Gestion de versions

- Historique des modifications

Qui ? **Quoi** ? **Quand** ?

- Travail collaboratif

Travailler à plusieurs sur la même base de code

- Branches de développement

Expérimentation, versions stables/instables...

Notion de dépôt

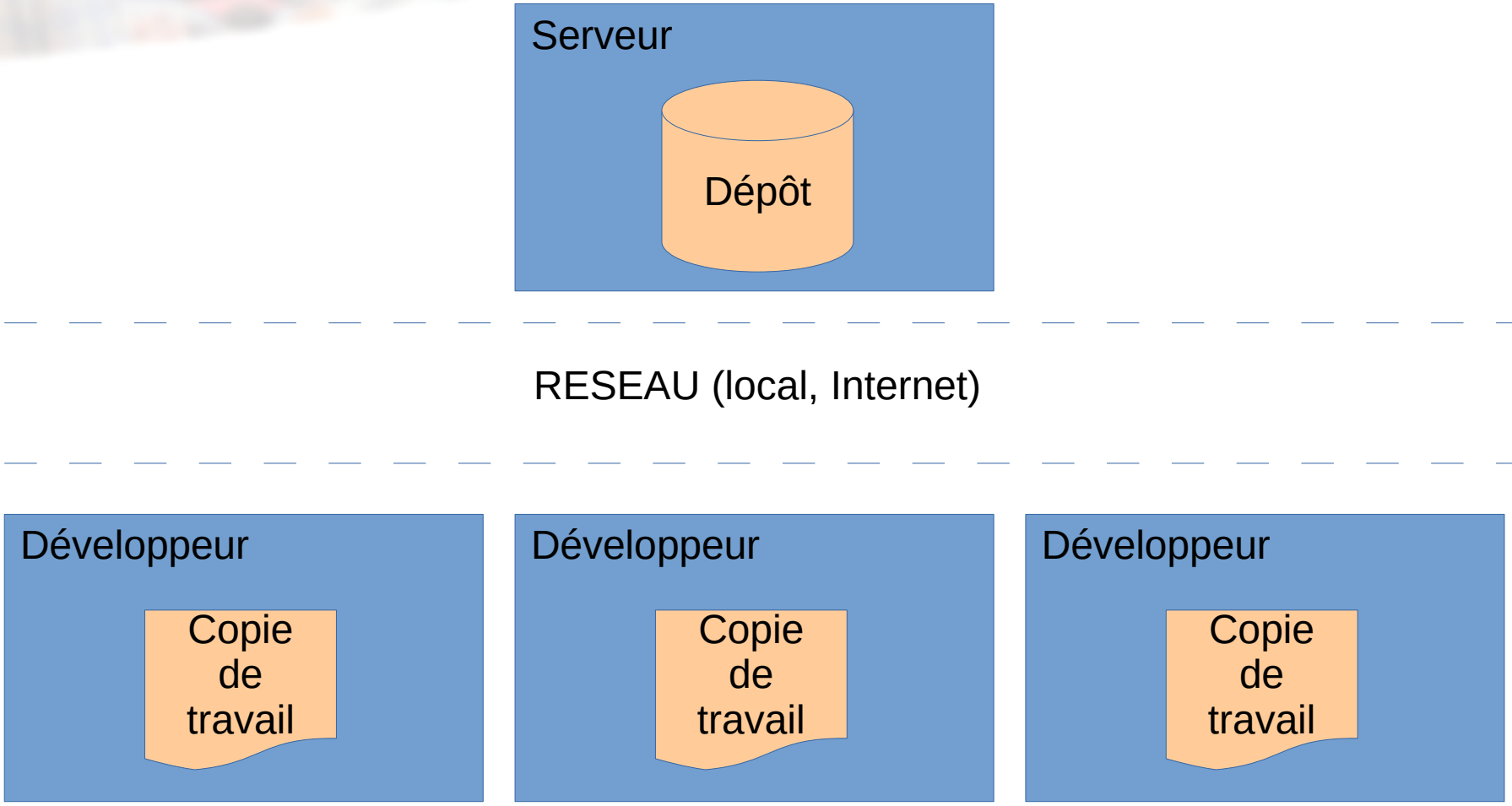
Repository

- Stockage de l'**historique complet** des modifications apportées aux fichiers du projet
- L'historique n'est pas nécessairement linéaire et peut comporter des **branches**, des **fusions**
- Ajout d'une modification : « **commit** »
- A chaque *commit* est associé un **message** expliquant la modification
- Un projet ↔ un dépôt

Succinct et explicite

Gestion centralisée

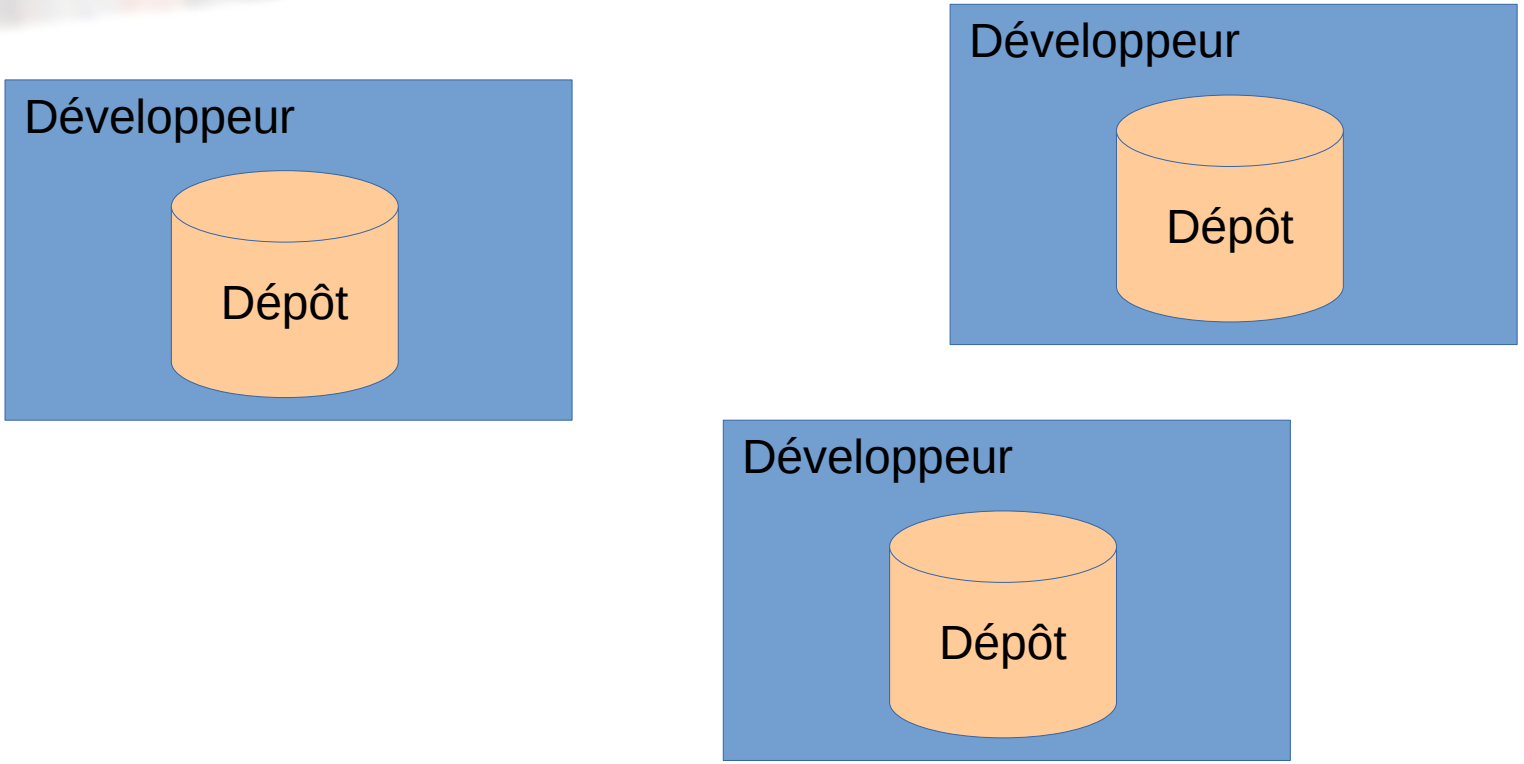
cvs, Subversion(svn), ClearCase, Visual Sourcesafe, ...



- Organisation simple : un dépôt unique centralise l'historique du projet
- Toute opération sur le dépôt nécessite une connexion au réseau
- Les droits de « commit » doivent être correctement gérés

Gestion décentralisée

git, Mercurial, Bazaar, BitKeeper, ...

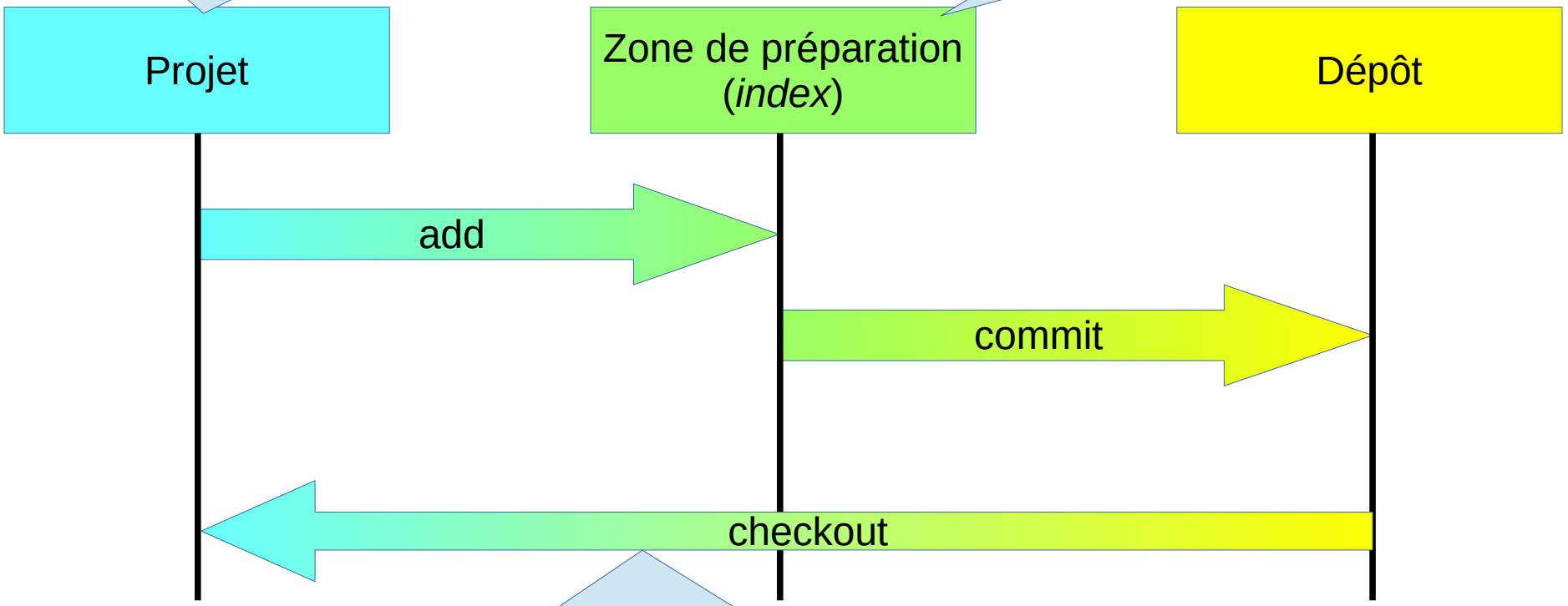


- Organisation libre : chaque développeur dispose de l'historique du projet
- Les opérations sur le dépôt sont locales (pas besoin de connexion réseau)
- Inutile de gérer les droits (chaque développeur a tous les droits sur son dépôt)
- Idéal pour un développeur solitaire...

Fonctionnement général de Git

Répertoire "normal" contenant les fichiers à versionner. Il peut également contenir des sous-répertoires, dont le contenu sera également versionné.

Emplacement "logique" auquel on ajoute les fichiers que l'on souhaite intégrer au prochain *commit*



Cette opération permet de restaurer un ou plusieurs fichiers dans l'état où ils étaient lors d'un *commit* antérieur

Configuration globale

A faire une seule fois

Terminal

```
git config --global user.name "Prénom Nom"
```

A remplacer par **votre** prénom et **votre** nom

```
git config --global user.email "votre@adresse"
```

A remplacer par **votre** adresse de courriel

```
git config --global core.editor "evim -f"
```

Éditeur pour les messages de *commit*. Par défaut, **git** utilise l'éditeur **vim** (que vous pouvez choisir de conserver si vous en maîtrisez l'usage). Sous Windows, vous pouvez utiliser **notepad**

```
git config --global credential.helper cache
```

Conservation en mémoire des identifiants de connexion pendant 15 minutes

```
git config --global push.default simple
```

Mode à utiliser pour les opérations de *push*

Initialisation d'un dépôt

Se placer dans le répertoire contenant le projet à versionner

Création d'un dépôt vide.

Attention à ne pas passer cette commande dans un sous-répertoire d'un projet déjà versionné par Git

Édition du fichier définissant ce qu'on ne souhaite pas versionner

Terminal

```
cd répertoire/du/projet

git init

evim .gitignore
```

Exemples de définitions dans le fichier `.gitgnore`

`fichier_a_ignorer.txt` — Ignorer le fichier portant ce nom, quel que soit le sous-répertoire dans lequel il réside

`*.tmp` — Ignorer les fichiers dont le nom se termine par `.tmp`, quel que soit le sous-répertoire dans lequel ils résident

`doc/` — Ignorer toute l'arborescence contenue dans ce répertoire

Commit

Se placer dans le répertoire contenant le projet

Ajout à l'*index* des fichiers du projet non exclus par le **.gitignore**. N'oubliez pas le point après **add.**

Ouvre l'éditeur de message de *commit*.

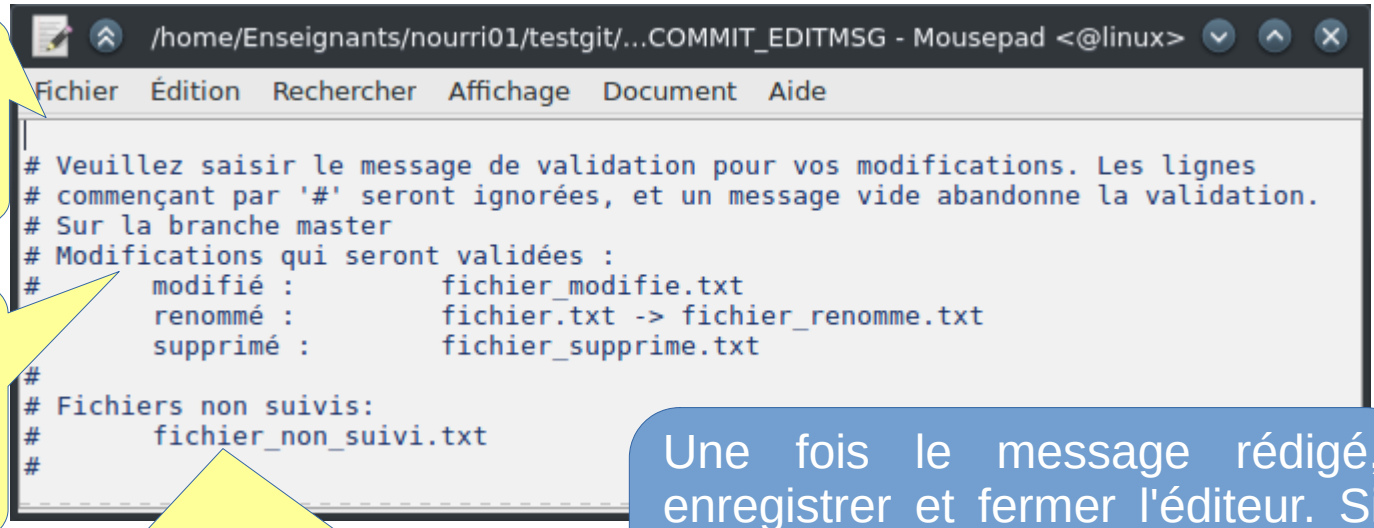
Terminal

```
cd répertoire/du/projet

git add .

git commit
```

Inutile de déplacer le curseur : écrivez le message dès que la fenêtre est ouverte



Vérifiez que toutes les modifications à intégrer au *commit* sont bien mentionnées dans cette section.

Fichier ne faisant pas partie du *commit.* C'est soit volontaire, soit dû à un oubli de passer la commande **git add .**

Une fois le message rédigé, enregistrer et fermer l'éditeur. Si l'éditeur est fermé sans enregistrer le message, l'opération de *commit* est annulée

Message de *commit*

Répondre à la question : pourquoi ?

La **ligne de sujet** doit résumer la modification en un minimum de caractères (idéalement : 50).

Une **ligne vide** doit séparer la ligne de sujet des explications détaillées

Les **explications détaillées** ne doivent pas porter sur le code, mais sur les raisons qui ont conduit à effectuer les modifications validées par ce *commit*.
Si le *commit* est simple, la ligne de sujet peut être suffisante.

Éditeur de message

```
Sujet (une seule ligne)

Explications détaillées
(autant de lignes que
nécessaire)

# commentaires (qui ne
# feront pas partie du
# message lors de la
# validation du commit)
```

Un message de *commit* est semblable à un **courriel**

- Le sujet résume le message. Le corps du courriel expose votre argumentaire.
- Le message doit être rédigé pour être lu par un destinataire qui n'est pas vous : assurez-vous donc qu'il contienne toutes les informations nécessaires à sa compréhension.

Historique

Terminal

```
cd répertoire/du/projet

git log

git log --oneline

git show HEAD

git show HEAD~12

git show 73ba44d

git blame fichier

gitk
```

Se placer dans le répertoire contenant le projet à versionner

Résumé paginé de l'historique (Appuer sur **Q** pour quitter)

Résumé (un *commit* par ligne)

Affiche le dernier *commit*

12 fois "avant"

Affiche l'avant … avant dernier *commit*

Affiche le *commit* identifié par **73ba44d** (obtenu grâce à **git log**)

Détail des modifications d'un fichier

Ouvre une interface graphique

Différences

Se placer dans le répertoire contenant le projet à versionner

État des fichiers du projet

Différences entre l'état actuel des fichiers et le dernier *commit*

Différences entre le dernier et l'avant-dernier *commit*

Interface graphique pour les différences entre l'état actuel des fichiers et le dernier *commit*

Terminal

```
cd répertoire/du/projet

git status

git diff

git diff HEAD HEAD~1

meld .
```

Site web de l'application Meld

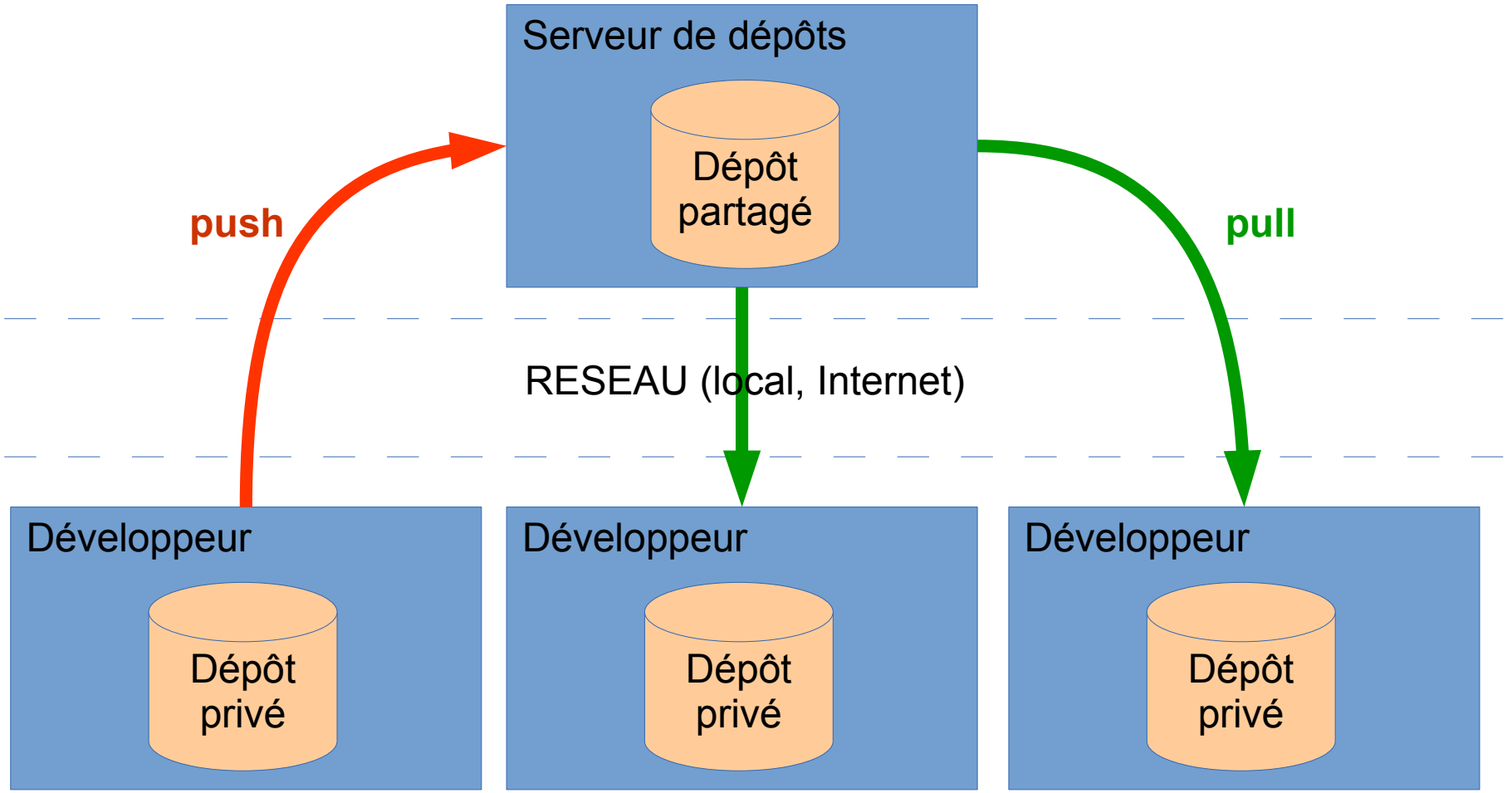
Ce qu'il ne faut surtout **pas faire**

- **`git init`** dans votre **répertoire d'accueil**, ou dans un **sous-répertoire** d'un dépôt **git**
- **`git commit`** alors que le projet comporte des **erreurs** : tous les *commits* doivent représenter un état valide du projet
- Conséquences du point précédent :
  - Pas de **`git commit`** alors que projet n'a **pas été testé**
  - Les **tests** doivent faire partie du **dépôt**
- Messages **peu informatifs, humoristiques et/ou grossiers** : une fois validé, un *commit* (et le message associé) ne peut plus être supprimé ni modifié.

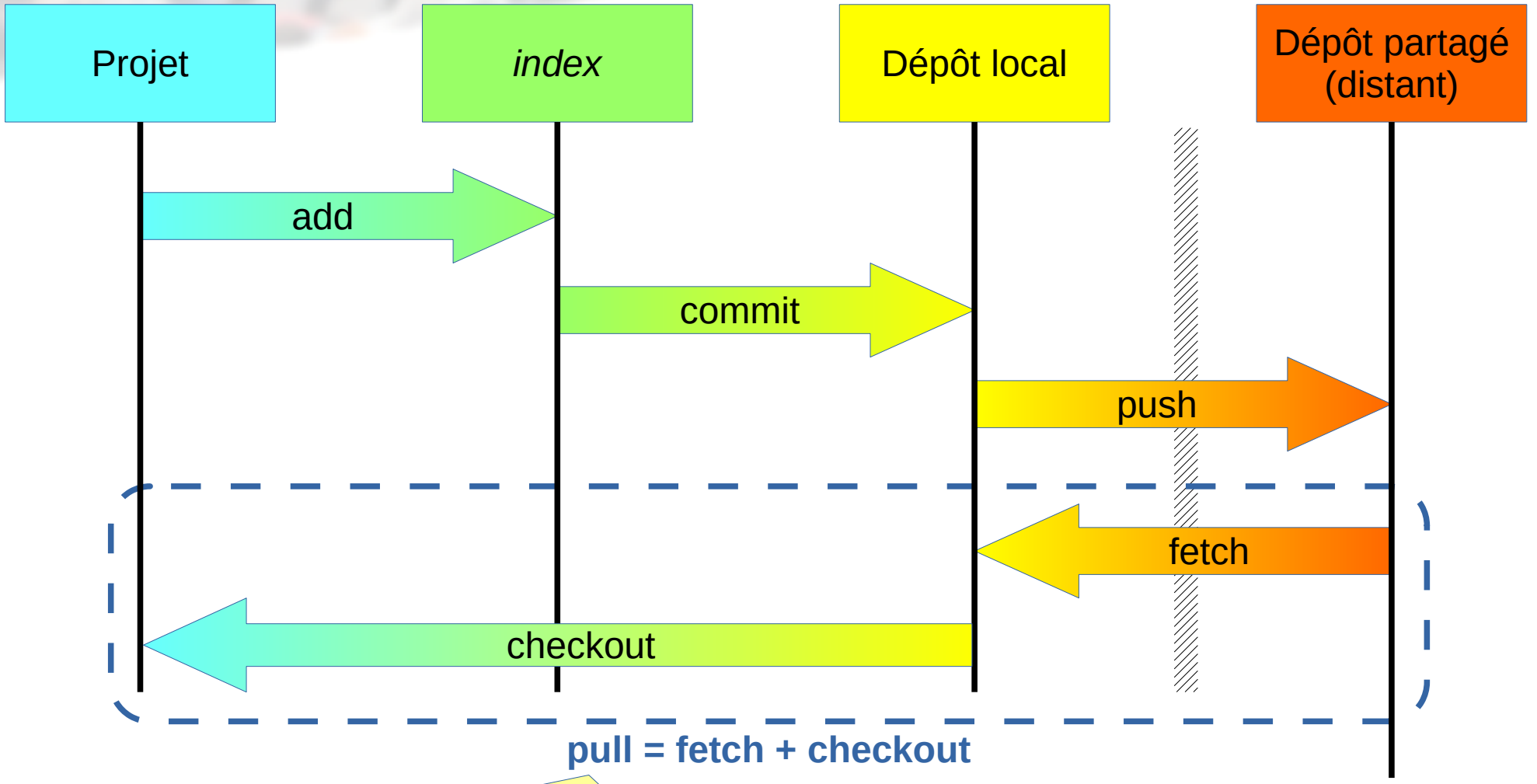
TRÈS IMPORTANT !

Utilisation d'un dépôt partagé

Un serveur de dépôts Git (comme Gitlab) permet aux membres d'une équipe de développement de partager les modifications qu'ils apportent à un projet commun



Fonctionnement général de Git (V2)



Si le dépôt local n'existe pas encore, on n'utilisera pas la commande **`pull`**, mais la commande **`clone`** qui le créera (en copiant l'intégralité de l'historique du dépôt distant) ainsi que le projet associé

Push : local → distant

Première opération de *push* vers le dépôt distant

Terminal

```
git remote add origin https://adresse/du/dépôt

git push -u origin main
```

Association de l'étiquette **origin** à l'adresse d'un dépôt distant (*remote* en anglais), par exemple sur le serveur **Gitlab** du département.

Vérification : `git remote -v`

Suppression de l'association : `git remote rm origin`

Envoi des *commits* de la branche locale **main** (branche par défaut, créée automatiquement lors de l'initialisation du dépôt à l'aide de **git init**) vers le dépôt distant identifié par l'étiquette **origin**.

Opérations de *push* ultérieures pour le même dépôt

Terminal

```
git push
```

Pour un autre dépôt, il faut répéter la procédure ci-dessus, en utilisant bien sûr un autre dépôt distant.

Pousse automatiquement la branche **main** vers **origin** (l'association des deux ayant été mémorisée en utilisant l'option **-u** lors du premier *push*)

Push : conditions

Dépôt local — Commit 1 — Commit 2 — Commit 3 — Commit 4

Dépôt distant avant *push* — Commit 1 — Commit 2

Dépôt distant après *push* — Commit 1 — Commit 2 — Commit 3 — Commit 4

Cette opération n'est possible que si le dépôt local contient l'ensemble des *commits* du dépôt distant. Si ce n'est pas le cas (historiques divergents), il faudra d'abord effectuer un *pull*

Dépôt local — Commit 1 — Commit 2 — Commit 3 — Commit 4

Dépôt distant avant *push* — Commit 1 — Commit 2 — Commit 5

Dépôt distant après *push* — impossible

Cloner un dépôt distant

Terminal

```
git clone https://adresse/du/dépôt
```

Crée une copie locale du dépôt distant :

- L'historique est intégralement copié
- Les fichiers du projet sont dans l'état du *commit* le plus récent
- Il est bien sûr possible d'ajouter de nouveaux *commits* à ce dépôt local

Attention à ne pas passer cette commande dans un sous-répertoire d'un projet déjà versionné par Git

Opérations de *push* ultérieures pour le même dépôt

Terminal

```
git push
```

La branche origin a été définie avec l'adresse du dépôt lors du *clone.* Les opérations de *push* envoient donc automatiquement les nouveaux *commits* vers le dépôt initial.

Pull : distant → local

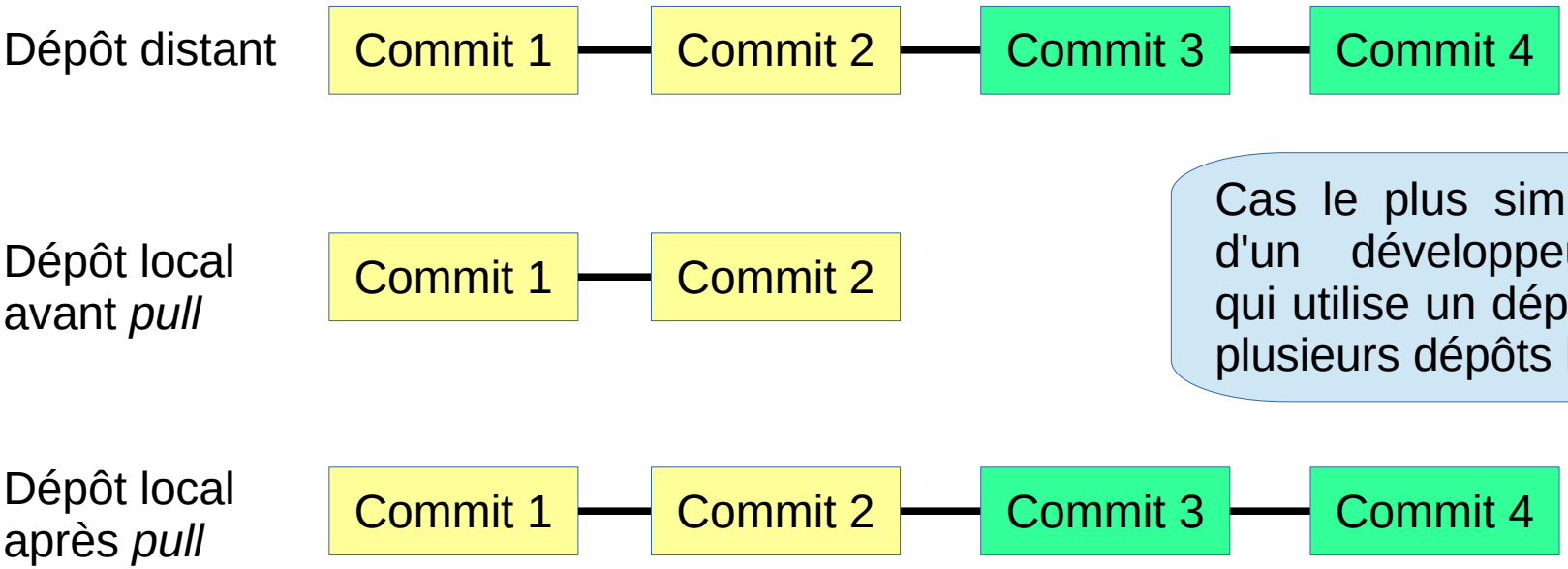
Terminal

```
git pull
```

Télécharge les *commits* du dépôt distant qui ne sont pas encore présents dans le dépôt local, les ajoute à celui-ci et met à jour les fichiers du projet pour qu'ils correspondent à l'état du *commit* le plus récent.

ATTENTION

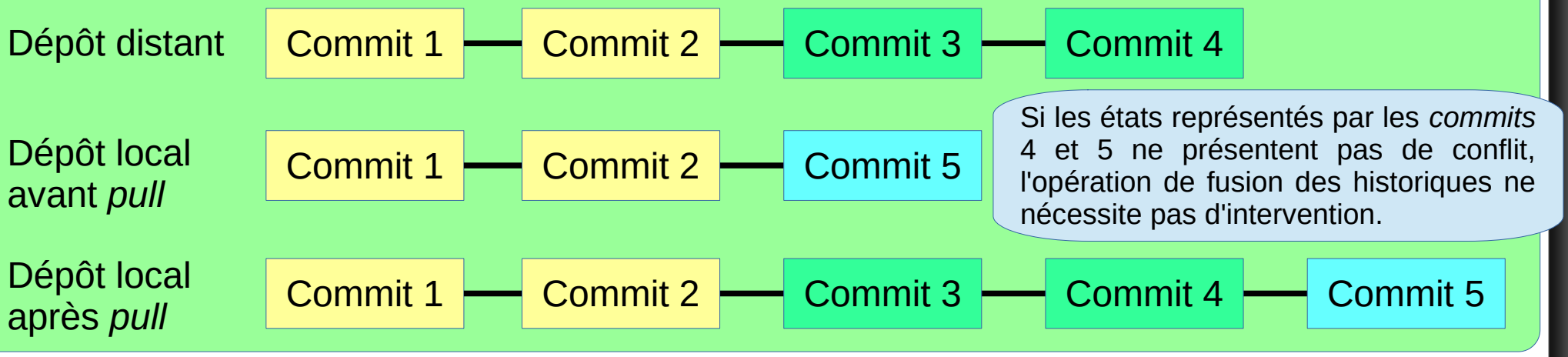
Cette opération n'est possible que si les fichiers du projet n'ont pas été modifiés depuis le dernier *commit* local



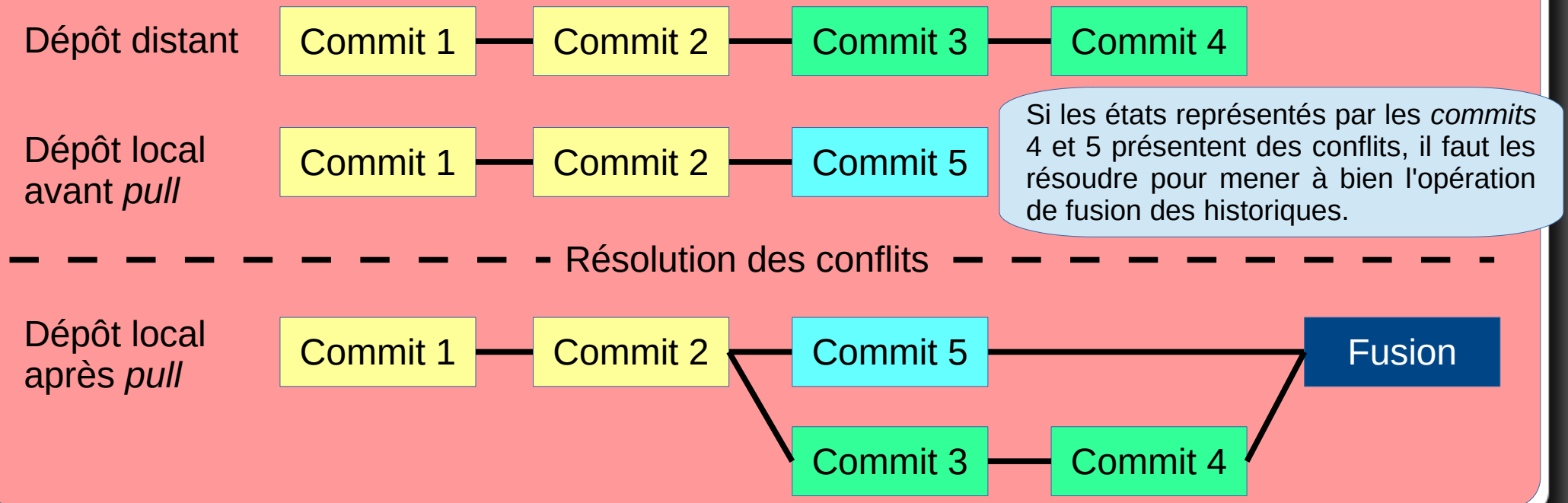
Cas le plus simple, typique d'un développeur solitaire qui utilise un dépôt central et plusieurs dépôts locaux

Pull : historiques divergents

Historiques divergents sans conflits



Historiques divergents avec conflits



Résolution des conflits

Terminal

```
cd répertoire/du/projet
git status
evim fichier.avec.conflits
git add fichier.avec.conflits
git commit
git push
```

Répéter ces opérations tant qu'il reste des conflits

Affiche (notamment) les fichiers contenant des conflits

Édition d'un des fichiers contenant un ou plusieurs conflit(s) (voir page suivante)

Après suppression des conflits du fichier

Après avoir résolu tous les conflits

Ces opérations peuvent être réalisées à l'aide d'un outil graphique. Ex :

```
meld .
```

Pousser les *commits* locaux (y compris le *commit* de fusion créé lors de la résolution des conflits)

Marqueurs de conflits

Lors du *pull*, git a inséré des marqueurs dans les fichiers concernés pour délimiter les sections comportant des conflits.

fichier.avec.conflits - Mousepad

```
...
<<<<<<< HEAD
...
=======
...
>>>>>>> b61ded13ae059947f7c488714164b703e449548b
...
```

Marqueur de début d'une section comportant un conflit

Lignes de la version du dépôt local

Séparateur

Lignes de la version du dépôt distant

Marqueur de fin

- Pour résoudre un conflit, il faut déterminer la version à conserver (version locale, version distante ou une combinaison des deux) et supprimer les marqueurs de conflit.
- Si vous utilisez un outil comme *meld*, celui-ci se charge de la suppression des marqueurs.

Remarques importantes

- Si **les conflits sont fréquents**, cela signifie que des développeurs différents travaillent souvent sur les mêmes fichiers : il faut donc **revoir la répartition des tâches** au sein de l'équipe de développement.
- Si les fichiers versionnés comportent des **informations sensibles** (comme des identifiants de connexion à un serveur de base de données), assurez-vous que les personnes ayant accès au dépôt partagé sont habilitées à connaître ces informations.

Rappel

pas de *commit* (ni, a fortiori, de *push*) d'un code insuffisamment testé.