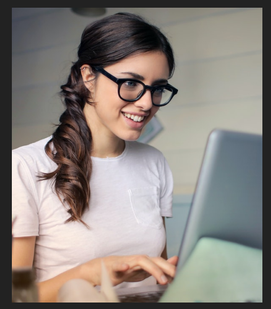
MOTIVATION

# GIT

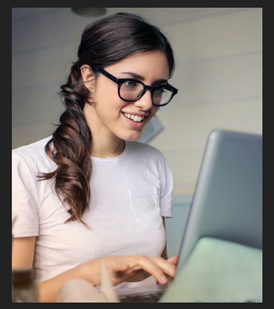
# ALICE TRAVAILLE SUR UN LOGICIEL

- Elle souhaite **sauvegarder son travail**
- Elle souhaite pouvoir **revenir en arrière** quand elle fait une erreur



mon-projet/

a.py

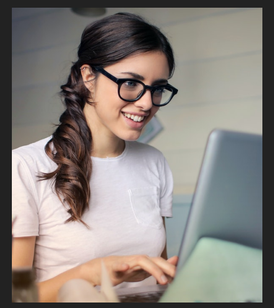


mon-projet/

a.py

Copie de travail (« Working Copy »)

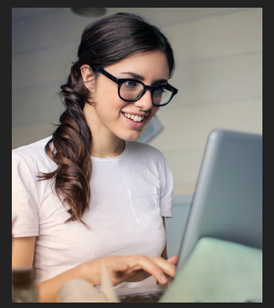
sauvegardes/



mon-projet/

a.py

Copie de travail (« Working Copy »)



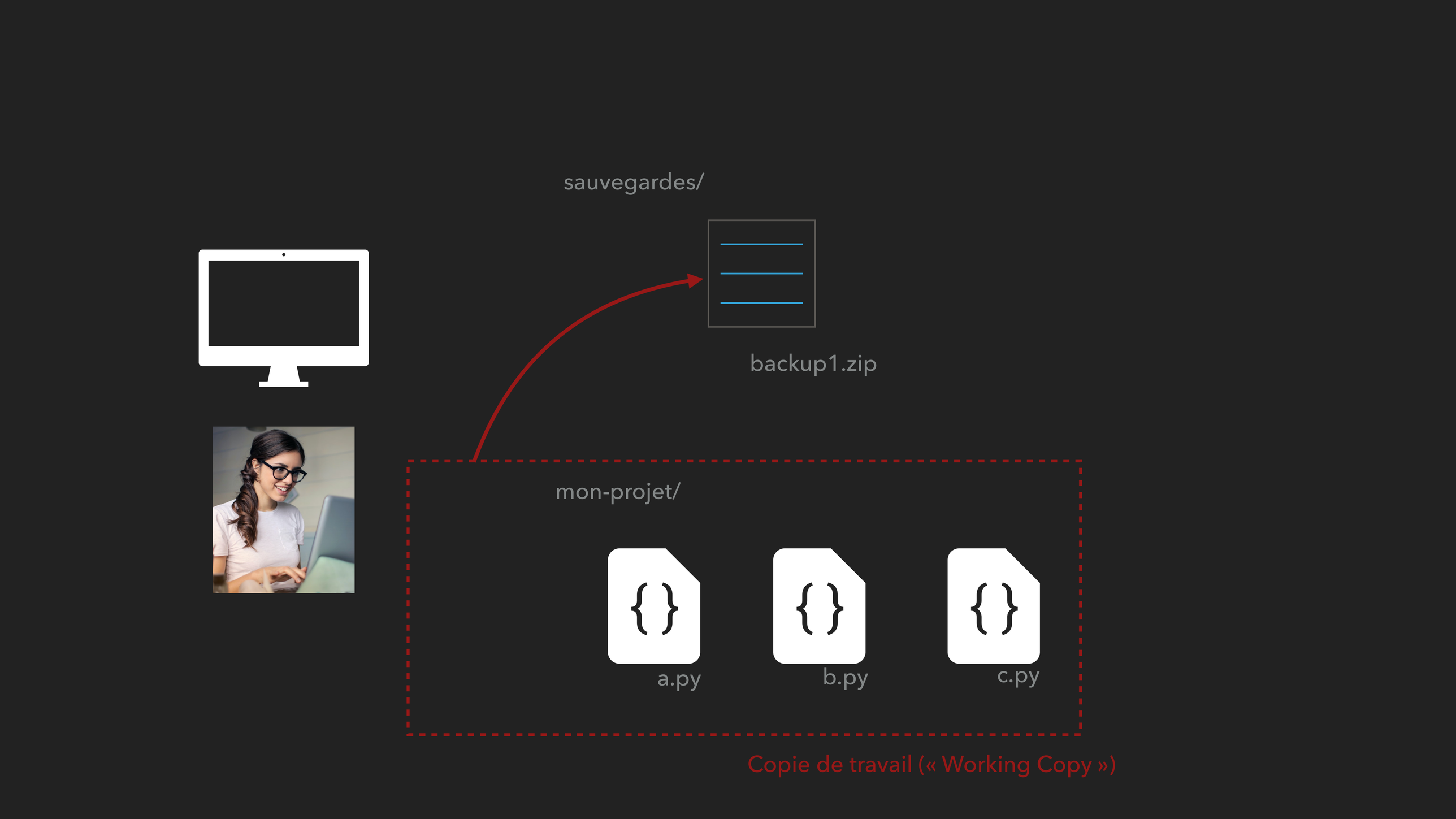
sauvegardes/

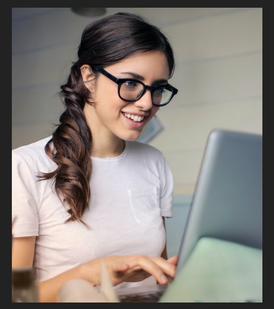
backup1.zip

mon-projet/

a.py

Copie de travail (« Working Copy »)


sauvegardes/
backup1.zip
mon-projet/
a.py
b.py
c.py
Copie de travail (« Working Copy »)



sauvegardes/

backup1.zip

backup2.zip

mon-projet/

a.py

b.py

c.py

Copie de travail (« Working Copy »)

# PROBLÈMES

A. Il est très dur de **savoir ce qui a changé**

B. Ça prend vite **beaucoup de place**

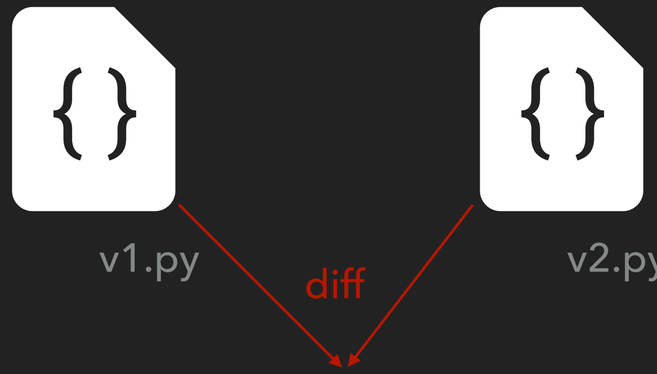
C. C’est très fastidieux, risque d’oubli

# COMPARAISON AVEC DIFF

v1.py

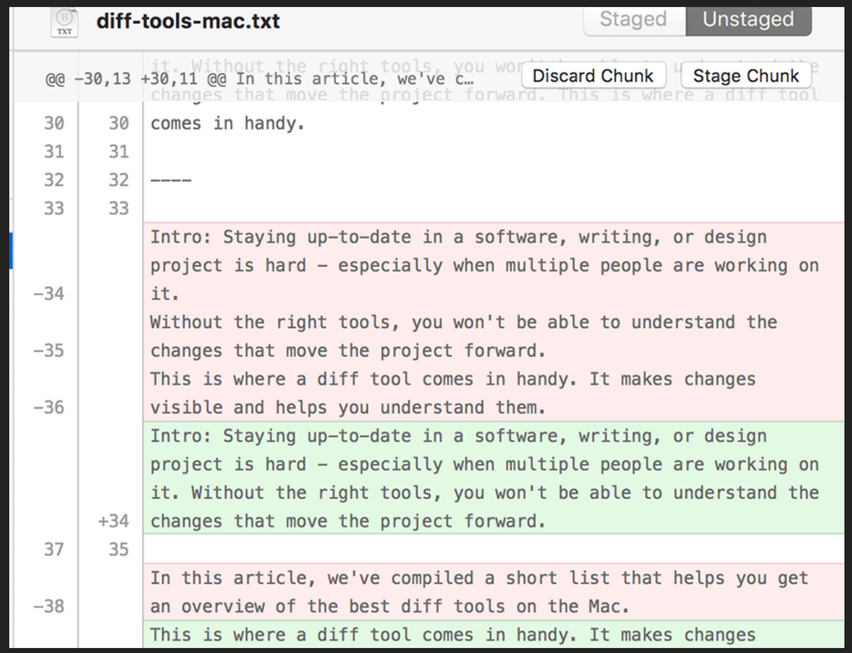
v2.py

# COMPARAISON AVEC DIFF




```
diff --git a/diff-colorize.py b/diff-colorize.py
old mode 100755
new mode 100644
--- a/diff-colorize.py
+++ b/diff-colorize.py
@@ -5,9 +5,9 @@
 import fileinput

 index_color      = int(os.environ.get('DIFF_INDEX_COLOR', 32))
-old_mode_color   = int(os.environ.get('DIFF_OLD_MODE_COLOR', 124))
+old_mode_color   = int(os.environ.get('DIFF_OLD_MODE_COLOR', 88))
 new_mode_color   = int(os.environ.get('DIFF_NEW_MODE_COLOR', 28))
-removed_color    = int(os.environ.get('DIFF_REMOVED_COLOR', 203))
+removed_color    = int(os.environ.get('DIFF_REMOVED_COLOR', 160))
 added_color      = int(os.environ.get('DIFF_ADDED_COLOR', 2))
 hunk_start_color = int(os.environ.get('DIFF_HUNK_START_COLOR', 32))
```



# LOGICIEL DE GESTION DE VERSIONS

- Utiliser un outil dédié
- Permet de rendre la sauvegarde plus rapide, d'acquérir des automatismes
- Ce qu'on a décrit correspond à l'outil RCS (1982)

# BOB SOUHAITE AIDER ALICE

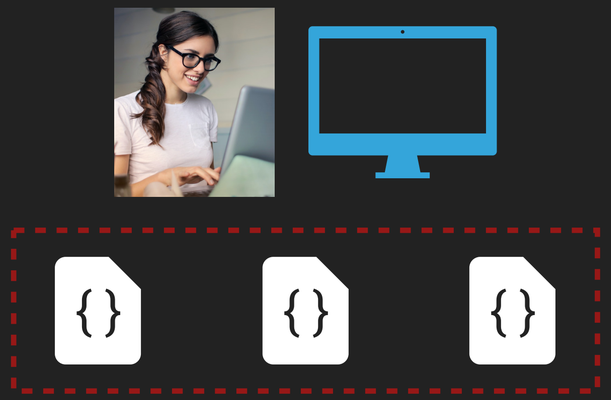
- Il souhaite pouvoir travailler **en même temps** qu'Alice sur le logiciel
- Ils doivent donc **synchroniser** leur travail régulièrement

# SOLUTIONS INTUITIVES

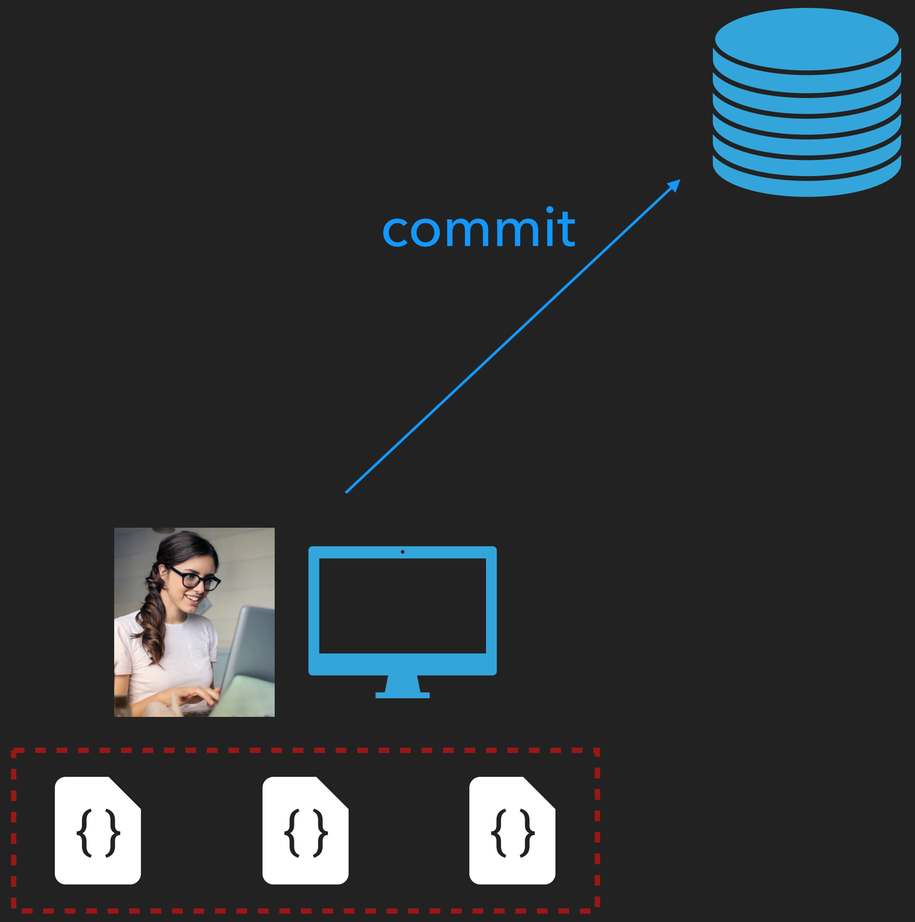
- Utiliser un disque partagé
- S’envoyer les changements par email

- On risque « d’écraser" les changements de l'autre
- Ça fait beaucoup d’emails…

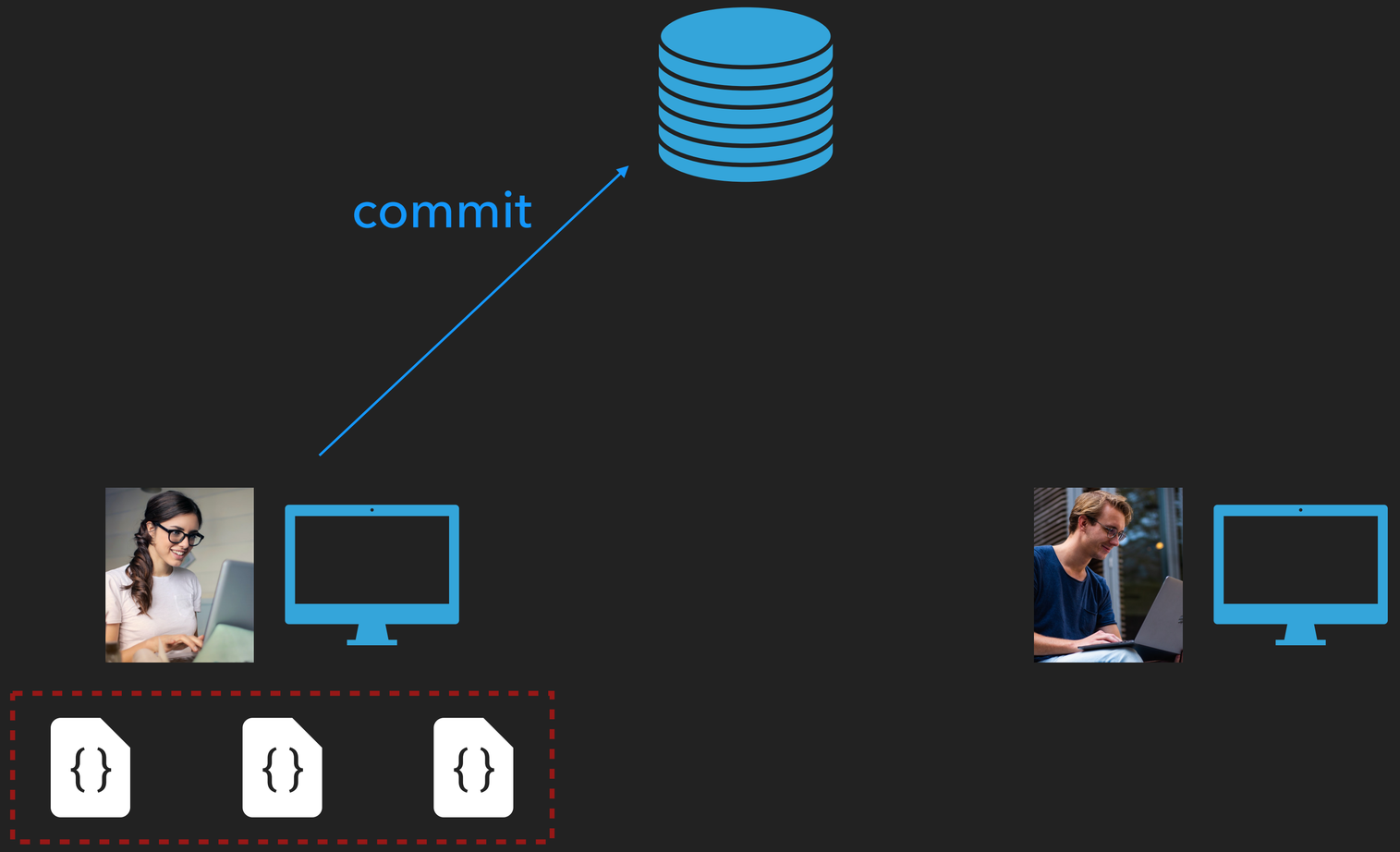
# SOLUTION: UTILISER UN OUTIL DE GESTION DE VERSIONS« AVEC SERVEUR »



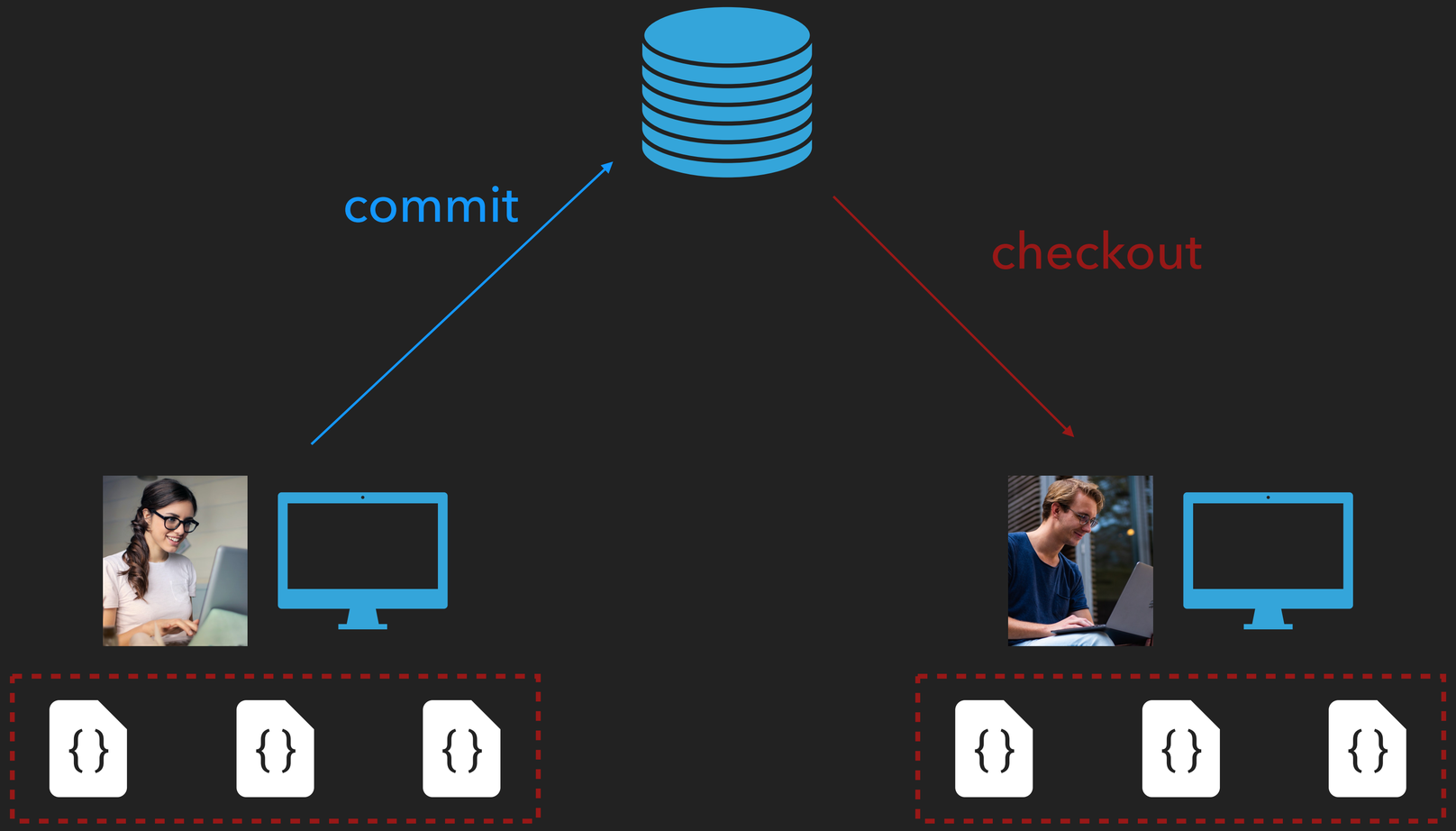
# SOLUTION: UTILISER UN OUTIL DE GESTION DE VERSIONS« AVEC SERVEUR »



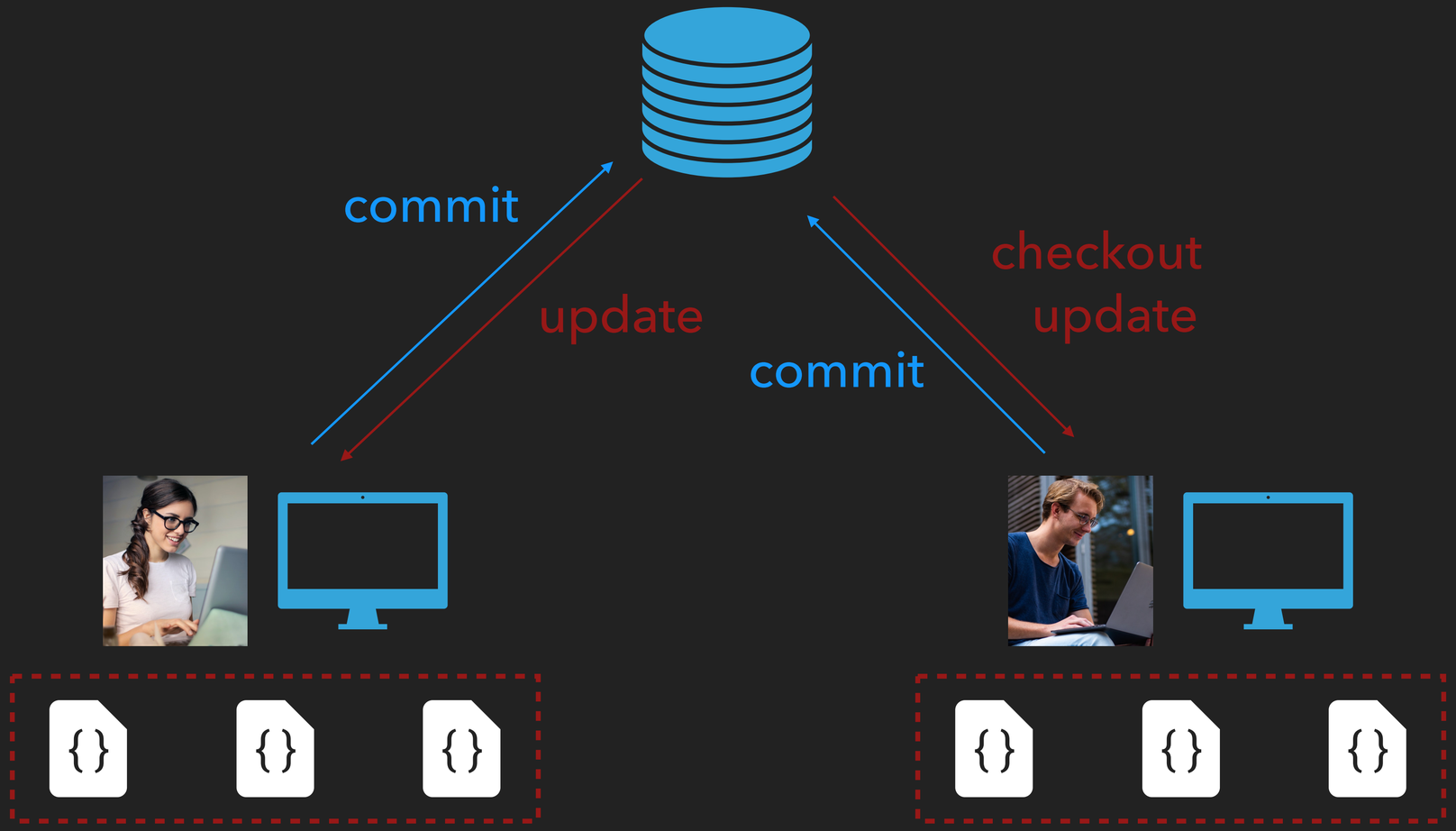
# SOLUTION: UTILISER UN OUTIL DE GESTION DE VERSIONS« AVEC SERVEUR »



# SOLUTION: UTILISER UN OUTIL DE GESTION DE VERSIONS« AVEC SERVEUR »



# SOLUTION: UTILISER UN OUTIL DE GESTION DE VERSIONS« AVEC SERVEUR »



# AVANTAGES / INCONVÉNIENTS

- L’outil encadre le process:
  pas d’email à envoyer !
- On récupère très facilement
  la dernière version
- Gestion automatisée des « conflits »

- Impossible de sauvegarder
  ses changements sans réseau
- Il y a un risque de « Hacking"
  du serveur

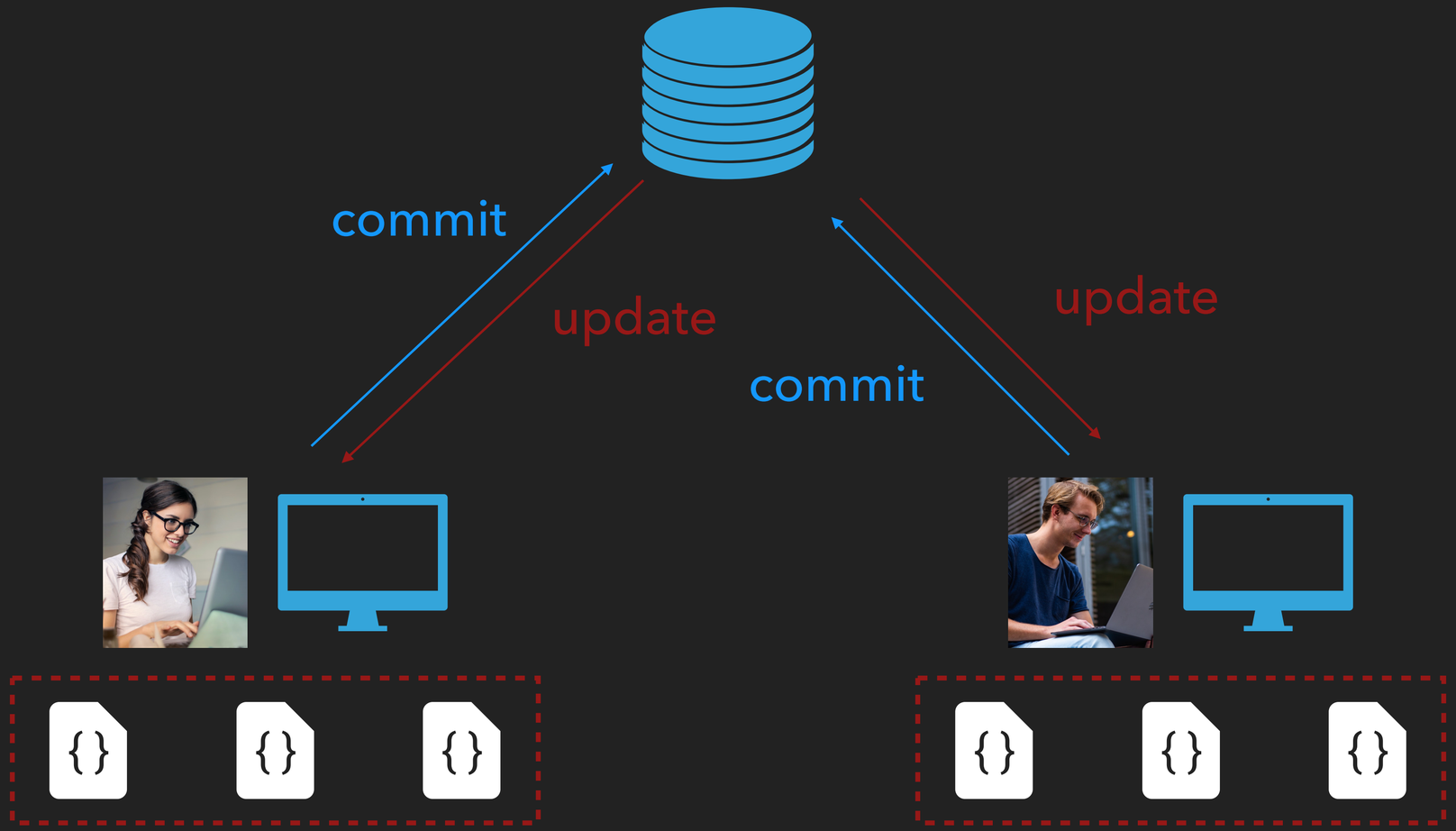
# IL FAUT UN OUTIL PLUS PUISSANT

# IL FAUT UN OUTIL PLUS PUISSANT

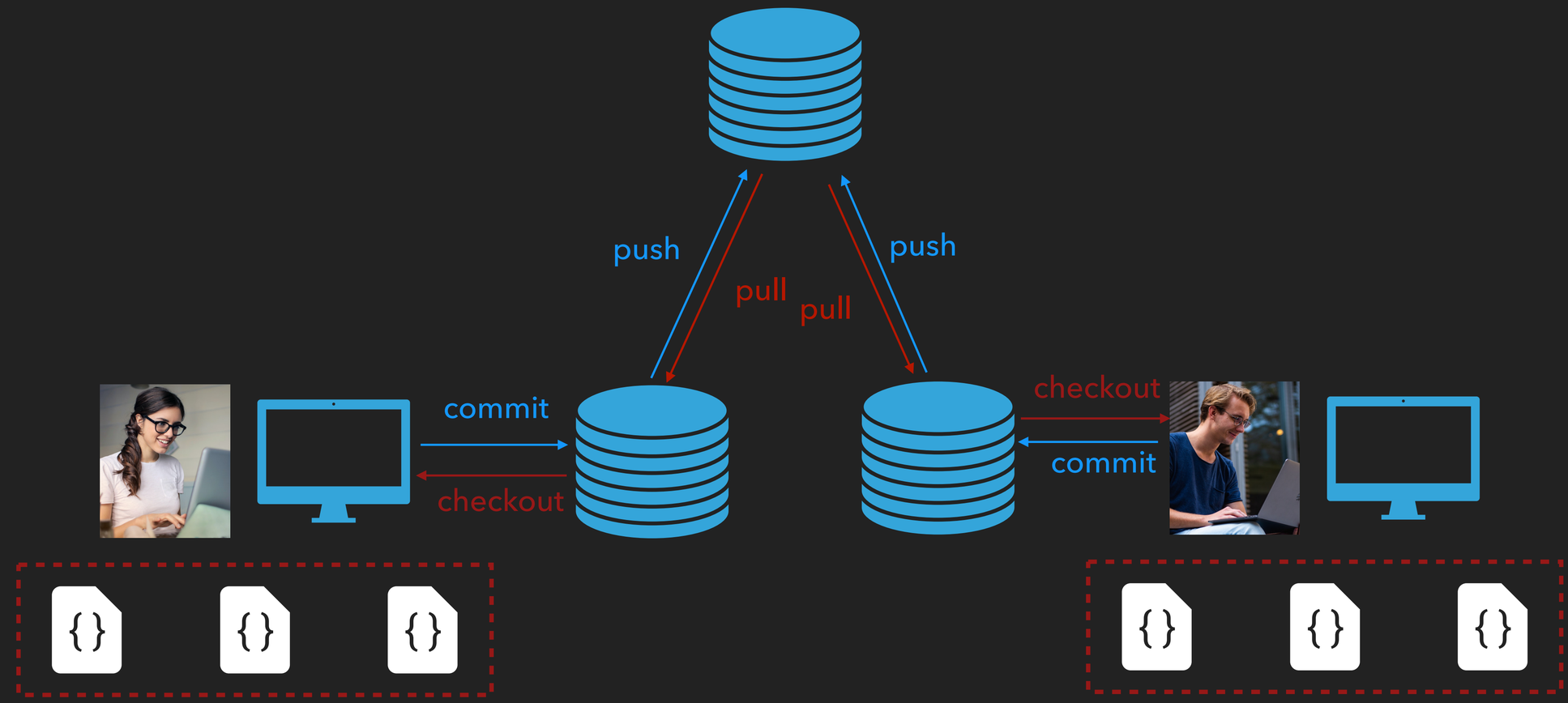
# ON TRANSFORME L'ARCHITECTURE

- on remplace le modèle centralisé
  - **un dépôt central**, n clients
- en un modèle décentralisé
  - **autant de dépôts** que de clients

# MODÈLE CENTRALISÉ = 1 DÉPÔT



# GIT: CHACUN VOIT 2 DÉPÔTS (LOCAL + DISTANT)



# CE QUI A CHANGÉ

- **chacun** a son propre **dépôt** local et peut donc travailler **sans réseau**
- on peut **synchroniser** à tout moment **les dépôts** (local/distant) entre eux

# AUTRES AVANTAGES

- git est sécurisé: on détecte facilement toute modification non autorisée
- git est extensible: on peut lui brancher un grand nombre d’outils annexes

# MAIS

- c’est un outil **complexe**: il y a énormément de commandes
- et pas toujours très cohérentes entre elles
  - ex: pour détruire une branche `$ git branch -d mybranch`
  - pour détruire un remote `$ git remote remove myremote`
- ne pas se priver d’utiliser une GUI (vs-code, sourcetree, …)
  - mais d’abord **bien comprendre les bases**

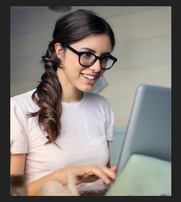
EN REPARTANT DE ZÉRO

# PETIT CAS PRATIQUES

# 2 SCÉNARIOS

- on va étudier les 2 cas d'usage les plus fréquents
- la séquence de démarrage est la même dans les 2 cas
  - Alice publie son travail
    Bob clone le travail d'Alice
    Bob fait une correction, il veut la transmettre à Alice
- les deux scénarios diffèrent selon qu'Alice connait/fait confiance à Bob, ou pas

# ALICE CRÉE UN PROJET



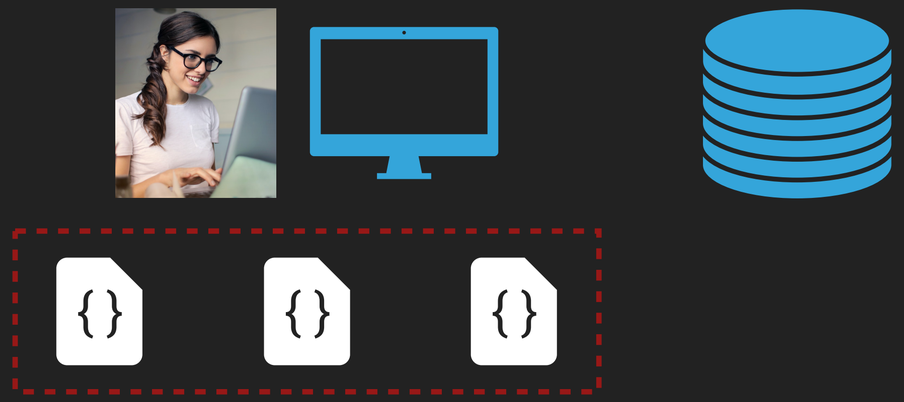
# ALICE CRÉE UN PROJET

- Elle crée une première version sur son ordinateur

# ALICE CRÉE UN PROJET

- Elle crée une première version sur son ordinateur

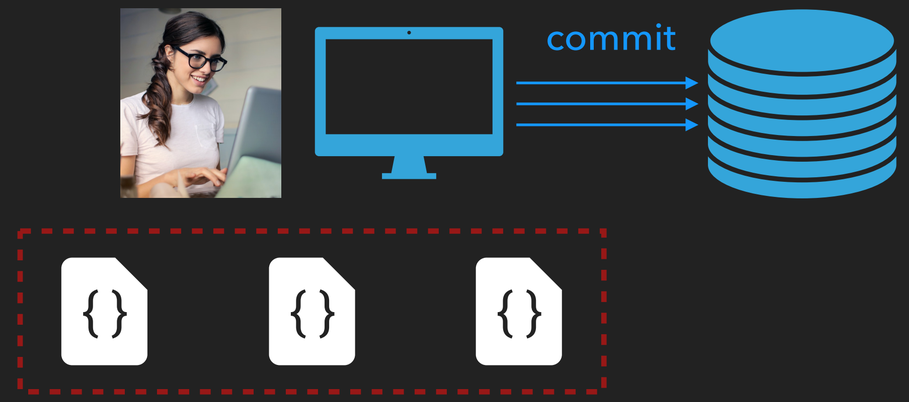
- Puis un dépôt git **local** pour travailler,
  et versionner son travail



# ALICE CRÉE UN PROJET

- Elle crée une première version sur son ordinateur

- Puis un dépôt git **local** pour travailler,
  et versionner son travail

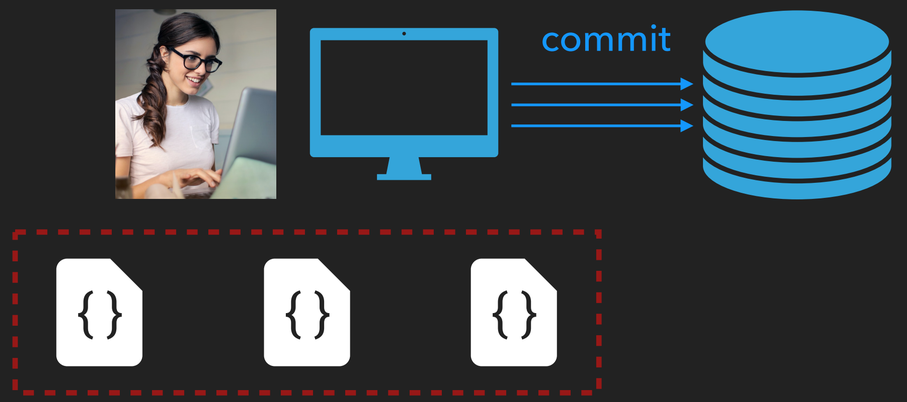


# ALICE CRÉE UN PROJET

- Elle crée une première version sur son ordinateur

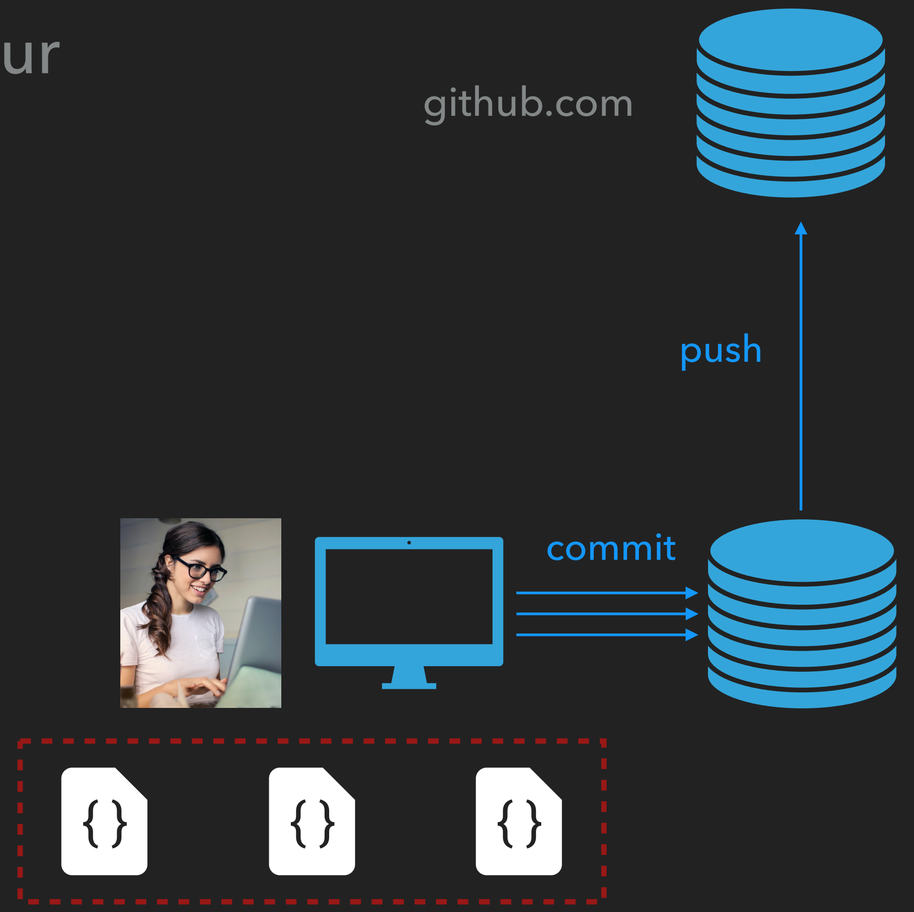
- Puis un dépôt git **local** pour travailler,
  et versionner son travail

- Elle crée ensuite un dépôt **public** sur
  github.com pour « open-sourcer » son projet

# ALICE CRÉE UN PROJET

- Elle crée une première version sur son ordinateur
- Puis un dépôt git **local** pour travailler,
  et versionner son travail
- Elle crée ensuite un dépôt **public** sur
  github.com pour « open-sourcer » son projet
- Et téléverse (« push ») son code
  sur le serveur GitHub

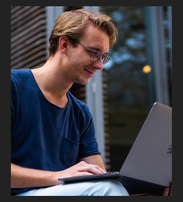


# BOB REMARQUE LE TRAVAIL D'ALICE

# BOB REMARQUE LE TRAVAIL D'ALICE

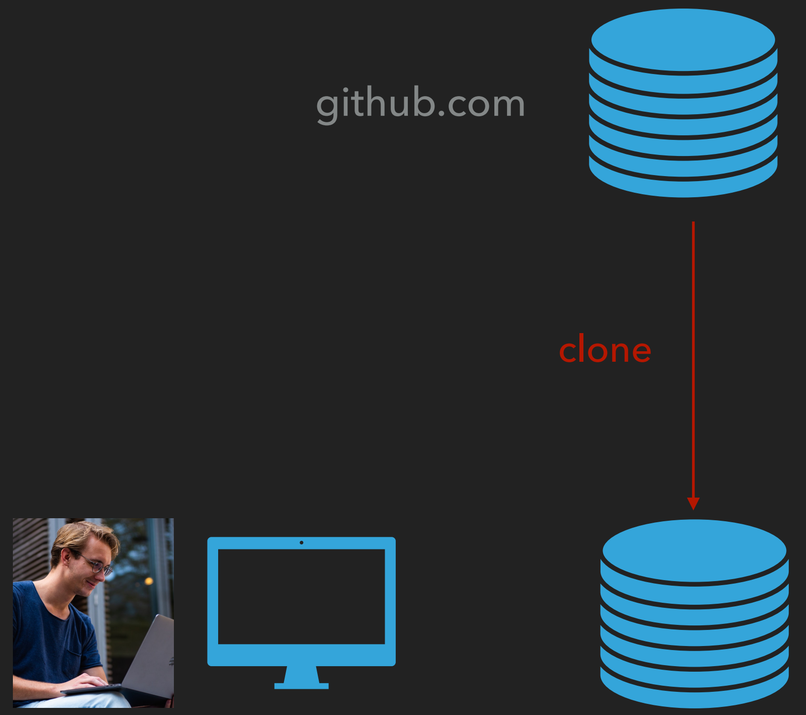
- Il remarque le travail d'Alice (via Google) et souhaite l'utiliser

github.com



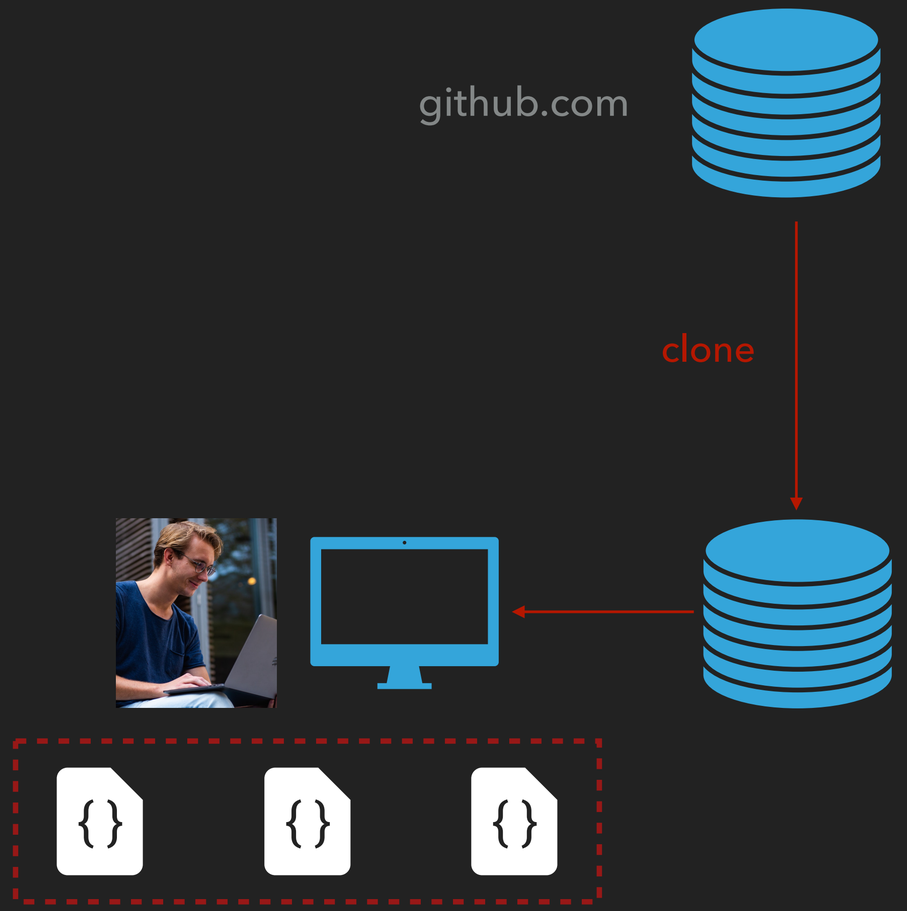
# BOB REMARQUE LE TRAVAIL D'ALICE

- Il remarque le travail d'Alice (via Google) et souhaite l'utiliser
- Il copie le dépôt distant localement

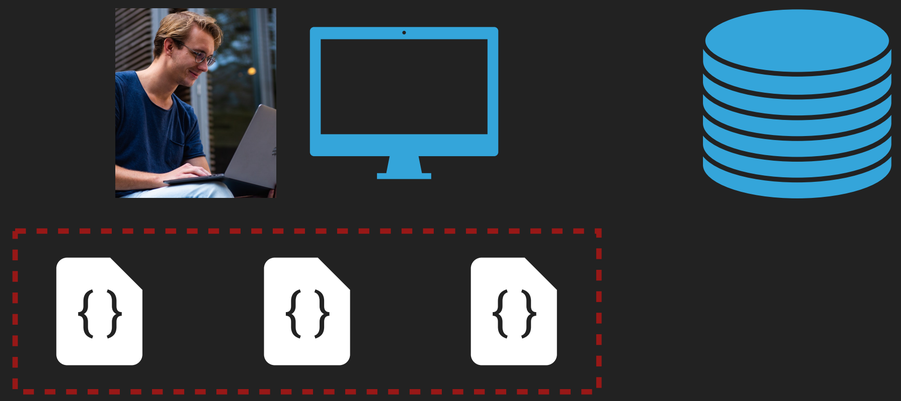


# BOB REMARQUE LE TRAVAIL D'ALICE

- Il remarque le travail d'Alice (via Google) et souhaite l'utiliser
- Il copie le dépôt distant localement
- Il peut désormais accéder aux fichiers
- (ainsi qu'à toutes les différentes versions)

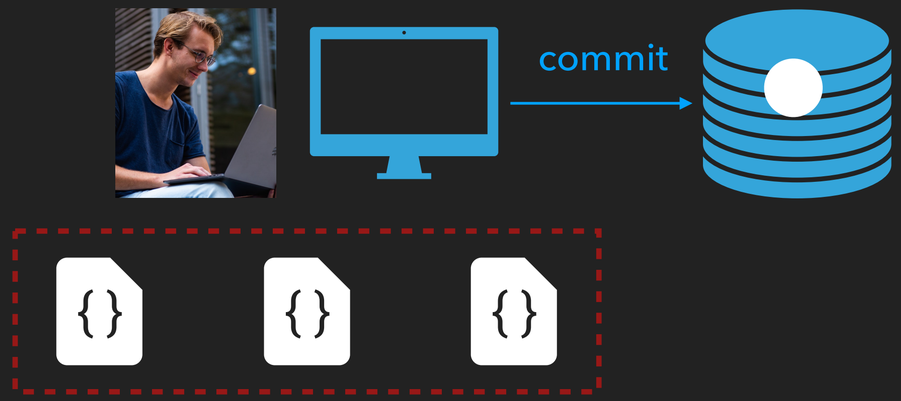


# BOB A TROUVÉ UN BUG (!)



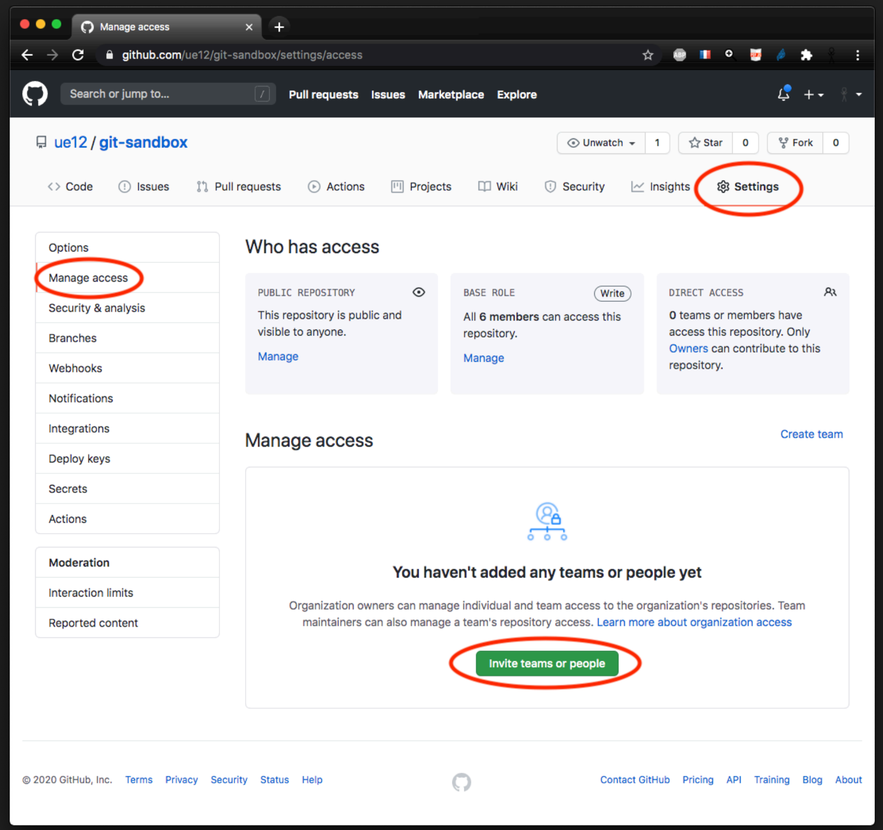
# BOB A TROUVÉ UN BUG (!)

- il corrige **localement** le logiciel
  - sans souci, le clone lui appartient
- pour transmettre la correction à Alice, le workflow dépend du scénario
  - selon que Bob a ou non le droit d'écrire dans le dépôt d'Alice



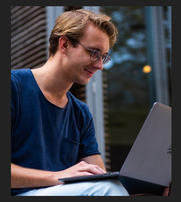
# SCÉNARIO #1 : DROITS D'ACCÈS PARTAGÉS

- Alice et Bob se connaissent, ils travaillent ensemble tous les jours
  - Alice donne à Bob **le droit d'écrire** dans son dépôt github
  - ils peuvent utiliser le dépôt github pour partager et synchroniser leur travail

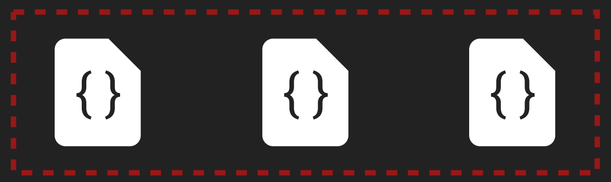


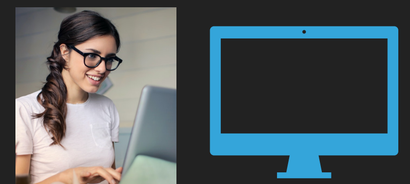
# BOB PEUT ÉCRIRE DANS LE DÉPÔT D'ALICE

github.com
(Alice)



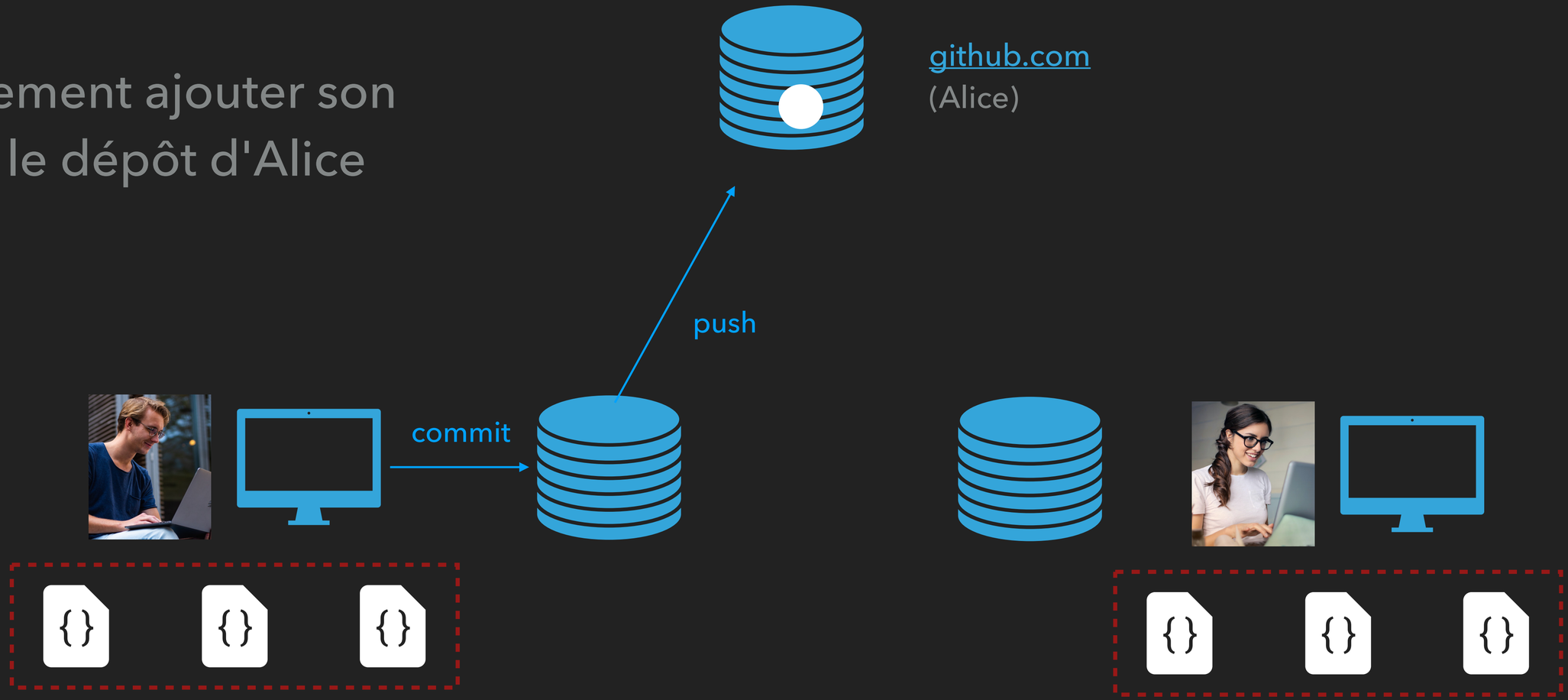
commit





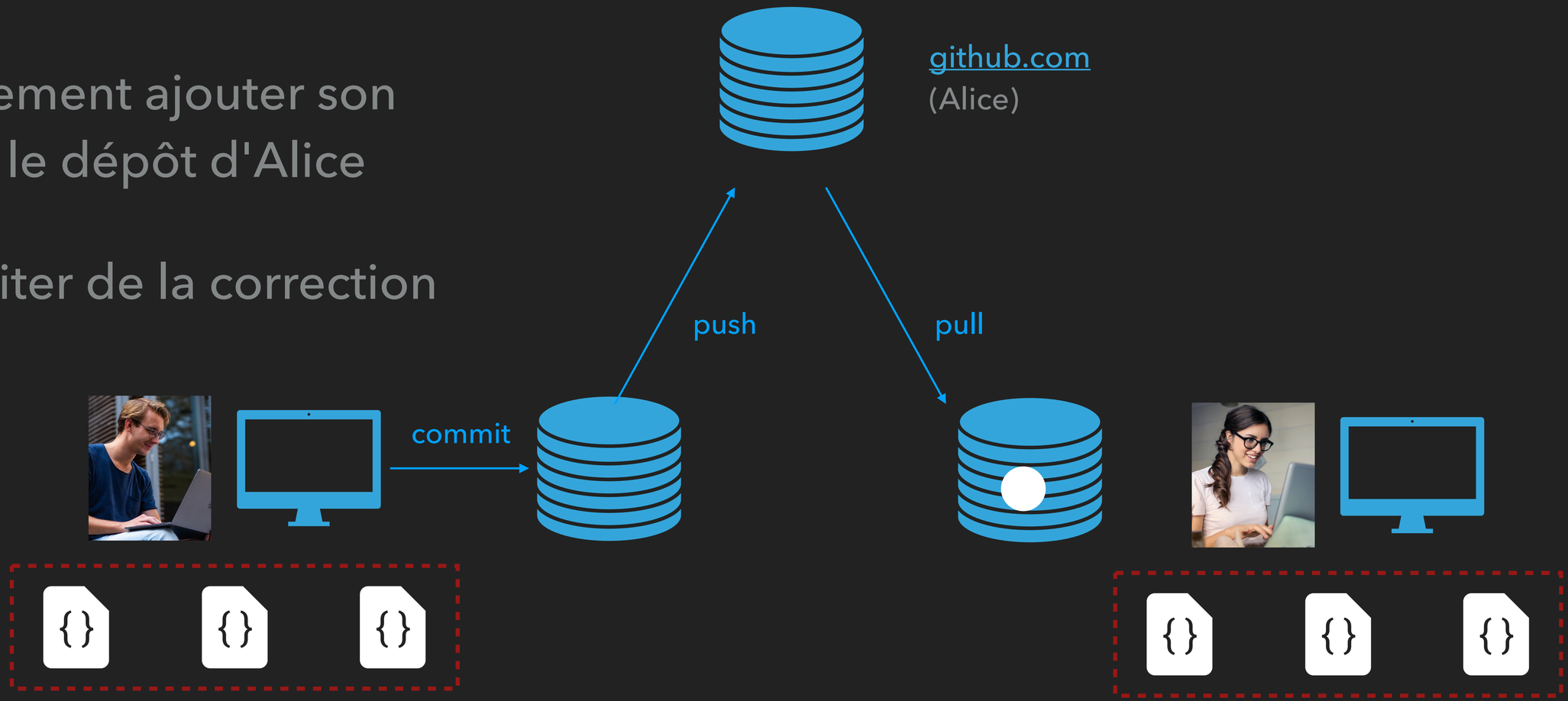
# BOB PEUT ÉCRIRE DANS LE DÉPÔT D'ALICE

- il peut directement ajouter son commit dans le dépôt d'Alice



# BOB PEUT ÉCRIRE DANS LE DÉPÔT D'ALICE

- il peut directement ajouter son commit dans le dépôt d'Alice
- qui peut profiter de la correction

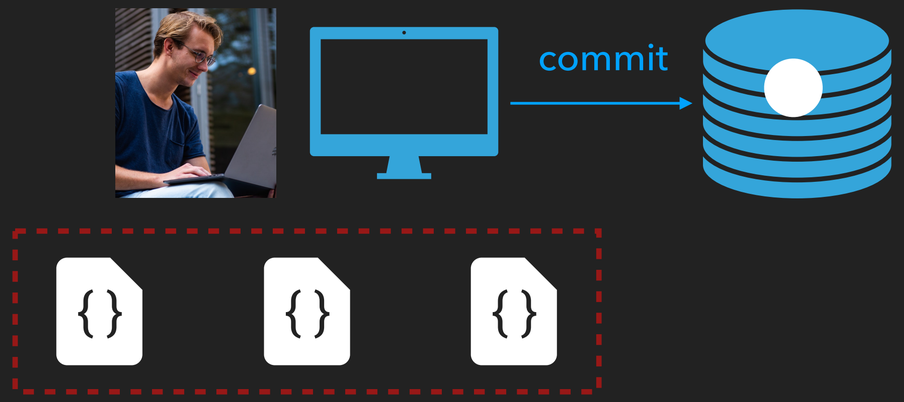


# SCÉNARIO #2: A TRAVERS UN “PULL REQUEST"

- Alice et Bob ne se connaissent pas
  - Bob n’a pas le droit d’écrire dans le dépôt d’Alice
  - c’est la raison d’être du *fork,* et des *Pull Requests :*
    - Bob se crée un *fork*, qui est (encore) un clone du dépôt d’Alice, hébergé sur github aussi, mais dans lequel il a le droit d’écrire
    - cela va lui servir à exposer sa proposition de changement à Alice au travers d’un *Pull Request*

# BOB PROPOSE SA CORRECTION VIA UN FORK

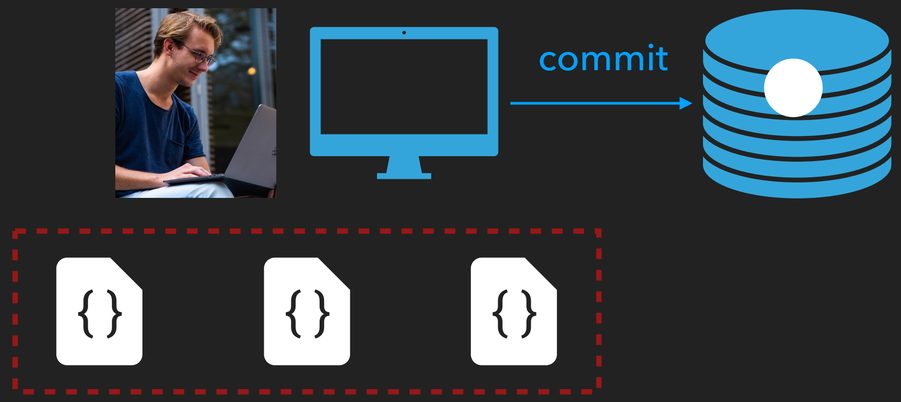
github.com
(Alice)



# BOB PROPOSE SA CORRECTION VIA UN FORK

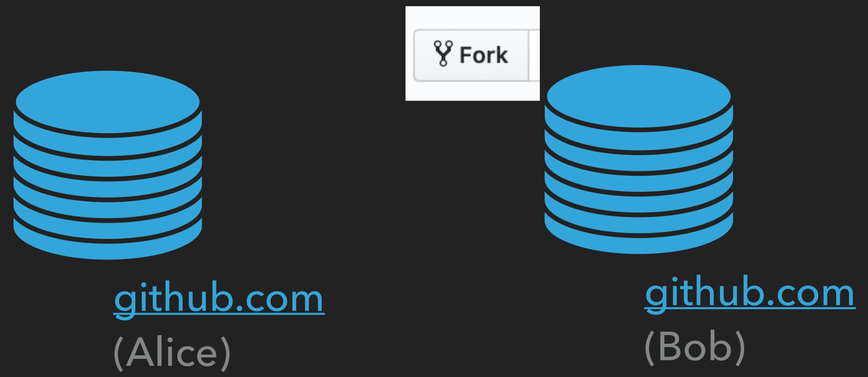
- il ne peut pas publier sur le dépôt distant de Alice: il doit créer **sa propre copie** publique (« *fork*")

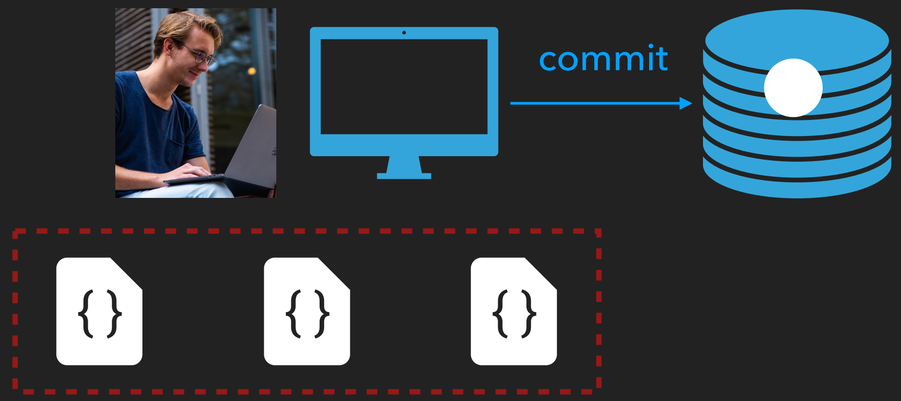
github.com
(Alice)



# BOB PROPOSE SA CORRECTION VIA UN FORK

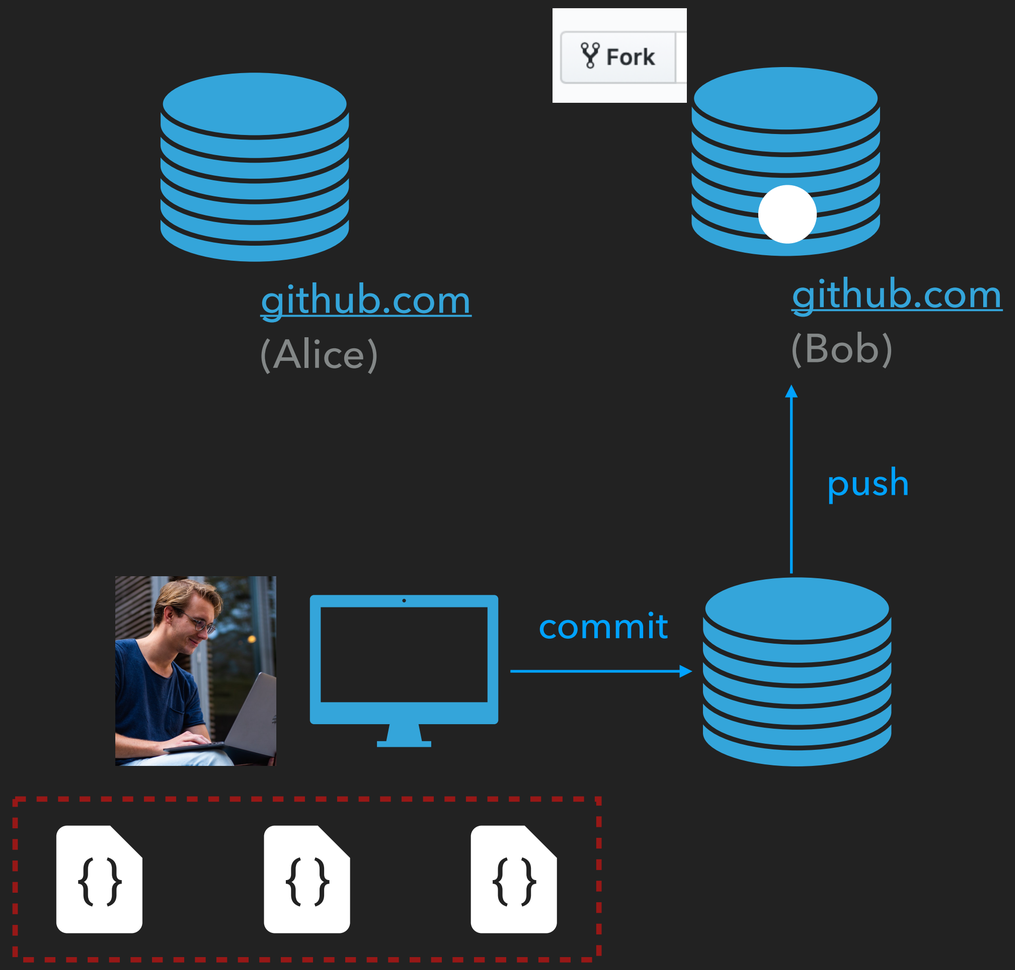
- il ne peut pas publier sur le dépôt distant de Alice: il doit créer **sa propre copie** publique (« *fork*")



# BOB PROPOSE SA CORRECTION VIA UN FORK

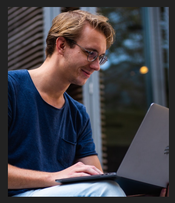
- il ne peut pas publier sur le dépôt distant de Alice: il doit créer **sa propre copie** publique (« *fork*")
- Il est désormais autorisé à publier ses changements, sur **son** dépôt **public** (son *fork*)

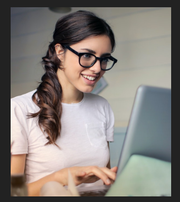


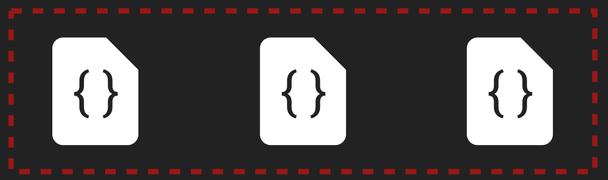
# ALICE RÉCUPÈRE LES CHANGEMENTS

github.com
(Alice)

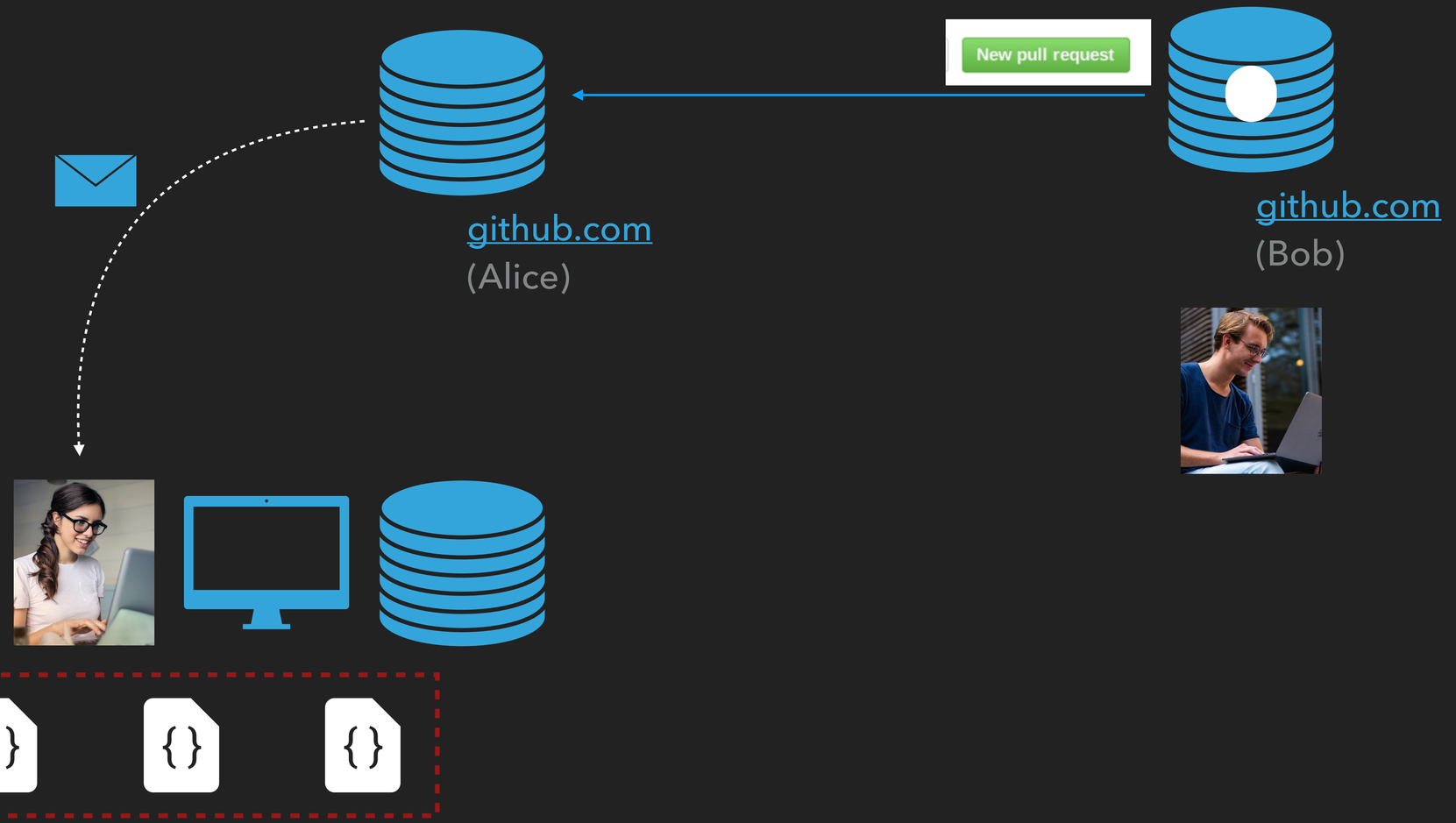
github.com
(Bob)



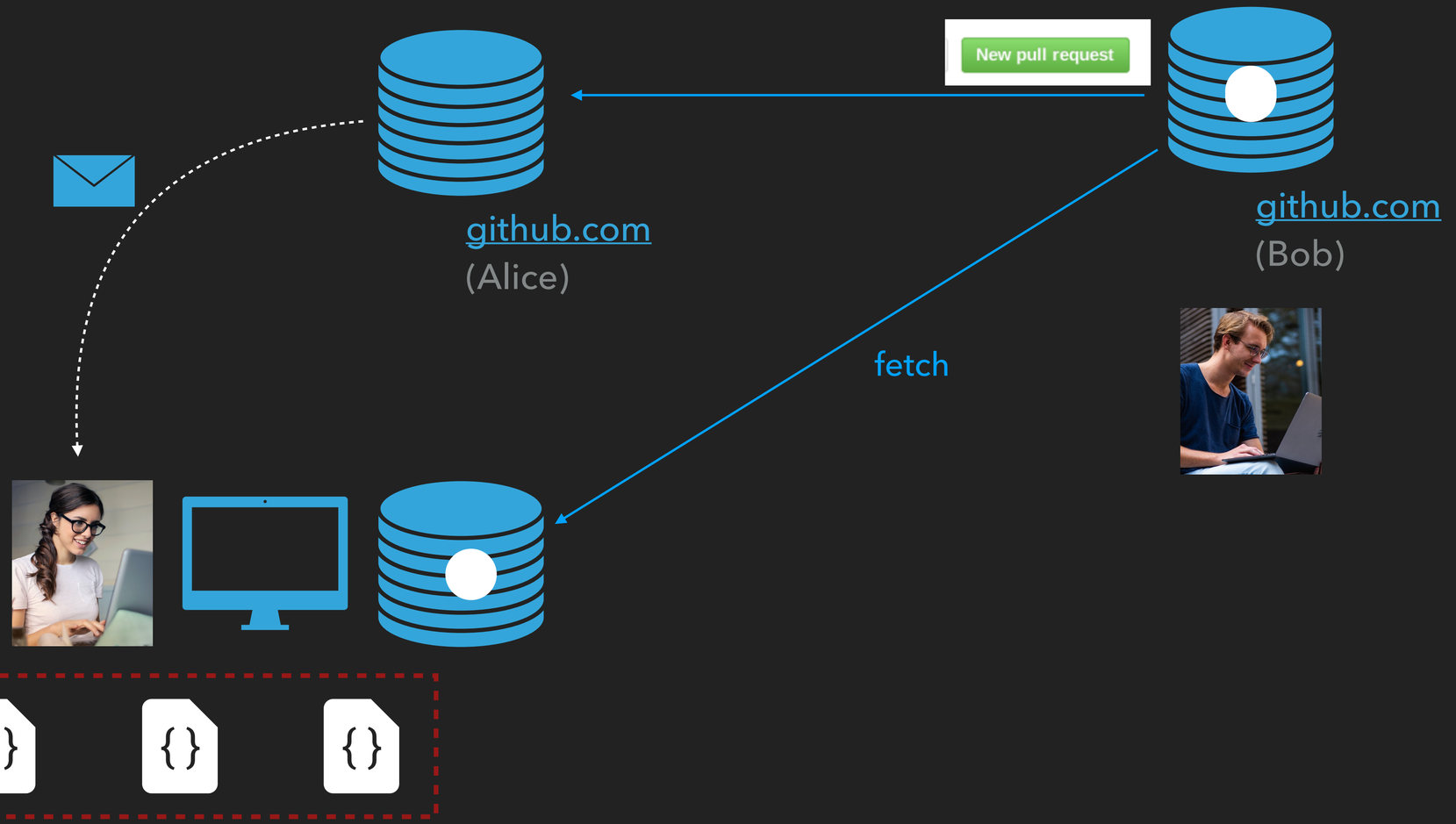




# ALICE RÉCUPÈRE LES CHANGEMENTS



# ALICE RÉCUPÈRE LES CHANGEMENTS



# ALICE RÉCUPÈRE LES CHANGEMENTS