Génie Logiciel

Gestion de versions

Nicolas Delanoue

Université d'Angers - Polytech Angers

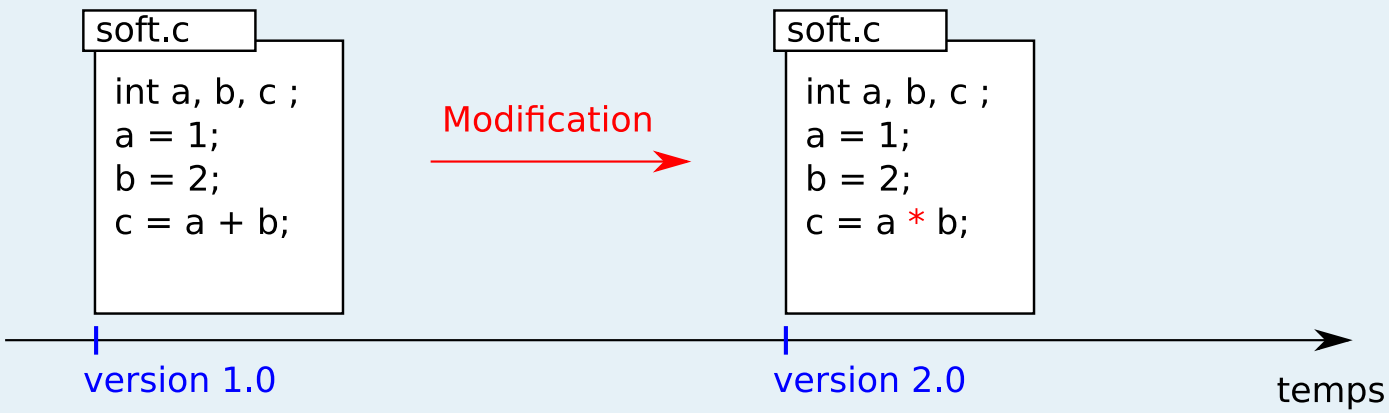
Définition

Les logiciels évoluant, chaque étape d'avancement est appelée version (ou revision).

Définition

Une modification constitue une évolution entre deux versions.

Illustration



Définition

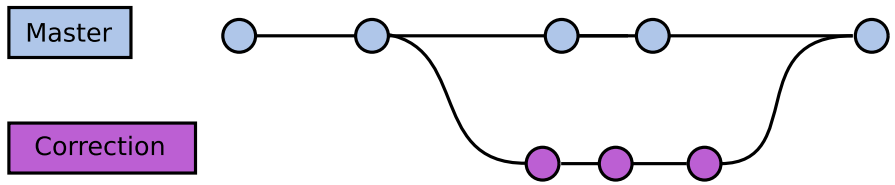
La gestion de versions consiste à gérer l'ensemble des versions d'un ou plusieurs fichiers (généralement en texte).

Remarques

- Il existe de nombreux logiciels capable d'aider le développeur dans cette tâche (e.g. git).
- En anglais, on parle de *version control* ou de *source code management*.

Possibilités offertes

- Sauvegarde (avec ou sans serveur distant)
- Conservation de l'historique des fichiers
  - l'auteur des modifications,
  - la date,
  - une note explicative de cette modification.
- Voir les changements,
- Expérimenter,
- Revenir aux versions précédentes,
- Collaborer.



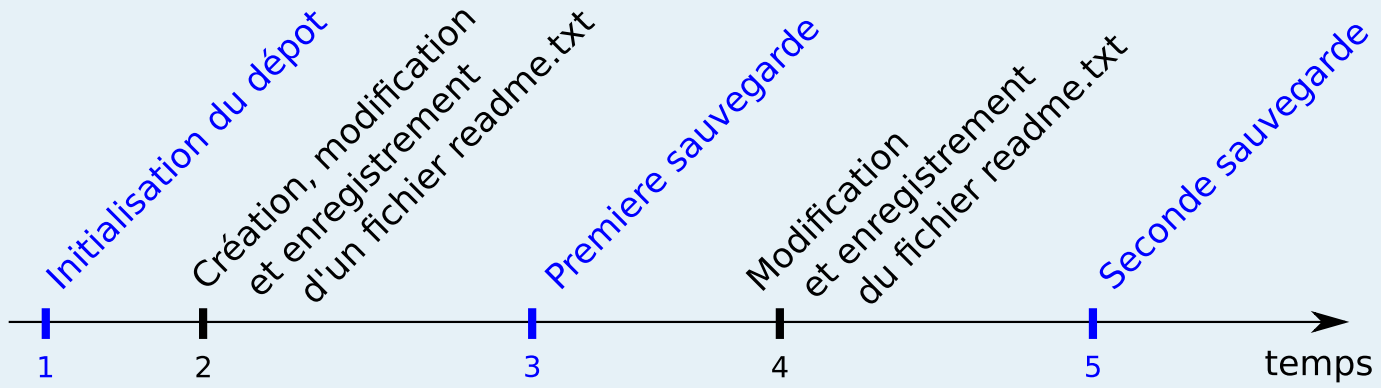
Avantages de la création de branches

- Expérimenter en sécurité, i.e. effectuer des grosses modifications sans faire prendre de risques à la branche principale.
- Collaborer, i.e. chacun sur sa branche. Il faudra tout de même fusionner. Exemples d'utilisation :
  - des correctifs complexes sans gêner les développements,
  - restructuration de logiciel,
  - compilation en local sans avoir à attendre ...

Conclusion

- Il est possible et risquer de développer des logiciels sans utiliser de contrôle de version.
- La question n'est donc pas de savoir s'il faut utiliser le contrôle de version, mais lequel.

Exemple

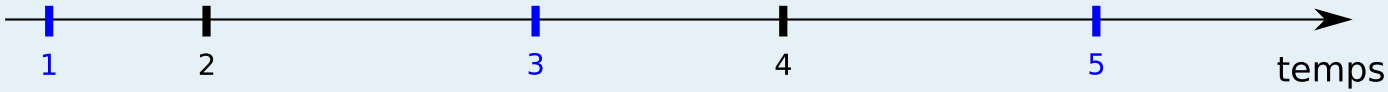


Remarque

Deux mécanismes d'écriture :

- En bleu : le système de gestion de versions,
- En noir : le système classique d'enregistrement de données.

Exemple



Exemple de scénario d'utilisation

1 →
```
$ git init
Dépôt Git vide initialisé dans .git/
```
2 →
```
$ echo "Bonjour tout le monde" > readme.txt
$ git add "readme.txt";
```
3 →
```
$ git commit -m "version 1.0"
master (commit racine) fc8dbda] version 1.0
 1 file changed, 1 insertion(+)
 create mode 100644 readme.txt
```
4 →
```
$ echo "\n Comment ca va ? " >> readme.txt
$ git add "readme.txt";
```
5 →
```
$ git commit -m "version 2.0"
[master 6096b1f] version 2.0
 1 file changed, 1 insertion(+)
```

Définition - Répertoire de travail

On appelle répertoire de travail (en anglais working copy) les fichiers effectivement présents dans le répertoire géré par GIT.

Définition - Un dépôt

Un dépôt (repository ou repo) est un dossier contenant l'ensemble des données associées au projet (les versions et les modifications d'un projet).

Création d'un dépôt local

```
nico@pc~/projet$ git init
Dépôt Git vide initialisé dans /home/nico/projet/.git/
nico@pc~/projet$ ls -la
drwxrwxr-x  7 nico nico 4096 nov.  12 15:32 .git
```

Remarque

Sauf cas rare, il n'est pas nécessaire d'intervenir manuellement dans le répertoire .git.

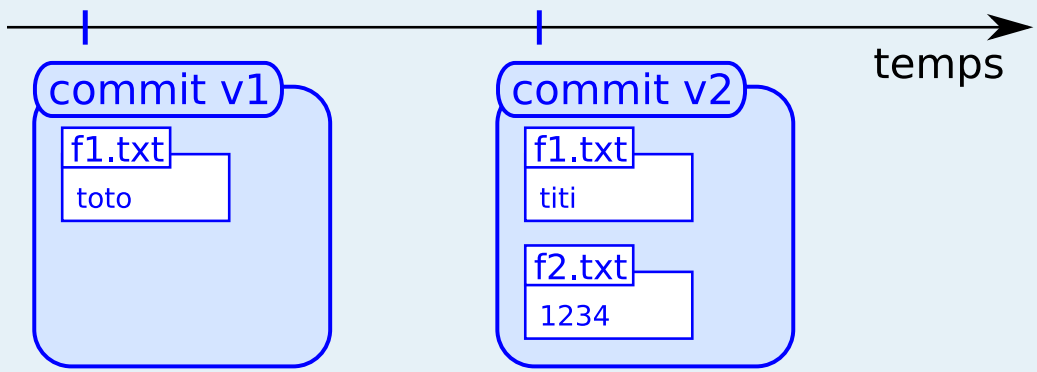
Avantages

- Un dépôt sont de simples fichiers dans un dossier,
- Copier un dépôt `.git` permet de saugarder l'historique,
- Tous les logiciels sont compatibles avec `git`.

Définition - commit

Sauvegarder au sens de `git`, c'est effectuer un `commit`.

Illustration

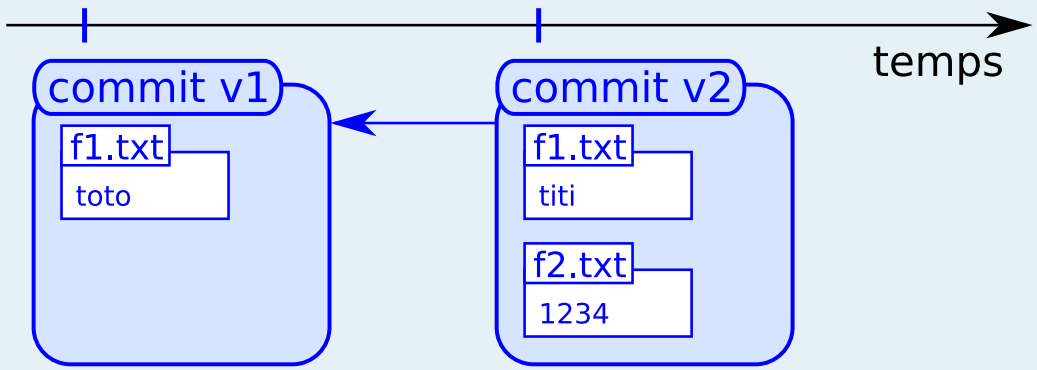


Remarque

En interne, les `commit` sont liés les uns aux autres.

D'une certaine manière, on s'affranchit du temps.

Illustration

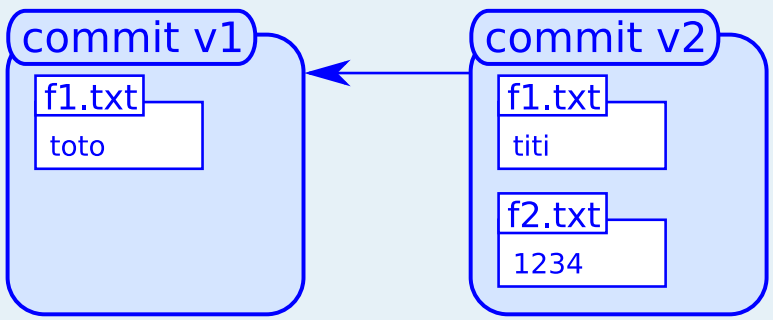


Remarque

En interne, les `commit` sont liés les uns aux autres.

D'une certaine manière, on s'affranchit du temps.

Illustration



Exemple simple

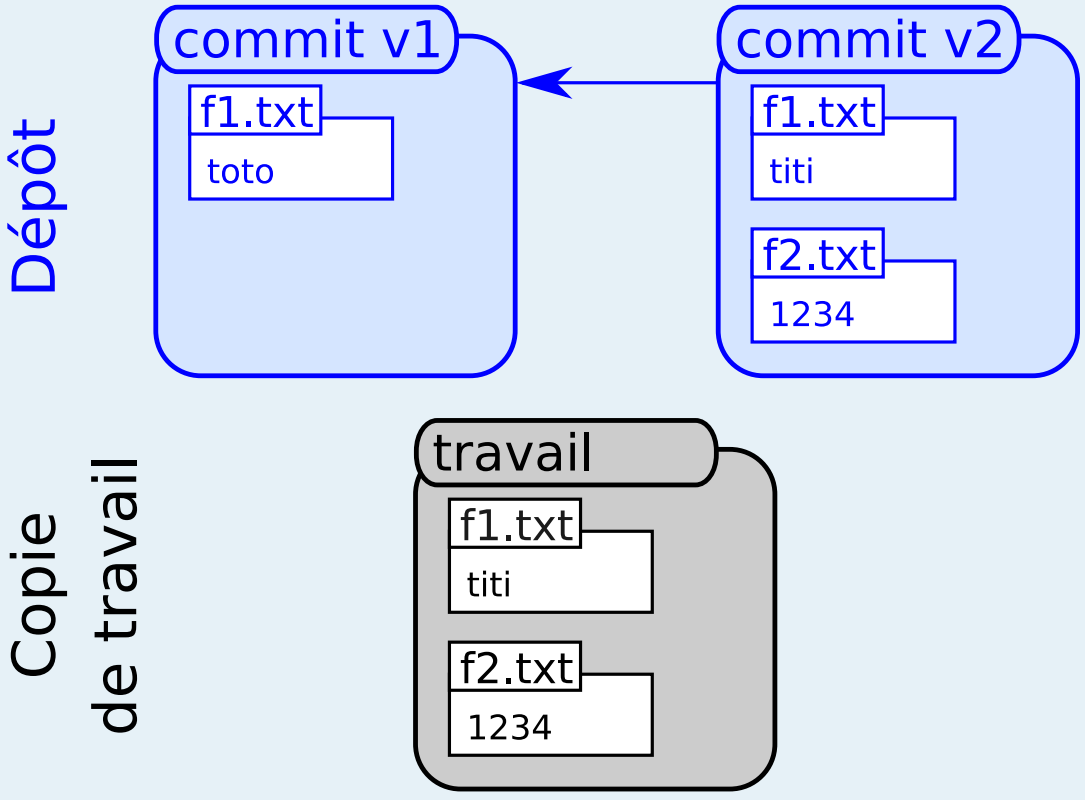


FIGURE – Résultat attendu.

Exemple plus complexe : présence d'un fichier qu'il n'est pas nécessaire d'historiser.

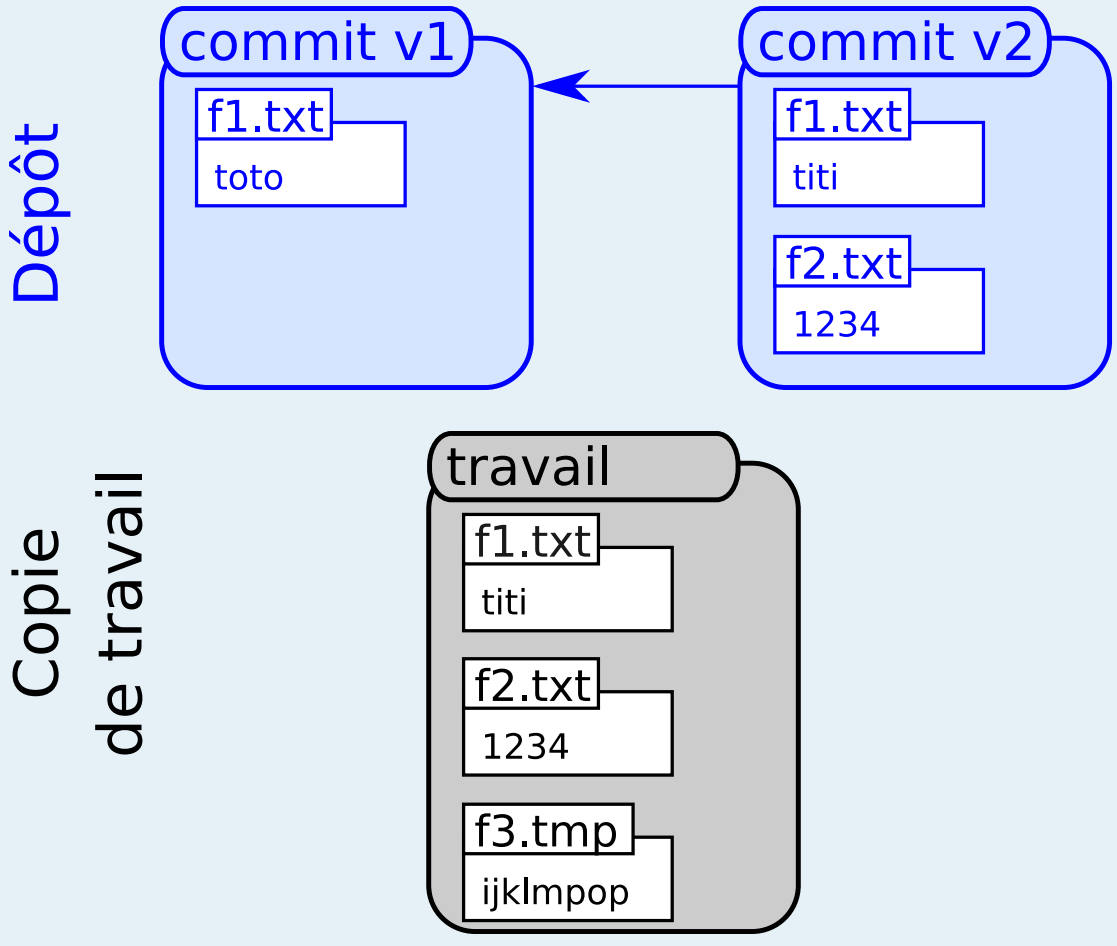


FIGURE – Résultat attendu.

Comment indiquer à git quels fichiers doivent être historisés lors du prochain `commit` ?

Comment indiquer à git quels fichiers doivent être historisés lors du prochain `commit` ?

Choix des concepteurs de git :

Le développeur doit explicitement préciser les fichiers qu'il souhaite historiser.

Commandes

```
$ git add f1.txt
$ git add f2.txt
```

Définition

Les fichiers choisis par la commande `git add` sont dans l'état *staged*. On dit aussi qu'il font partie de l'index.

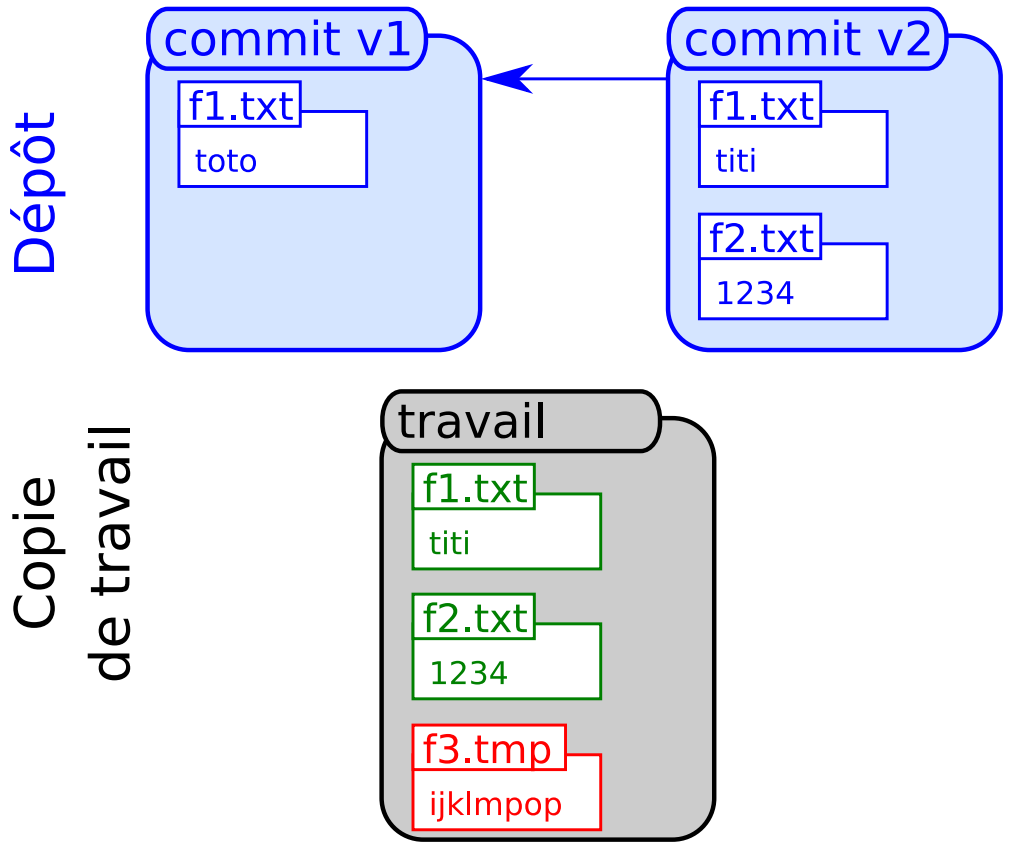
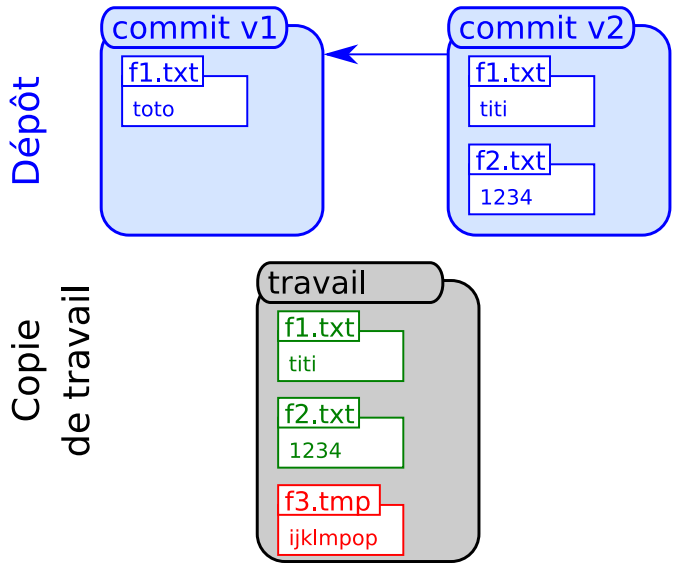


FIGURE – La solution.



Commandes

```
$ echo "titi" > f1.txt
$ echo "1234" > f2.txt
$ echo "ijklmnpop" > f3.tmp
$ git add f1.txt
$ git add f2.txt
$ git commit -m "v2"
```

Dans quel état se trouvent les fichiers f1.txt, f2.txt et f3.tmp après le `commit v2` ?

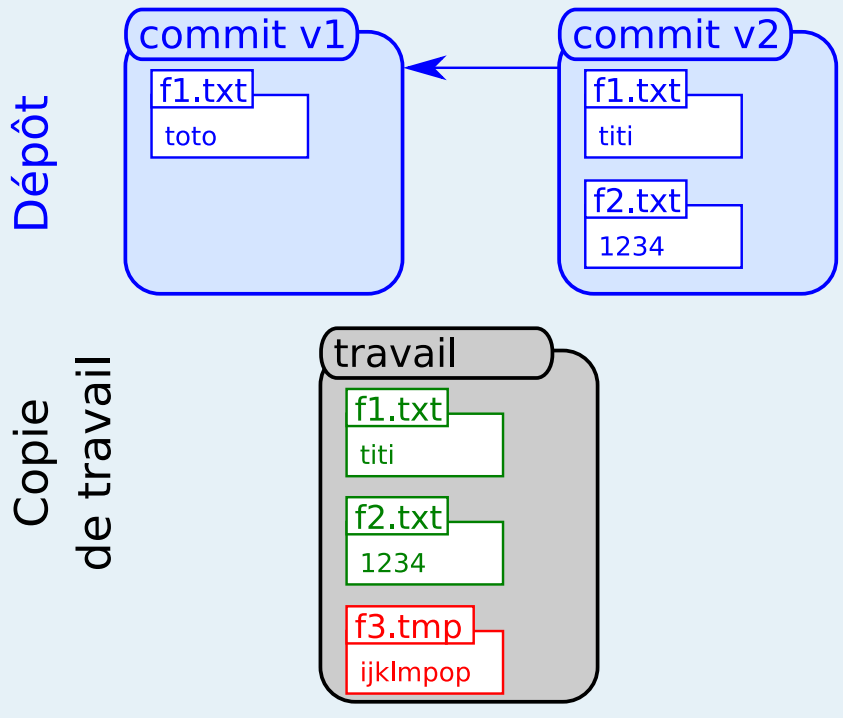
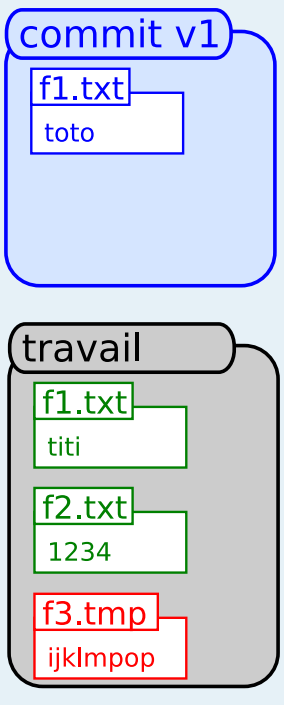


FIGURE – Ici, les fichiers f1.txt et f2.txt sont indexés.

Définition - Etat

La commande `git status` permet de connaitre l'*état* d'un fichier du répertoire de travail vis à vis du dernier commit.




```
nico@pc:~/projet$ git add f1.txt
nico@pc:~/projet$ git add f2.txt
nico@pc:~/projet$ git status
Sur la branche master
Modifications qui seront validées :
  (utilisez "git reset HEAD <fichier>..." pour désindexer)
        modifié :         f1.txt
        nouveau fichier : f2.txt
Fichiers non suivis:
  (utilisez "git add <fichier>..." pour inclure dans ce qui
        f3.tmp
```

Différences plus précisément

La commande `git diff` permet de voir les modifications entre le dernier `commit` et le contenu du répertoire de travail.

Historique git

La commande `git log` permet de voir l'ensemble des `commit`.

Exemple

```
nico@pc:~/projet$ git log
commit e190938a57cf041549311dddabaa75 (tag: v2)
Author: nico <nico@delanoue.fr>
Date:   Thu Nov 12 16:23:59 2020 +0100
    version avec fichier2.txt
commit fb9e90522ad57375103d646b4d0003 (tag: v1)
Author: nico <nico@delanoue.fr>
Date:   Thu Nov 12 16:07:07 2020 +0100
    version 1
```

git checkout

La commande `git checkout` restaure le dossier de travail en accord avec le commit passé en argument.

Exemple : `git checkout v2`

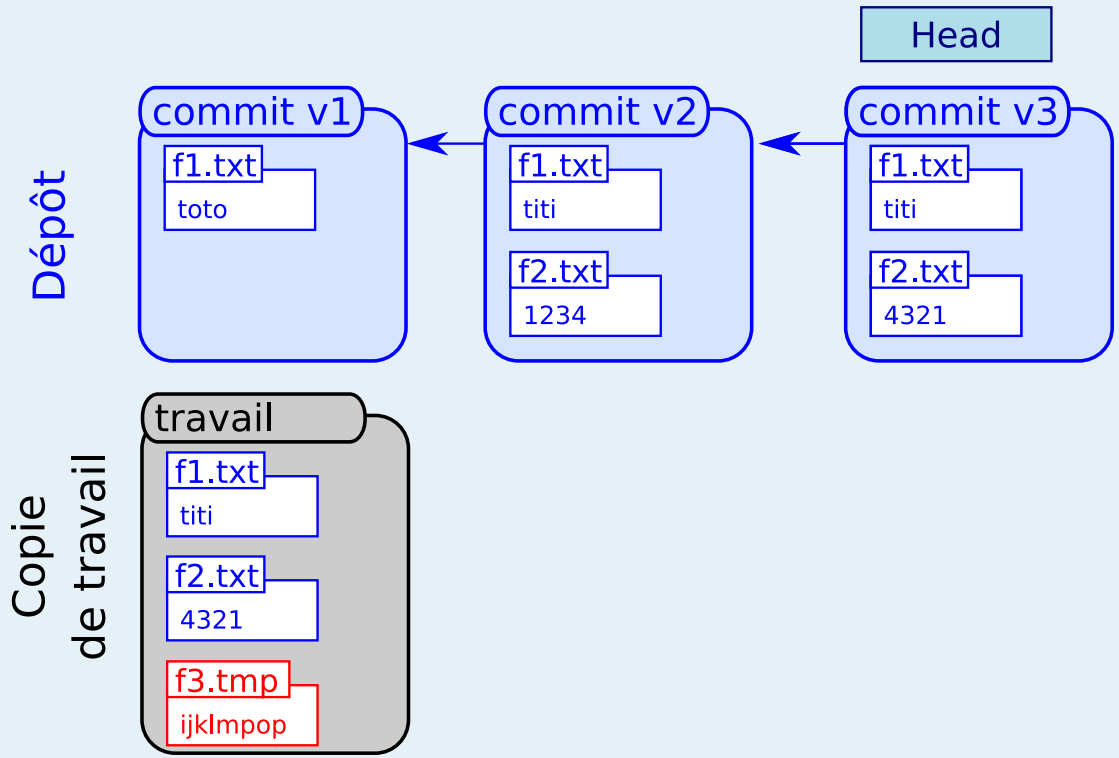


FIGURE – Avant git checkout v2

git checkout

La commande `git checkout` restaure le dossier de travail en accord avec le commit passé en argument.

Exemple : `git checkout v2`

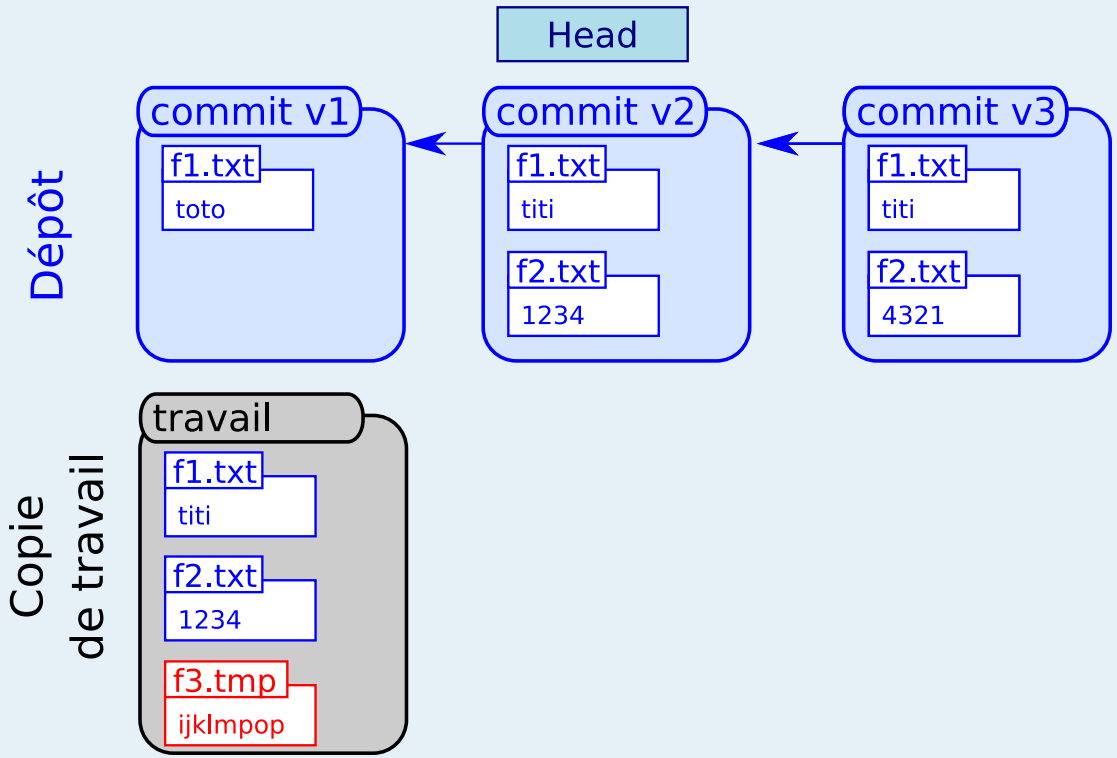
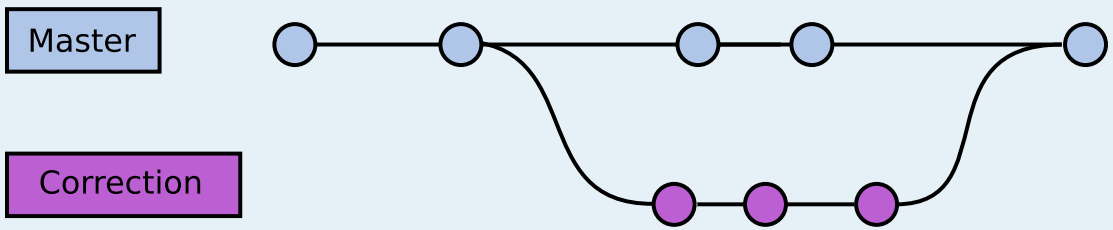
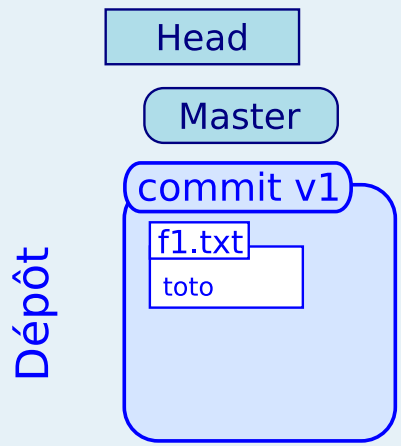


FIGURE – Après git checkout v2

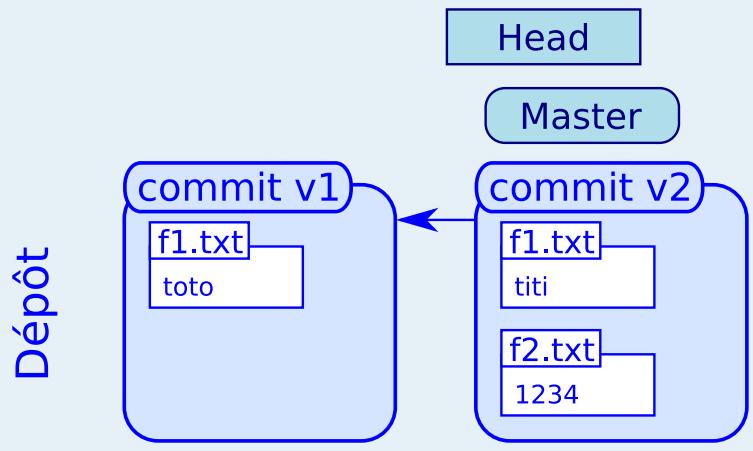
Collaborer avec soi même ou bien avec d'autres



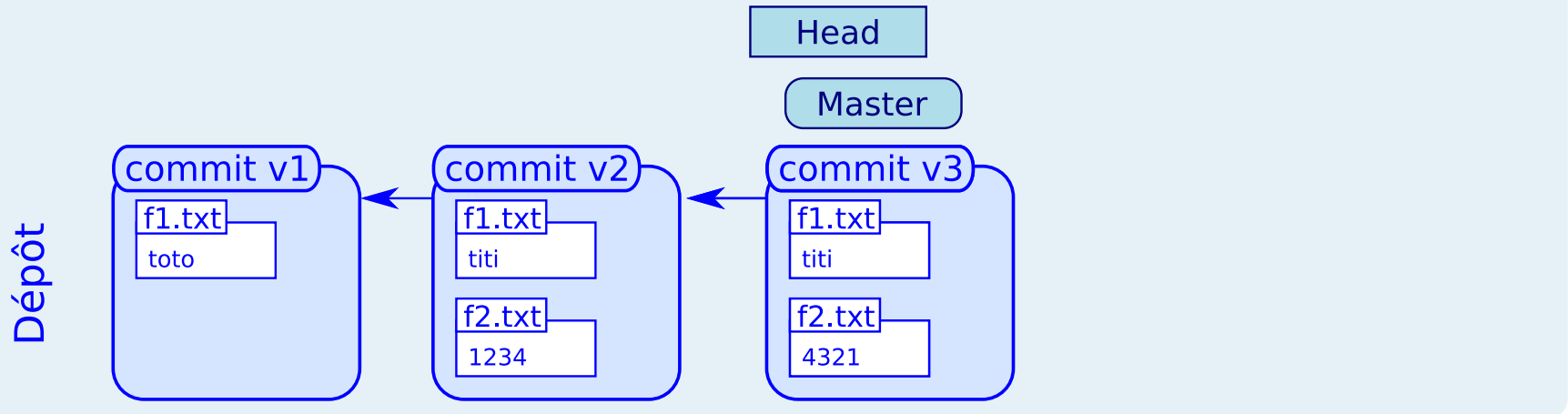
Illustration



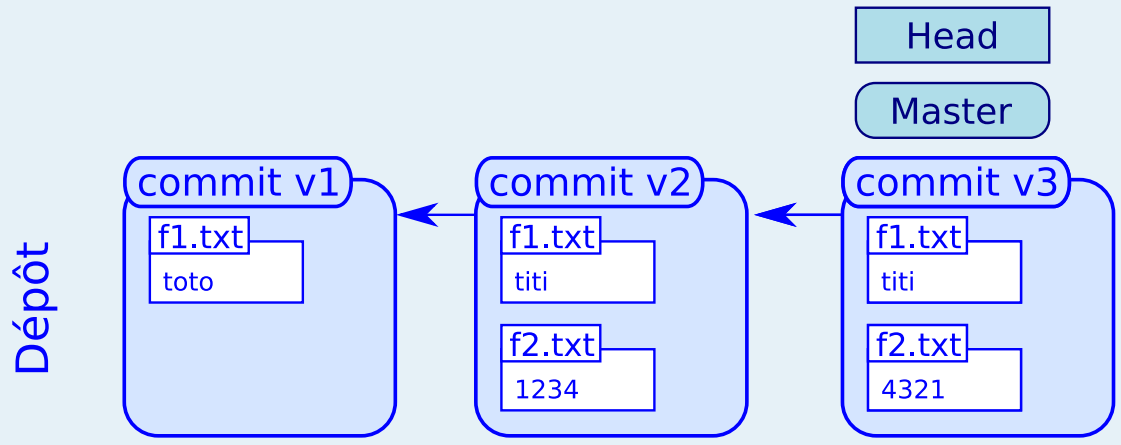
Illustration



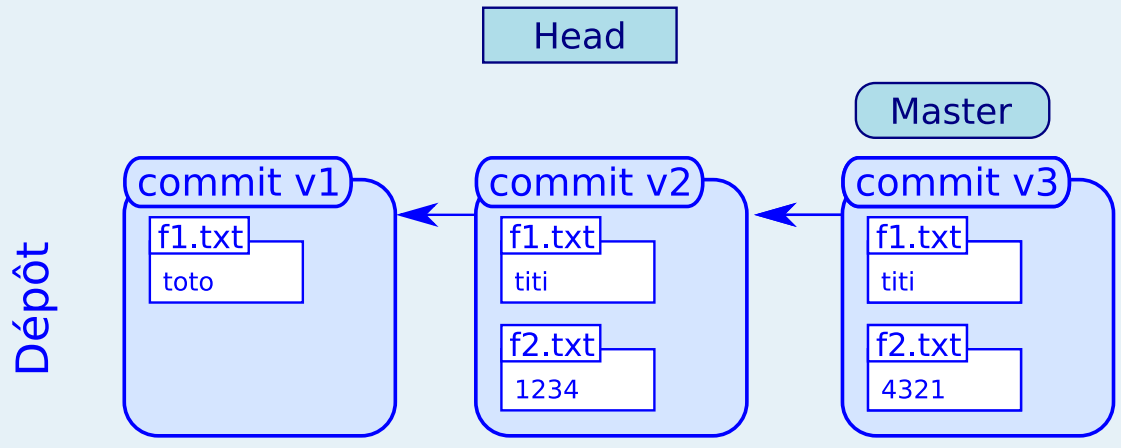
Illustration



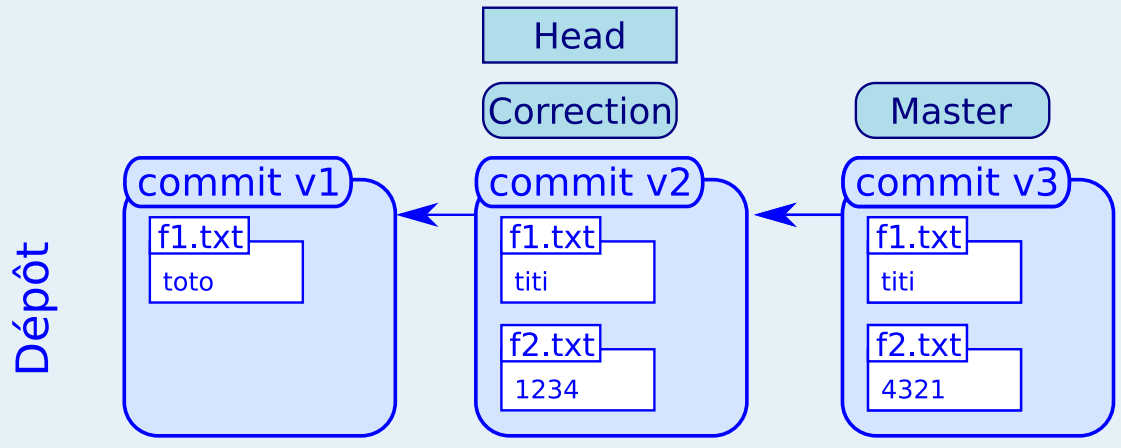
Illustration



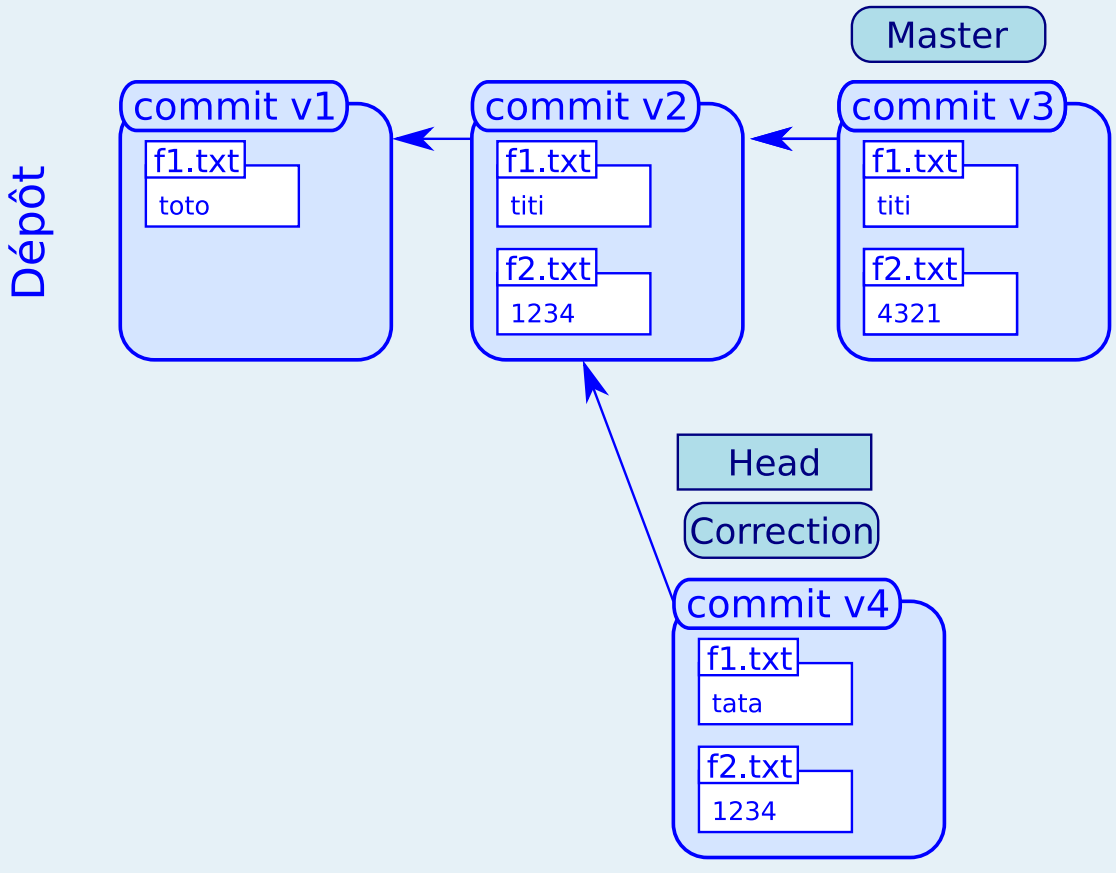
Illustration

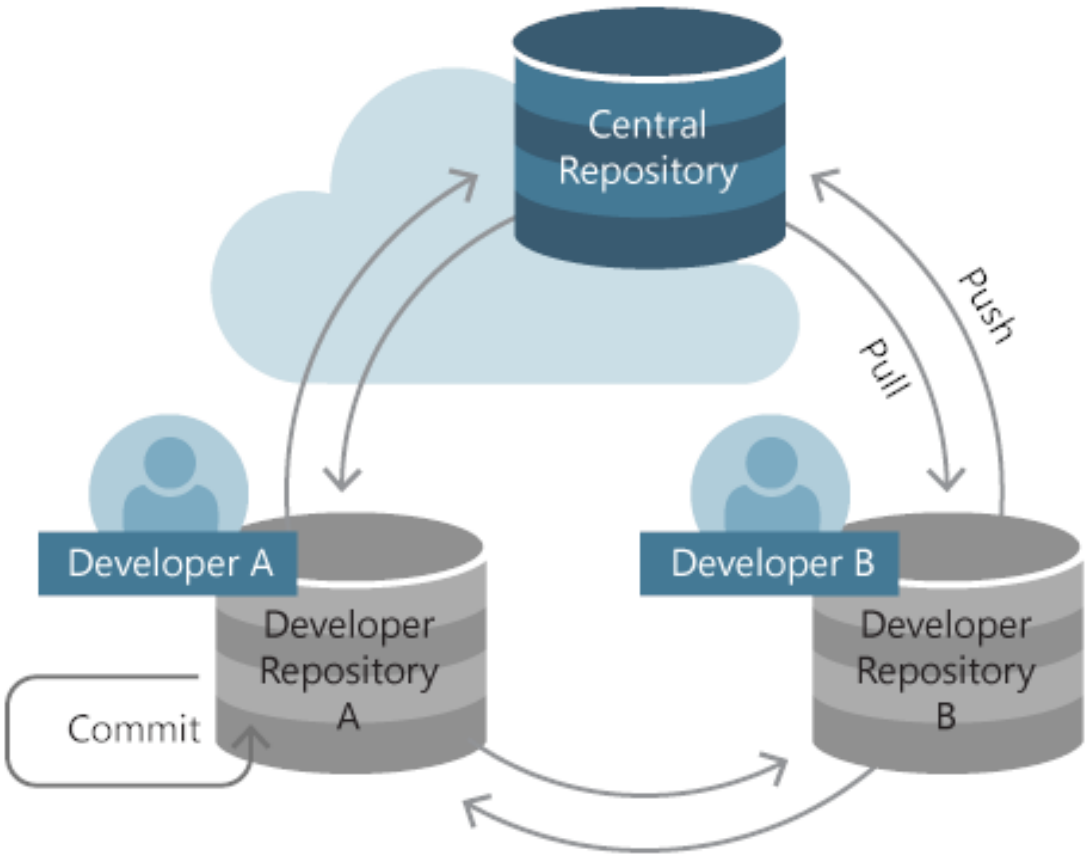


Illustration



Illustration




Central
Repository
Push
Pull
Developer A
Developer B
Developer
Repository
A
Developer
Repository
B
Commit

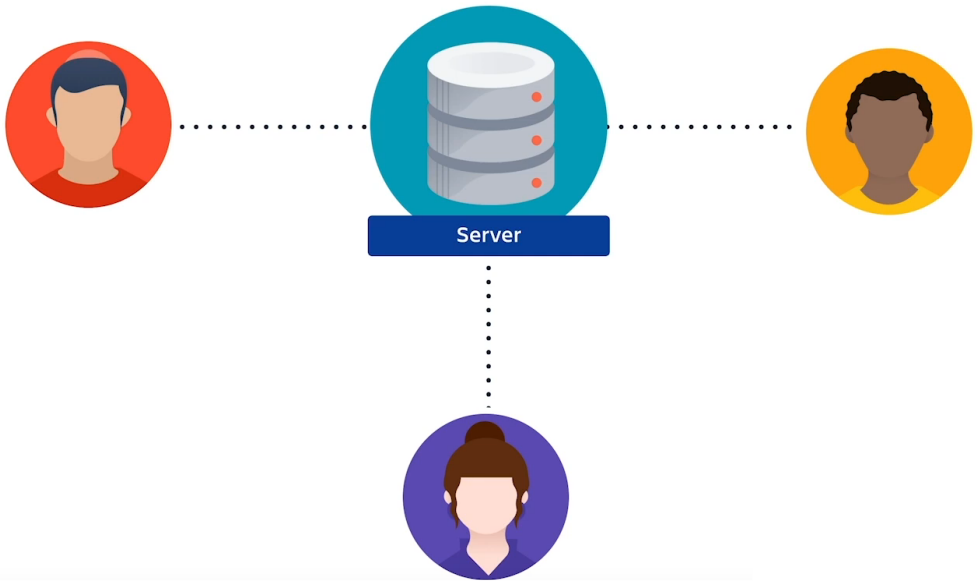


FIGURE – Les collaborateurs travaillent en étant connectés au système.

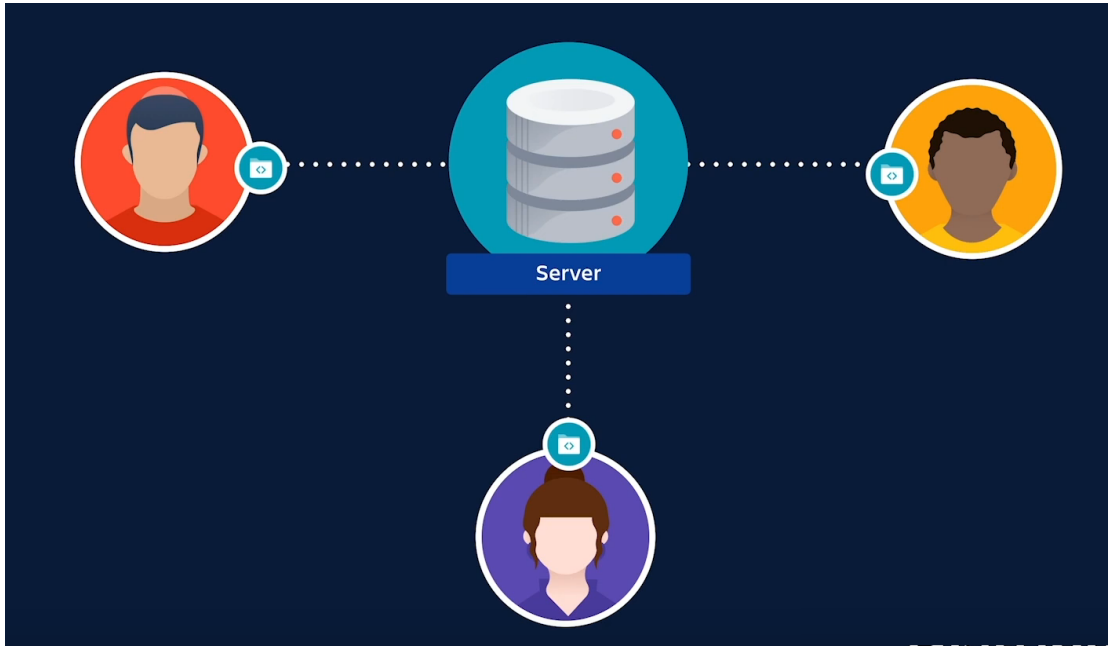


FIGURE – Les collaborateurs travaillent avec une copie locale.

Avantages

- Les problèmes réseaux sont presque indépendants de l'avancé des développeurs.
- Ils ont aussi chacun l'historique du projet.