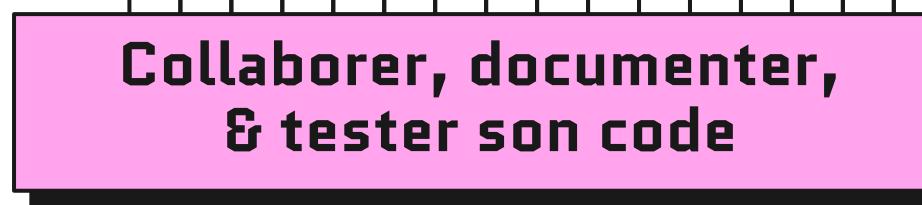
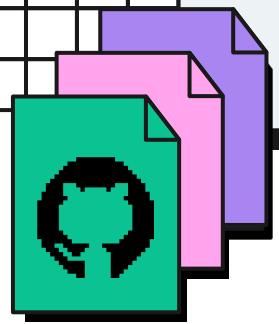


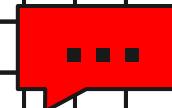
BONNES PRATIQUES DE PROGRAMMATION



Collaborer, documenter,
& tester son code



Du code efficace,
robuste & élégant



Plan du cours

7 OCTOBRE 2024 | 15h/17h
LES BASES DE GIT
ENVIRONNEMENT DE CODE

Introduction au fonctionnement et aux principales commandes Git
Familiarisation avec un écosystème de développement (IDE)
Pratique de la collaboration sur un *repository*

14 OCTOBRE 2024 | 15h/17h
COLLABORATION AVEC GITHUB
PULL REQUEST, ETC.

Aller plus loin avec Git et GitHub
Développement collaboratif et la mentalité *open-source*
Explorer, gérer et publier du code sur GitHub

9 DÉCEMBRE 2024 | 15h/17h
BONNES PRATIQUES DE CODE
TEST DRIVEN DEVELOPMENT

Comment rendre son code maintenable, agréable à relire,
debuggable facilement
Constitution de tests unitaires

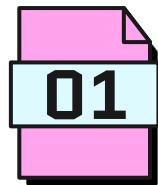
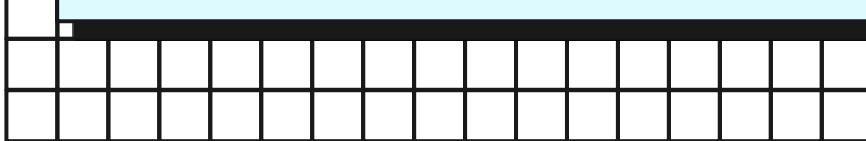
10 DÉCEMBRE 2024 | 15h/17h
MÉTHODOLOGIE DEVOPS
CI/CD AVEC TESTS AUTOMATIQUES

Gestion de projet informatique pour aboutir ses développements,
ne pas couler sous les *to-dos* et automatiser l'intégration de
son code



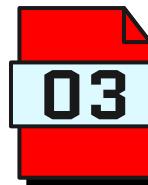
segolene.albouy[at]gmail.com

DÉROULÉ DE LA SÉANCE



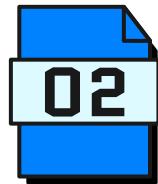
Git

Comment & pourquoi
utiliser le
versionning



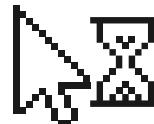
Github

Environnement de
collaboration



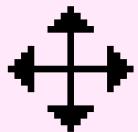
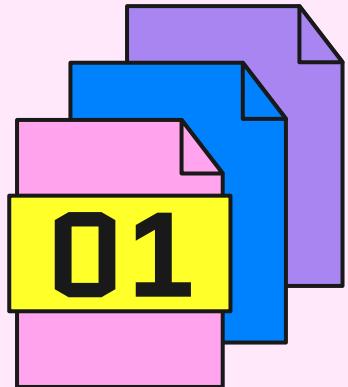
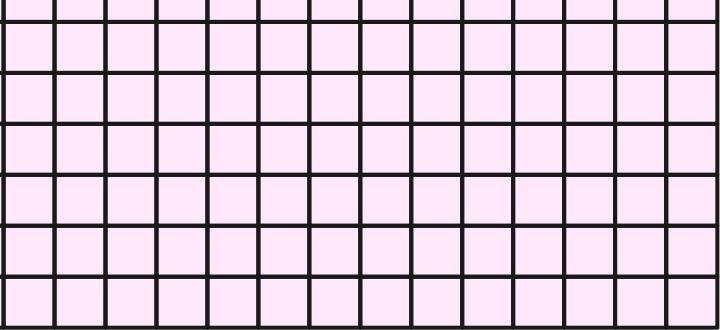
Install IDE

Prise en main d'un
logiciel de
développement



Exercice

Mise en pratique
des commandes

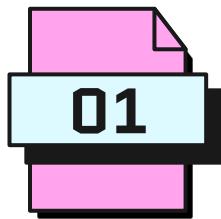


La magie de la gestion de version



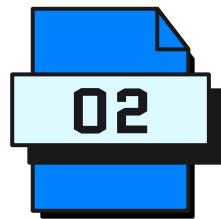
git est un outil de gestion de
versions décentralisé
créé par Linus Torvald
(le même que Linux) en 2005 sous
licence *open-source* GPL

À quoi ça sert le *versioning* ?



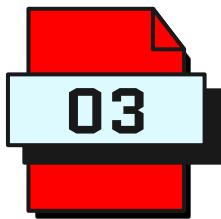
SAUVEGARDER DES VERSIONS

Git permet de créer des instantanés du contenu d'un dossier



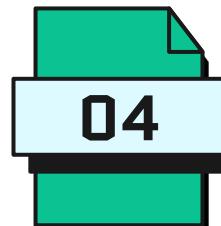
CONSERVER UN HISTORIQUE

En sauvegardant des versions successives, on a accès à la liste de toutes les modifications



CRÉER DES ÉTATS PARALLÈLES

Possibilité de disposer de plusieurs versions simultanées d'un dossier



COLLABORER SUR UN PROJET

Les modifications effectuées sur un même fichier peuvent être fusionnées



01

Sauvegarder des versions

Fini les fichiers aux noms pourris...



01

Sauvegarder des versions

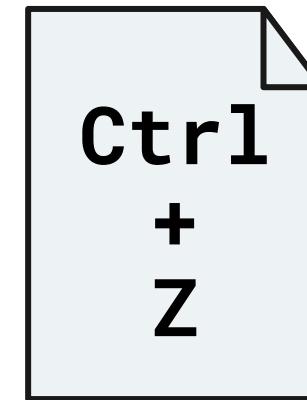
Avec git, en plus de sauvegarder normalement vos fichiers, vous pouvez décider **capturer des états** donnés d'un de vos dossiers.

Un dossier dont vous enregistrez des versions se nomme **repository**.

Une sauvegarde de ce repository est nommée **commit**.

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



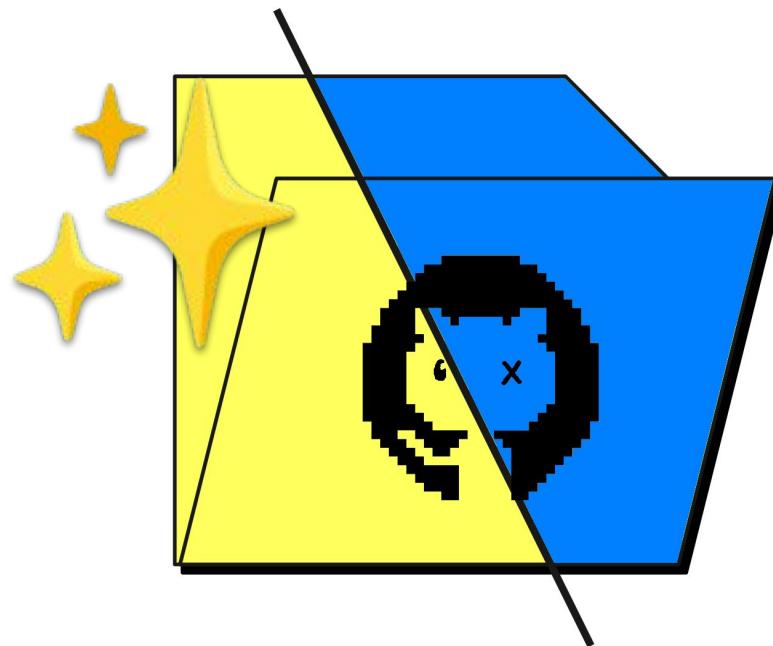
Chaque *commit* conserve en mémoire :

- sa date de création
- les lignes précises où les fichiers ont été édités
- un message expliquant le but des modifications
- la personne qui l'a effectué

En remontant les **commits** successifs, il est possible de suivre précisément l'évolution d'un **repository** et de revenir à un état précédent au besoin

03

Créer des états parallèles



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

03

Créer des états parallèles

Il est possible de faire coexister différentes versions des fichiers contenus dans un *repository* : ces versions sont appelées **branches**.

Les **branches** permettent de faire cohabiter de multiples états d'avancement d'un projet et de se lancer dans des modifications sans casser l'existant.

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

04

Collaborer sur un projet

Comme système de version décentralisé, git permet de :

- travailler sur un même projet avec plusieurs personnes, chacun depuis son ordinateur
- réconcilier les historiques pour mettre en commun les modifications.

En travaillant chacun sur sa **branche**, aucun risque de supprimer le travail de ses collaborateurs (ou de soi-même).

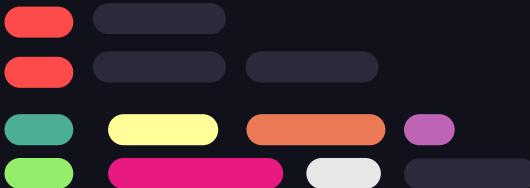
Les **branches** peuvent ensuite être fusionnées : on parle de **merge** ou de **rebase**.



01 { ..

Initialiser git

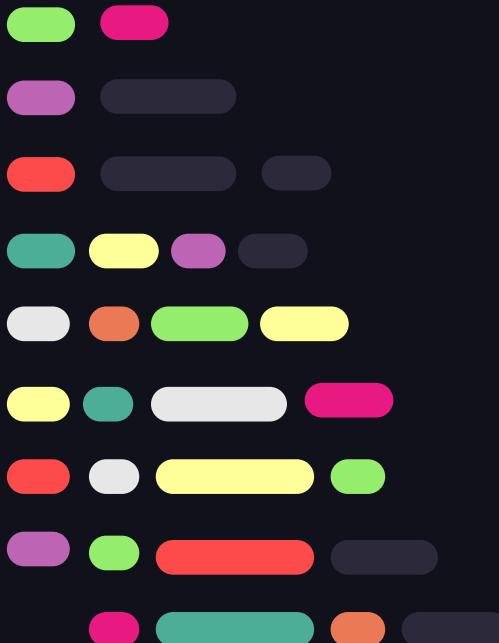
< le moment où on ouvre son terminal >



}

..

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 git



```
# Définition de l'utilisateur  
git config --global user.name "<github-user>"  
git config --global user.email "<github-email>"
```



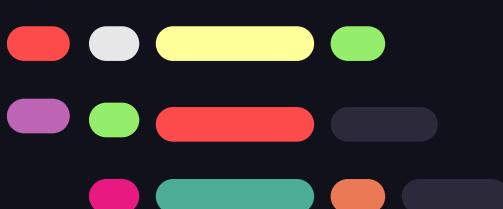
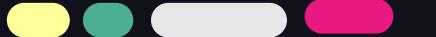
```
# Définition de l'éditeur de par défaut  
git config --global core.editor "nano"
```



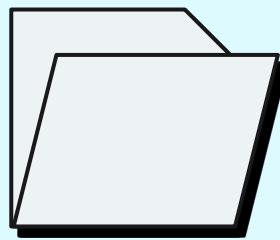
```
# Terminal avec les couleurs  
git config --global color.ui auto
```



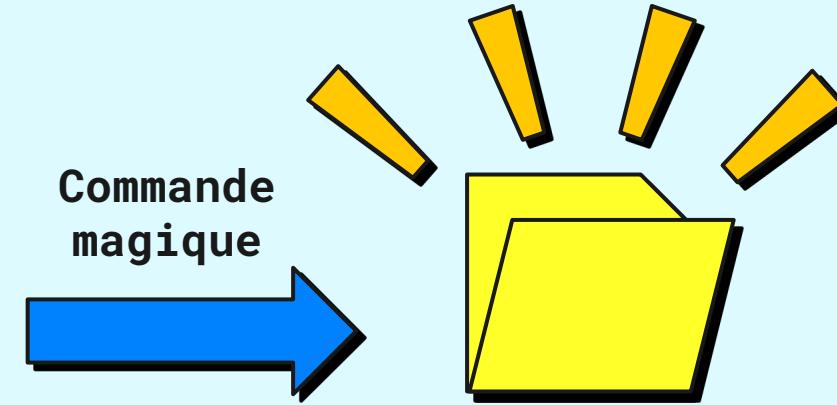
```
# Afficher vos configurations  
git config --list
```



INITIALISATION D'UN REPOSITORY

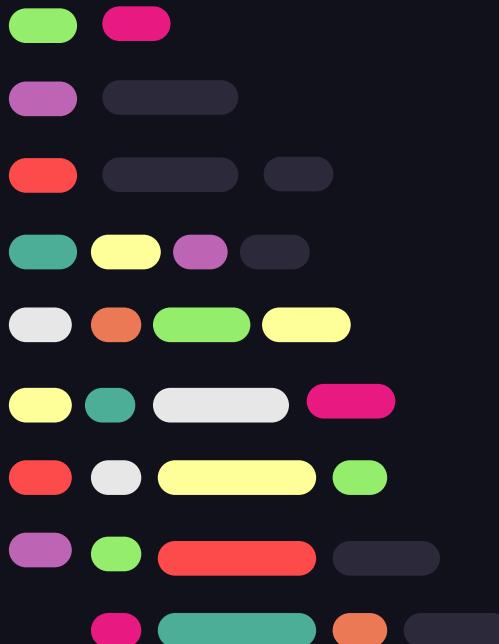


Création normale
d'un dossier



Transformation en
un *repository*

Initialisation d'un repository



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

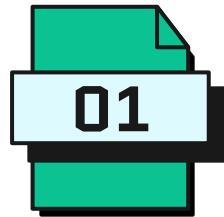
```
# Initialiser le repository  
git init
```

```
# Afficher le contenu du dossier  
ls -al
```

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

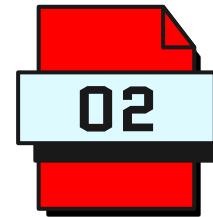
Sauvegarder un état du *repo*

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



CHOIX DES FICHIERS

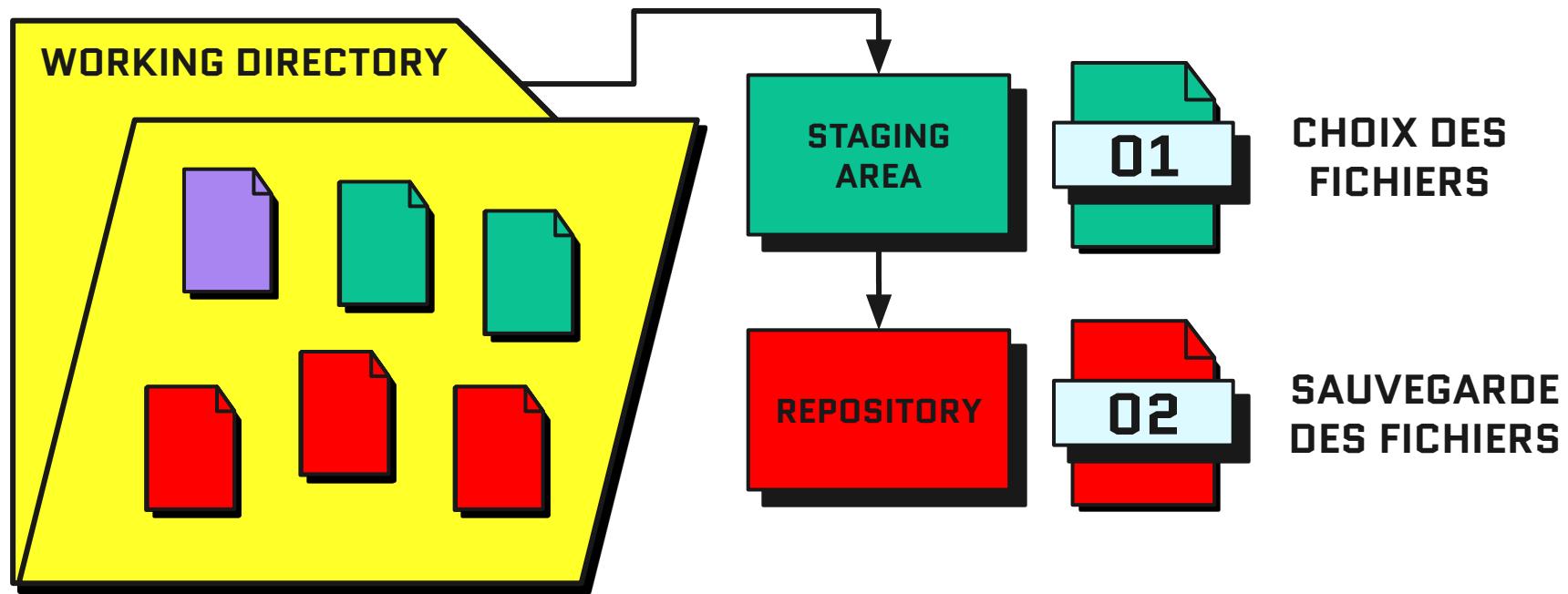
Sélection des fichiers à inclure dans le commit : ajout à la *staging area*



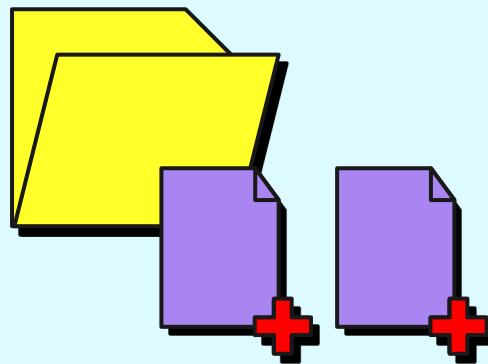
CRÉATION DE LA SAUVEGARDE

Enregistrement effectif de l'état des fichiers dans le commit

Processus de commit

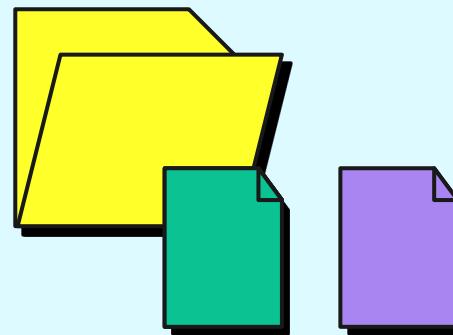
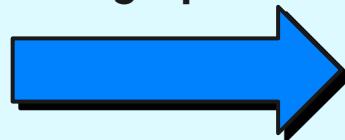


AJOUT DE FICHIERS À LA STAGING AREA



Création de
fichier(s) dans le
repository

Commande
magique



Ajout d'un fichier
dans la **staging**
area

Ajout à la *staging area*



```
# lister tous les fichiers modifiés  
git status
```



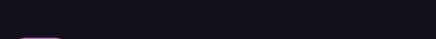
```
# Afficher les modifications  
git diff <path/to/filename>
```



```
# ajout d'un fichier précis  
git add <path/to/filename>
```



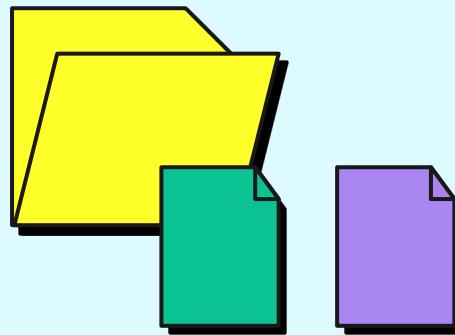
```
# ajout de tous les fichiers d'un dossier  
git add <path/to/directory>
```



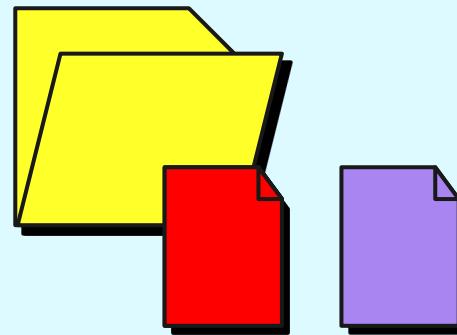
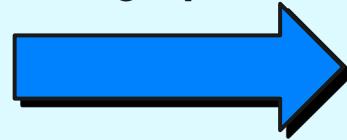
```
# ajout de tous les fichiers du repo  
git add -a
```



CRÉATION D'UN COMMIT



Commande
magique



Certains fichiers
ont été “**addés**”

Les fichiers
“**addés**” seulement
sont “**commités**”

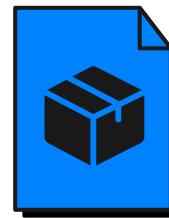
Réussir ses commits

Pour un bon commit :



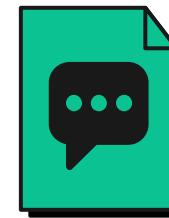
LE BON MOMENT

Committer lorsqu'on a achevé une modification significative.
Privilégier les petits commits.



LE BON CONTENU

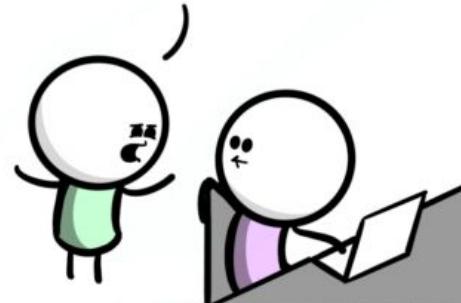
Ne commiter que les fichiers pertinents, si on entamé plusieurs chantiers, les commiter un par un



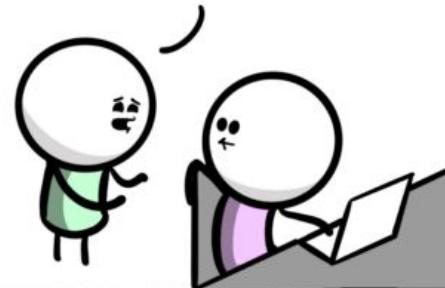
LE BON MESSAGE

Chaque commit est documenté avec un message résumant les modifications : il doit être explicite

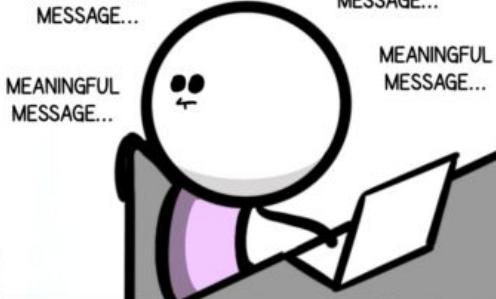
BOB, ALL YOUR
COMMIT MESSAGES SAY
"SMALL CHANGES!"



NEXT TIME, PLEASE
WRITE A MEANINGFUL
MESSAGE



MEANINGFUL
MESSAGE...
MEANINGFUL
MESSAGE...
MEANINGFUL
MESSAGE...
MEANINGFUL
MESSAGE...

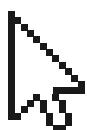


GIT COMMIT -M "WHAT IS LIFE IF
FULL OF CARE. IF WE HAVE NO
TIME TO STAND AND STARE..."



Bon message de commit ?

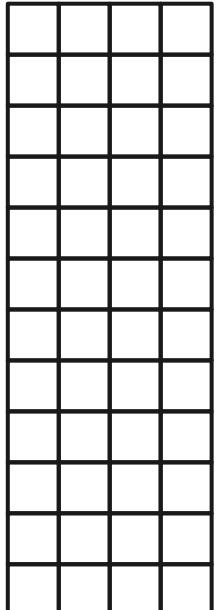
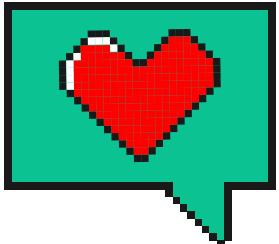
Modification du fichier de config		X	
[FIX] correction du formulaire de recherche	✓		
Ajout d'un nouveau champ descriptif correspondant à l'adresse postale de l'entité personne dans la base de données		X	
Clean files and rename		X	
[FEAT] Create task for image file post-processing	✓		
Correct typo in index title	✓	X	



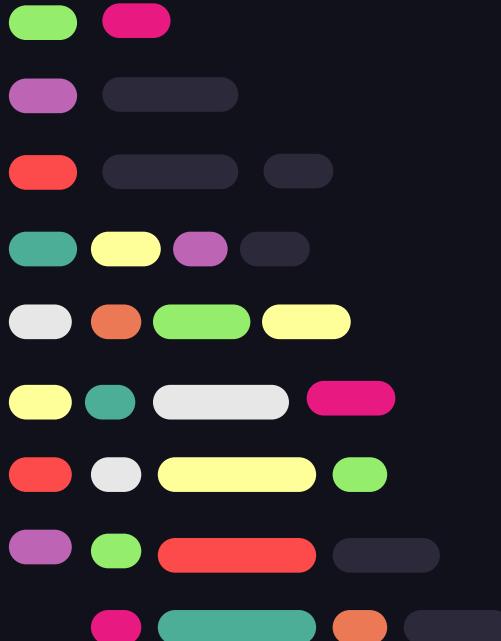
[type] (scope) description

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

- **type**: la catégorie de modification (*fix, feat, docs, refactor, 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)



Création d'un *commit*



```
# commit des fichiers de la staging area  
git commit  
  
# commit avec message  
git commit -m "<commit-message>"
```

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

```
UW PICO 5.09      File: /Users/seglinglin/Desktop/dossier sans titre/.git/COMMIT_EDITMSG

#
# 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 master
#
# Validation initiale
#
# Modifications qui seront validées :

^G Get Help      ^O WriteOut     ^R Read File    ^Y Prev Pg      ^K Cut Text     ^C Cur Pos
^X Exit          ^J Justify      ^W Where is     ^V Next Pg      ^U UnCut Text   ^T To Spell
```

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

```
■ # 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  
■ #  
■ ~  
■ ~  
■ ~  
■ ~
```

C'est normal
de paniquer la
première fois

Pour sauver
:x!

Pour sortir
:q!

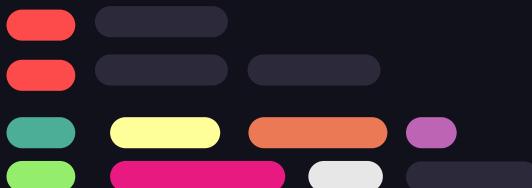


02 { ..

Créer un *repository*

Ajouter des fichiers

Effectuer 3 *commits*



}

..



..

Dossier

Créer un dossier et l'ouvrir dans un terminal

Diff

Lister les modifs avec git status et git diff



Init

Initialiser un *repository* dans le dossier

Add

Ajouter un/des fichiers à la *staging area*

Fichier

Créer un ou plusieurs fichiers

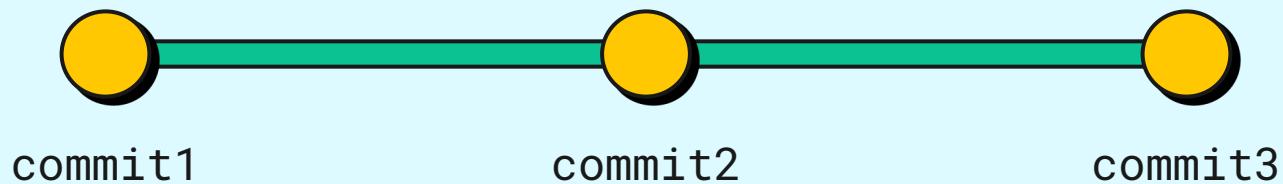
Commit

Committer les fichiers addés, recommencer 3 fois



..

HISTORIQUE DES COMMITS



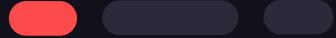
Visualiser l'historique



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



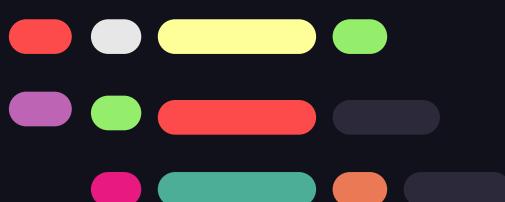
```
# historique compact (récent en haut)  
git log -oneline
```



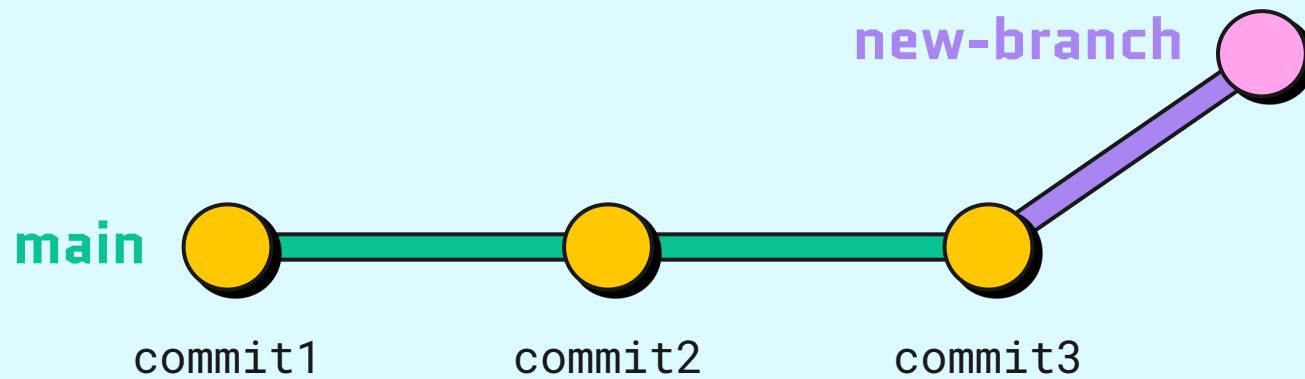
```
# graphe de l'historique  
git log -oneline --graph
```



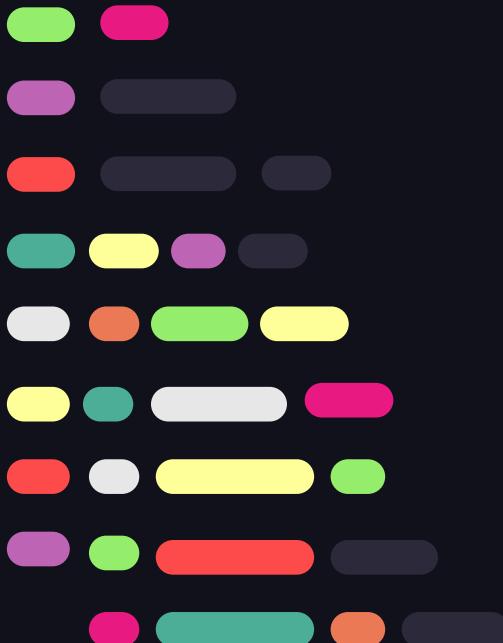
```
# quitter avec q
```



CRÉATION DE BRANCHE



Création d'une branche



```
# afficher 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>
```

Quand créer une branche

À quel moment est-il adéquat de créer une branche ?



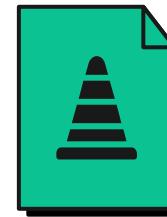
FEATURE

Vous avez une fonctionnalité à développer mais ne voulez pas intervenir sur le code principal



COLLABORATION

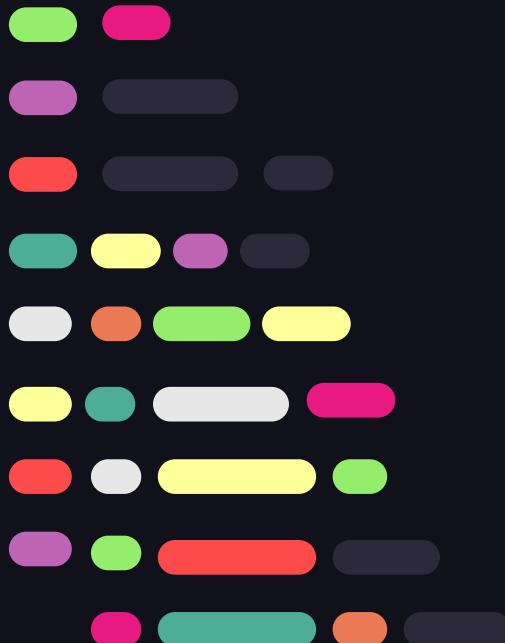
Une branche par personne pour ne pas se marcher sur les pieds, à update régulièrement avec main



TEST

Pour tenter des trucs sans risquer de tout casser, n'a pas forcément vocation à être intégrée

Afficher la branche courante

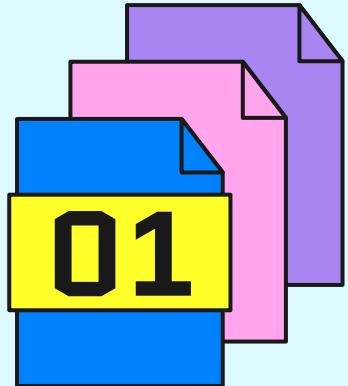
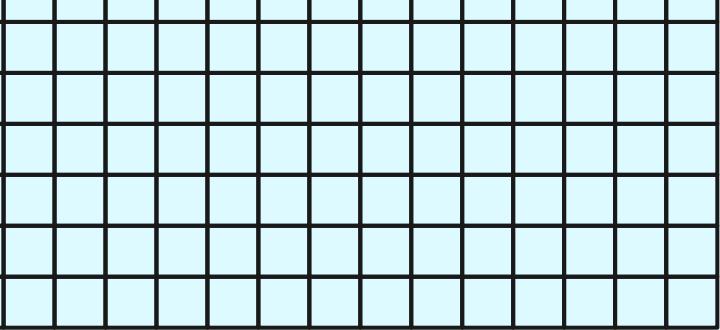


```
# ouvrir le fichier de config du terminal  
nano ~/.bashrc
```

```
# coller le contenu suivant à la fin du fichier  
⇒ EXAMPLE .BASHRC
```

```
# enregistrer et quitter avec Ctrl+S puis Ctrl+X
```

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



UTILISER UN IDE



Prendre en main un environnement de dev

Choisir son IDE



PyCharm

Version *community* gratuite
Licence complète avec ENC

Édité par JetBrains

Comprend de base de
nombreuses fonctionnalités



VS Code

Gratuit et partiellement
open source

Édité par Microsoft

Modulable à l'infini avec les
extensions de la communauté

VS

Obtenir licence étudiant JetBrains

Création d'un compte avec son adresse email de l'École pour avoir accès à toute la suite JetBrains

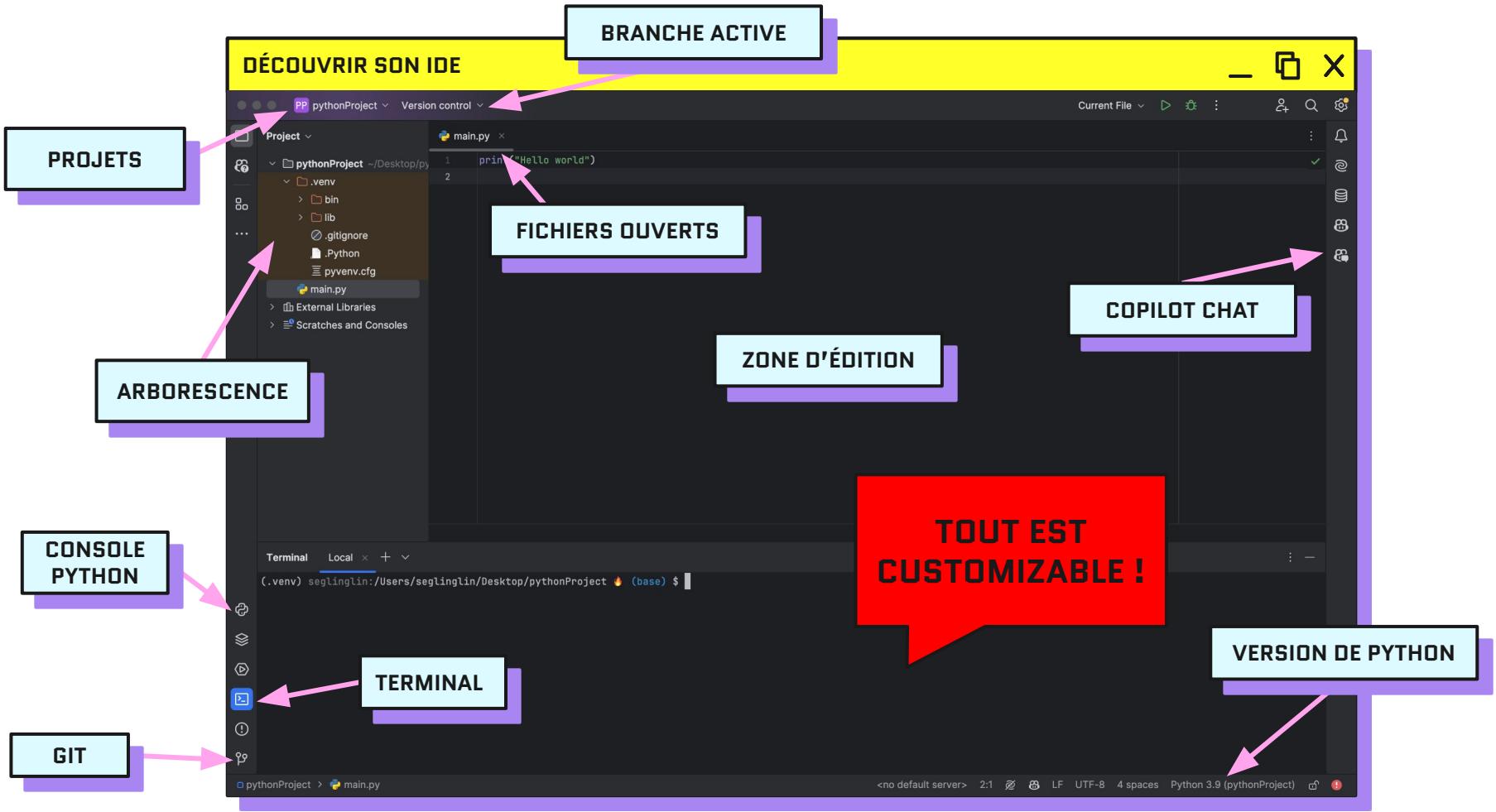
jetbrains.com/shop/eform/students

Télécharger la Toolbox pour une installation facile

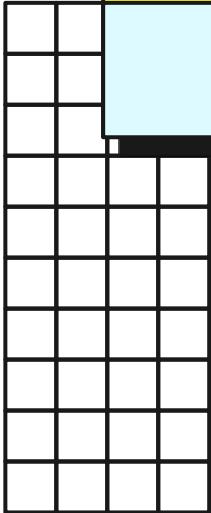
jetbrains.com/toolbox-app

The screenshot shows a web-based application for obtaining a student license. At the top, there's a navigation bar with tabs: "S'inscrire avec" (Inscription with), "E-mail de l'université" (University email) which is selected and underlined, "Carte ISIC/ITIC", "Document officiel", and "GitHub". Below the tabs, there are several input fields and dropdown menus:

- Situation :** A radio button group where the first option, "Je suis étudiant-e", is selected.
- Pays/Région :** A dropdown menu showing "France".
- Niveau d'études :** A dropdown menu showing "Étudiant universitaire".
- L'informatique ou l'ingénierie est-elle votre principal domaine d'étude ?** A radio button group where the first option, "Oui", is selected.
- Adresse e-mail :** An input field containing "Adresse e-mail de l'université, par exemple : js@mit.edu". Below it is a small text box stating: "Je confirme que l'adresse e-mail de l'université indiquée ci-dessus est valide et qu'elle m'appartient."
- Nom :** A text area containing "Votre nom complet, tel qu'il figure sur votre passeport, votre carte d'identité, votre permis de conduire ou tout autre document légal." Below this are two input fields: "Prénom" and "Nom".

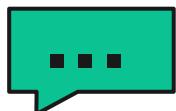


Mes raccourcis favoris



Ctrl+?

Sélectionner
la prochaine
occurrence



Ctrl+?

Supprimer une
ligne entière



Ctrl+?

Commenter le
texte
sélectionné



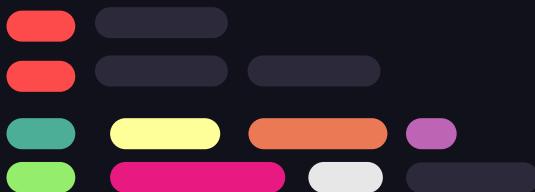
Ctrl+?

Rechercher
dans tout le
code



03 { ..

Ouvrir son *repository*
dans son IDE
Modifier le contenu



}

..



{

..

Repository

Ouvrir votre
repository depuis
votre IDE

Add/commit

Ajouter ces
modifications dans
la nouvelle branche



Branche

Créer une branche
depuis le
terminal de l'IDE

Main

Revenir à la
branche principale
et afficher les
logs

Dossier

Créer un sous-dossier
avec des fichiers
dans le *repo*

Repository

Observer
l'arborescence de
fichiers

}

..

La fusion : 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

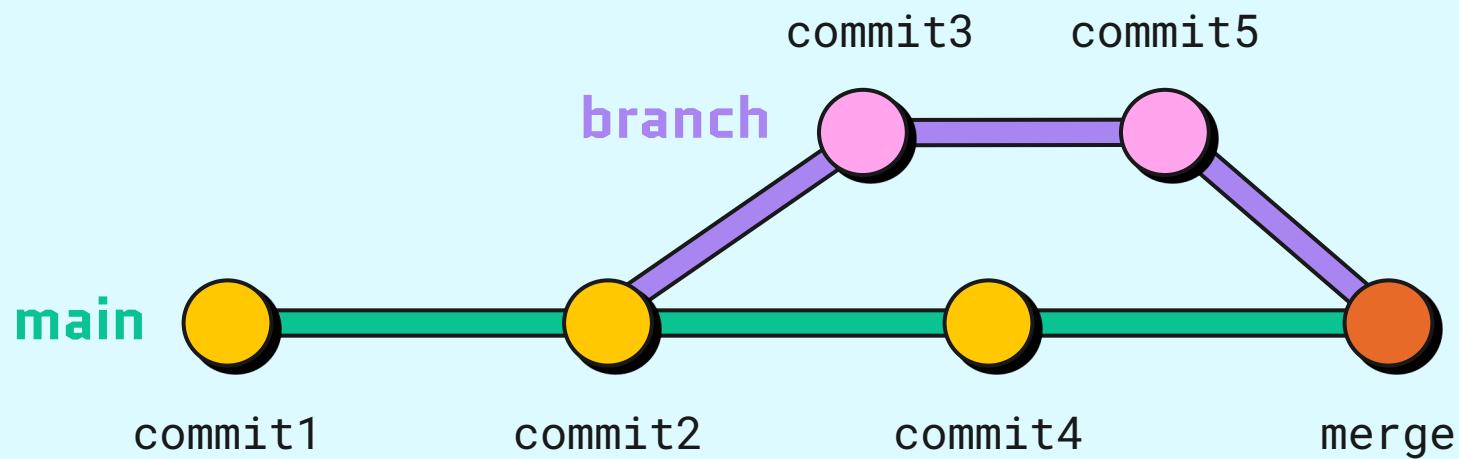
Rebase

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

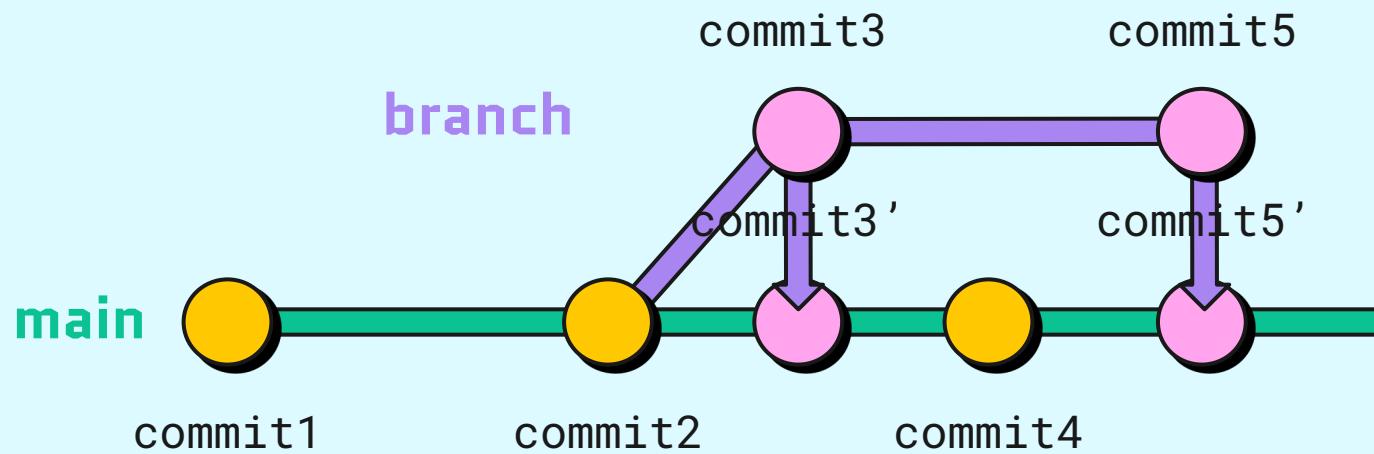
Résolution des conflits **commit après commit**

L'historique des commits est **linéaire**

MERGE DE BRANCHES



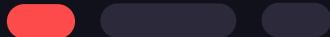
REBASE DE BRANCHES



Fusionner des branches



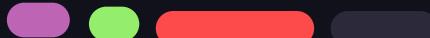
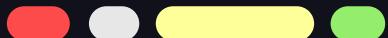
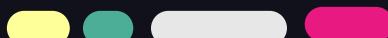
```
# merge de main dans my-branch  
[my-branch] git merge main
```



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



de manière générale, privilégiez **merge**



!! À retenir

Quand on veut intégrer sa branche dans une autre :

1. depuis sa branche, on merge/rebase l'autre branche
2. depuis l'autre branche, on merge sa branche

Cas classique

```
# on fusionne main
/my/repo [my-branch] git merge main
# ou
/my/repo [my-branch] git rebase main

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

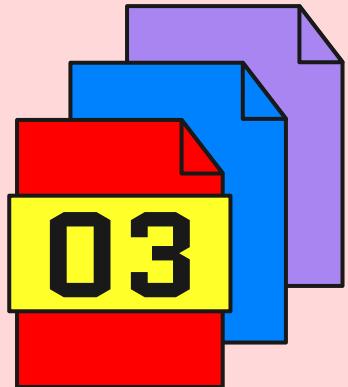
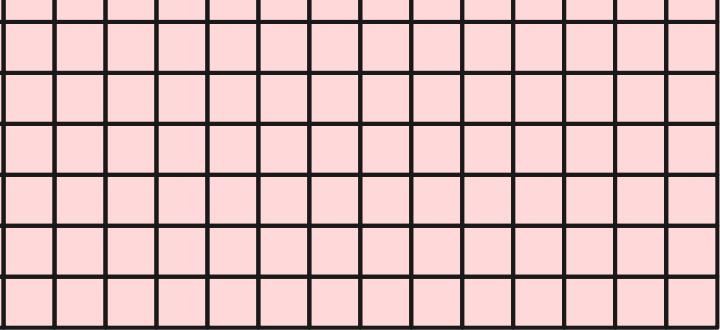
# on MERGE my-branch
/my/repo [main] git merge my-branch

# ! on rebase jamais sur une branche
# ! où on travaille à plusieurs
```

Conflits

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





La puissance de git en ligne



github est un service en
ligne pour héberger son
repository

Plateformes en ligne



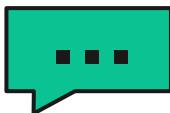
La plus répandue,
sert de portfolio
aux développeurs



Principale alternative,
peut être hébergé sur
des serveurs privés



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



git / github

git

Gestionnaire de versions

Outil pour effectuer des modifications sur un *repository*

Ensemble de commandes à exécuter dans son terminal

VS

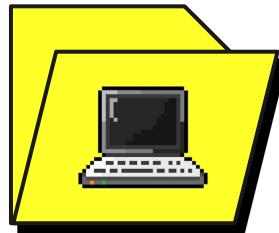
GitHub

Service d'hébergement en ligne

Là où stocker et publier ses *repositories*

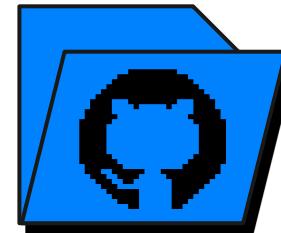
Plateforme web avec interface pour gérer/exposer ses *repo*

Dépôt local ou distant



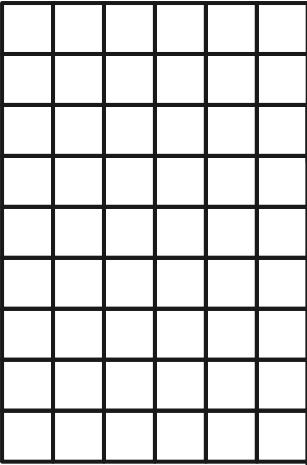
Local

Dossier contenant le *repo*,
situé sur votre ordinateur

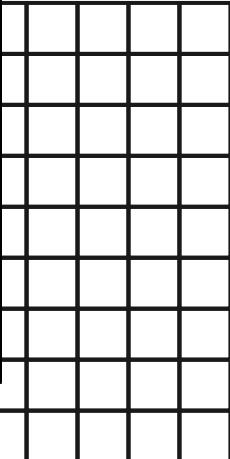


Remote

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



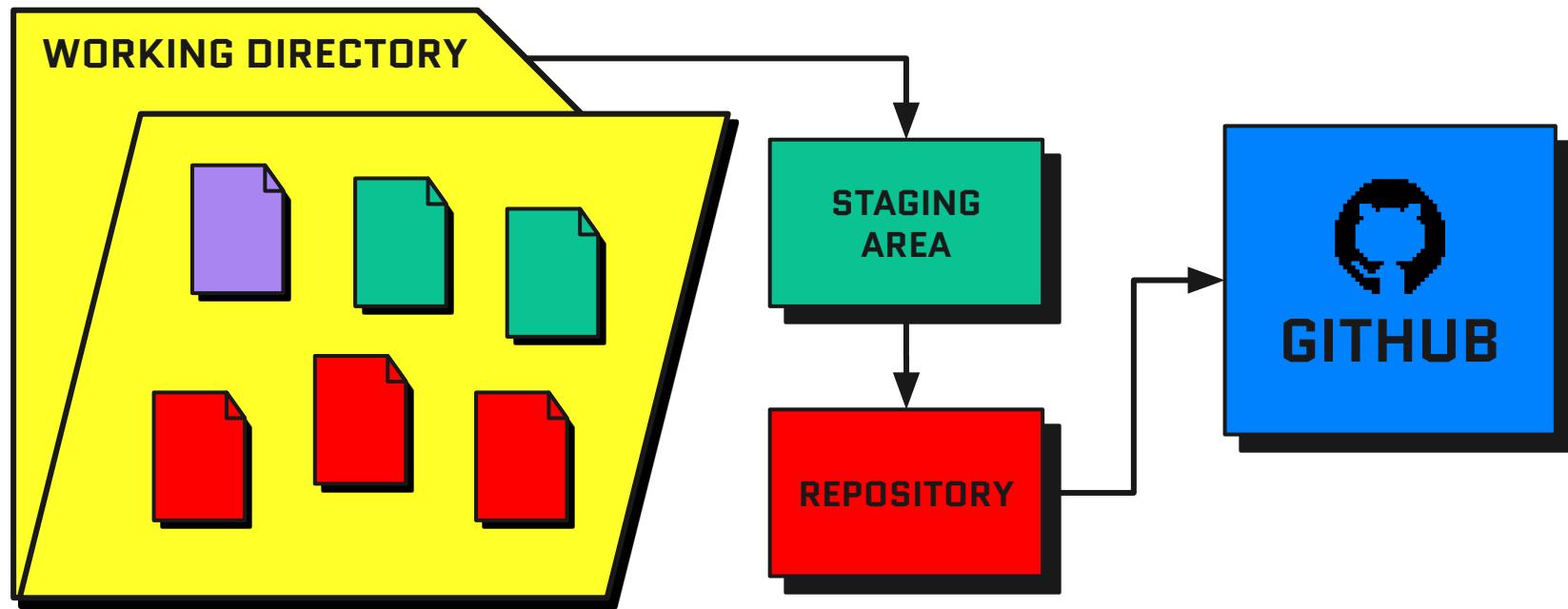
Remote repository



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.



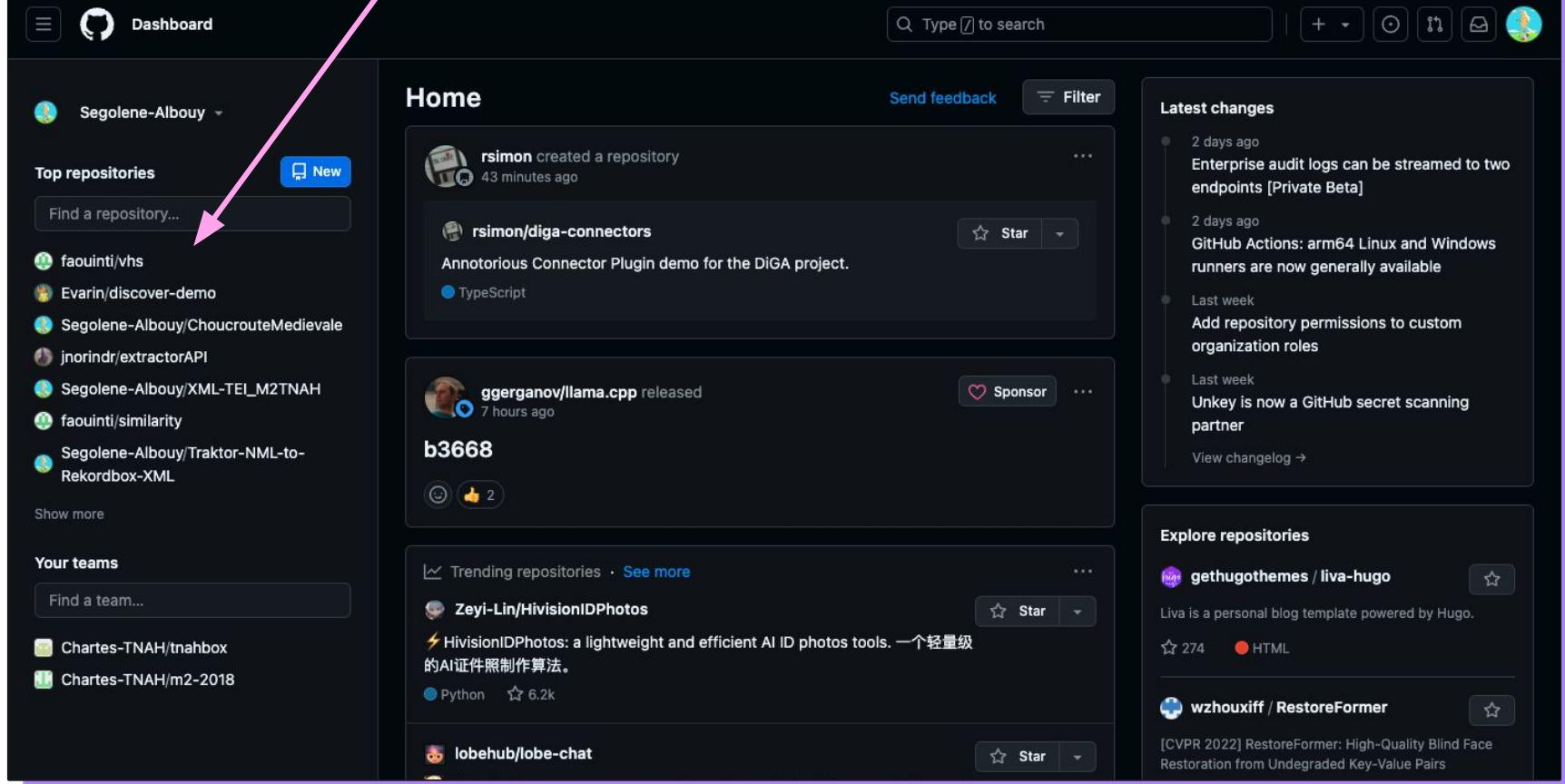
Processus de publication



VOS REPOSITORIES

RECHERCHE

Page d'accueil



Page de profil

Overview Repositories 37 Projects Packages Stars 83



Ségalène Albouy

Ségalène-Albouy · she/her

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

Edit profile

40 followers · 37 following

CNRS

Paris

<https://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

XML-TEI_M2TNAH Public

Ressources et présentations pour le cours en XML-TEI des M2 TNAH de l'École des chartes

HTML ⭐ 8 📂 8

kanon Public

Forked from ALFA-project-erc/kanon

Python

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



Public

Pin

Unwatch 1

Fork 0

Star 0

main

1 Branch 0 Tags

Go to file

t

+

Code

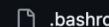
README



Segolene-Albouy [EXAMPLE] Add documentation to functions

c3bdb90 · 3 hours ago

3 Commits



[EXAMPLE] Add documentation to functions

3 hours ago



Initial commit

last month



Initial commit

last month



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

LICENCE

1 watching

0 forks

Releases

No releases published

Create a new release

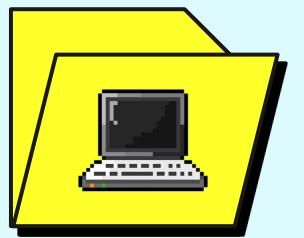
Packages

No packages published

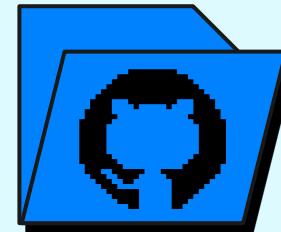
Publish your first package

Contributors

DÉPOSER SON CODE SUR LE DÉPÔT DISTANT

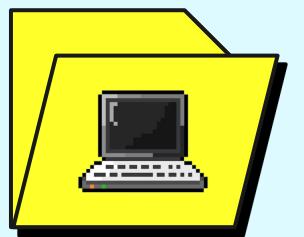


Local

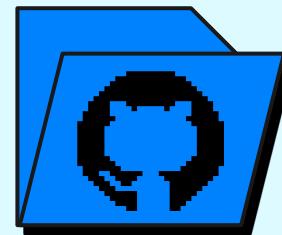


Remote

RÉCUPÉRER DU CODE DEPUIS LE DÉPÔT DISTANT



Local



Remote

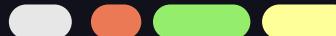
Publier son code



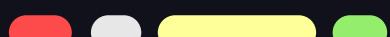
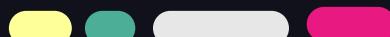
```
# déposer son code local sur GitHub  
git push
```



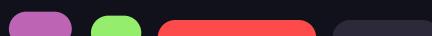
```
# récupérer du code depuis GitHub  
git pull
```



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



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



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 pull.rebase false
```

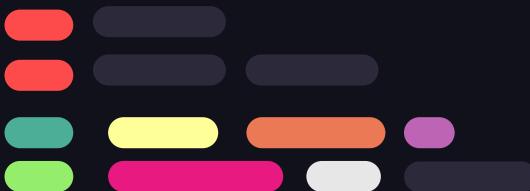
```
# git pull opère un rebase des branches locale et remote  
git config pull.rebase true
```

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



04 { ..

Publier son *repository*
sur GitHub



}

..

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 <nom> <url>

et poussez alors en utilisant le dépôt distant

git push <nom>

New repository

+ ▾Import a repository+ImportIssuesPull requestsCommitsFileProfile

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 * Repository name *

Segolene-Albouy /

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:

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

Choose a license

License:

NOUVEAU

Lier son dépôt local à un remote



GIT-M2TNAH Public

main · 1 Branch · 0 Tags

Segolene-Albouy [EXAMPLE] Add documentation to functions · c3bdb90 · 2 hours ago

.bashrc
LICENSE
README.md

README MIT license

GIT-M2TNAH

Cours de Git et bonnes pratiques de dé

Go to file · Add file · Code

Local Codespaces Copilot

Clone

HTTPS SSH GitHub CLI

git@github.com:Segolene-Albouy/GIT-M2TNAH.! · Copy url to clipboard

Use a password-protected SSH key.

Open with GitHub Desktop

Download ZIP

Pin · Unwatch 1 · Fork 0 · Star 0

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

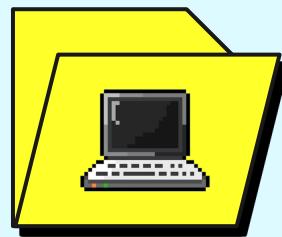
No packages published
[Publish your first package](#)

Languages

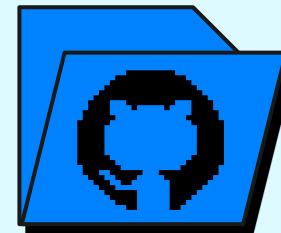
Shell 100.0%

A blue callout box highlights the "Code" dropdown menu at the top right. A red number "1" is placed over the "Code" button. A second blue callout box highlights the "Copy url to clipboard" button in the "Clone" section. A red number "2" is placed over this button. A yellow border surrounds the entire highlighted area.

LIER SON DOSSIER LOCAL À UN REMOTE REPOSITORY

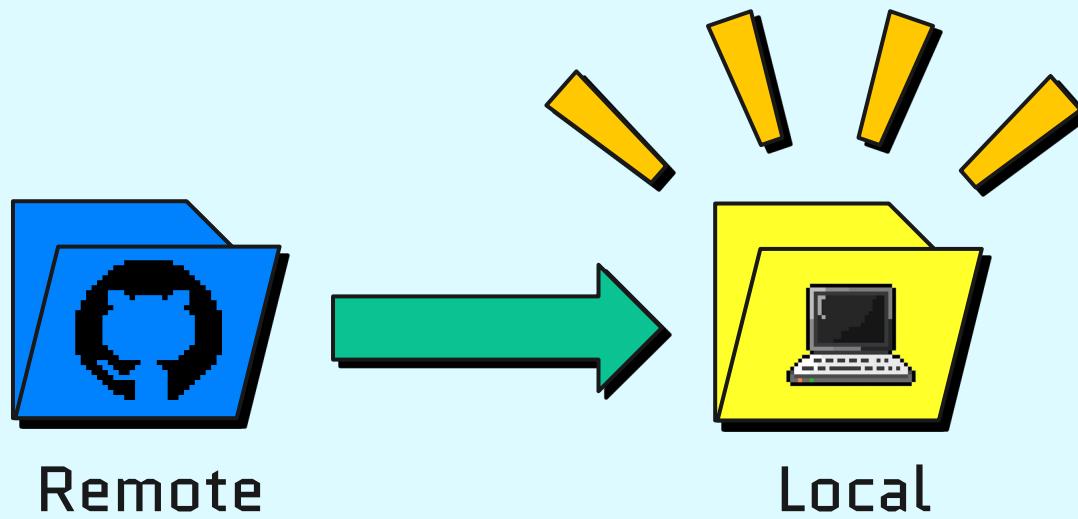


Local



Remote

COPIER UN REPOSITORY SUR GITHUB SUR SON ORDINATEUR



Lier un remote



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



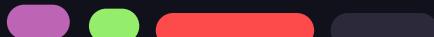
```
# Copier un repo distant sur son ordinateur  
git clone <url-repo-distant>
```



```
# De manière générale, on privilégie  
# de cloner un repo distant en local
```



```
# Cloner le repository du cours
```



```
git clone git@github.com:Segolene-Albouy/GIT-M2TNAH.git
```



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. Nommer et coller la clef publique

Danger zone



Code Issues Pull requests Actions Projects Wiki Security Insights Settings

General

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



Danger Zone

Change visibility

Change repository visibility

This repository is currently public.

Disable branch protection rules

Disable branch protection rules

Disable branch protection rules enforcement and APIs

Transfer

Transfer ownership

Transfer this repository to another user or to an organization where you have the ability to create repositories.

Archive this repository

Archive this repository

Mark this repository as archived and read-only.

Delete this repository

Delete this repository

Once you delete a repository, there is no going back. Please be certain.

Ajouter un collaborateur



Code Issues Pull requests Actions Projects Wiki Security Insights Settings

General

Who has access

Access

Collaborators

Moderation options

Code and automation

Branches

Tags

Rules

Actions

Webhooks

Environments

Codespaces

Pages

Security

Code security

Deploy keys

Secrets and variables

Integrations



Public repository

This repository is public and visible to anyone

Manage



PUBLIC REPOSITORY

This repository is public and visible to anyone.

Manage

DIRECT ACCESS

0 collaborators have access to this repository. Only you can contribute to this repository.



Manage access

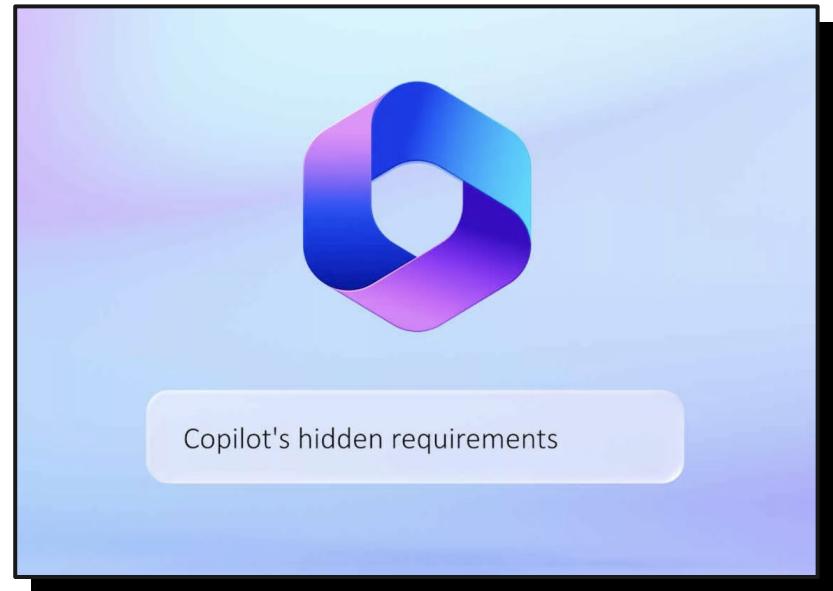


You haven't invited any collaborators yet

Add people

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](https://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
education.github.com/discount_requests/application
2. Prendre en photo sa carte étudiant
3. Attendre la validation
4. Installer le plugin copilot sur son IDE
5. Profiter de copilot gratuitement !



Home / Benefits application

Access free GitHub Education benefits

Complete the fields below to unlock tools and resources for your educational journey

Select your role in education *

Teacher Student School

Enhance your tech skills with real-world tools

STUDENT

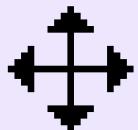
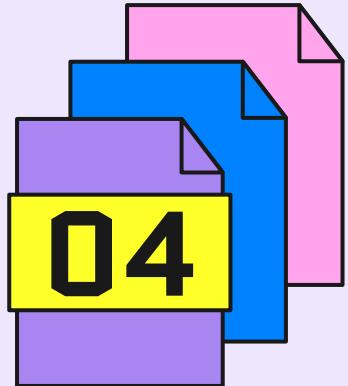
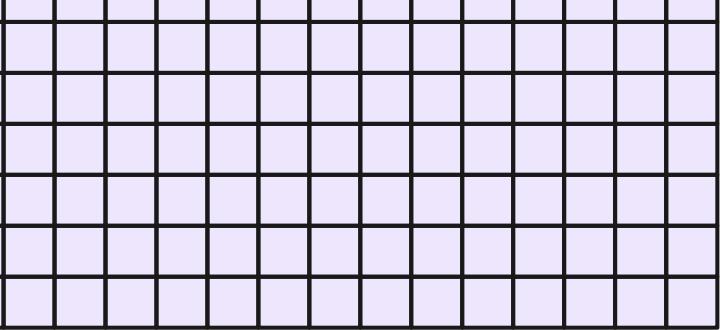
Hub Pro while you are a

STUDENT

Valuable GitHub Student Developer Pack partner offers

STUDENT

GitHub Campus Expert training for qualified applicants

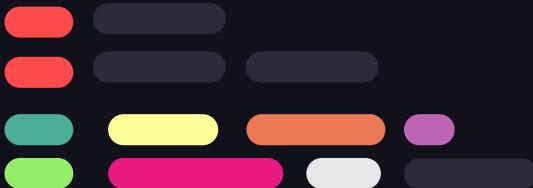


La puissance de git en ligne



05 { *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

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



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
```



{ .. Partie 2

Branche

Après avoir *pull*,
chacun crée une
branche en local

Modif

Chacun modifie
les mêmes bouts
de fichiers

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
sur 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
avec git add et
git commit



Trouver les conflits



Ctrl+?

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

Commit après résolution

```
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
```

Après avoir *add* les fichiers où des conflits ont été résolus à la *staging area*,

git génère un message de commit automatique.

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