15/04/2023
Version 3

GIT

JÉRÉMY PERROUAULT

INTRODUCTION

INTRODUCTION À GIT

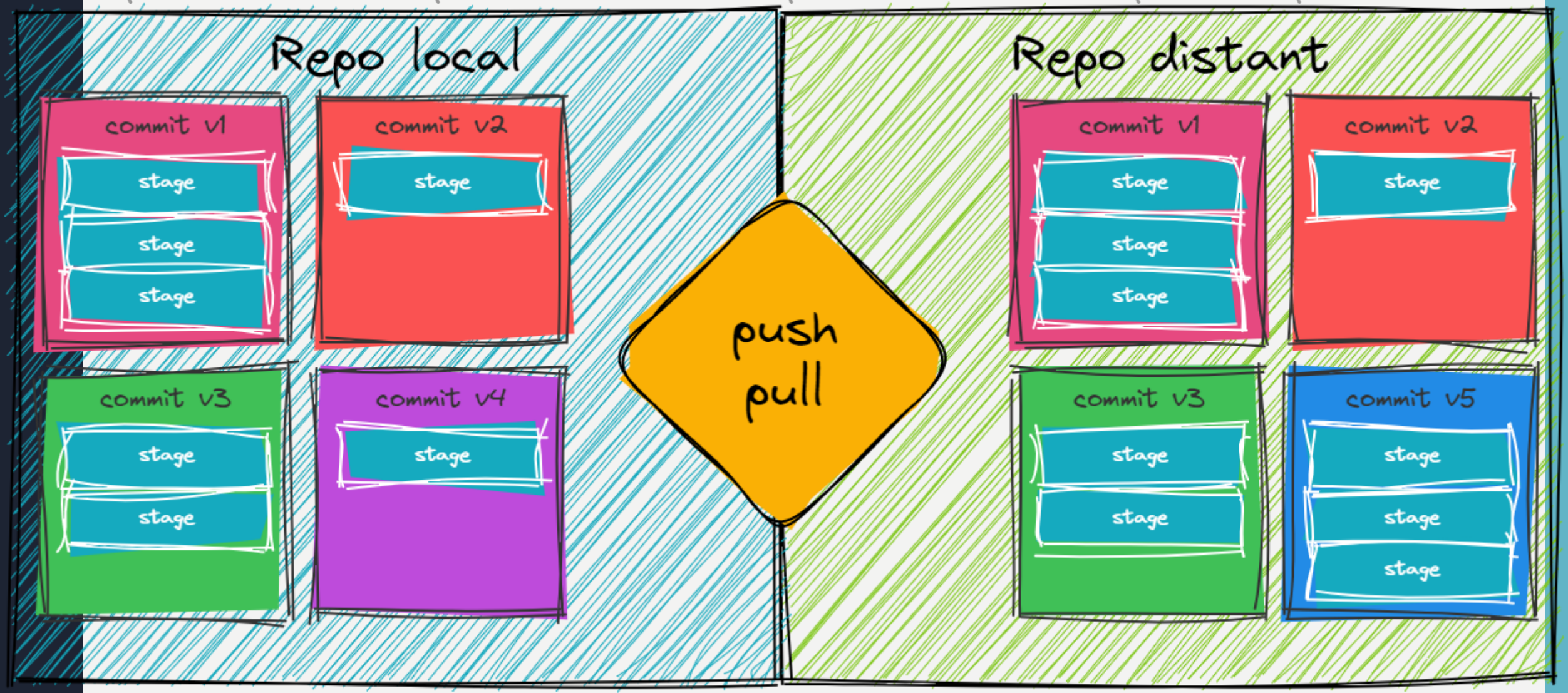
INTRODUCTION

- Système de contrôle de versions (ou gestionnaire de sources) **décentralisé**
  - Chaque personne possède sa propre copie du dépôt sur son poste
- Autorise plusieurs personnes à travailler sur des documents communs
  - Chaque personne a une copie locale
- Synchronise les différentes versions des documents
- Permet un rollback des fichiers sur des versions précédentes
- Permet de suivre les modifications au cours du temps

INTRODUCTION

- Pour exécuter les commandes **git**
  - Il faut se placer dans le répertoire **git**

1 stage, c'est un fichier ajouté, modifié ou supprimé
1 commit, c'est un ou plusieurs stages
1 push, c'est l'envoie d'un ou plusieurs commits sur le repo distant
1 pull, c'est la récupération d'un ou de plusieurs commits depuis le repo distant



INTRODUCTION

- Les commandes de `stage`, de `commit`, de `pull` et de `push` se font sur une branche
  - On peut avoir plusieurs branches, mais on ne travaille que sur une seule branche à la fois

INTRODUCTION

Action	Description
`init`	Initialiser un nouveau repo (ou nouveau projet)
`clone`	Récupérer un repo existant, et en créer une copie locale
`status`	Vérifier si on est en avance ou en retard, voir les fichiers à gérer, etc.
`add`	Ajouter des fichiers dans le prochain commit
`reset`	Retirer des fichiers du prochain commit
`fetch`	Récupérer les données du repo distant
`merge`	Assembler des parties
`pull`	Se mettre à jour depuis le repo distant
`commit`	Versionne dans le repo local
`push`	Envoie les commits locaux vers le repo distant
`revert`	Revenir à une ancienne version
`diff`	Différence entre deux versions
`branch`	Lister les branches / créer une nouvelle branche
`checkout`	Naviguer vers une branche

Lorsque le git n'a pas été initialisé

1. On initialise le repo

`git init`

2. On ajoute les fichiers

`git add`

3. On commit

`git commit`

4. On crée la branche

`git branch`

5. On associé le repo local à un repo distant

`git remote`

5. On push sur le repo distant

`git push`

Lorsque le git a été initialisé

1. On clone le repo
sur la machine locale

Lorsqu'on veut récupérer des modifications

1. On vérifie si on est à jour

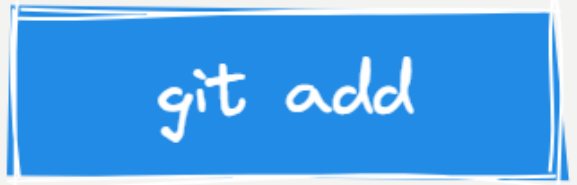
```
git fetch
git status
```

2. On récupère les mises à jour si nécessaire

```
git pull
```

Lorsqu'on veut envoyer des modifications

1. On ajoute les fichiers



2. On commit

3. On vérifie si on est à jour

4. On récupère les mises à jour si nécessaire

5. On push sur le repo distant

INTRODUCTION

- Récupérer les modifications du repo distant, sans les intégrer

```
git fetch
```

- Vérifier les modifications à faire, les retards ou les avances

```
git status
```

COMMIT

- Un **commit** est la création d'une nouvelle révision (version)
- Avant un **commit**, il est nécessaire d'indiquer quelles sont les modifications
  - On parle alors de **stage**, d'organisation de fichiers
- Un **commit** contient un ensemble de stages
  - Fichiers ajoutés
  - Fichiers modifiés
  - Fichiers supprimés
- Les répertoires ne sont pas présents
  - Ils le sont au travers des fichiers existants
  - Donc un répertoire vide ne sera pas commitable !
- Lorsqu'un **commit** est réalisé, il est important de préciser un message pour mieux s'y retrouver

COMMIT

- Lorsqu'un fichier est dans un **stage**, on dit qu'il est *tracké*
- Si le **commit** a lieu, le nouveau fichier *tracké* ne pourra plus être ignoré
  - Chaque modification sera identifiée
  - Sa suppression sera identifiée
- Tant que le **commit** n'est pas fait, on peut retirer un fichier d'un **stage**
  - Si c'est un nouveau fichier, il n'est alors plus *tracké*
  - Si ce n'est pas un nouveau fichier, on retire du **stage** les modifications de ce fichier

COMMIT

- Ajouter tous les fichiers

```
git add .
```

- Ajouter un seul fichier

```
git add repertoire/fichier
```

- Retirer un fichier

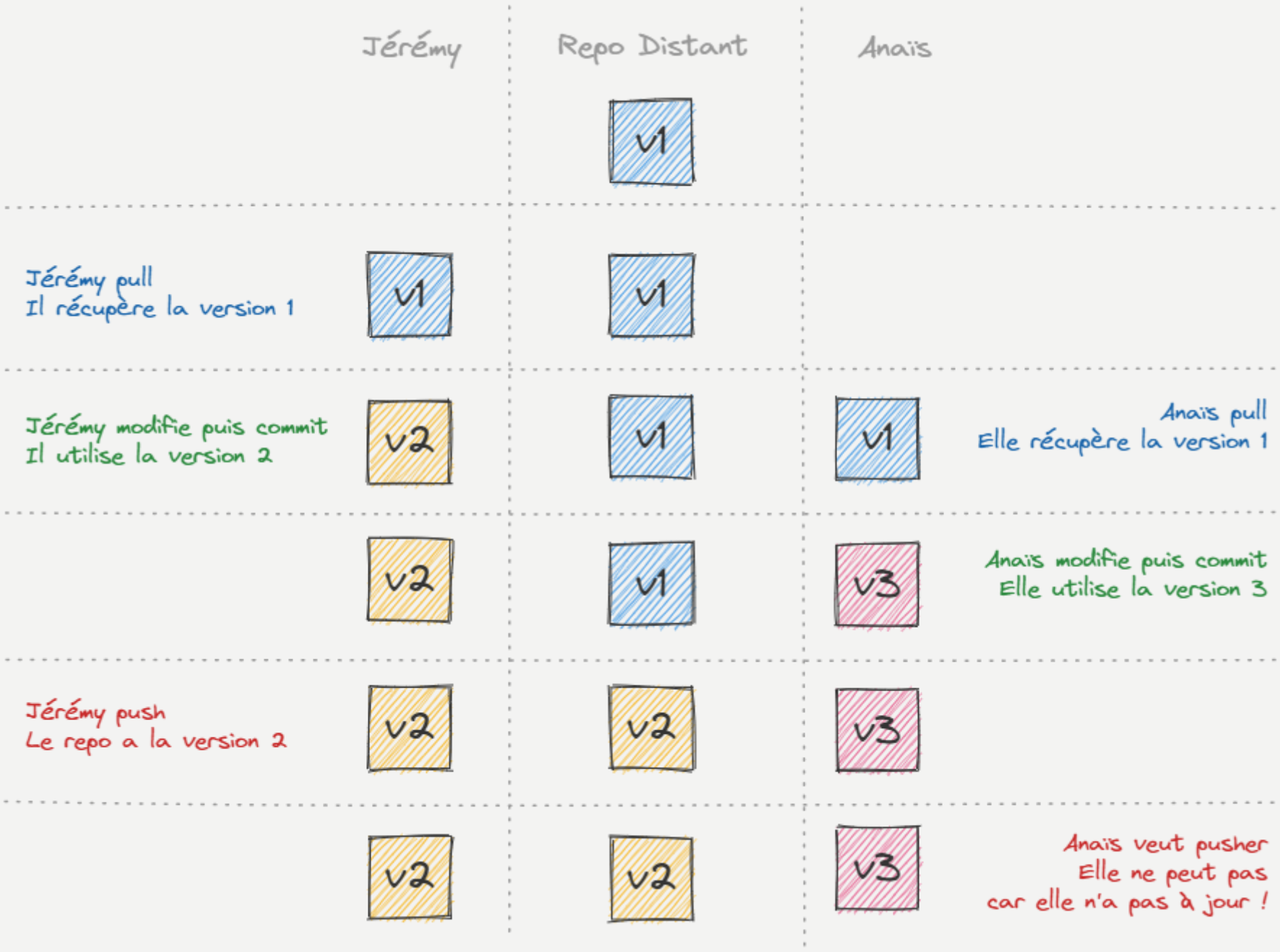
```
git reset repertoire/fichier
```

- Réaliser un commit

```
git commit -m "Message du commit"
```

EXERCICE

- Créer un compte gratuit sur **Github**
- Télécharger et installer le client Git
- Initialiser un premier projet (`git init`)
- Tester de créer un fichier et de le placer sur le dépôt local
  - `commit` (avec commentaires !)
  - `push`


Jérémy
Repo Distant
Anaïs
v1
Jérémy pull
Il récupère la version 1
v1
v1
Jérémy modifie puis commit
Il utilise la version 2
v2
v1
v1
Anaïs pull
Elle récupère la version 1
v2
v1
v3
Anaïs modifie puis commit
Elle utilise la version 3
Jérémy push
Le repo a la version 2
v2
v2
v3
v2
v2
v3
Anaïs veut pusher
Elle ne peut pas
car elle n'a pas à jour !

TRAVAILLER TOUJOURS À JOUR

- Dans le dernier cas, ça coince
  - Il est impossible de pusher sur une branche qui n'est pas à jour
- Il est nécessaire de mettre à jour sa branche
  - Si des conflits sont détectés, il faudra les résoudre
    - Il est impossible de faire un commit si les conflits ne sont pas résolus

REMISE

REMETTRE SON TRAVAIL À PLUS TARD

LA REMISE

- Imaginons que dans le dernier cas, le commit n'ait pas été fait
  - Mais que Anaïs cherche à mettre à jour sa branche (à récupérer les modifications)
- D'une manière générale, le travail n'est pas terminé mais il faut soit :
  - Changer de branche
  - Faire un merge
  - Récupérer le travail d'un partenaire
  - Mettre à jour la branche
- Périlleux de faire ces actions s'il y a du travail en cours !

LA REMISE

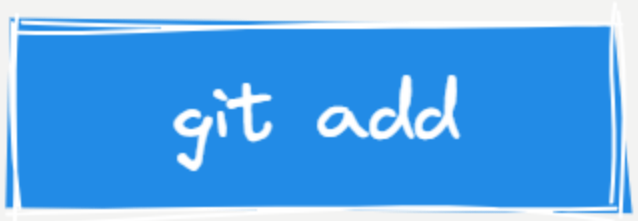
- Il faut donc faire (au choix)
  - Supprimer les changements (on perd le travail)
  - Stager les fichiers et
    - Faire un commit (pour être sûr d'enregistrer son travail)
    - Remettre les changements à plus tard : c'est la remise, avec la commande `git stash`
- Si remisage, une fois la manipulation faite
  - Il faut réappliquer les changements
    - Pour appliquer les changements `git stash apply`
    - Pour supprimer le stash `git stash drop`
    - Pour faire ces deux actions en une `git stash pop`
  - Puis résoudre les éventuels conflits

LA REMISE


Jérémy
Repo Distant
Anaïs
v2
v2
v1
Anaïs stage et stash
Elle récupère la version 1
v2
v2
v2
Anaïs pull
Elle récupère la version 2
v2
v2
v4
Anaïs pop
Elle récupère sa version 3
Elle doit gérer les conflits
Ce qui en fait une version 4
v2
v4
v4
Anaïs push
Le repo a la version 4
Jérémy pull
Il récupère la version 4
v4
v4
v4

Lorsqu'on veut mettre dans la remise

1. On ajoute les fichiers



2. On remise

Lorsqu'on veut sortir de la remise

1. On ressort de la remise

```
git stash pop
```

LA REMISE

- Mettre dans la remise

```
git stash
```

- Voir la liste des remises

```
git stash list
```

- Récupérer de la remise le dernier **stash** (puis le supprimer)

```
git stash pop
```

- Récupérer de la remise un **stash** en particulier (puis le supprimer)

```
git stash pop {id}
```

```
git stash pop 0
```

```
git stash pop 1
```

EXERCICE

- Faire des modifications
- Les remiser
- Lister les remises
- Ressortir de la remise les modifications

.GITIGNORE

IGNORER DES FICHIERS

.GITIGNORE

- Fichier texte qui s'appelle *.gitignore* (avec le point devant)
- Fichier qui permet d'ignorer des fichiers et/ou des répertoires (les fichiers du répertoire)
  - A placer généralement à la racine du projet
  - Peut également être dans des sous-répertoires
    - (sont ignorés les fichiers et répertoires à partir de l'emplacement de *.gitignore*)

.GITIGNORE

- Pour un projet **JAVA**, on ignorera toujours

```
.project
.classpath
.local
.settings/
bin/
target/
```

.GITIGNORE

- Attention, dès qu'un fichier a été *tracké* puis *commité*, il ne peut plus être ignoré !
  - Il faut alors le supprimer, soit manuellement, soit par la commande
    - l'option `cached` permet de garder le fichier en local

```
git rm --cached fichier
```

  - Peu importe la façon de procéder, il faudra
    - Commiter ce changement
    - Ajouter le fichier au *.gitignore*
  - Le fichier sera alors supprimé du repo
  - La prochaine mise à jour (`pull`) par les collègues supprimera ce fichier de leur repo local
    - Veiller à bien communiquer cette modification à l'ensemble de l'équipe

EXERCICE

- Créer un fichier *test.txt*
- Ignorer ce fichier
- Vérifier avec un `status` que le fichier n'est pas à commiter

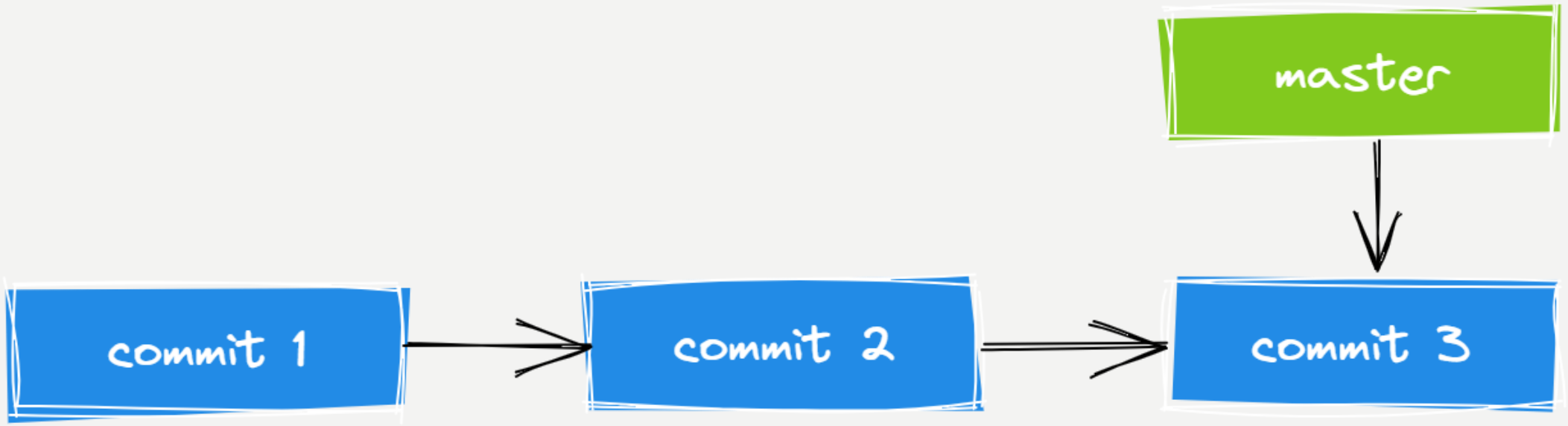
BRANCHES

PASSER DE BRANCHE EN BRANCHE

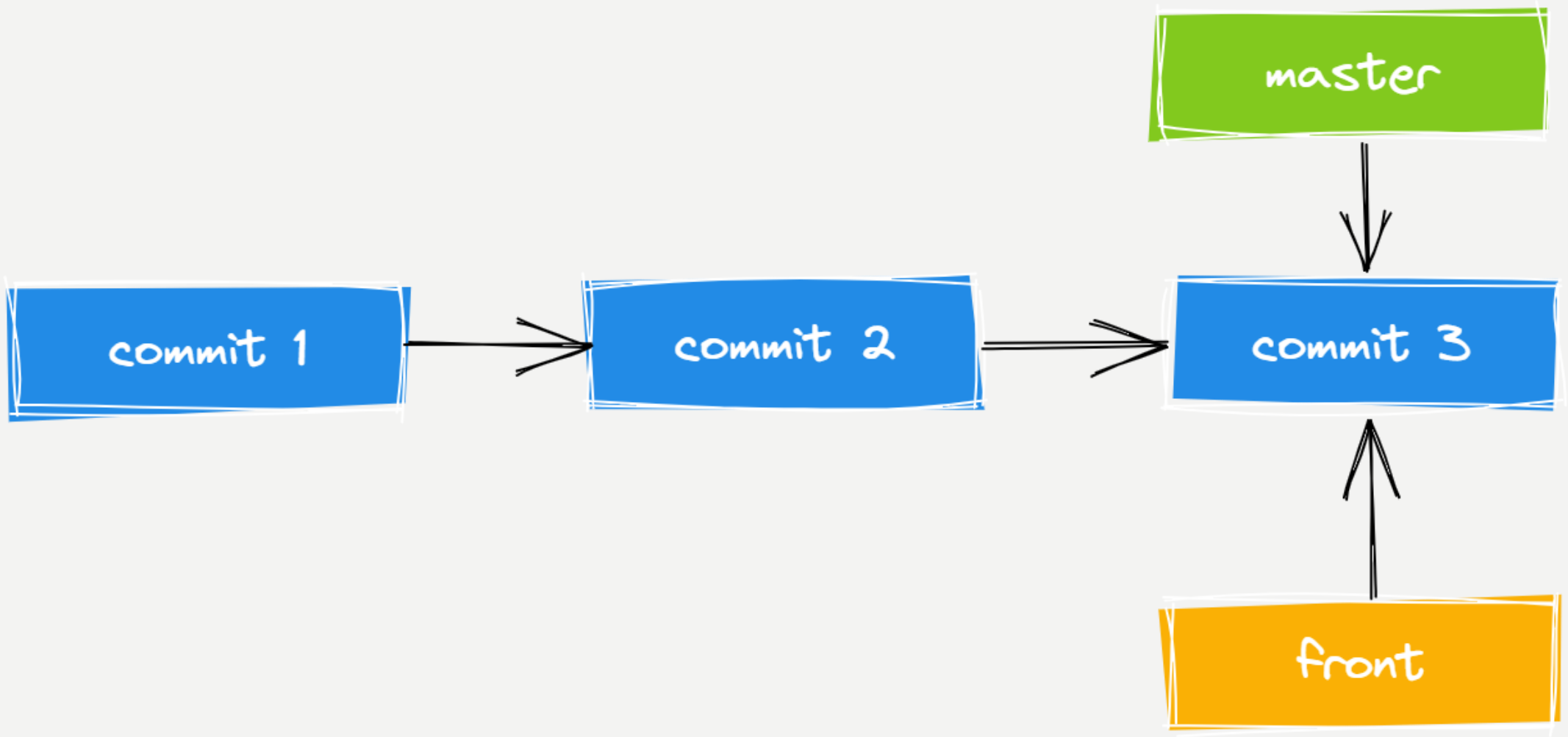
BRANCHES

- Pointeur sur un **commit** en particulier
- Permet d'isoler des modifications, par exemple
  - Corriger un bug
  - Ajouter une fonctionnalité bien particulière
  - Séparer le travail entre développeurs
  - …

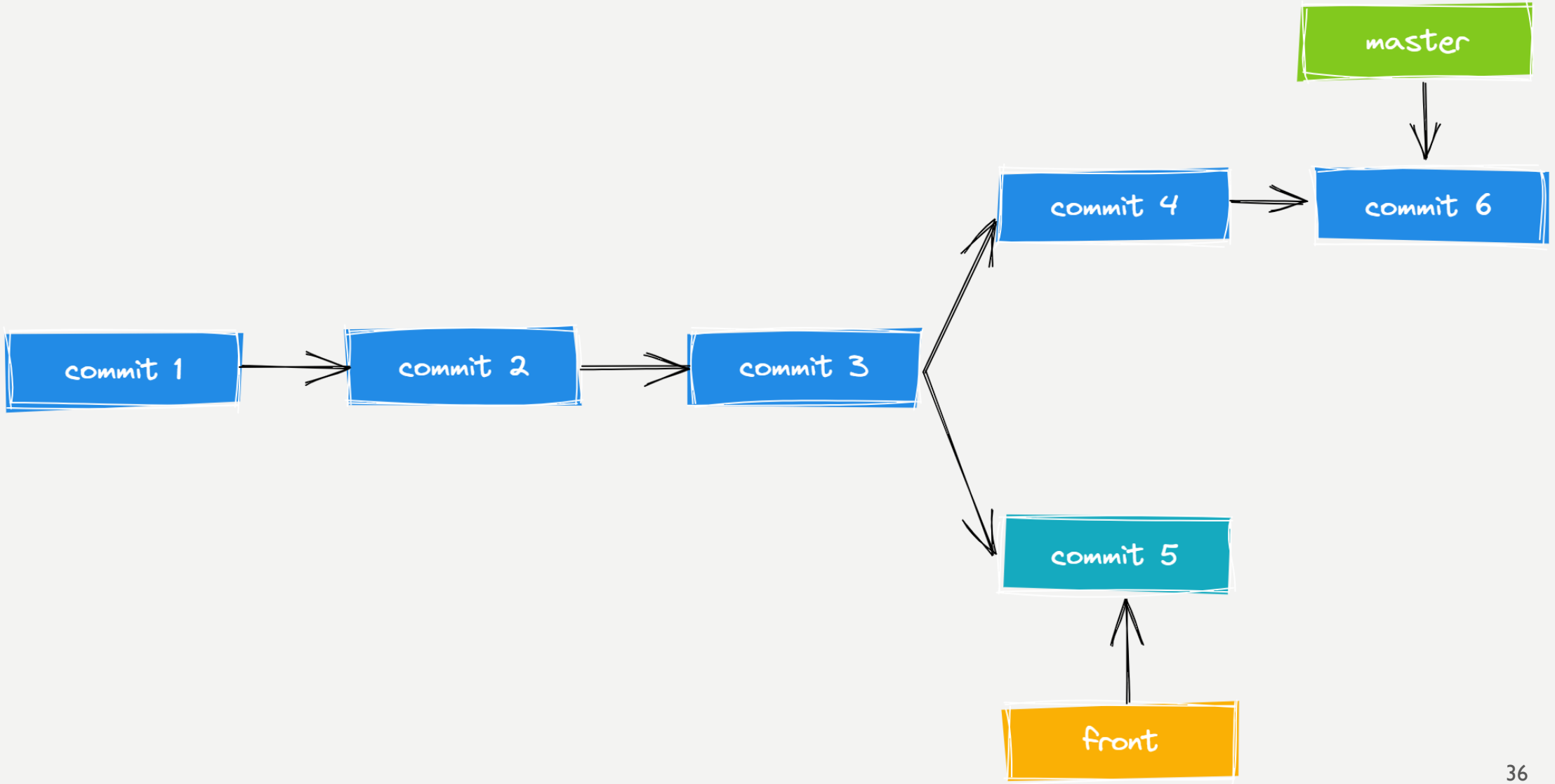
BRANCHES



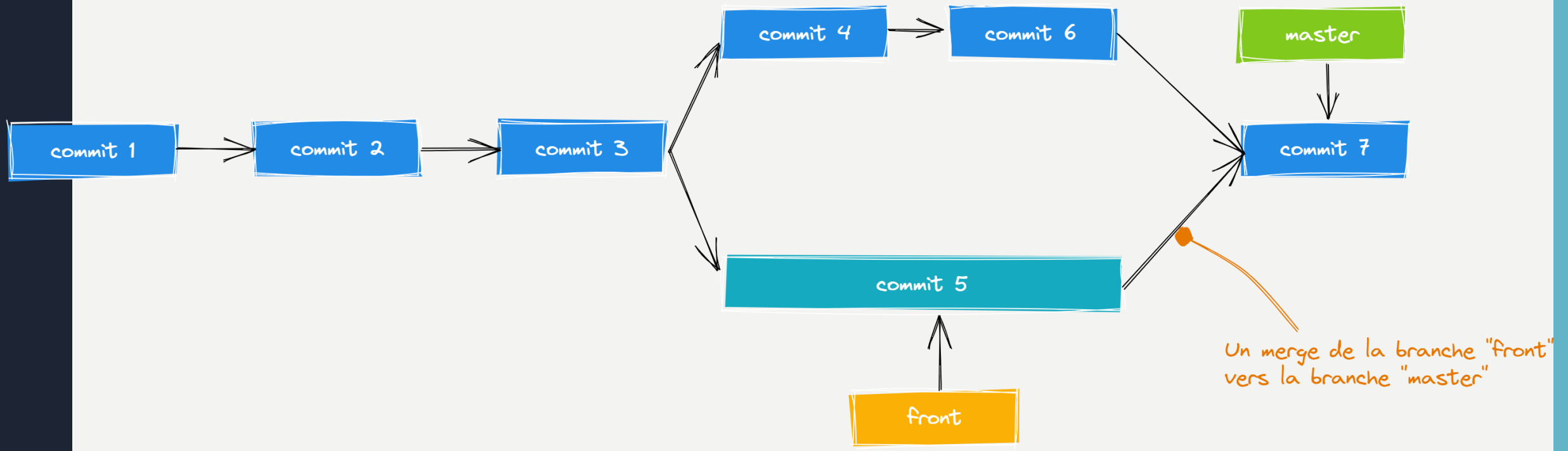
BRANCHES



BRANCHES


master
commit 1
commit 2
commit 3
commit 4
commit 6
commit 5
front

BRANCHES


commit 1
commit 2
commit 3
commit 4
commit 6
master
commit 7
commit 5
front
Un merge de la branche "front"
vers la branche "master"

Lorsqu'on veut créer une branche et naviguer dessus

1. On crée la branche

```
git branch
```

2. On naviguer vers la branche

```
git checkout
```

Lorsqu'on veut rassembler 2 branches

1. On commit les changements de la branche en cours

2. On navigue vers l'autre branche

3. On merge les 2 branches

BRANCHES

- Lister toutes les branches

```
git branch -a
```

- Créer une nouvelle branche

```
git branch nombranche
```

- Naviguer vers une branche

```
git checkout nombranche
```

- Navigurer vers une branche qui n'existe pas (et la créer au passage)

```
git checkout -b nombranche
```

EXERCICE

- Créer une nouvelle branche
- Modifier un ou plusieurs fichiers
- Stager et commiter
- Revenir sur la branche principale
- Assembler les 2 branches