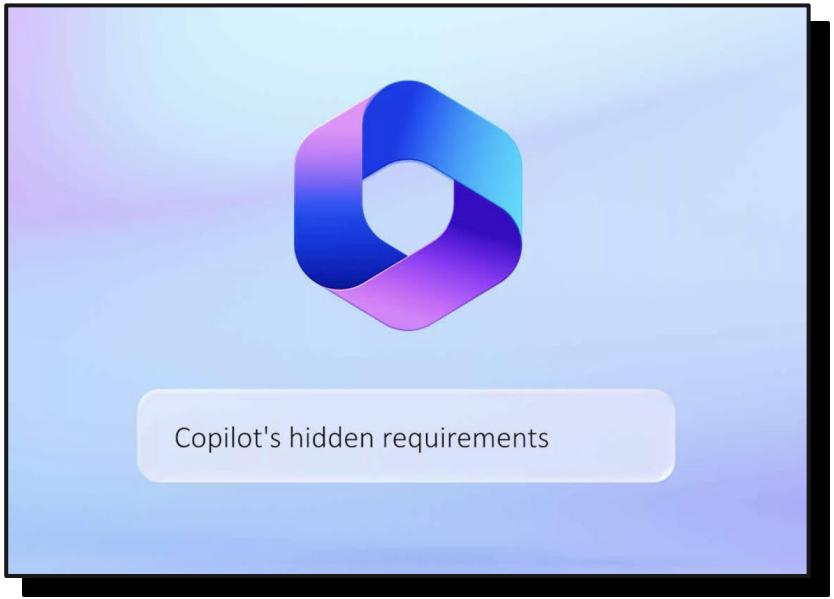
Bitbucket

Version développée par Atlassian, il en existe plein d'autres



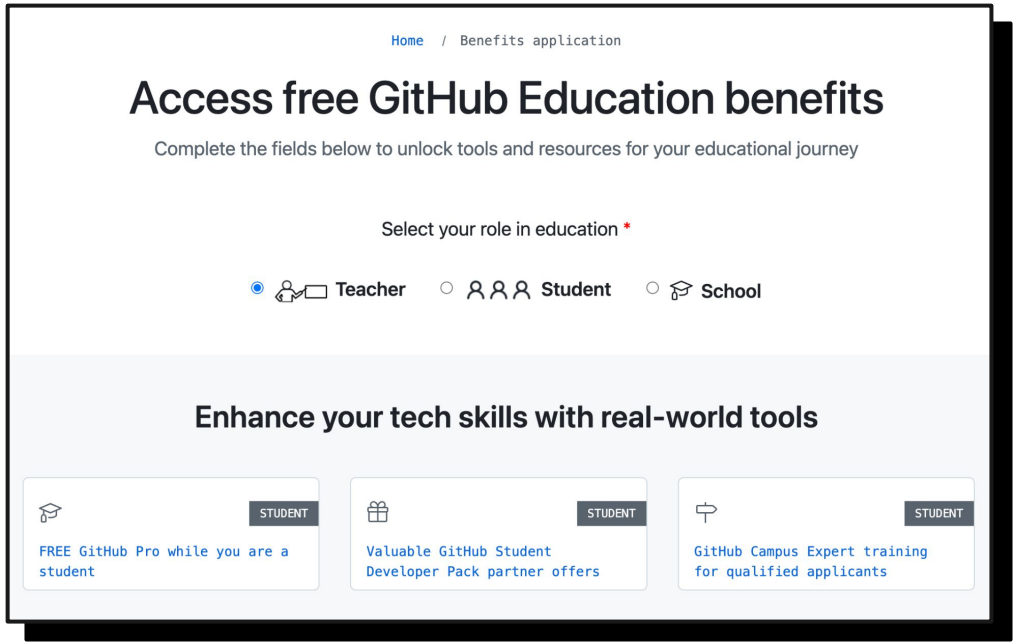
Prérequis licence étudiant Github

1. Ajouter son adresse mail de l'École
   github.com/settings/emails
2. Remplir son profil GitHub
   github.com/settings/profile
3. Customiser son **README**
   github.com/<your-username>
4. Renseigner ses infos bancaires
   github.com/settings/billing/payment_information
5. Activer la 2 factors Authentication
   a. Par SMS
   b. Puis avec l'app mobile



Obtenir licence étudiant Github

1. Renseigner son institution github.com/settings/education/benefits
2. Prendre en photo sa carte étudiant
3. Attendre la validation
4. Installer le plugin copilot sur son IDE
5. Profiter de copilot gratuitement !



git / github

git

Logiciel de gestion de versions

Outil local sur sa machine pour modifier ses *repositories*

Ensemble de commandes disponibles dans le terminal

VS

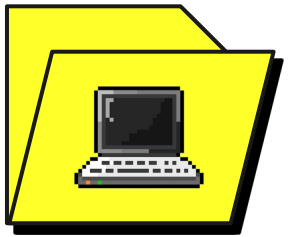
GitHub

Service d'hébergement en ligne

Service en ligne pour stocker et partager ses *repositories*

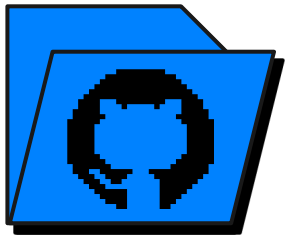
Interface web pour visualiser et collaborer sur des *repos*

Dépôt local ou distant



Local

Dossier contenant le *repo*
situé sur votre ordinateur



Remote

Version en ligne du *repo*,
à la manière d'un *drive*

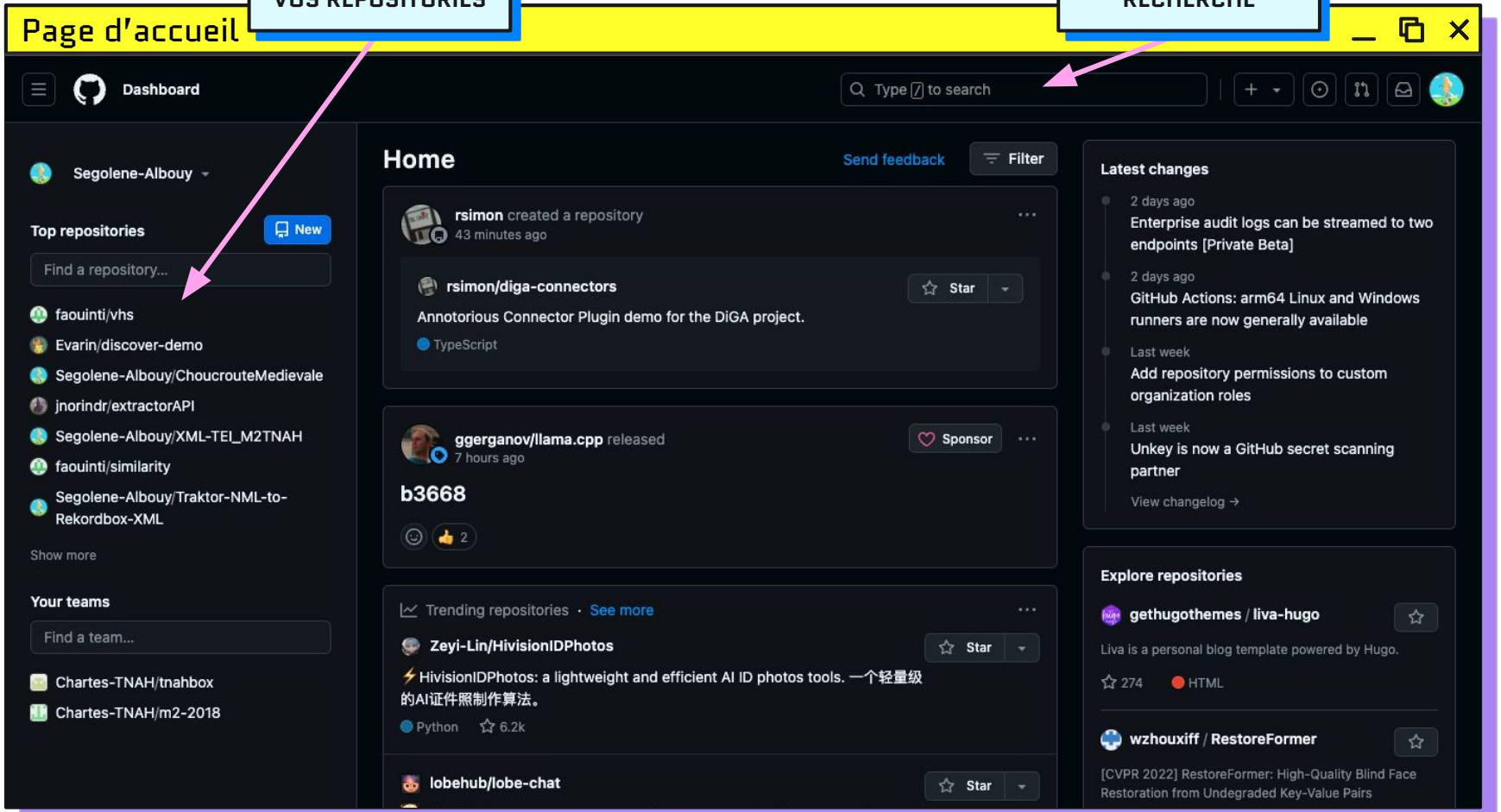
Remote repository

Version du *repository* (aussi désignée par **origin**) en ligne qui permet de centraliser l'historique et les modifications. Public ou privé, plusieurs collaborateurs peuvent y contribuer, avec différents droits.

Page d'accueil

VOS REPOSITORIES

RECHERCHE



Page de profil

Overview | Repositories 37 | Projects | Packages | Stars 83

Ségolène Albouy

Segolene-Albouy · she/her

Computer vision for digital humanities
Teacher @ École des chartes Research
@ École des Ponts

Edit profile

40 followers · **37** following

CNRS
Paris
htpps://discover-demo.enpc.fr
https://orcid.org/0009-0008-0998-978X
@AlbouySegolene

Segolene-Albouy / README.md

- 👋 Hi, I'm @Segolene-Albouy
- 🔧 I build tools that enables researchers in social sciences to use computer vision algorithms
- 👩‍🏫 I teach to Masters students Git, XML TEI, Machine Learning, Dataviz & Digital Humanities
- 👀 I'm interested in sauerkraut and process automation

Pinned Order updated. Customize your pins

vayvi/HDV Public
Historical Diagram Vectorization
Jupyter Notebook ☆ 11

kanon Public
Forked from ALFA-project-erc/kanon
Python

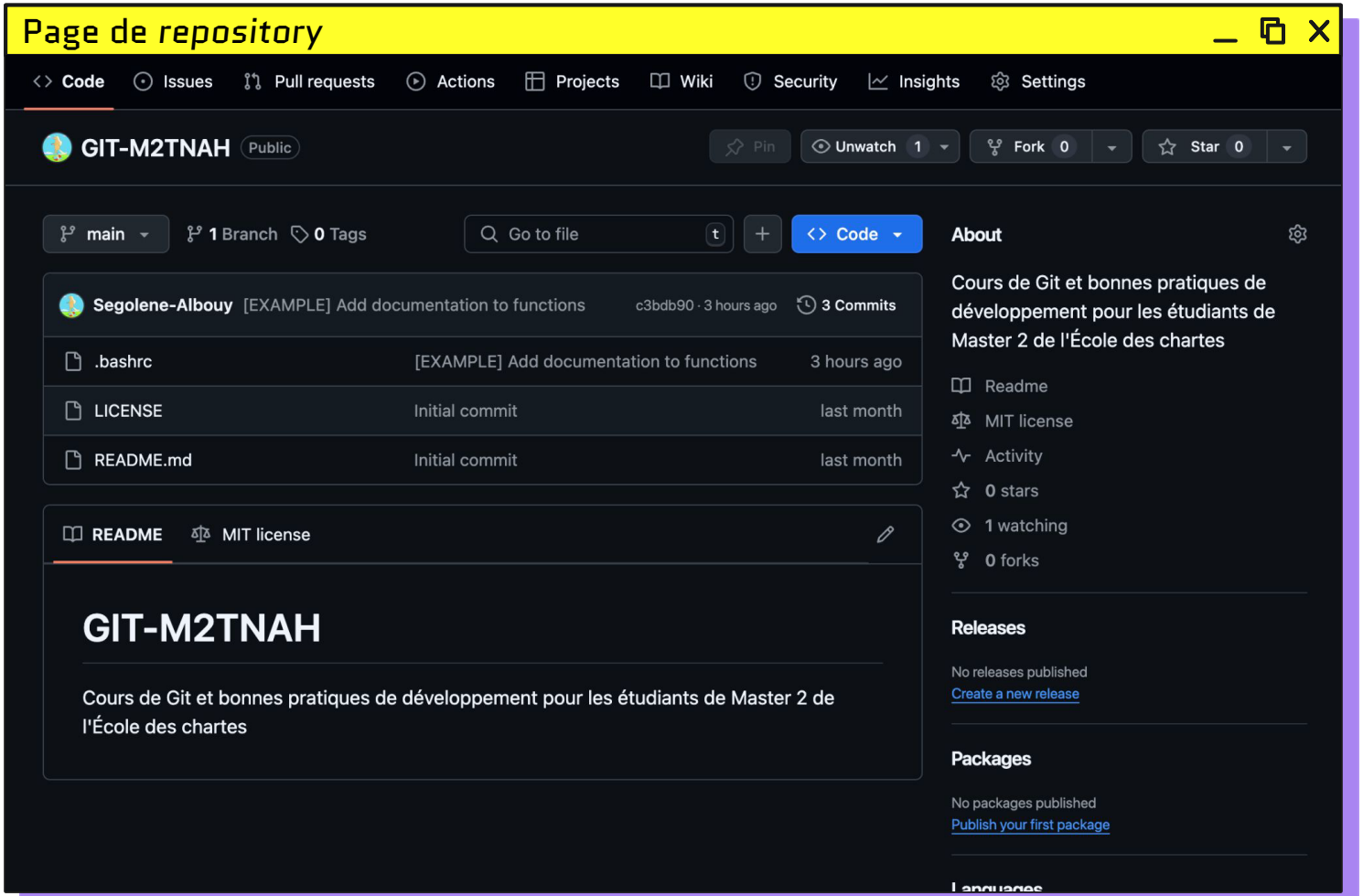
XML-TEI_M2TNAH Public
Ressources et présentations pour le cours en XML-TEI des M2 TNAH de l'École des chartes
HTML ☆ 8 ⑂ 8

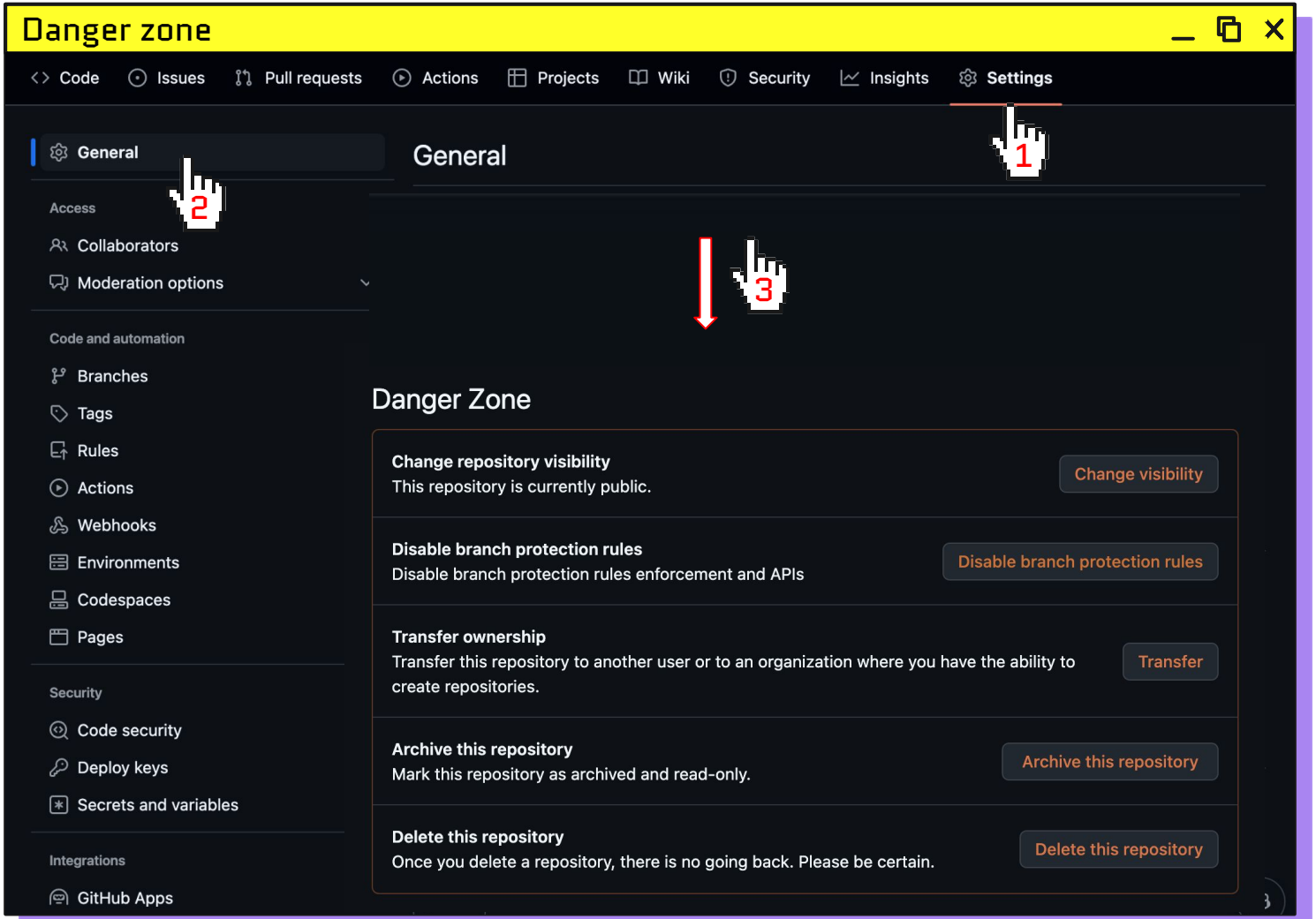
iiif-downloader Public
Script for automatic image retrieval from IIIF manifests
Python ☆ 1

Traktor-NML-to-Rekordbox-XML Public
Python script to convert your Traktor playlists to a Rekordbox format (with Hot Cues and Loops)
Python ☆ 8 ⑂ 1

ChoucrouteMedievale Public
Simple website update for medieval party
JavaScript ☆ 2

920 contributions in the last year Contribution settings 2024


Page de *repository*
Code
Issues
Pull requests
Actions
Projects
Wiki
Security
Insights
Settings
GIT-M2TNAH
Public
Pin
Unwatch 1
Fork 0
Star 0
main
1 Branch
0 Tags
Go to file
t
Code
Segolene-Albouy [EXAMPLE] Add documentation to functions
c3bdb90 · 3 hours ago
3 Commits
.bashrc
[EXAMPLE] Add documentation to functions
3 hours ago
LICENSE
Initial commit
last month
README.md
Initial commit
last month
README
MIT license
GIT-M2TNAH
Cours de Git et bonnes pratiques de développement pour les étudiants de Master 2 de l'École des chartes
About
Cours de Git et bonnes pratiques de développement pour les étudiants de Master 2 de l'École des chartes
Readme
MIT license
Activity
0 stars
1 watching
0 forks
Releases
No releases published
Create a new release
Packages
No packages published
Publish your first package
Languages


Danger zone
Code
Issues
Pull requests
Actions
Projects
Wiki
Security
Insights
Settings
1
General
2
Access
Collaborators
Moderation options
Code and automation
Branches
Tags
Rules
Actions
Webhooks
Environments
Codespaces
Pages
Security
Code security
Deploy keys
Secrets and variables
Integrations
GitHub Apps
General
3
Danger Zone
Change repository visibility
This repository is currently public.
Change visibility
Disable branch protection rules
Disable branch protection rules enforcement and APIs
Disable branch protection rules
Transfer ownership
Transfer this repository to another user or to an organization where you have the ability to create repositories.
Transfer
Archive this repository
Mark this repository as archived and read-only.
Archive this repository
Delete this repository
Once you delete a repository, there is no going back. Please be certain.
Delete this repository

Fichiers spéciaux du *repository*

Liste de fichiers à ignorer par git

.gitignore

Fichier de présentation du repo

README

Licence précisant les conditions d'utilisation du repo

LICENSE

.gitignore example

```
# Jupyter Notebook
.ipynb_checkpoints

# Environment variables
.env

# Virtual environments
env/
venv/

# PyCharm
.idea/
```

README.md example

Installation

Step-by-step instructions on how to install the project and its dependencies.

1. Clone the repository:

```bash
git clone git@github.com:<username>/<repository>.git
cd <repository>
```

2. Set up environment:

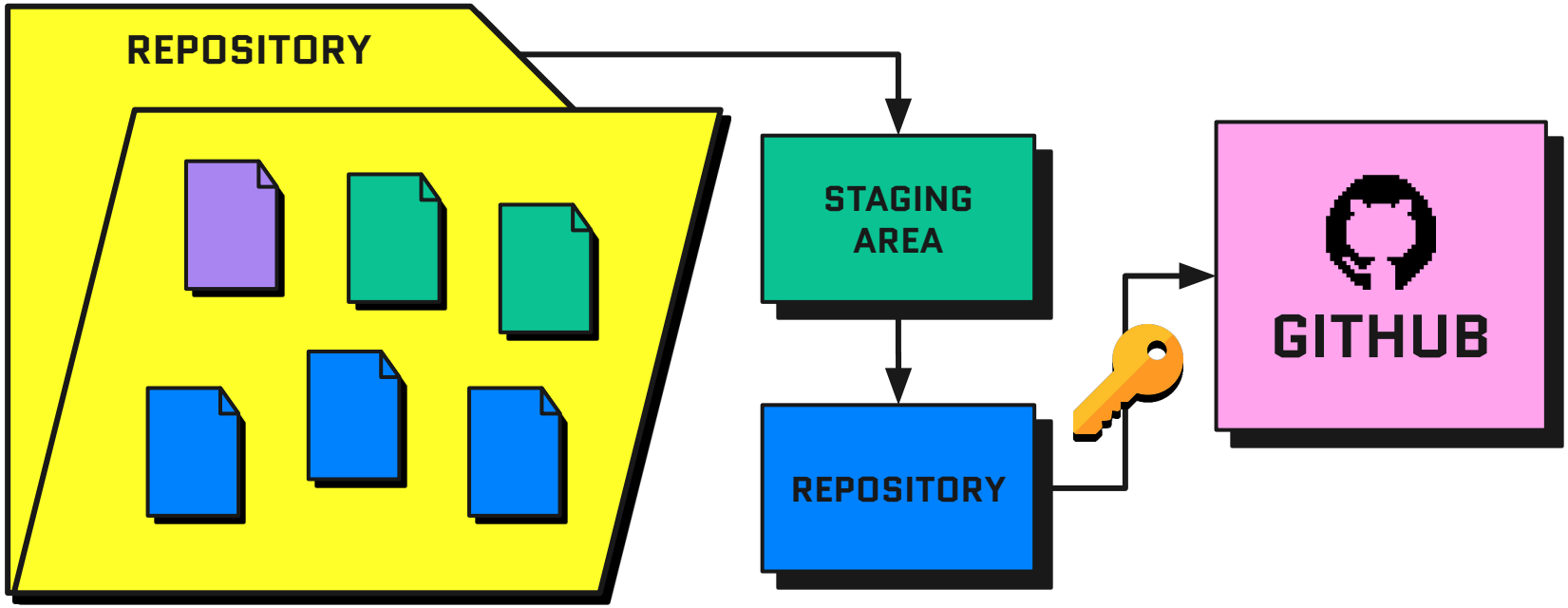
```
...
```

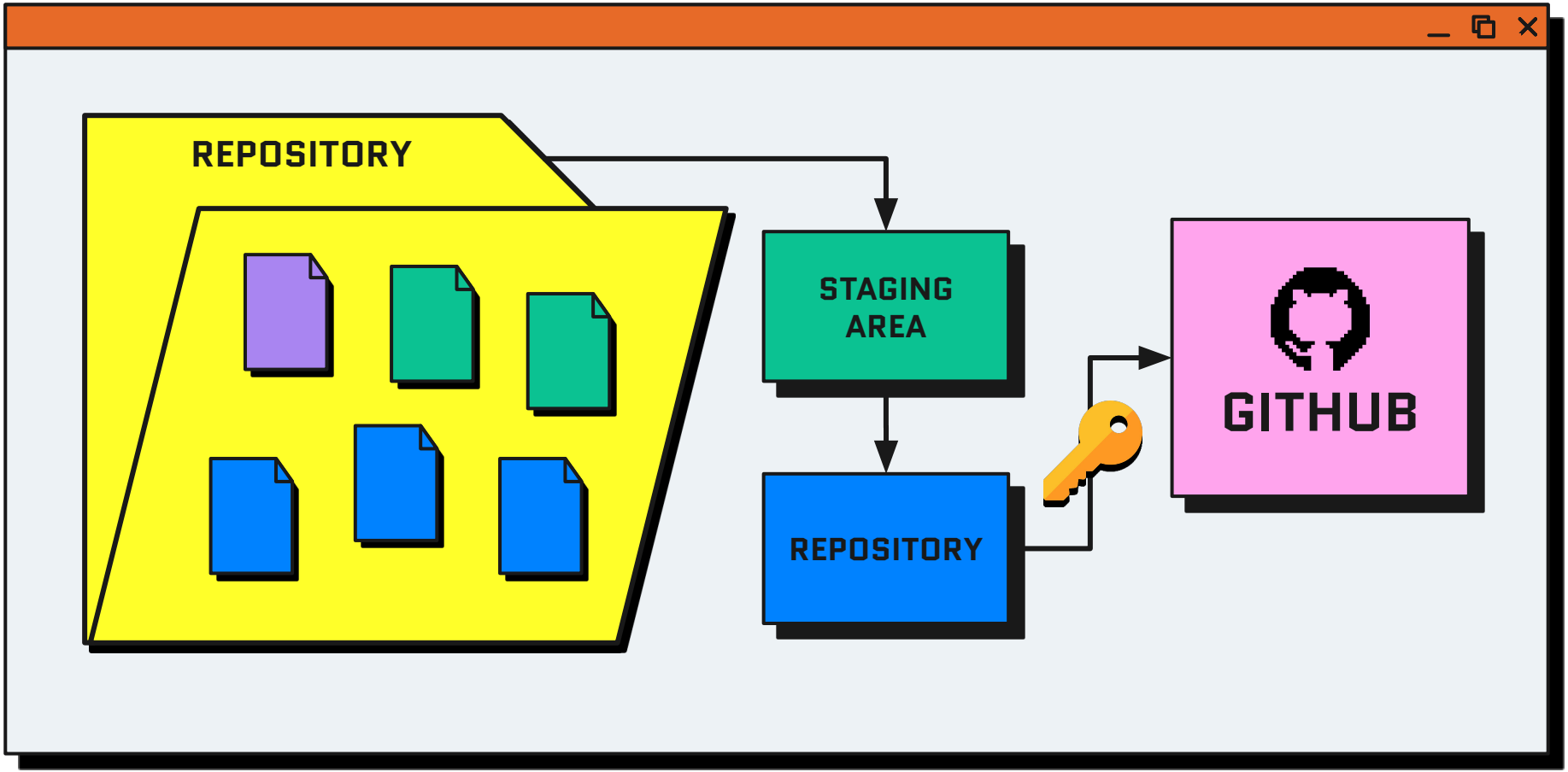
LICENSE

Sélection de la licence à la création du *repository* sur GitHub

choosealicense.com

Processus de publication



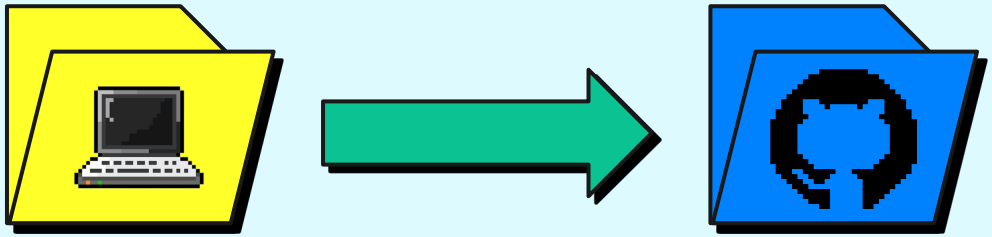

REPOSITORY
STAGING AREA
REPOSITORY
GITHUB

Ajouter une clef SSH pour son ordinateur

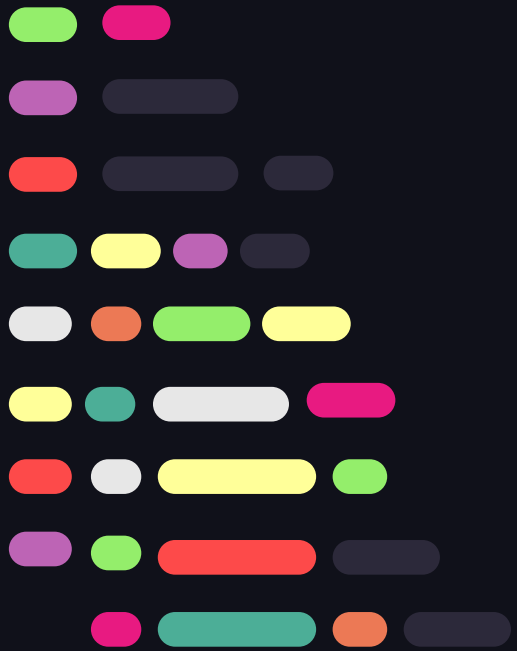
1. Créer une clef SSH depuis le terminal
   ```
   ssh-keygen -t ed25519
   cat ~/.ssh/id_ed25519.pub
   ```
2. GitHub > Settings > SSH and GPG keys > New
3. Coller la clef publique de votre terminal
4. Nommer la clef pour désigner votre ordinateur


BYE!

DÉPOSER SON CODE SUR LE DÉPÔT DISTANT



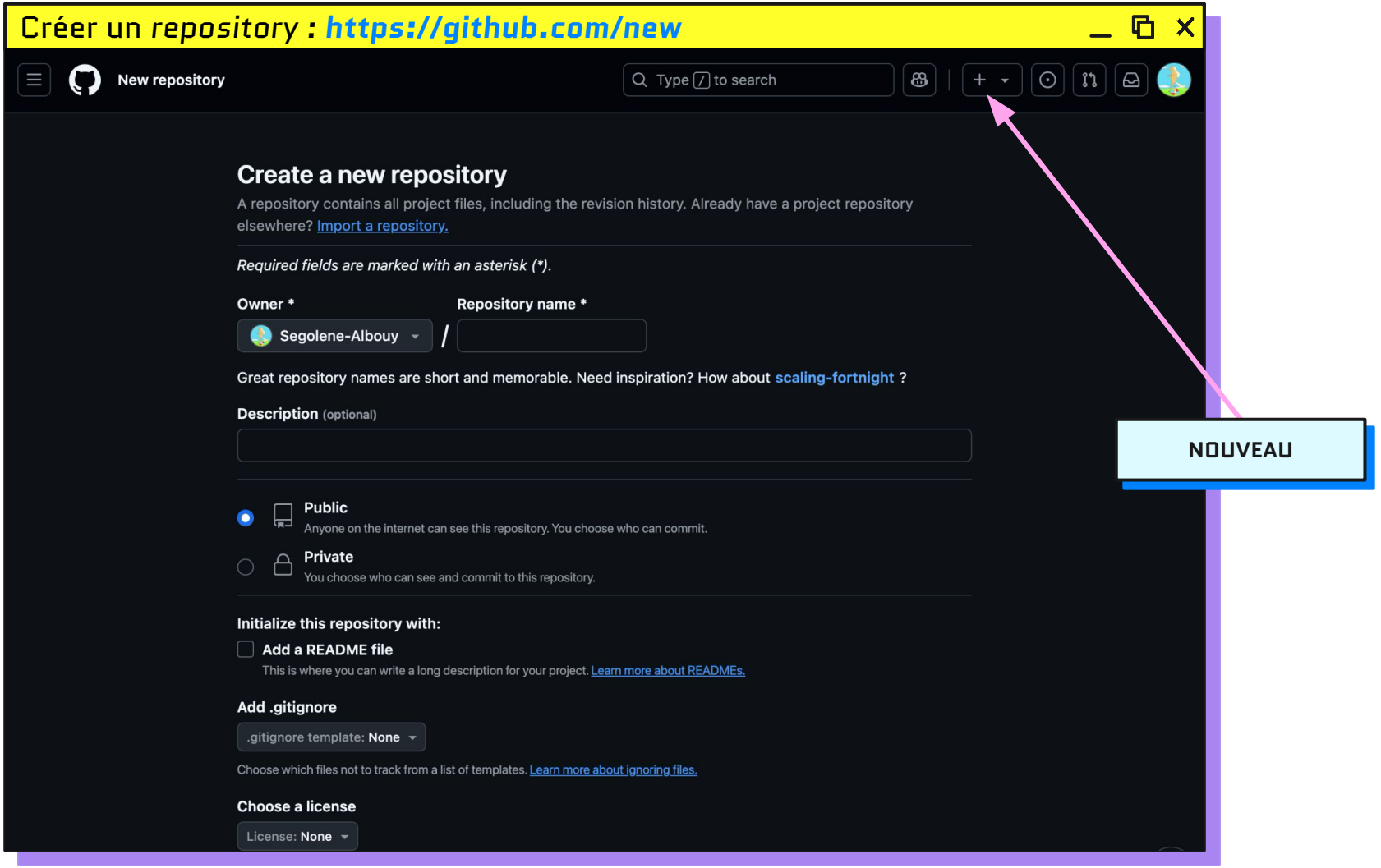
Publier son code



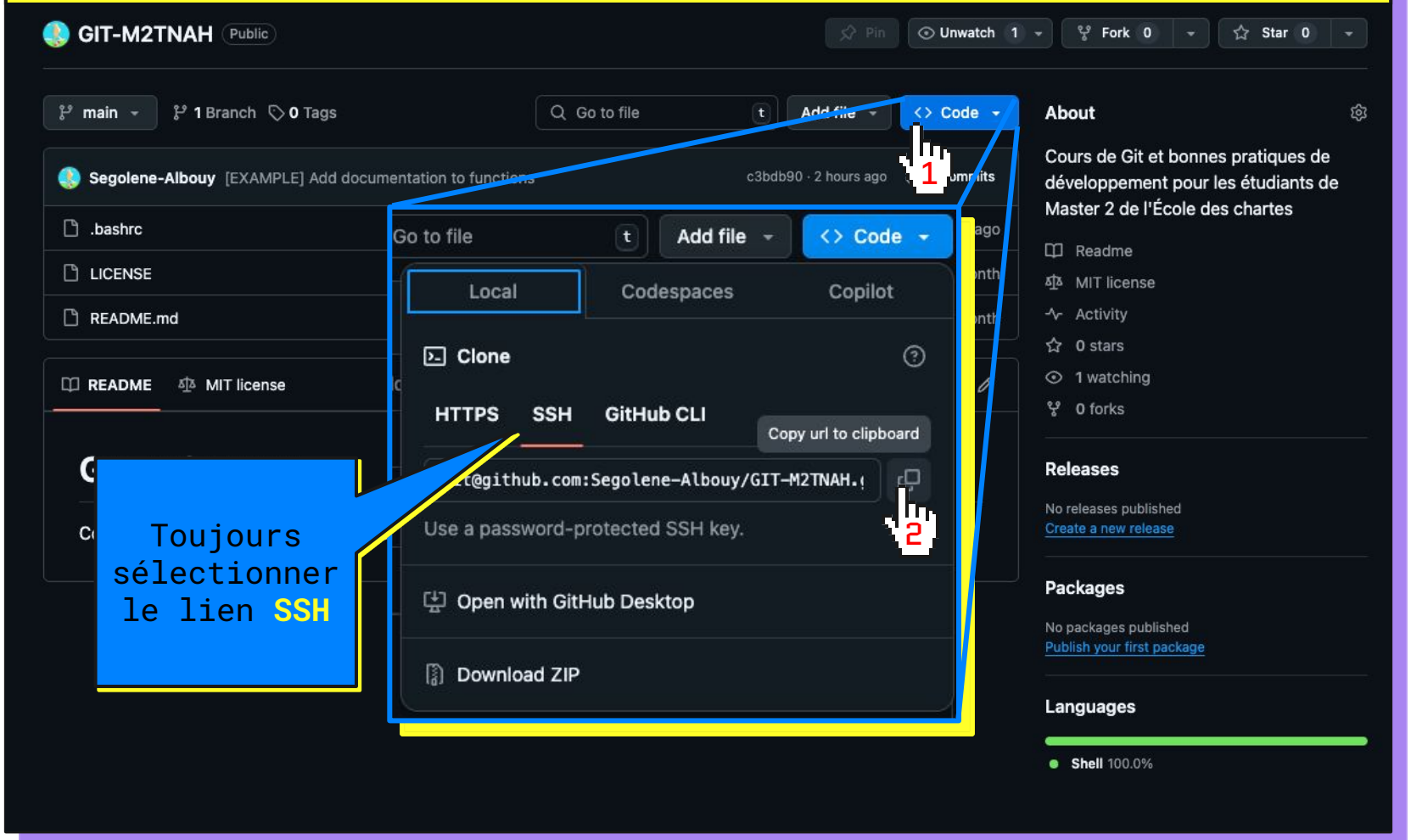
```
# déposer son code local sur GitHub
git push
```

Pas de dépôt distant !

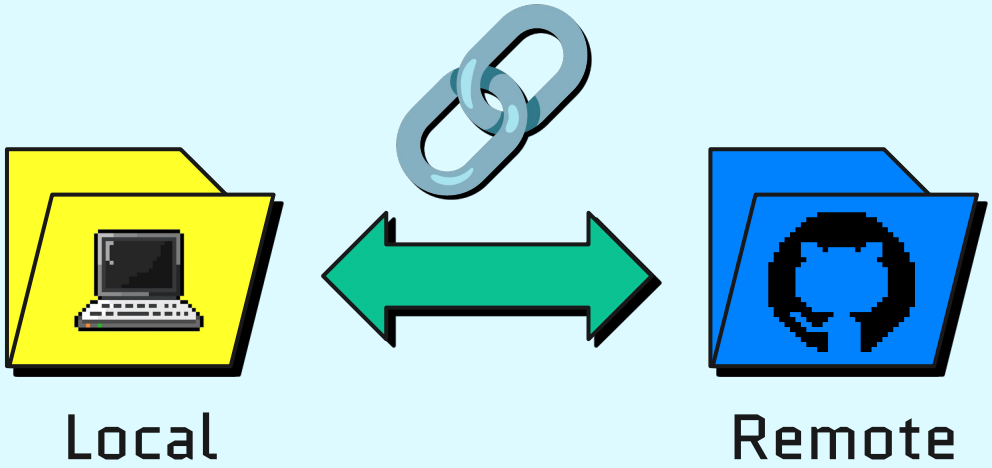
```
$ git push

fatal : Pas de destination pour pousser.
Spécifiez une URL depuis la ligne de commande
ou configurez un dépôt distant en utilisant

    git remote add <origin> <url>

et poussez alors en utilisant le dépôt distant

    git push <origin>
```


Créer un repository : https://github.com/new
New repository
Type / to search
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 *
Segolene-Albouy
/
Repository name *
Great repository names are short and memorable. Need inspiration? How about scaling-fortnight ?
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
NOUVEAU

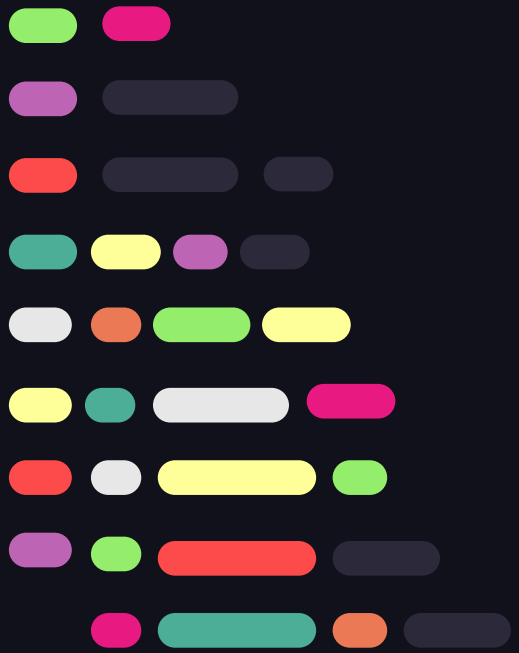
Lier son dépôt local à un remote



LIER SON DOSSIER LOCAL À UN REMOTE REPOSITORY

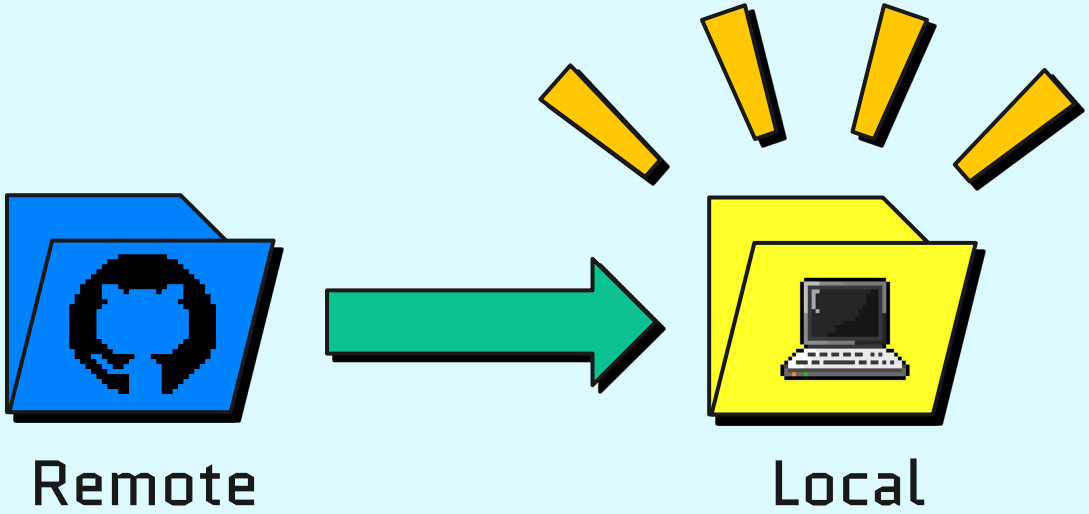


Lier un remote à son local

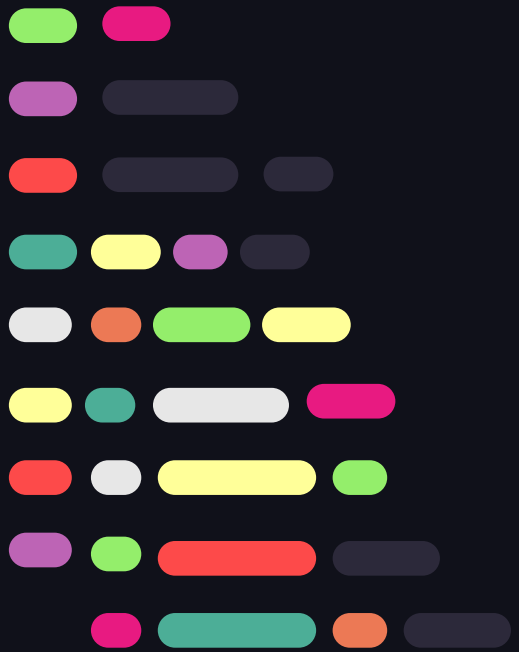


```
# Lier son repo local à un repo distant
git remote add origin <url-repo-distant>

# Il est ensuite possible de publier son repo
git push
```

COPIER UN *REPOSITORY* SUR GITHUB SUR SON ORDINATEUR



Récupérer un remote en local



```
# Copier un repo distant sur son ordinateur
git clone <url-repo-distant>

# De manière générale, pour créer un nouveau projet,
# on le crée sur GitHub, puis le clone en local

# Cloner le repository du cours
git clone git@github.com:Segolene-Albouy/GIT-M2TNAH.git
```

OSSEKOUR !

GitHub veut pas m'autoriser à cloner le *repo* ! 😭

On commence par lire le message de son terminal :

```
git@github.com: Permission denied (publickey).
fatal: Could not read from remote repository.

Please make sure you have the correct access
rights and the repository exists
```

Permission denied

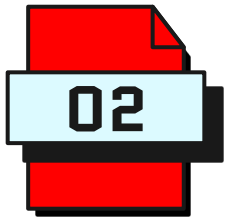
L'erreur peut être due à :

COMPTE GIT LOCAL MAL CONFIGURÉ

```
# Vérifier dans sa config locale que user & email
# correspondent bien à ceux du compte GitHub
git config --list

# Si ce n'est pas le cas, corriger avec
git config --global user.name "<github-user>"
git config --global user.email "<github-email>"
# (⚠️ retirer les chevrons)
```

Permission denied

L'erreur peut être due à :



CLEF SSH DÉFECTUEUSE

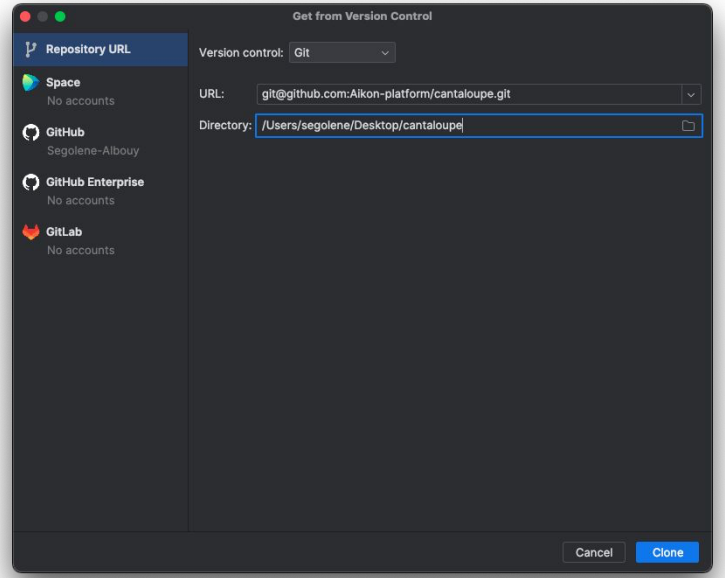
```
# Vérifier la config SSH avec GitHub
ssh -T git@github.com

# Si cela renvoie une erreur,
# recréer une paire de clefs SSH
ssh-keygen -t ed25519

# Afficher le contenu de la clef publique
cat ~/.ssh/id_ed25519.pub

# Copier la clef publique et la coller sur GitHub
```

Cloner un projet Git sur son IDE

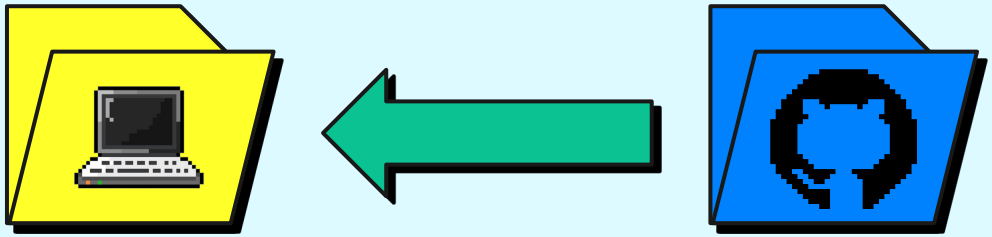
Copier l'URL SSH du *repository*

VSCode

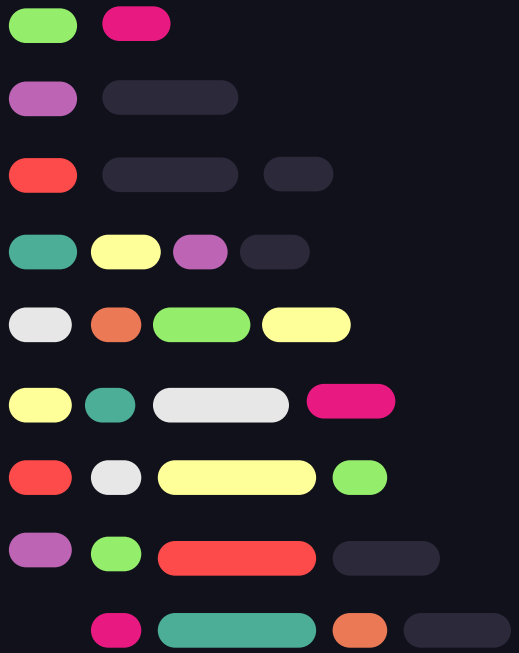
1. Ouvrir la *Command palette* `Ctrl+Shift+P`
2. Rechercher "*Git clone*"
3. Connexion à GitHub
4. Sélection du *repository*
5. Choix de l'emplacement
6. Ajout au *Workspace*



RÉCUPÉRER DU CODE DEPUIS LE DÉPÔT DISTANT

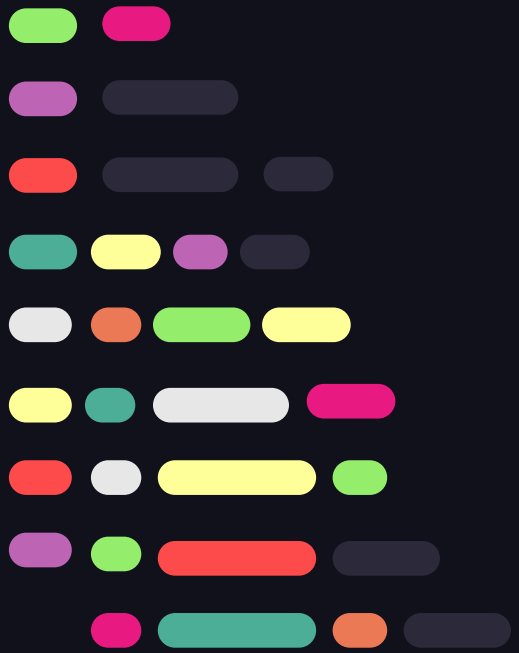


Récupérer du code



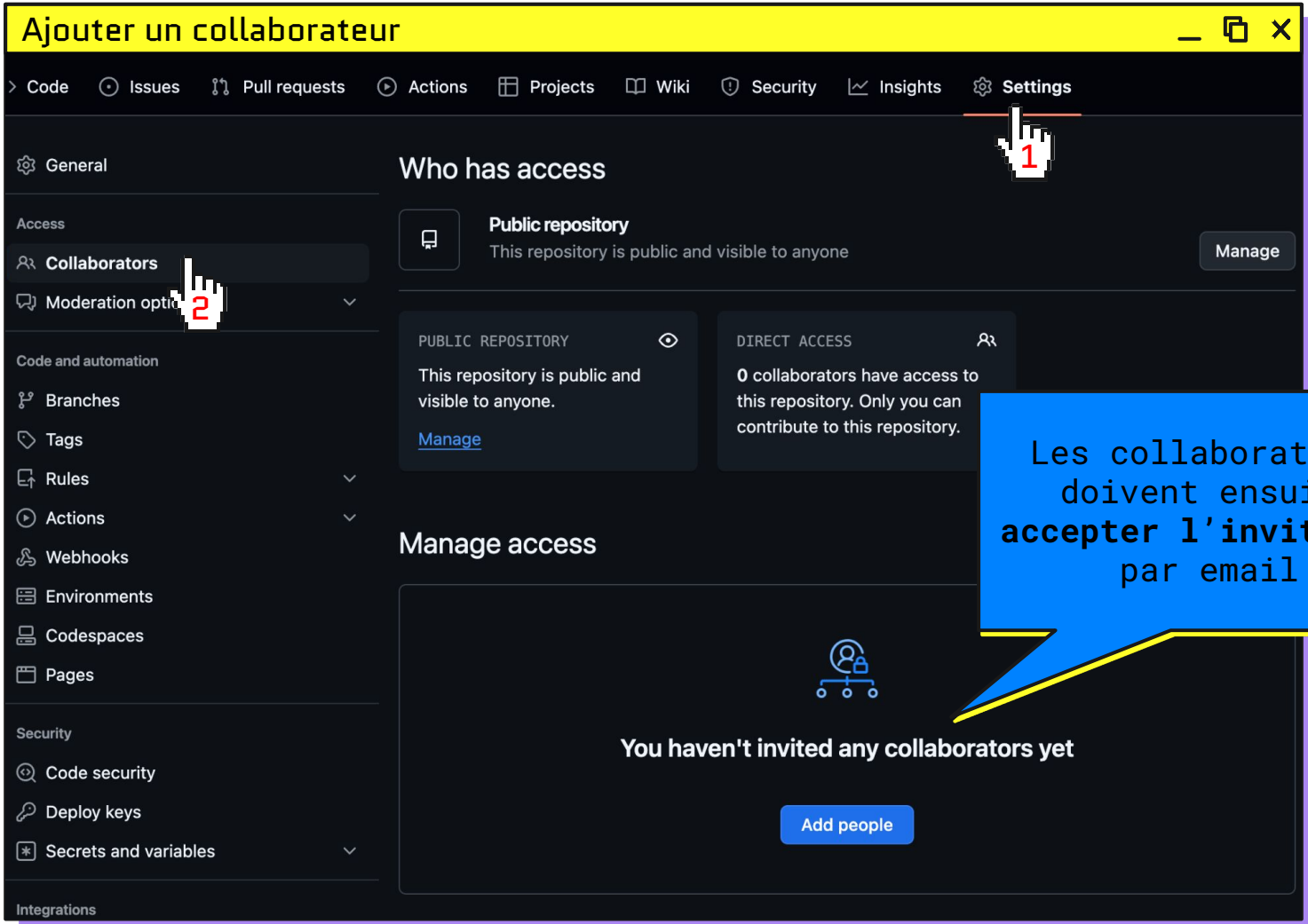
```
# récuperer du code depuis GitHub
git pull

# git pull est une forme de fusion de branches
# puisqu'on réunit le contenu d'une branche
# distante avec une branche locale

# récuperer du code depuis GitHub sans fusion
git fetch
```

Fusion de branche distante



```
# Cet alias permet d'utiliser
# rebase s'il n'y a pas de conflit avec remote
# ou merge si un conflit doit être résolu
git config --global alias.pull '!f() { \
  git fetch && \
  (git rebase "$@") || \
  (git merge "$@"); \
}; f'
```

Ajouter un collaborateur



Les collaborateurs doivent ensuite **accepter l'invitation** par email

Exercice 4

04 { *par deux*

Collaborer sur un *repository*

} ..

{ .. Partie 1

Création

Un des deux crée un *repository* sur GitHub

Collab

Il ajoute l'autre dans les collaborateurs

Clone

Chacun clone le *repository* sur son ordinateur

Modif

Depuis son IDE, chacun effectue des commits sur la branche **main**

Push

Chacun push ses commits sur le *repo* distant

Quoi ?

Tenter de comprendre ce qui se passe

OSSEKOUR !

L'autre a réussi à pusher mais moi ça marche paaaas ! 😭

On commence par lire le message de son terminal :

```
To github.com:utilisateur/nom-du-repo.git
 ! [rejected] <local> -> <remote> (fetch first)
error: failed to push some refs to '<remote>'
...
hint: (e.g., 'git pull ...') before pushing again.
```

Condition pour *pusher*

Il n'est pas possible de *pusher* ses modifications **sans avoir auparavant intégré les commits déjà publiés** sur le *remote*

Pour mettre à jour sa version du code et pouvoir *pusher*, il faut d'abord *pull* le dépôt distant

```
To <remote> ! [rejected] <b> -> <b> (fetch first)
error: failed to push some refs to '<remote>'
...

$ git pull
```

OSSEKOUR !

GitHub veut pas m'autoriser à *pusher* ! 😭

On commence par lire le message de son terminal :

```
remote: Support for password authentication
was removed on August 13, 2021. Please use a
personal access token instead.

fatal: Authentication failed for
'https://github.com/username/repo.git/'
```

Authentication failed

Lorsqu'on utilise le **lien HTTPS** pour cloner un *repository*, il n'est plus possible de publier son code.

Il faut alors changer le remote pour utiliser **lien SSH** du *repository*

```
# Modifier l'URL du remote pour le lien SSH
git remote set-url origin git@github.com:ssh.git
```

Configurer *pull*

```
warning: Pulling without specifying how to reconcile divergent branches is
discouraged. You can squelch this message by running one of the following
commands sometime before your first pull:

# git pull opère un merge des branches locale et remote
git config --global pull.rebase false # ⬅️ choisir celle-là

# git pull opère un rebase des branches locale et remote
git config --global pull.rebase true

# git pull opère un merge des branches locale et remote seulement si aucun conflit
git config --global pull.ff only # fast-forward only
```

{ .. Partie 2

Pull

Pull de la branche **main** locale avec les modifs remote

Branche

Chacun crée une branche à partir de **main**

Modif

Chacun modifie les mêmes bouts de fichiers déjà existants

Push

Chacun *push* sa branche sur le *repo* distant

Quoi ?

Tenter de comprendre ce qui se passe

*

} ..

Pusher une nouvelle branche

Lorsqu'une nouvelle branche est créée en local, il faut configurer une branche distante sur laquelle pusher

```
$ git push

fatal : La branche courante <branch> n'a pas de
branche amont.

Pour pousser la branche courante et définir la
distante comme amont, utilisez

    git push --set-upstream origin <branch>
```

{ .. Partie 3

Fetch

Récupérer la branche de l'autre en local

Switch

Changer de branche pour aller sur l'autre

Merge

Merger sa branche dans celle de l'autre

Conflit

Ouvrir le fichier avec conflit sur l'IDE

Résolution

Modifier le fichier pour ne conserver qu'une version des modifs

Fusion

Finir le merge et publier le code sur GitHub


BYE!