GIT

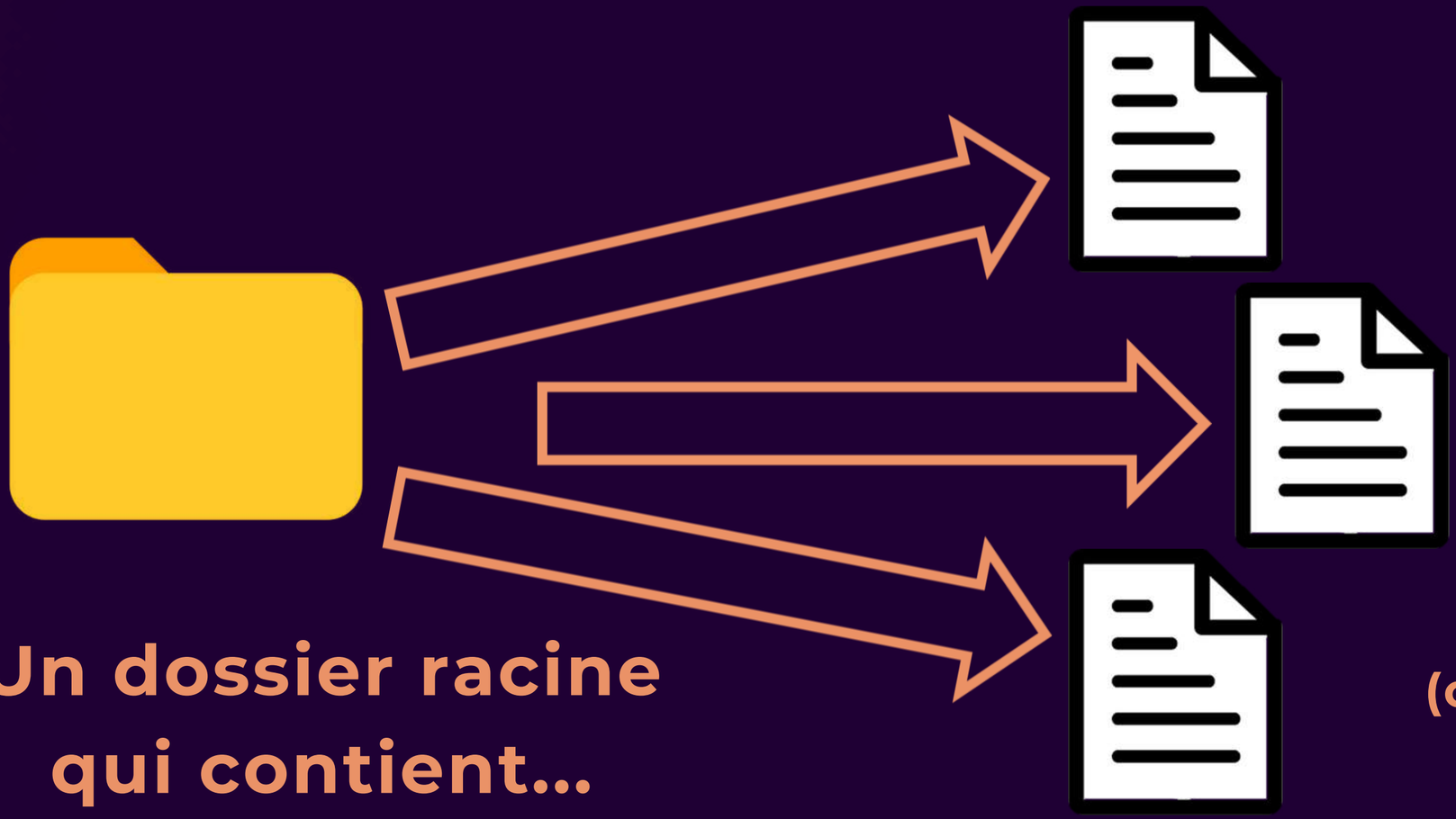
FROM NOOB TO GIGACHAD

git c’est
INDISPENSABLE dès
que tu fais du code

Genre VRAIMENT

Vraiment Vraiment

Un projet de code c’est toujours :



1 Un dossier racine qui contient…

2 …pleins de fichiers (ou d’autres dossiers)

ON PREND UN EXEMPLE

JOUR 1

Projet

Script 1

1) Je crée mon projet
2) J’écris un premier script

ON CONTINUE

JOUR 2

Projet

Script 1 bis

Script 2

1) Je crée un 2ème script
2) Je modifie le premier

JE TENTE QUELQUE CHOSE

JOUR 3

Projet

Script 1 ter

Script 2 bis

Script 3

Je rajoute encore un script, et je modifie les autres

ET LÀ ! JE ME RENDS COMPTE QUE J'AI FAIS NIMP

Projet

Script 1 ter

Script 2 bis

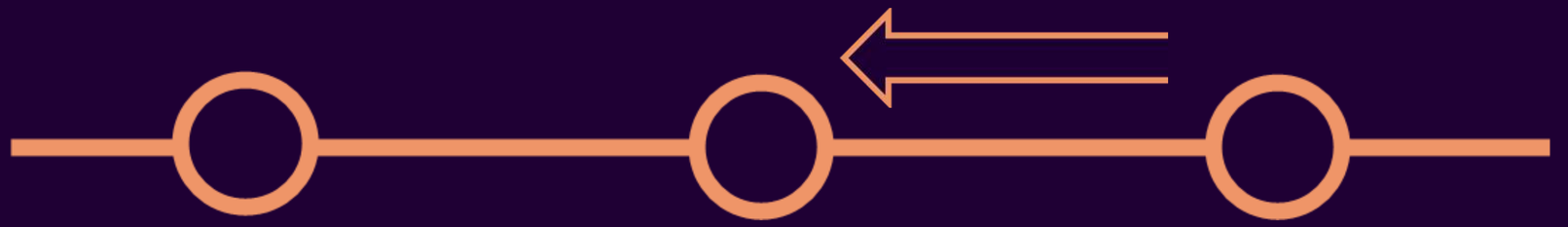
Script 3

J'aimerai revenir à ce que j'avais fais le jour 2

C’est là que GIT intervient !

Git permet de faire des “checkpoints” !

Et on peut revenir à un précédent checkpoint



c’est pratique pour tester des trucs sans risque

En git : checkpoint = commit

A VOUS DE JOUER !

Téléchargez git : **$ sudo apt install git**
Créez un dossier et allez dedans : **$ mkdir dossier**
Ajouter git à votre “projet” : **$ git init**
Créez un fichier et écrivez des trucs dedans :
$ touch fichier puis $ nano fichier

Essayez la commande : **$ ls -a**

Vous verrez un dossier caché .git
Il contient toutes les infos liées à git

A VOUS DE JOUER !

Faites un commit : $ git add * puis $ git commit -m “message”

Rajoutez un autre fichier, modifiez le premier, puis faites un 2ème commit

Afficher l’historique des commits : $ git log

Voilà tous les commits que vous avez jusque ici, avec un hash associé

```
valt@Valt:~/gittest$ git log
commit 0c2fc9df1462154cf6b6a55559c7aaee649392f9 (HEAD -> master)
Author: Valt7 <v.lantigny@gmail.com>
Date:   Sun Oct 13 18:57:34 2024 +0200

    fichier 2

commit 6e7fc2914a1414fe20c1fe58423562c9a0dfdb1f
Author: Valt7 <v.lantigny@gmail.com>
Date:   Sun Oct 13 18:56:45 2024 +0200

    fichier 1
```

A VOUS DE JOUER !

Copiez le hash du premier commit
Revenez au premier commit :
$ git reset --hard “hash”

Affichez vos fichiers, vous verrez que tout a été réinitialisé : **$ ls** puis **$ nano fichier**

Essayez de revenir au commit 2

Et si je veux...

Collaborer sur un projet avec d’autres personnes ?

Et si je veux...

Eviter que mon projet disparaisse si mon PC meurt ?

Et si je veux...

Automatiser le déploiement de mon projet ?

La solution ?
Un hébergeur de dépôts Git (repositories)

Centraliser le dév des projets
Partager le code, collaborer

Dépôts Privés ou publics

POUSSER LES MODIFS SUR LE DÉPÔT

Ajouter le dépôt distant

$ git remote add origin
https://gitlab.com/utilisateur/projet.git

Pousser les modifications

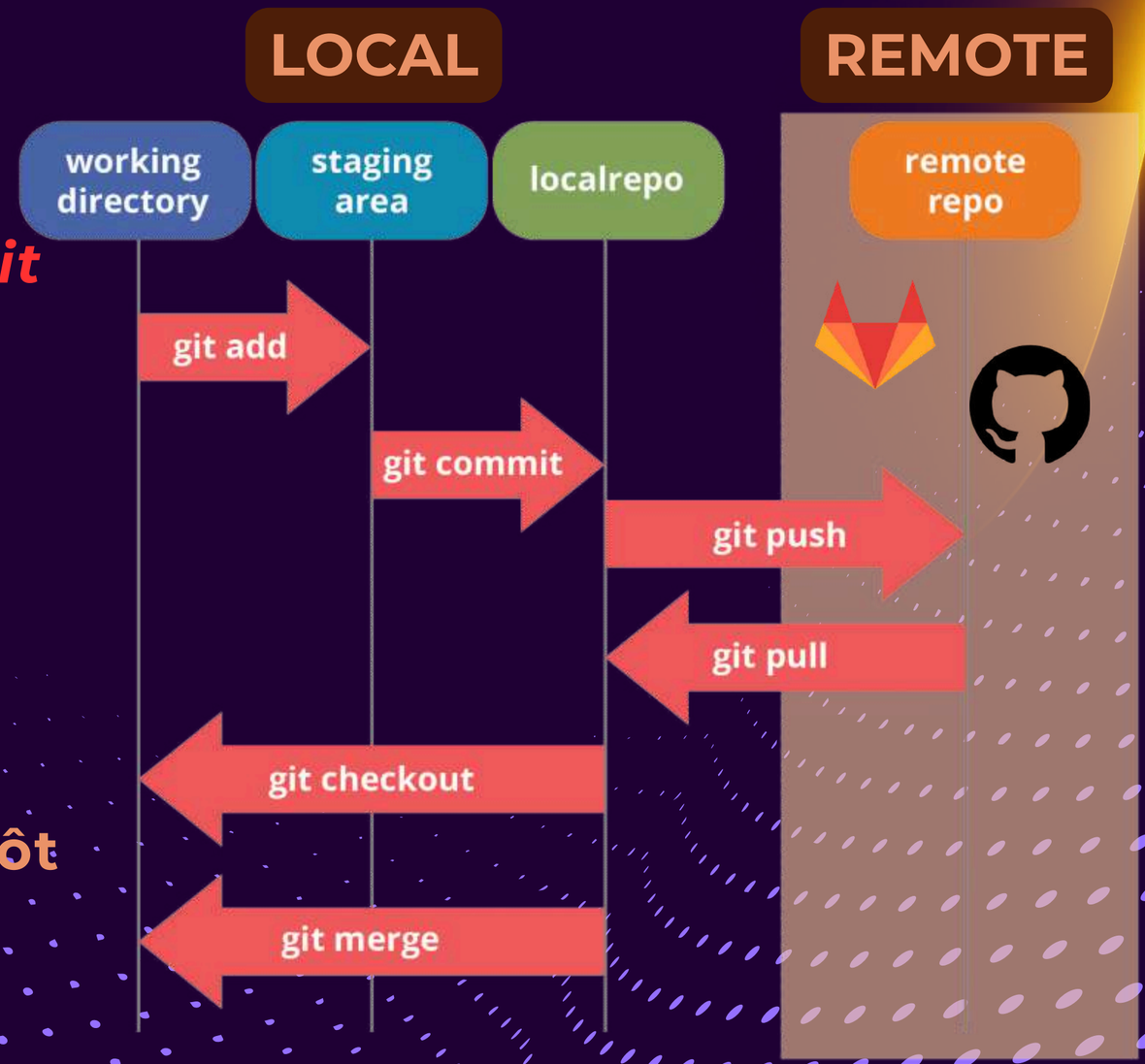
$ git push origin main

Lister les dépôts distants

$ git remote -v

Récupérer les modifications sur le dépôt

$ git pull origin main

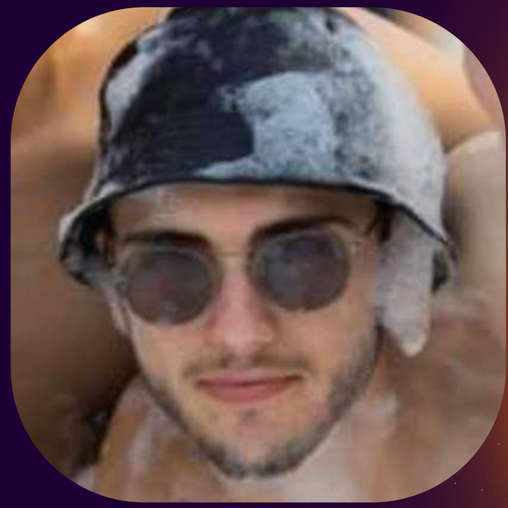


On récapitule

Raphaël crée un projet et le publie sur le dépôt Gitlab

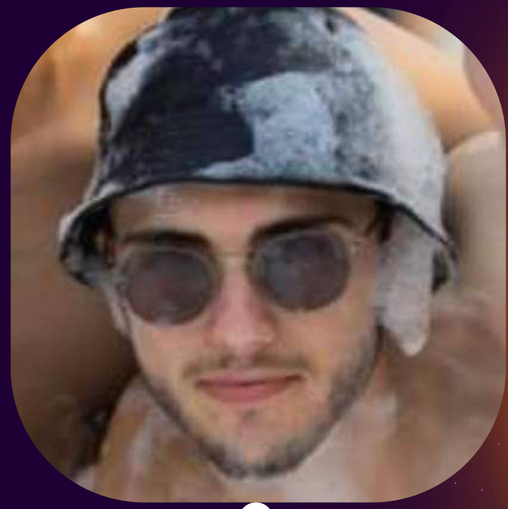
$ git init
*$ git add **
$ git commit -m “First commit”
$ git remote add origin https://gitlab.com/notre-projet.git
$ git push origin main

On récapitule



git push

On récapitule



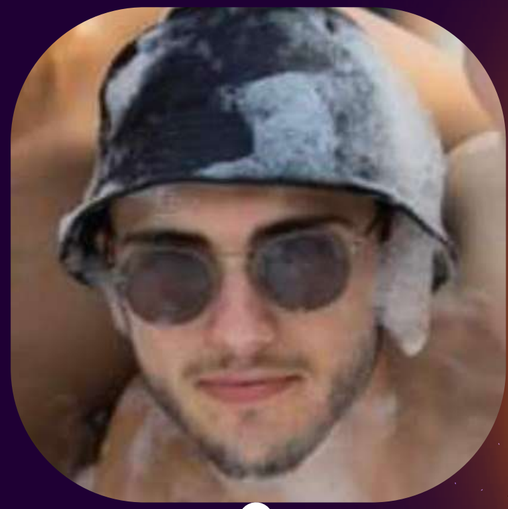
git clone

git clone

Zoé et moi récupérons le projet
sur notre PC

$ git clone
https://gitlab.com/notre-projet.git

On récapitule



Je fais une modification

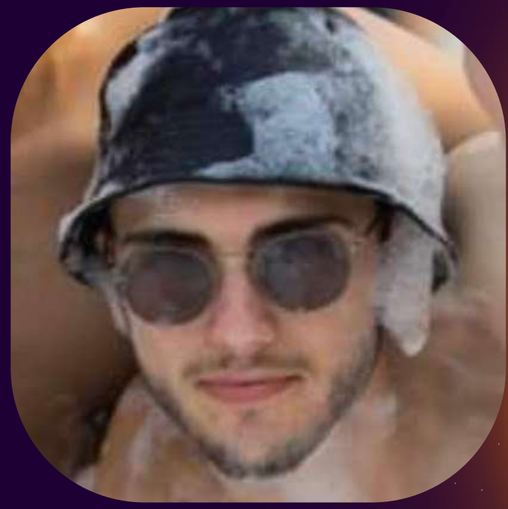
$ git commit -m "Ajout nouvelle fonctionnalité"

A la fin de ma session, je fais

$ git push origin main

git push

On récapitule



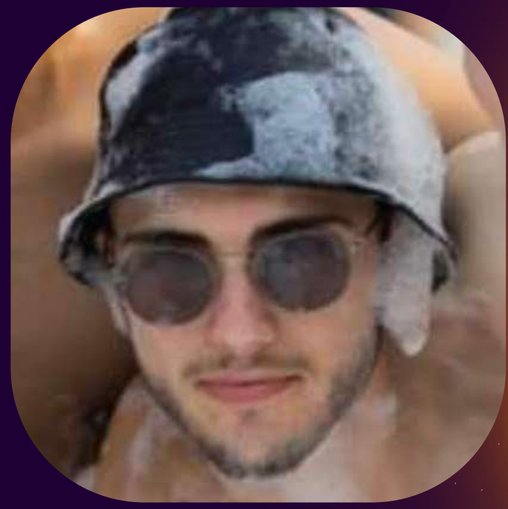
git pull

git pull

Raph et Zoé reprennent le projet,
ils récupèrent la dernière version

$ git pull origin main

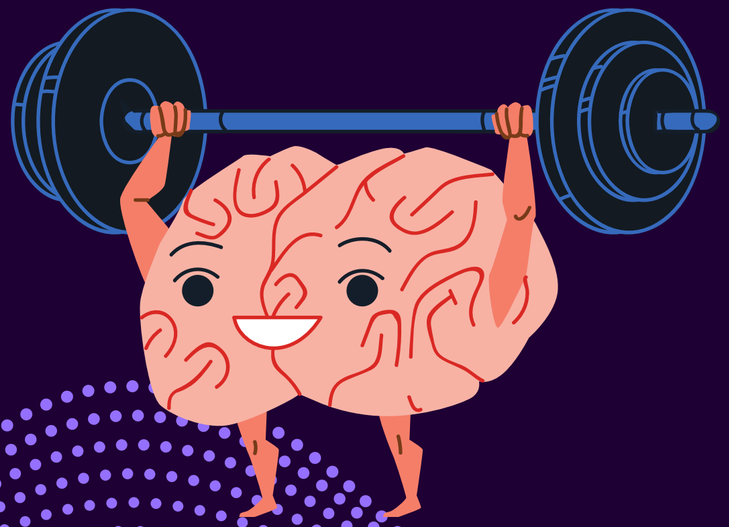
On récapitule



Une fois la session de code terminée, on PUSH nos modifications, etc.

A toi de jouer

Crée un compte sur gitlab.minet.net
publie ton dépôt local
(définir le remote, push...)
supprime-le à la fin

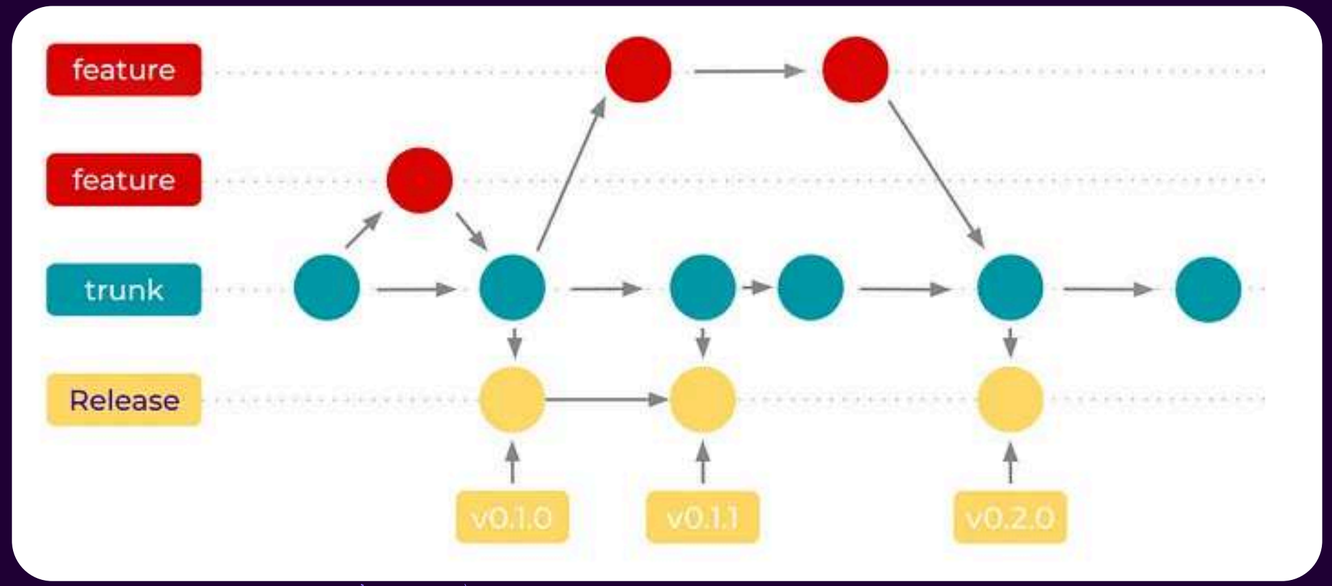


OK… J’en sais un peu plus sur Git

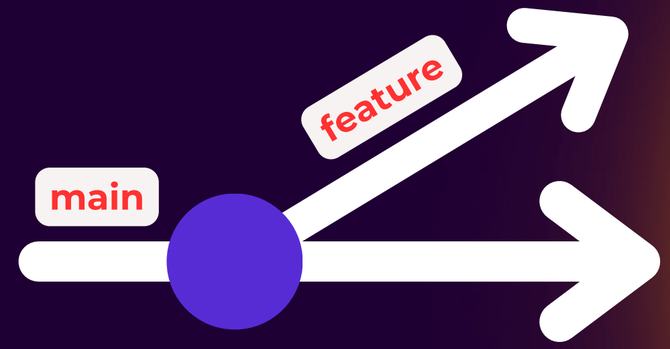
Et si je veux…

Développer des nouveautés sans casser la version en prod

Le branching



Un exemple de workflow
(trunk-based development)



$ git checkout -b feature

(git branch feature
git checkout feature)

Lister les branches :
$ git branch

**Branche principale,
Branches secondaires pour
les nouvelles fonctionnalités**

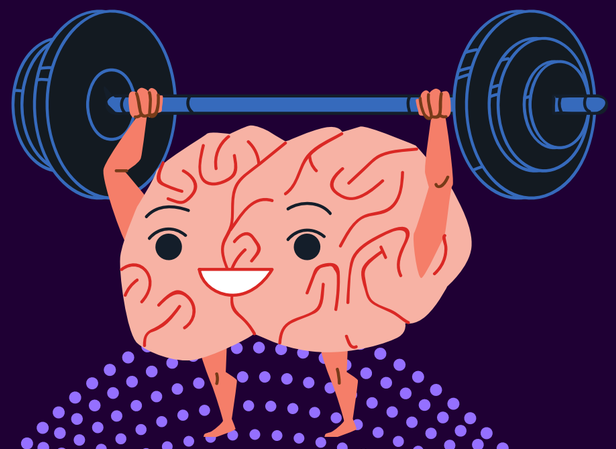
Fusionner des branches : Le MERGE

On a terminé la feature,
On veut fusionner la branche
feature sur la branche main

$ git checkout main
$ git merge feature

A toi de jouer

Sur ton dépôt local,
Crée une branche avec un nom de feature de ton choix
Ajoute un fichier txt et fais un commit
Reviens sur la branche principale et observe
Merge ta branche feature et observe



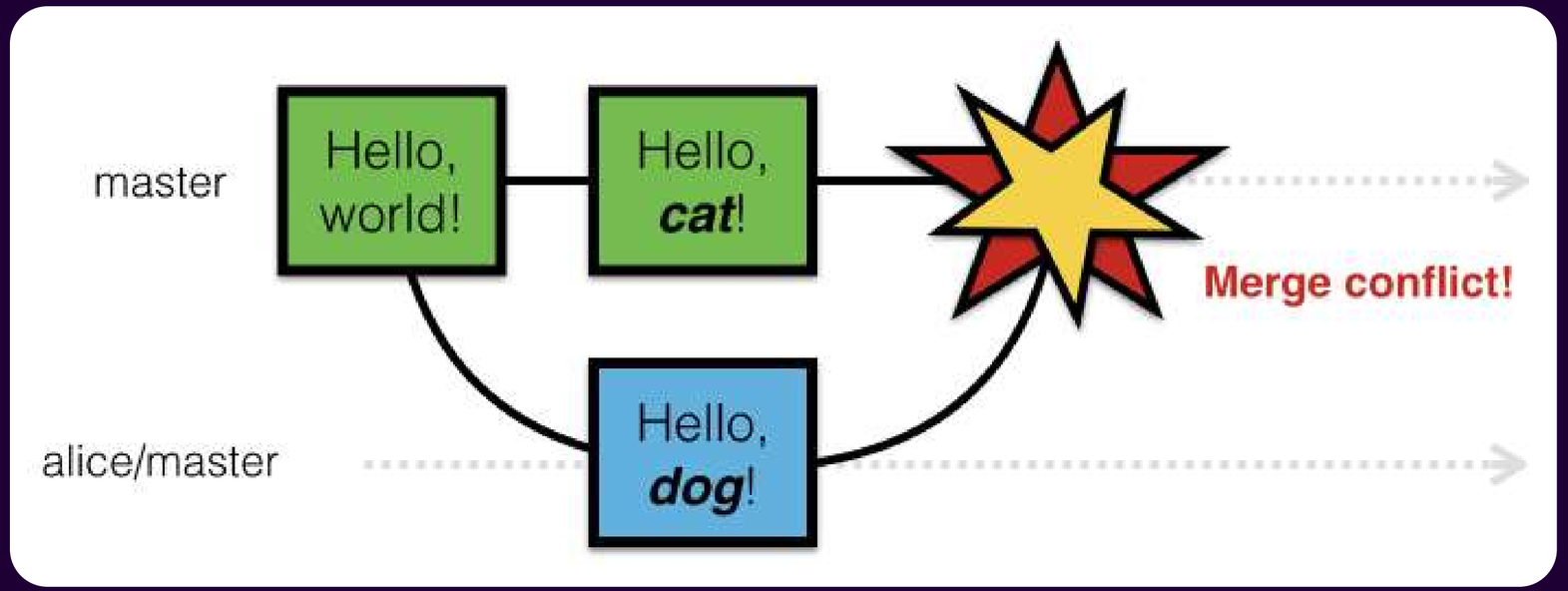
Pense bête

```
git checkout -b myfeature
touch file.txt
nano file.txt (editer)
git commit -m “message”’
git checkout branche
git merge brancheafusionner
```

Dernier truc à savoir sur git :

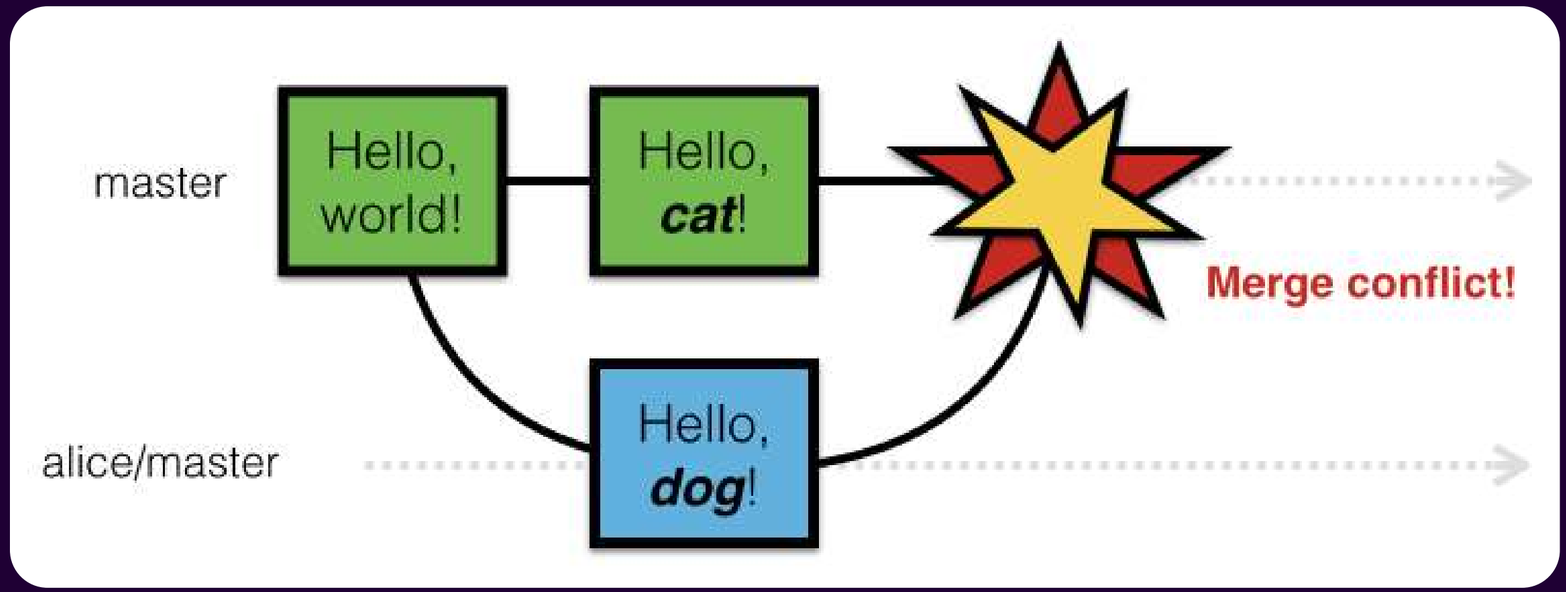
LES MERGE CONFLICTS

Si vous êtes plusieurs à bosser sur le même projet



Y’a un moment où vous allez modifier le même fichier chacun de votre côté

Dans la “plupart des cas”, Git est intelligent et fusionne automatiquement



Mais des fois, c’est pas possible

En résumé :

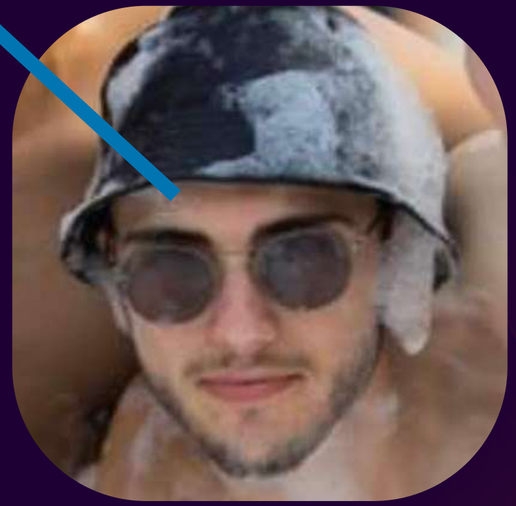

Automatic merge failed
Fix conflicts and then commit the result
CHANGELOG

Donc quand on fusionne 2 branches divergentes, les dévs doivent choisir quelle ligne on garde au final

On prend le mien !

Hello, Cat!

Non le mien est mieux !



Hello, Dog!


Ici la réponse est évidente

CRÉER SON MERGE CONFLICT

Choisissez un fichier présent sur vos 2 branches master et test
Pour changer de branche : **$ git switch branchName**

Modifiez une ligne de ce fichier sur master : **$ nano fichier** puis **$ git add *** puis **$ git commit**

Allez sur la branche test : **$ git switch test**
Modifier le même fichier, la même ligne, mais avec un texte différent

CRÉER SON MERGE CONFLICT

Commitez, puis allez sur master : $ git switch master
Fusionnez votre branche sur master :
$ git merge test

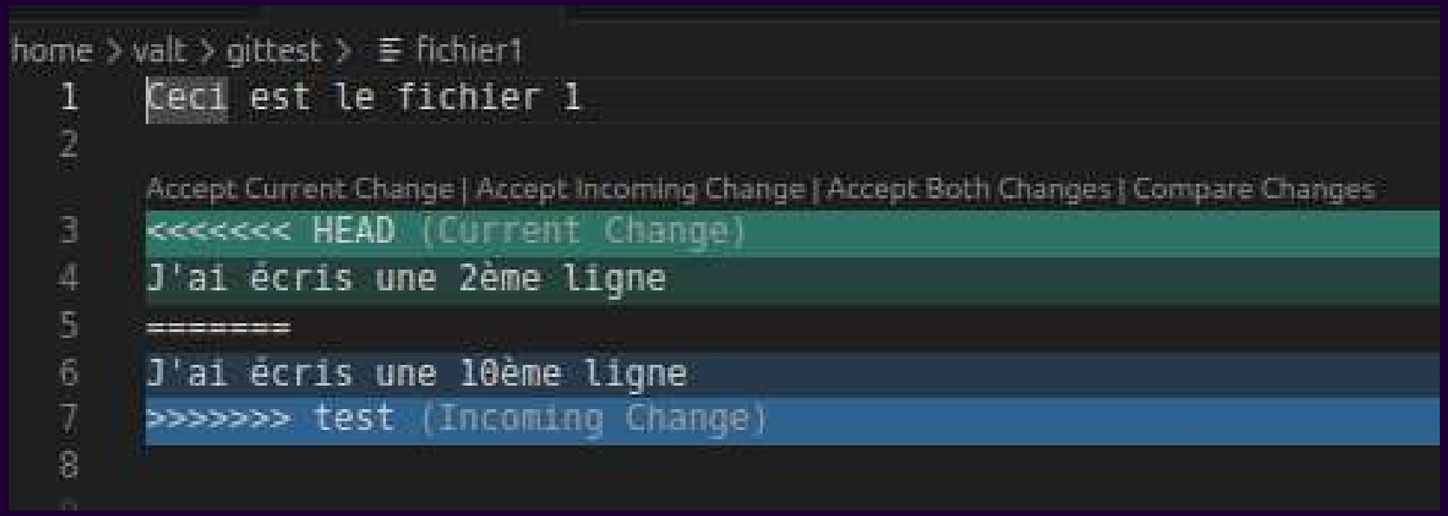
```
valt@Valt:~/gittest$ git merge test
Fusion automatique de fichier1
CONFLIT (contenu) : Conflit de fusion dans fichier1
La fusion automatique a échoué ; réglez les conflits et validez le résultat.
```

Si vous avez tout “bien” fait, git ne saura quelle ligne choisir entre vos 2 branches

Pour résoudre le conflit, c’est bien d’utiliser VSCode

CRÉER SON MERGE CONFLICT

Dans VSCode, faites : File > Ouvrir > “votreFichier”
VSCode vous affiche les 2 versions de votre fichier



Vous pouvez accepter l’un, l’autre, ou les 2
Sauvegardez, puis commitez

MAIS NAN !!!

VOUS AVEZ RÉSOLU VOTRE PREMIER MERGE CONFLICT !

Le premier d’une longue série...

GIT

C’est fini… (pour ajourd’hui)