Formation Git

Theo Bessel ≔ theo.bessel[at]etu.inp-n7.fr

1/ Le système Git ?

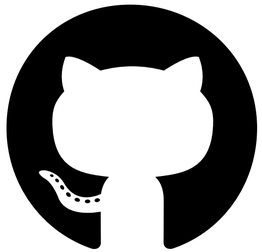
1/ Le système Git ?

Ça sert à gérer l’évolution d’une arborescence de fichiers …

… mais aussi à sauvegarder facilement ton code

1/ Le système Git ?

Différentes instances :




2/ Les repository ?

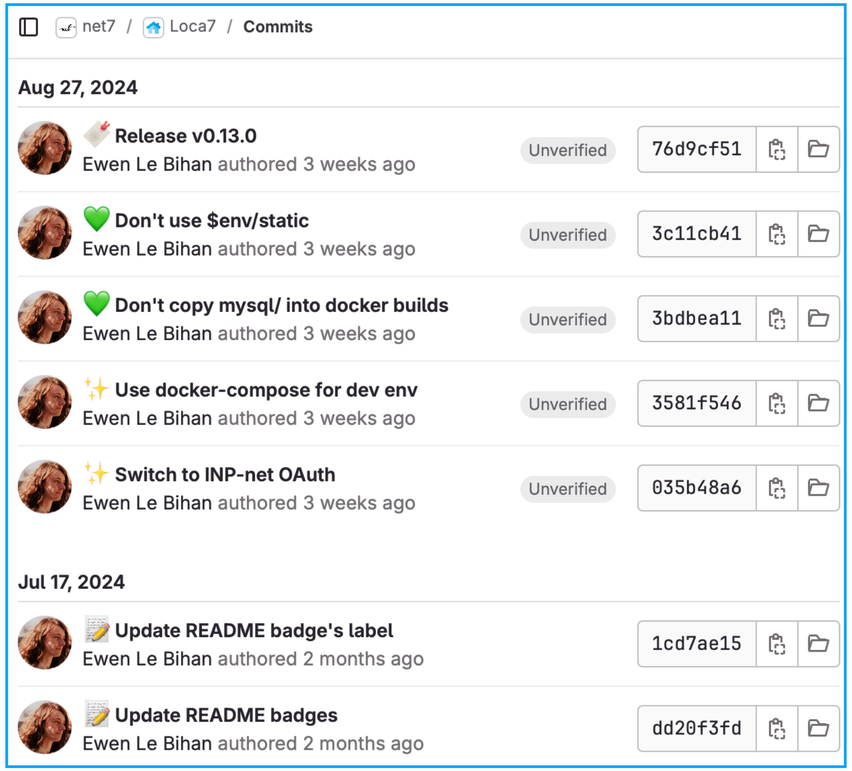
Theo Bessel ≔ theo.bessel@etu.inp-n7.fr

2/ Les repository ?

Un repository c’est un dossier mais versioné

Dossier + Historique de modifications

2/ Les repository ?



Theo Bessel ≔ theo.bessel@etu.inp-n7.fr

2/ Les repository ?

Un repository peut exister en local mais aussi sur un serveur

Synchronisation Local / Distant

Permet une sauvegarde efficace du code

2/ Les repository ?

Initialiser un repository en créant un dossier :

```
# git init <nom du dossier>
```

Initialiser un repository dans un dossier existant :

```
# git init
```

3/ Les commits ????

3/ Les commits ????

Un commit c’est une nouvelle version du code du repository

Une sorte de sauvegarde locale

3/ Les commits ????

Ajouter des fichiers/dossiers à la création d'une version :

```
# git add <fichier 1> <fichier 2> <…>
```

Créer une nouvelle version avec un descriptif :

```
# git commit -m <descriptif>
```

3/ Les commits ????

Suivre quels fichiers sont actuellement ajoutés à la version :

```
# git status
```

Afficher l’historique des commits :

```
# git log
```

3/ Les commits ????

Revenir à un commit précédent :

```
# git reset <hash>
```

Modifier le dernier commit effectué :

```
# git commit --amend
```

3/ Les commits ????

Évolution d’un repository au fil des commits :

Exercices

https://qrco.de/bfRK9q

Exercices

Avec un petit récapitulatif :

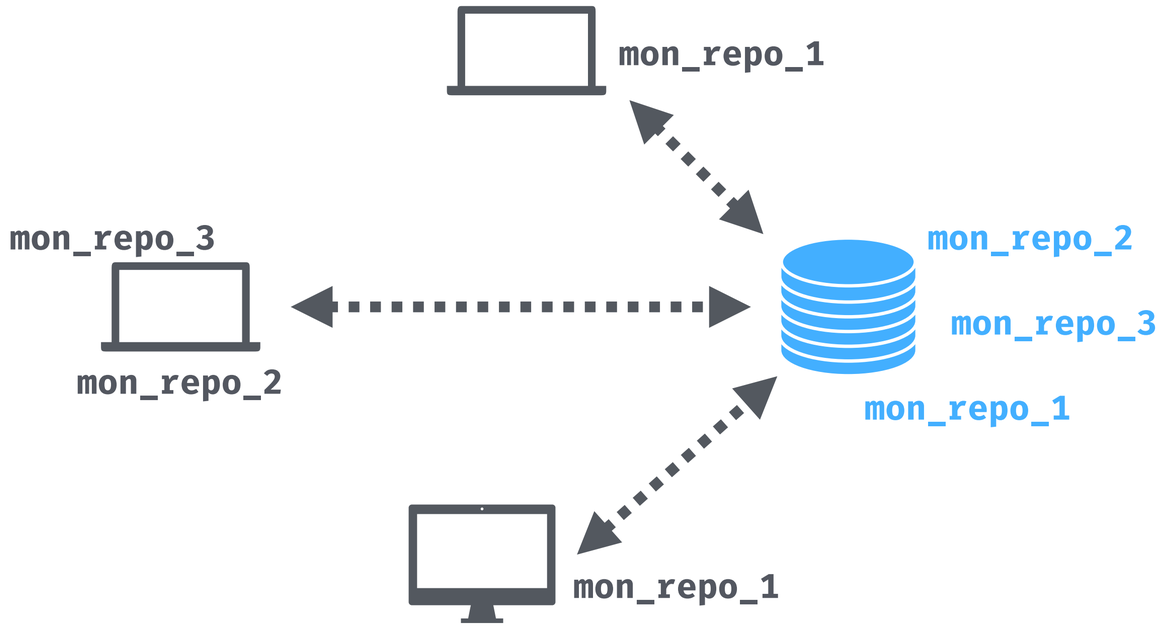
Créer un repository :	`# git init <nom du dossier>`
Créer un repository dans un dossier :	`# git init`
Ajouter des fichiers au commit :	`# git add <fichier 1> <fichier 2> <…>`
Créer un commit (= une version) :	`# git commit -m <descriptif>`
Voir les fichiers ajoutés :	`# git status`
Afficher l'historique :	`# git log`
Revenir à un commit antérieur :	`# git reset <hash>`
Modifier le dernier commit :	`# git commit --amend`

4/ Git distant ????

4/ Git distant ????

Git peut fonctionner en distant …

… un peut comme un Drive



4/ Git distant ????

Cloner un repository à partir d'un URL :

```
# git clone <url>
```

Connecter un repository local à un repository distant :

```
# git remote add origin <url>
```

4/ Git distant ????

Synchroniser un repository local et un repository distant :

Local → Distant :

```
# git push
```

Distant → Local :

```
# git pull
```

L'option `--force` permet d'appliquer des changements destructifs

5/ Les branches ???

5/ Les branches ???

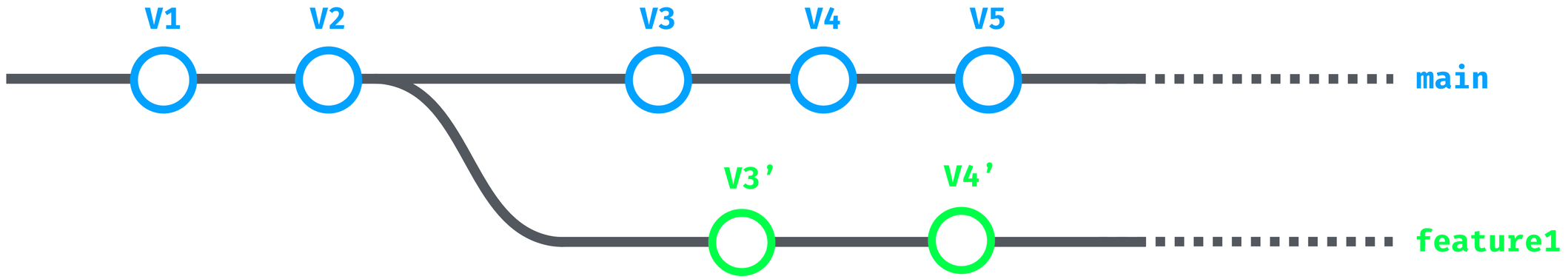
Les branches permettent de créer des historiques parallèles

La branche principale s’appelle main ou master

Les autres branches servent à développer des fonctionnalités annexes

5/ Les branches ???

Un premier exemple :



5/ Les branches ???

Créer une nouvelle branche :

```
# git branch <nom de la branche>
```

Afficher les branches :

```
# git branch
```

Changer de branche :

```
# git switch <branche destination>
```

5/ Les branches ???

Réunir deux branches :

Lorsqu’on est satisfait de l’évolution d’une branche

On veut réunir des branches pour rassembler les fonctionnalités

Theo Bessel ≔ theo.bessel@etu.inp-n7.fr

5/ Les branches ???

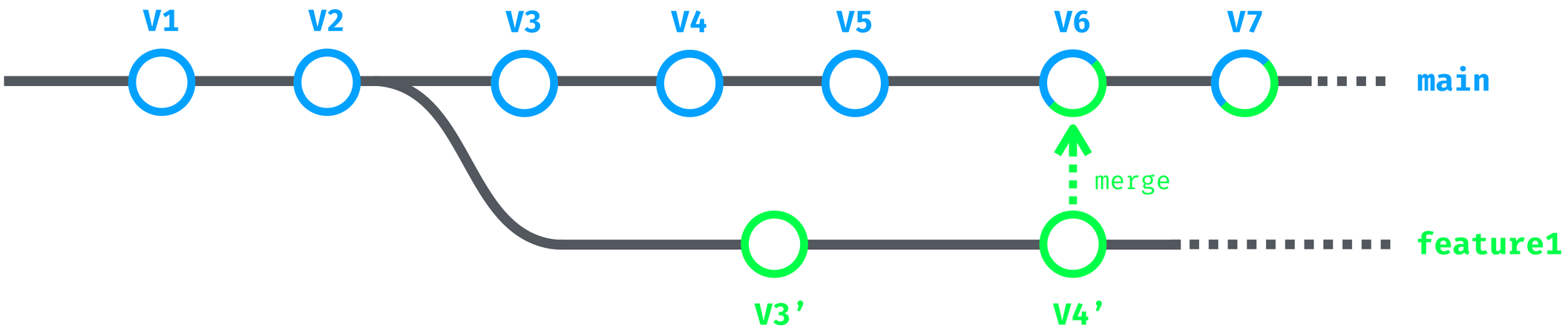
Réunir deux branches :

On se place dans la branche destination et on écrit :

```
# git merge <branche source>
```

Git va alors fusionner les historiques des deux branches

5/ Les branches ???



Après ce merge, **main** contient l'historique des modifications faites dans **feature1**

Exercices

https://qrco.de/bfRK9q

Exercices

Avec un petit récapitulatif :

Cloner un repository :
```
# git clone <url>
```

Se connecter à un repository distant :
```
# git remote add origin <url>
```

Uploader sa dernière version locale :
```
# git push
```

Télécharger la dernière version distante :
```
# git pull
```

Créer une nouvelle branche :
```
# git branch <nom de la branche>
```

Lister les branches :
```
# git branch
```

Changer de branche :
```
# git switch <branche destination>
```

Réunir deux branches :
```
# git merge <branche source>
```

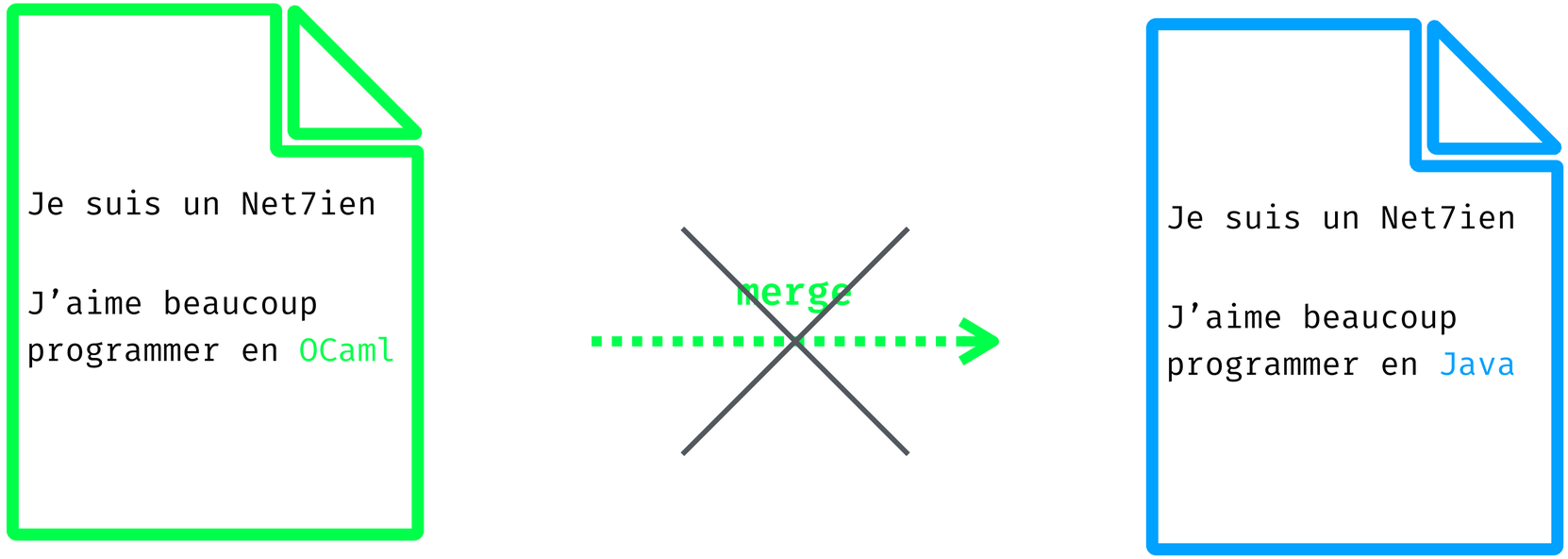
6/ Les conflits ???

6/ Les conflits ???

Si un même fichier a été modifié sur deux branches distinctes ces dernières peuvent entrer en conflit

Git ne pourra alors pas nécessairement résoudre ce conflit seul

6/ Les conflits ???

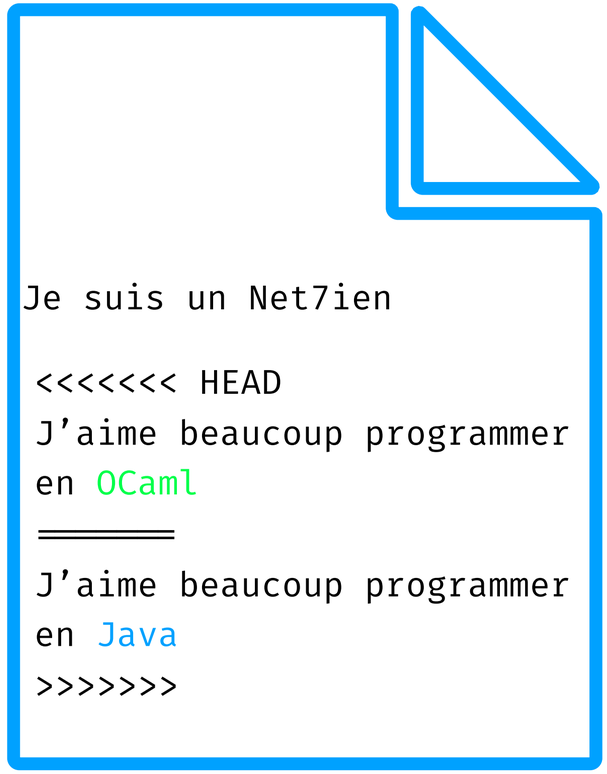


Ici Git ne sait pas si il doit choisir entre OCaml et Java, c’est à l’utilisateur de choisir

6/ Les conflits ???

Git va donc modifier le fichier concerné de la manière suivante :

Et c’est à l’utilisateur de choisir la version qu’il préfère.



6/ Les conflits ???

Certains IDE proposent des outils intégrés pour gérer plus facilement le traitement manuel des conflits

C’est le cas de l’extension Git de VSCode

Bonne pratique pour les éviter : faire régulièrement de petits commits

Quelques ressources :

https://trygit.js.org/cheatsheet/

https://cli.github.com/

https://learngitbranching.js.org

https://gitlab.com/gitlab-org/cli

https://git.inpt.fr

https://github.com/jesseduffield/lazygit

https://git-scm.com/

https://desktop.github.com/download/

Fin de la forma !

https://qrco.de/bfPKlG