

# Formation git

→ 28 novembre 2024

**onepoint.**  
beyond the obvious

# Qui sommes-nous ?



**Capucine BOIS**

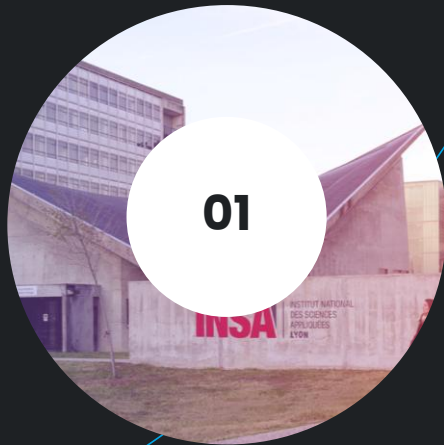
Dev fullstack  
Associate Onepoint  
[c.bois@groupeonepoint.com](mailto:c.bois@groupeonepoint.com)



**Rahul RAMSAHA**

Dev backend  
Associate Onepoint  
[rr.ramsaha@groupeonepoint.com](mailto:rr.ramsaha@groupeonepoint.com)

# Pourquoi cette formation ?



Pour vos projets de 3A  
4A, 5A à l' INSA



Pour vos stages



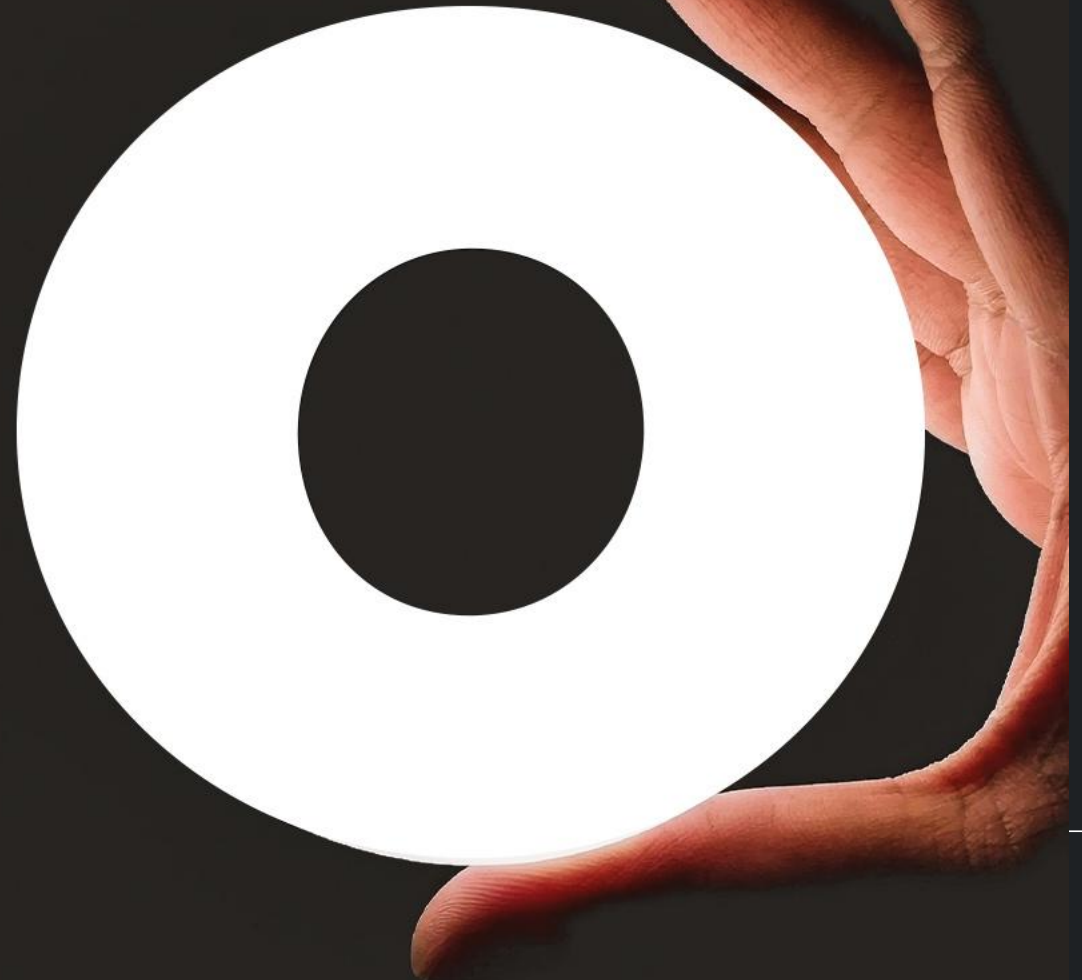
Plus tard en  
entreprise...

# Sommaire

1. Git 101
2. Git 201
3. Bonus

# 1. Git 101

- Les fondamentaux de git
- Les commandes de bases
- Comment collaborer sur git ?





# Qu'est-ce que git ?

**Git** est un logiciel de **gestion de versions** **open source**

Débute en 2005 et est maintenant implémenté sur des plateformes en ligne comme **Github**

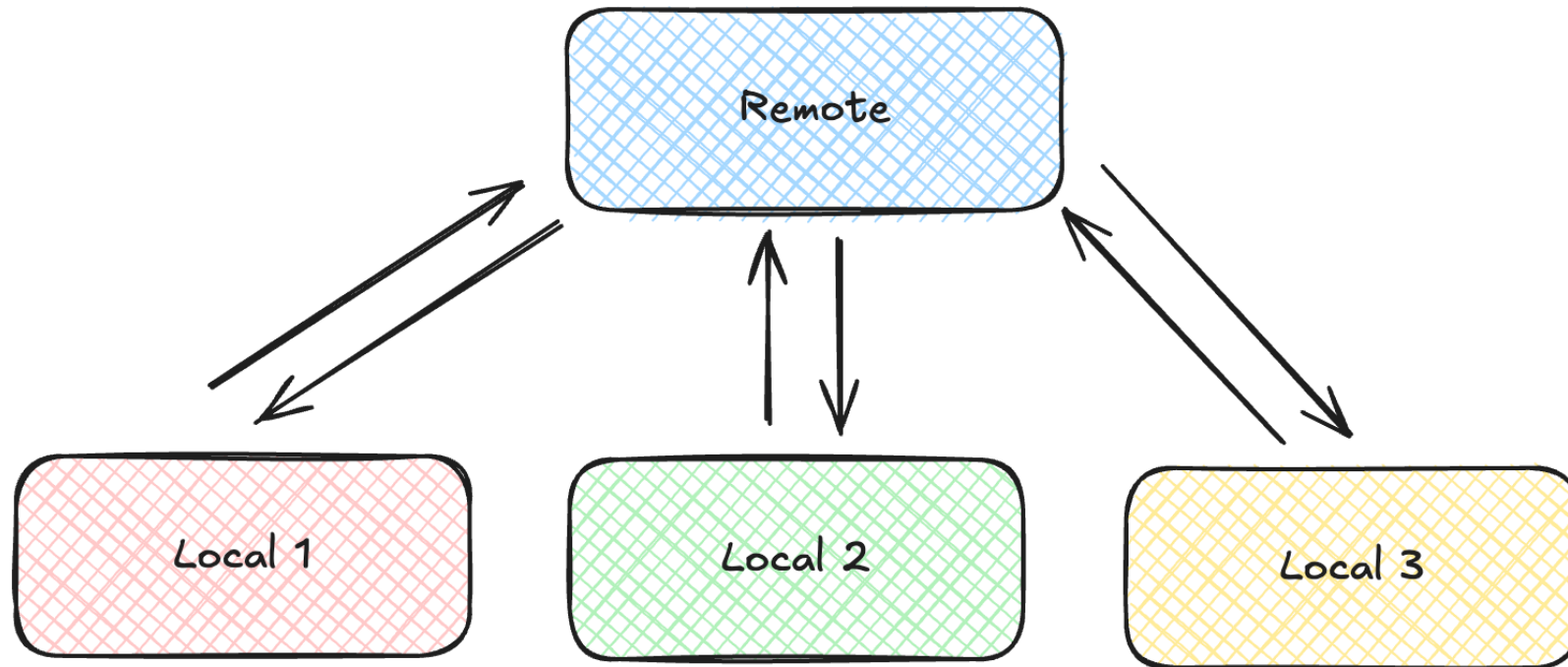
Garde un historique de toutes versions du code, des changements et de ceux les ayant effectués

Tout le monde y a accès et peut contribuer au développement



# L'intérêt de git

- Pouvoir collaborer à plusieurs sur le même projet
- Conserver un historique de l'état d'un projet



# Installer git

Linux : `apt-get install git`

Mac OS : `brew install git`

Windows : [Installer git bash](#)

```
MINGW64/c/Users/me/git
me@work MINGW64 ~
$ git clone https://github.com/git-for-windows/git
Cloning into 'git'...
remote: Enumerating objects: 500937, done.
remote: Counting objects: 100% (3486/3486), done.
remote: Compressing objects: 100% (1415/1415), done.
remote: Total 500937 (delta 2494), reused 2917 (delta 2071), pack-reused 497451
Receiving objects: 100% (500937/500937), 221.14 MiB | 1.86 MiB/s, done.
Resolving deltas: 100% (362274/362274), done.
Updating files: 100% (4031/4031), done.

me@work MINGW64 ~
$ cd git

me@work MINGW64 ~/git (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean

me@work MINGW64 ~/git (main)
$ |
```

```
blog-11ty — andreaverlicchi@Mac — ..ode/blog-11ty — zsh — 105x26
➔ Code cd blog-11ty
➔ blog-11ty git:(main) ✗ clear

➔ blog-11ty git:(main) ✗ git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   content/blog/how-integrate-git-autocomplete-mac-shell-terminal-bash-zsh/index.md

no changes added to commit (use "git add" and/or "git commit -a")
➔ blog-11ty git:(main) ✗ |
```



# Set up de git pour un projet

## ● git init

Sert à créer un context git en local au sein d'un répertoire voulu

## ● git clone <url du repo>

Sert à récupérer un projet git et son historique depuis une plateforme comme github

# Comment sauvegarder son code ?

1 git add .

Sélectionner les fichiers à mettre à jour.  
"." signifie qu'on prend tout le dossier en excluant les fichiers spécifiés dans le .gitignore

2 git commit -m "message"

On crée une nouvelle version avec les fichiers modifiés accompagné d'un message explicatif

3 git push

On "pousse" la nouvelle version sur le répertoire Github en ligne

# Commandes utiles

- **git clone :** On fait une copie du répertoire sur son ordinateur
- **git pull :** Git Pull permet de télécharger le contenu d'un dépôt distant pour mettre à jour son dépôt local. C'est le meilleur moyen de vous assurer **de travailler sur la version actuelle.**
- **git merge / rebase :** Commandes qui permettent de fusionner ses travaux avec des travaux distants

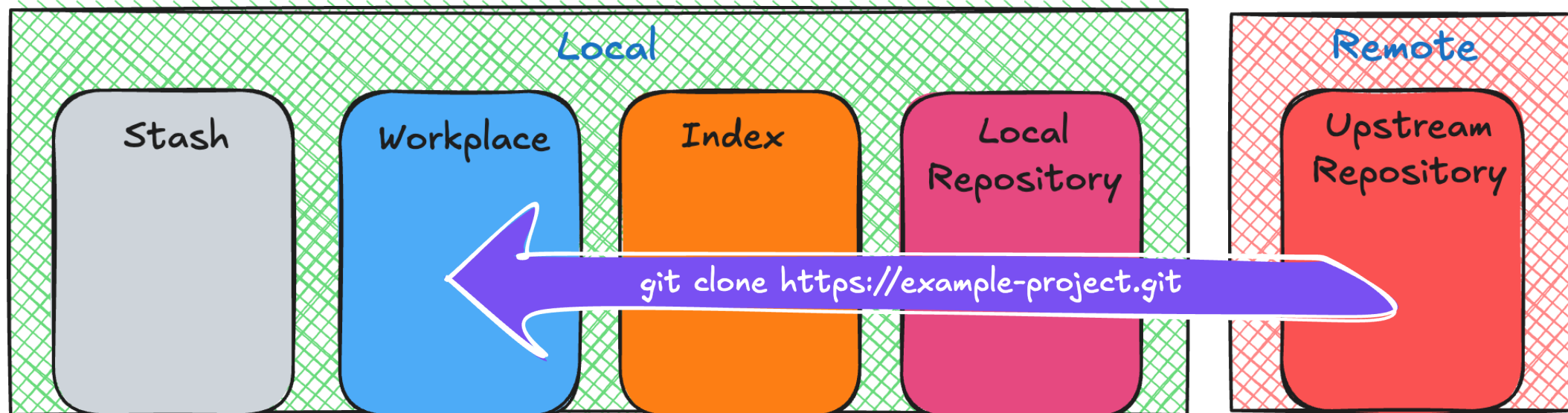
# Commandes utiles

- **git switch**  
**<branch>** Permet de créer des branches / basculer d'une branche à l'autre
- **git switch -c**  
**<branch>** Permet de créer une nouvelle branche en partant de la branche où on est
- **git stash /**  
**pop :** Stash : mettre de côté ses changements  
Pop : récupérer les changements qu'on a précédemment mis de côté



# Git en images

```
~ git clone <lien du repo>
```

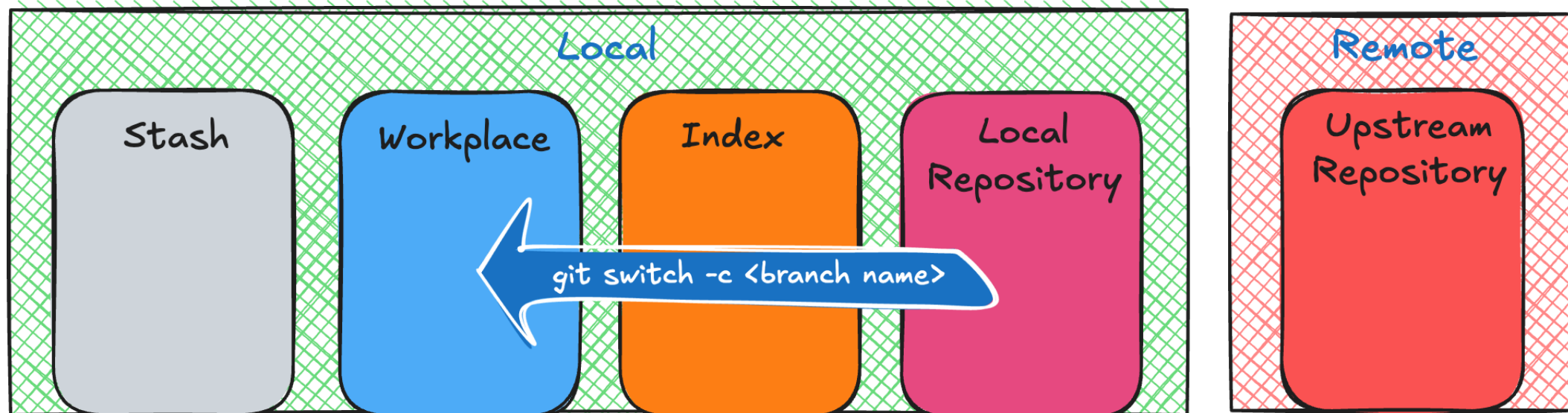




# Git en images

```
~ git switch -c <nom de la branche>
```

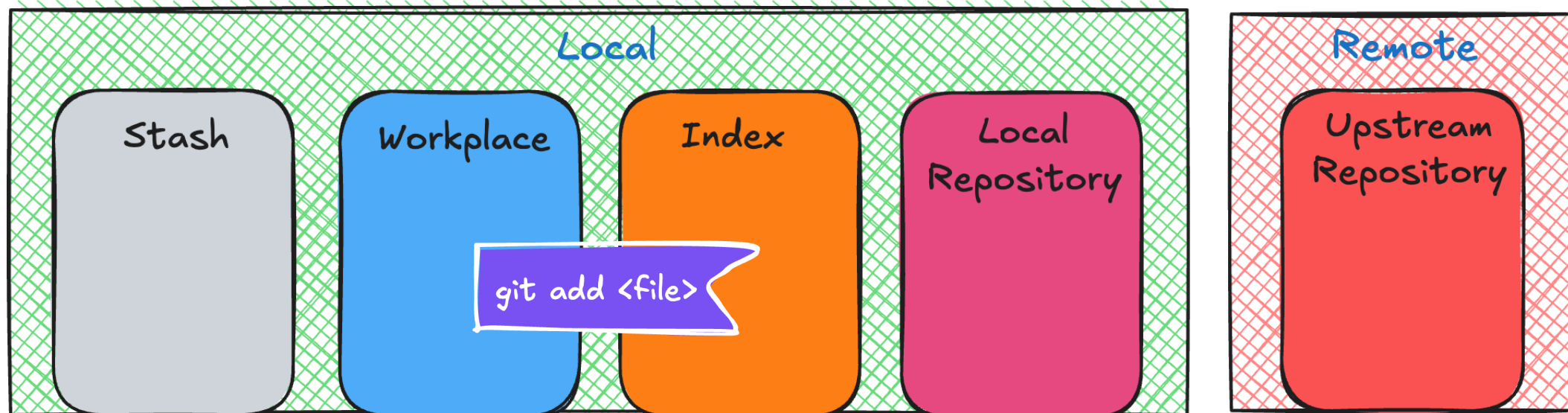
```
~ git checkout -b <nom de la branche>
```



# Git en images

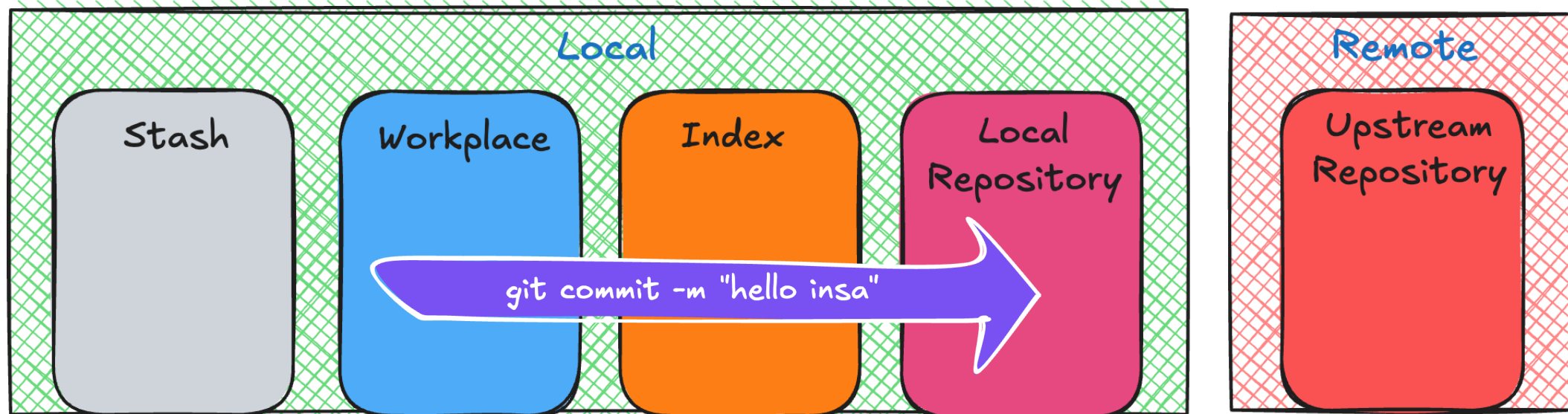
```
~ git add .
```

```
~ git add <nom fichier>
```



# Git en images

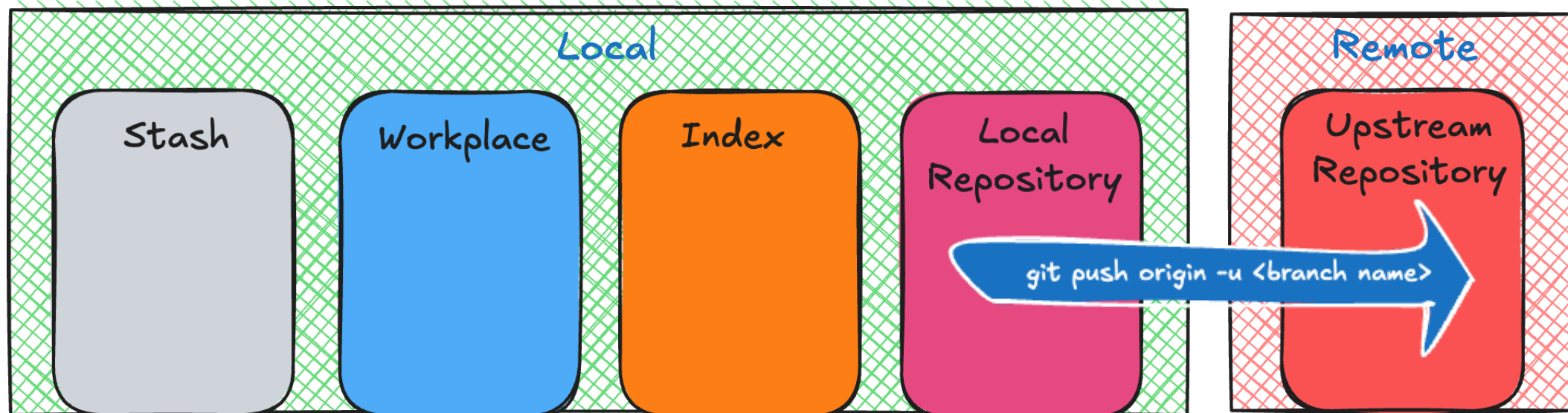
```
~ git commit -m "Hello INSA"
```



# Git en images

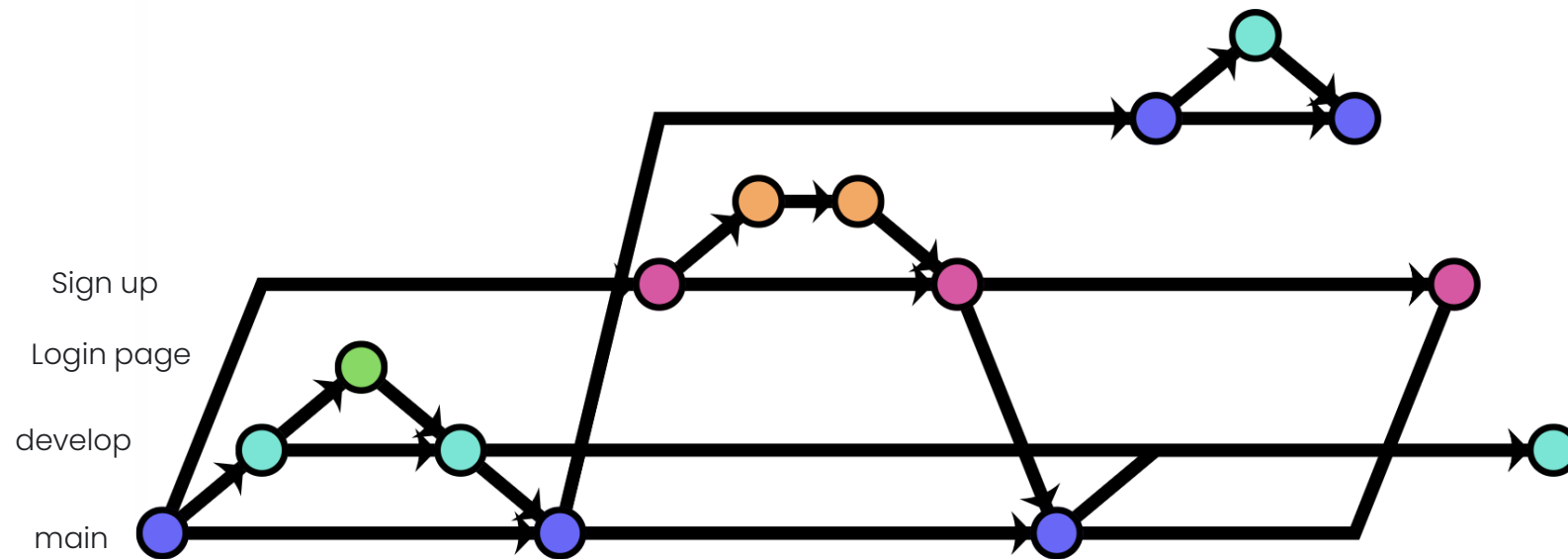


```
~ git push origin -u <branch name>
```



# Graph git

Un graph git est une représentation visuelle d'un projet, de ses commits, branches et tags



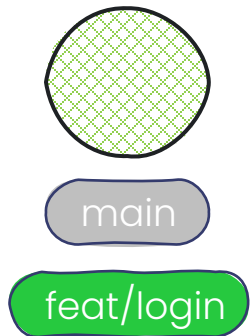


# Graph git en images

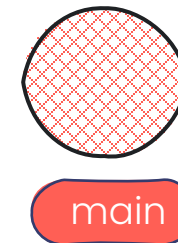
Créer une branche pour la tâche login

```
~ git switch -c feat/login
```

Local



Remote



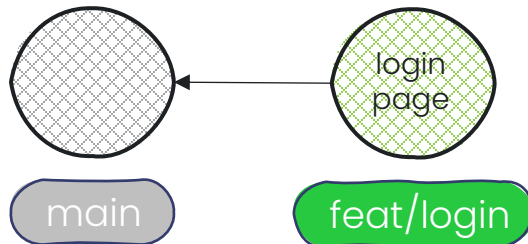
# Graph git en images

Premier commit fait !

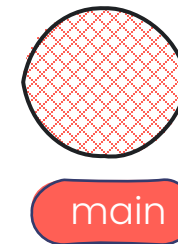


```
~ git add .  
~ git commit -m "login page"
```

Local



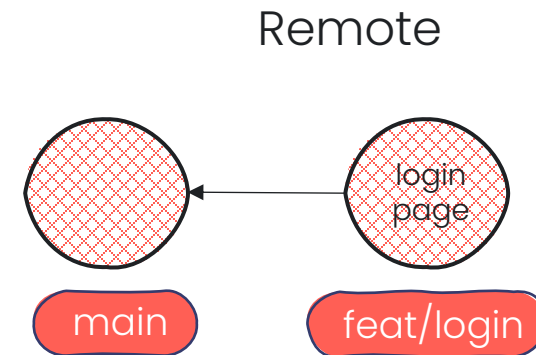
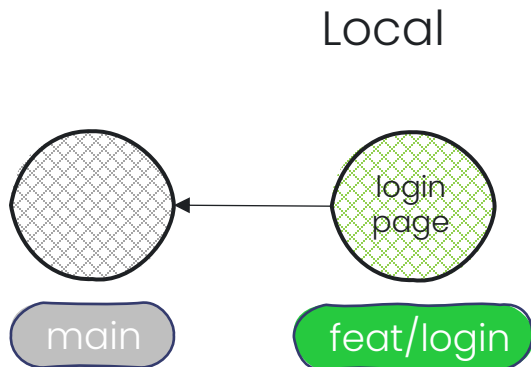
Remote



# Graph git en images

Push de la branche

```
~ git push origin -u feat/login
```



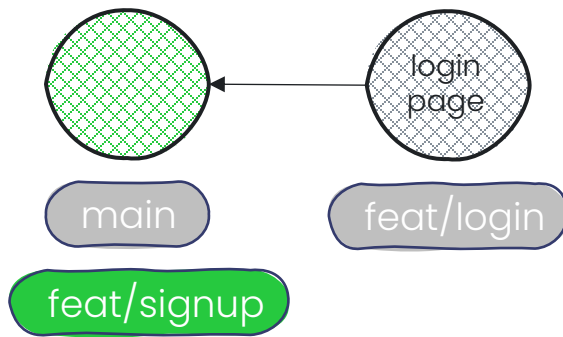
# Graph git en images

Deuxième branche créée depuis main

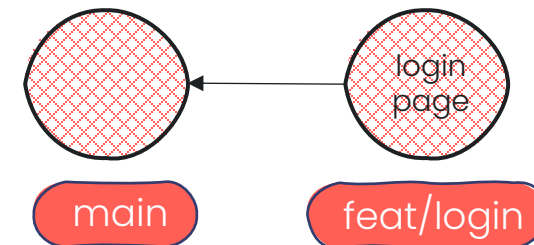


```
~ git switch main  
~ git switch -c feat/signup
```

Local



Remote

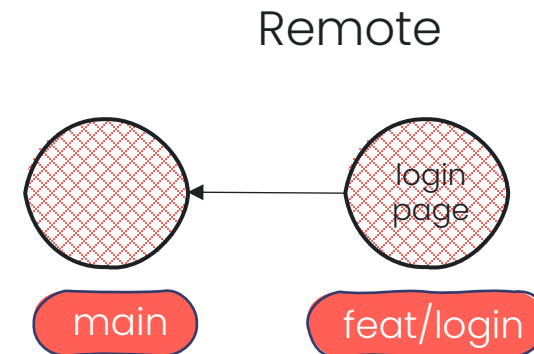
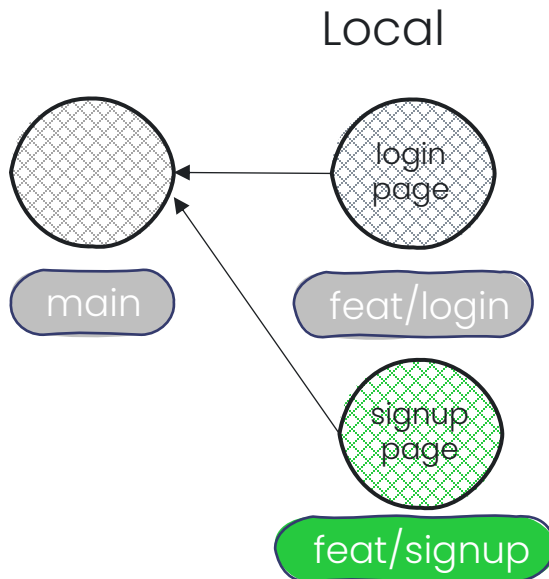


# Graph git en images

Développement et commit fait sur la branche signup



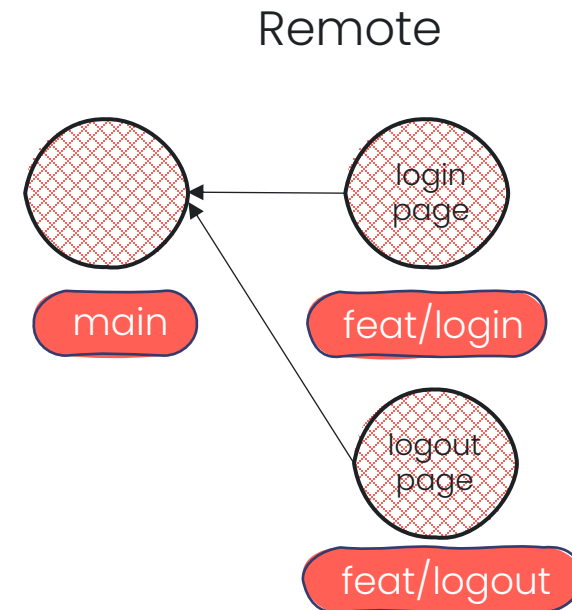
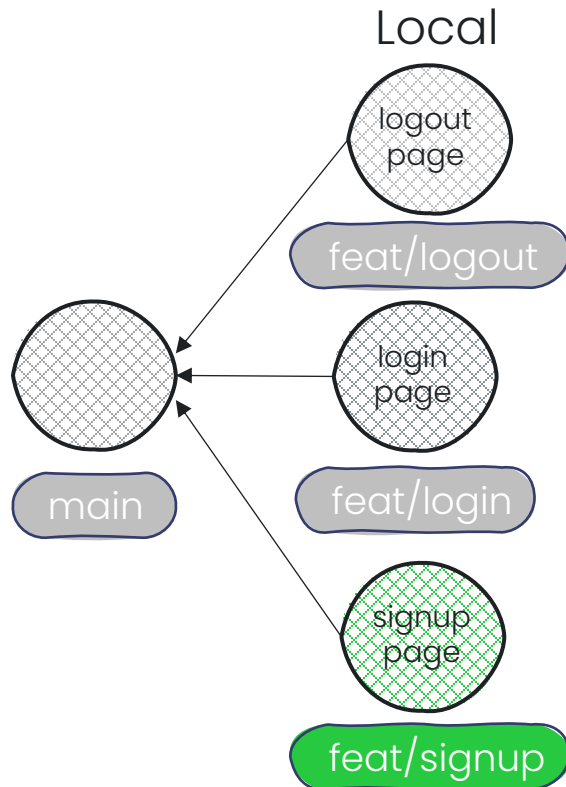
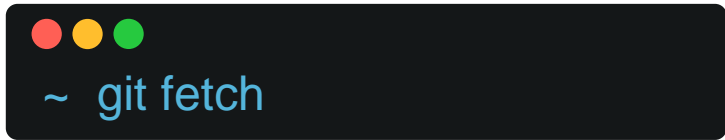
```
~ git add .  
~ git commit -m "signup page"
```





# Graph git en images

Récupérer les branches en remote

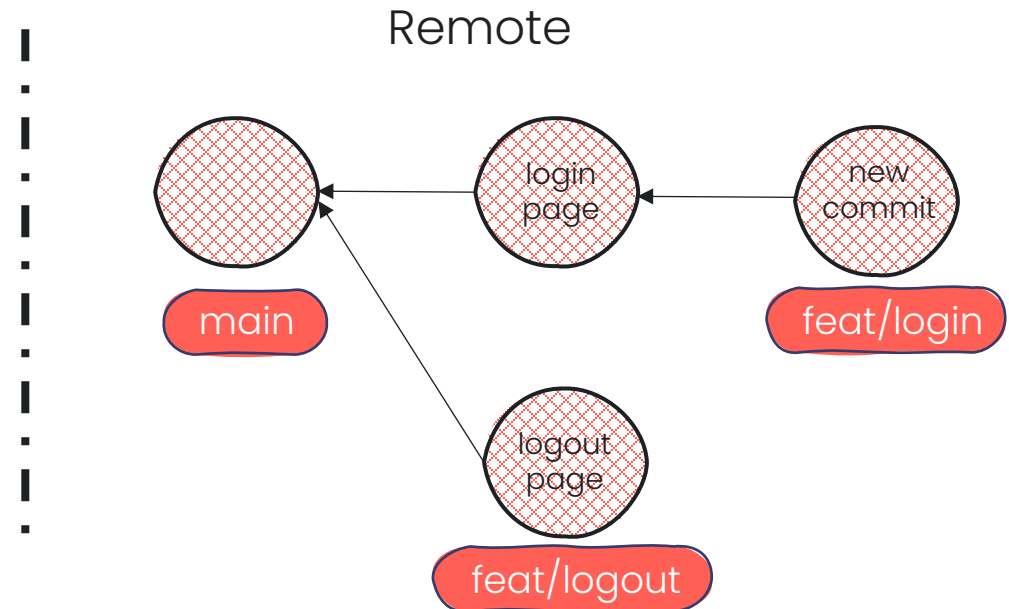
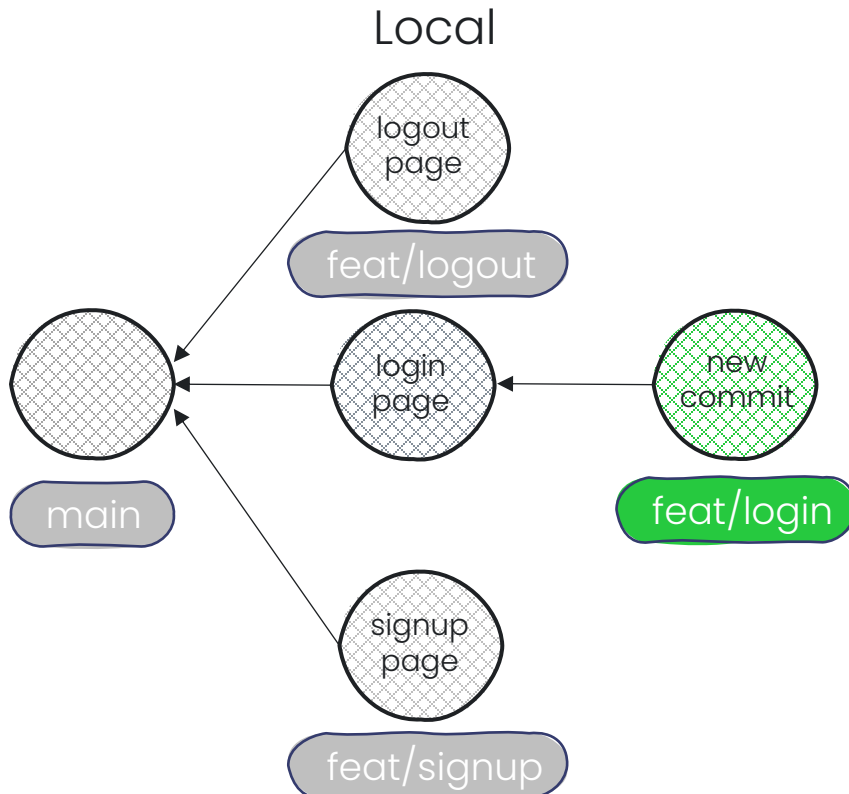


# Graph git en images

Changements sur une branche remote qu'on souhaite récupérer



```
~ git checkout feat/login  
~ git pull
```



# Point sur git rebase

Lorsque votre repository local a divergé de celui en remote, il est parfois nécessaire d'effectuer un rebase afin de récupérer les modifications.

Des commits peuvent générer des conflits quand ils modifient les **mêmes lignes de code ou fichiers de manière différente** dans leurs branches respectives.

Ces conflits doivent être résolus manuellement pour intégrer correctement les modifications.

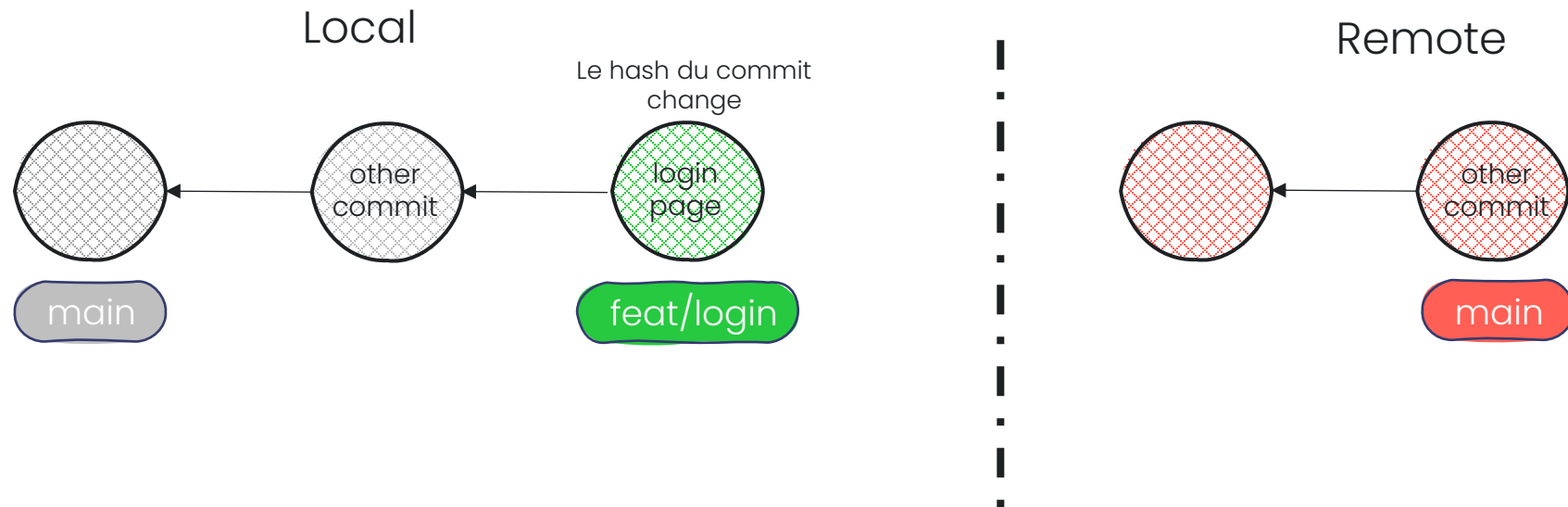


# 1<sup>er</sup> cas : pas de conflits

Les commits ne génèrent pas de conflits :

Exemple : Le commit "*login page*" touche au fichier `login.js` alors que le commit "*other commit*" touche au fichier `other.js`

```
~ git rebase main
```



## 2ème cas : Il y a des conflits !

Les commits génèrent un ou plusieurs conflits :

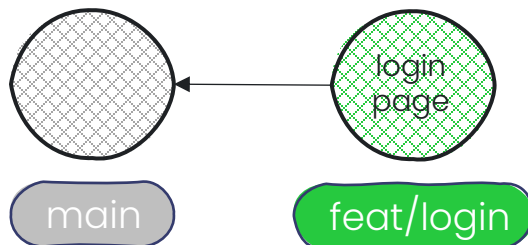
Exemple : Les deux commits touchent au fichier MyApplication.java

```
~ git rebase main
```

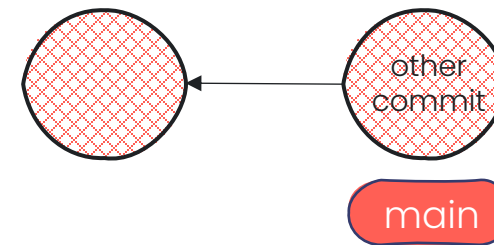


```
Auto-merging MyApplication.java
CONFLICT (content): Merge conflict in MyApplication.java
error: could not apply <commit-hash>... <commit-message>
hint: Resolve all conflicts manually, mark them as resolved with
hint: "git add/rm <conflicted-files>" and then run "git rebase --continue".
```

Local



Remote





# Régler des conflits

modifs en local

code de la  
branche origin/master  
modifié par les autres

```

1  /*
2   * To change this license header, choose License Headers in Project Properties.
3   * To change this template file, choose Tools | Templates
4   * and open the template in the editor.
5   */
6  package myapplication;
7
8  <<<<<<< HEAD
9  import java.util.Scanner;
10 =====
11 import java.io.Scanner;
12 >>>>>>> origin/master
13
14 /**
15  *
16  * @author CCannon
17  */
18 public class MyApplication {
19
20     /**
21     * @param args the command line arguments
22     */
23     public static void main(String[] args) {
24         // TODO code application logic here
25         Scanner keyboard = new Scanner(System.in);
26         <<<<<<< HEAD
27         System.out.print("Enter an input: ");
28         int input = keyboard.nextInt();
29
30         System.out.println(input * 5);
31         =====
32         System.out.println("Enter user input: ");
33         int operand = keyboard.nextInt();
34         >>>>>>> origin/master
35     }
36
37 }
38

```



# Résoudre les conflits

1. Lancer la commande rebase `git rebase`
2. Identifier à partir du message dans le terminal, les fichiers avec conflits
3. Aller dans ces fichiers et corriger les conflits
  - Soit conserver les changements de la branche actuelle
  - Soit ceux de la branche cible
  - Soit modifier tout autrement
4. Ajouter les fichiers corrigés `git add <file>`
5. Continuer le rebase `git rebase --continue`

The screenshot shows the GitHub interface for the repository 'capucine-bois / PLD-Comp'. The repository is marked as 'Private'. The main navigation bar includes links for Code, Issues, Pull requests, Actions, Projects, Security, Insights, and Settings. The repository name 'PLD-Comp' is highlighted with a callout 'Titre du projet'. Below the name, the 'main' branch is selected, with a callout 'branches' pointing to the branch dropdown. To the right of the branch, it says '8 branches' and '0 tags'. A callout 'Pour télécharger une copie du répertoire' points to the 'Code' button. The file list shows the last commit by 'Pilou1542' with the message 'clean', dated '104b25d on Apr 4, 2022', and '131 commits'. A callout 'Dernier commit' points to this commit information. The file list includes folders like '.vscode', 'antlr', 'compiler', and 'tests', and files like '.DS\_Store', '.gitignore', 'README.md', 'install-antlr.sh', 'test', and 'test.c'. A callout 'Fichiers du répertoire dans la branche principale « main »' points to the file list. The right sidebar shows the 'About' section with no description, website, or topics provided, and the 'Releases' and 'Packages' sections, both indicating no published items.

capucine-bois / PLD-Comp

Code Issues Pull requests Actions Projects Security Insights Settings

PLD-Comp Private

Unwatch 1

branches

main 8 branches 0 tags

Go to file Add file Code

About

No description, website, or topics provided.

Readme Activity 0 stars 1 watching 0 forks

Releases

No releases published  
[Create a new release](#)

Packages

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

Pilou1542 clean 104b25d on Apr 4, 2022 131 commits

Dernier commit




Fichiers du répertoire dans la branche principale « main »







.vscode antlr compiler tests .DS\_Store .gitignore README.md install-antlr.sh test test.c

last year last year last year last year last year last year last year last year last year last year

update pour windows clean clean je sais pas trop je sais pas trop



README.md

  capucine-bois / PLD-Comp 
























     

[Code](#) [Issues](#) [Pull requests](#) [Actions](#) [Projects](#) [Security](#) [Insights](#) [Settings](#)

## Commits

 main 

Commits on Apr 4, 2022

<b>clean</b>  Pilou1542 committed on Apr 4, 2022	 104b25d 
<b>add test</b>  Pilou1542 committed on Apr 4, 2022	 09fead1 
<b>ajout de masse tests</b>  Pilou1542 committed on Apr 4, 2022	 29e706a 
<b>rename des tests</b>  Pilou1542 committed on Apr 4, 2022	 ca6f38d 
<b>fin de la belle grammaire</b>  Pilou1542 committed on Apr 4, 2022	 a71468a 
<b>Merge branch 'dev' into belleGram</b>  Pilou1542 committed on Apr 4, 2022	 08a1265 
<b>ca degage</b>  Pilou1542 committed on Apr 4, 2022	 c1b4e94 
	 9e8a839 

<https://github.com/capucine-bois/PLD-Comp>

Noms des  
commits

Hash du  
commit



# Quiz

**Adrien et Bastien récupèrent le répertoire en ligne avec git clone sur leur ordinateur**

**Adrien fait des modifications**

**Adrien add/commit/push sur le repo**

**Le répertoire en ligne est à jour**

**Le code sur l'ordinateur de Bastien s'est-il mis à jour ?**



# Réponse

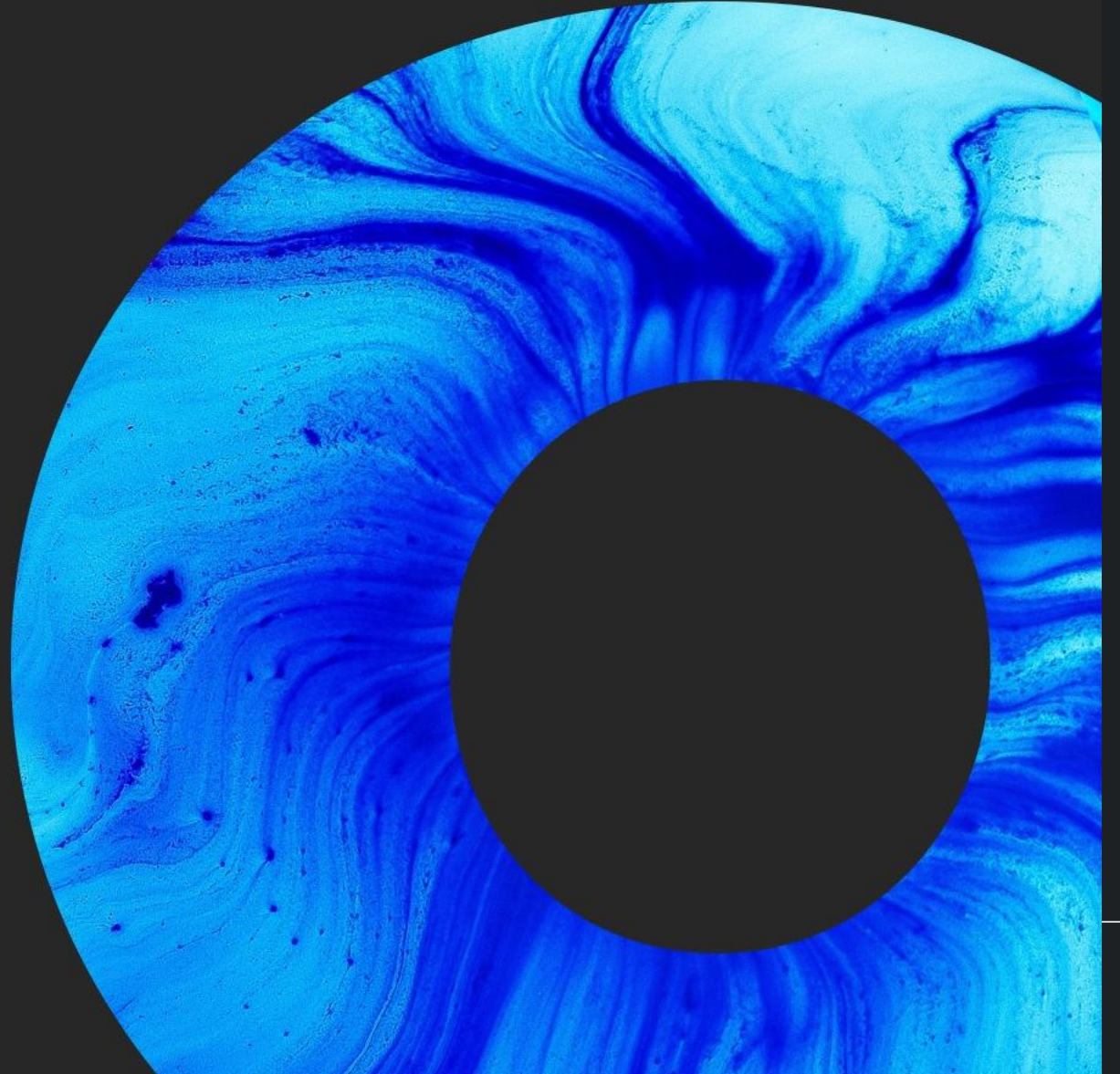
**NON**

**Il faut mettre à jour son code “en local” par rapport au répertoire en ligne.**

**Bastien doit lancer la commande : `git pull`**

# Git 201

- Mener un projet avec git
- Les best practices

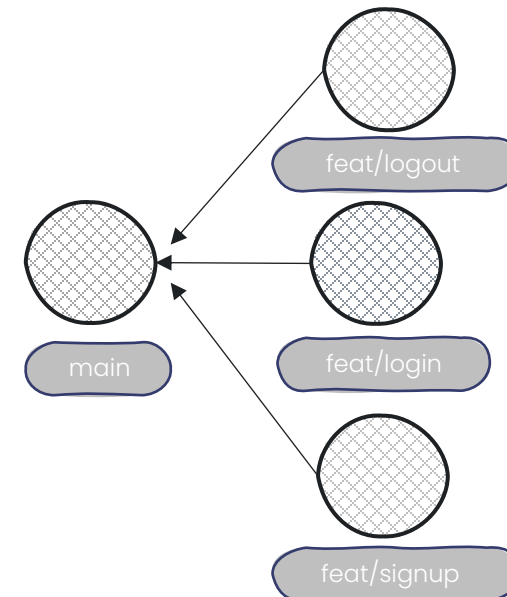


# Les workflows git

Quand on réalise un projet voici le workflow à mettre en place :

- Faire une branche par feature/fonctionnalité : *feat/nom-fonctionnalité*
- Faire une branche par bug/hot-fix : *fix/bug-à-résoudre*

-> Voir [Workflow-Gitflow](#)





# Commit: best practices

Rappel : `git commit -m "mon-message"`

- Nommer ses commits correctement :

*feat(nom-fonctionnalité): description de ma modification*

- En anglais !

+ Gitmoji :

- Ajouter des emojis dans son message pour définir l'objectif / le scope de son commit.

Ex : `git commit -m ":art: feat(login): refactor login function"`

- 🎨 :art: Improve structure / format of the code.
- ⚡ :zap: Improve performance.
- 🔥 :fire: Remove code or files.
- 🐛 :bug: Fix a bug.
- 🚑 :ambulance: Critical hotfix.
- ✨ :sparkles: Introduce new features.
- 📝 :memo: Add or update documentation.
- 🚀 :rocket: Deploy stuff.

Exemples de gitmojis

# Bonus

- Outils intéressants pour git



# Git Tools

Il existe des outils très puissants de visualisation git qui peuvent vous servir d'interface pour vos manipulations git comme des rebase, résoudre des conflits et bien d'autres.

- Git Kraken : Application gratuite avec la licence étudiante
- Git Lens : Plugin VS code gratuit

## Bonus – Les outils utiles



# Git kraken

The screenshot displays the GitKraken application interface for a repository named 'git-graph-example'. The interface is divided into several panels:

- Left Panel:** Contains a sidebar with sections for 'LOCAL' (21/21), 'REMOTE' (0/0), 'ISSUES' (with a list of issue trackers like GitHub, GitLab, etc.), 'TEAMS', 'TAGS' (4/4), and 'SUBMODULES' (0).
- Top Panel:** Shows the repository name 'git-graph-example' and the current branch 'b-6'. It includes a menu bar (File, Edit, View, Help) and a toolbar with various Git actions like Undo, Redo, Fetch, Push, Branch, Stash, Pop, Terminal, Boards, and Timelines.
- Graph View:** The central area displays a graph of branches and commits. The 'b-6' branch is selected and highlighted in blue. The graph shows a series of commits connected by lines, with commit messages visible on the right side of the graph.
- Right Panel:** Contains a 'Commit Message' section with a 'Summary' and 'Description' field. Below it, there are sections for 'Unstaged Files (2)' (listing 'add\_file.cs' and 'remove\_file.cs') and 'Staged Files (1)' (listing 'README.md'). A 'Commit Message' section is also present at the bottom right.



# Hook de pré-commit

## Qu'est-ce qu'un hook de pre-commit ?

Un script qui s'exécute avant chaque commit.


Il permet d'automatiser des tâches et d'appliquer des règles.

### Pour :

- Vérifier la syntaxe (linter).
- Formater le code (prettier).
- Valider les messages de commit (commitlint).

### Deux outils :

**Husky** : Le manager des hooks, il configure et exécute les scripts. **Husky** 🐶

**Commitlint** : Le policier des messages de commit, il vérifie que les messages respectent un format précis. **cl** 

# Ressources utiles

- Learn git branching : <https://learngitbranching.js.org/>
- Git cheatsheet : <https://education.github.com/git-cheat-sheet-education.pdf>

# Merci.

→ 28 novembre 2024

**onepoint.**  
beyond the obvious