Cloud Computing

Mise à niveau Git[1]

Danilo Carastan dos Santos

danilo.carastan-dos-santos@univ-grenoble-alpes.fr

2024

[1] Adapté du cours DevOps de Thomas ROPARS (cf dernier slide)

Motivations

Une équipe de développeurs participe à la réalisation d'une application.

Pour pouvoir être efficace dans votre nouvel environnement, il vous faudra:

- Comment conserver un historique ?
- Comment revenir en arrière ?
- Comment travailler à plusieurs en parallèle sur le même code ?
- Comment gérer plusieurs versions du code à la fois ?
- Comment savoir ce qui a été modifié et par qui (et pourquoi)?

Solution : utiliser un VCS (Version Control Software)

Ce qu'on y stocke

Ce qu'on y met

- Fichiers source (.java, .c, .html. etc.)
- Fichiers binaires non dérivés des sources (images, pdfs non générés par code source)
- Fichiers de configuration, compilation (Makefile, .yaml, etc)

Ce qu'on y stocke

Ce qu'on y met

- Fichiers source (.java, .c, .html. etc.)
- Fichiers binaires non dérivés des sources (images, pdfs non générés par code source)
- Fichiers de configuration, compilation (Makefile, .yaml, etc)

Ce qu'on n'y met pas

- Fichiers temporaires (.log, .out, etc)
- Fichiers générés (.jar, .class, .o, .exe, .pdf)

L'essentiel : Commit

Un Commit stocke l'état d'une partie du dépôt à un instant donné.
Quelques informations importantes stockées dans un commit :

- Un pointeur vers un ou plusieurs autres Commits (historique)
- Les informations sur l'auteur du Commit
- Une description sous forme d'une chaîne de caractères

Les commandes

Git est un ensemble de commandes. Les commandes sont de la forme:

```
git commande options
```

Exemple

```
git add file1.txt
```

Création d'un dépôt

Création d'un dépôt *serveur*

```
$ mkdir projet.git
$ cd projet.git
$ git --bare init
```

- Ne contient pas les fichiers versionnés, mais juste l'historique
- On ne travaille pas sur cette version

Création d'un dépôt

Initialisation d'un dépôt

```
$ cd myproject
$ git init
$ git add .
$ git commit -m 'initial commit'
$ git remote add origin git@gitserver:/XX/XX/project.git
$ git push origin master
```

- Créer un répertoire local `myproject` pour stocker notre version du projet.
- Associer le dépôt local avec le dépôt distant
- Envoyer l'état initial du dépôt vers le serveur
- À partir de ce moment, tout le monde peut obtenir sa copie locale du dépôt en utilisant `git clone`

Cloner un dépôt existant

Très souvent, un dépôt existe déjà. On veut alors récupérer une copie de ce dépôt.

Cloner un dépôt

```
$ git clone URL
```

- Crée une copie locale du dépôt entier.
- L'URL peut être de la forme :
  - file://./myproject/project.git
  - http://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux-2.6.git
  - git://github.com/schacon/grit.git

Quelques commandes

`add` : Ajoute dans l'index un fichier à commiter dans son état actuel.

`commit` : enregistre dans le dépôt local les modifications qui ont été ajoutées dans l'index par une commande add

`restore --staged` : supprime la référence d'un fichier de l'index ajouté par une commande add.

L'index est aussi appelé staging area.

Souvent on veut simplement commiter toutes les modifications en cours (= fichiers de l'index + fichiers modifiés):

```
$ git commit -a
```

Question

Supposons la structure du dossier suivant. Les fichiers `1-git.XX` sont générés par la compilation du fichier `1-git.tex`

```
$ 2024-2025-MIAGE/Cours/1-git$ ls -lah
total 608K
drwxrwxr-x 3 dancarastan dancarastan 4,0K juil.  5 16:36 .
drwxrwxr-x 4 dancarastan dancarastan 4,0K juil.  5 10:06 ..
-rw-rw-r-- 1 dancarastan dancarastan 2,5K juil.  5 16:36 1-git.aux
-rw-rw-r-- 1 dancarastan dancarastan    0 juil.  5 10:47 1-git.bbl
-rw-rw-r-- 1 dancarastan dancarastan 104K juil.  5 16:36 1-git.bcf
-rw-rw-r-- 1 dancarastan dancarastan  41K juil.  5 16:36 1-git.fdb-latexmk
-rw-rw-r-- 1 dancarastan dancarastan 130K juil.  5 16:36 1-git.fls
-rw-rw-r-- 1 dancarastan dancarastan  67K juil.  5 16:36 1-git.log
-rw-rw-r-- 1 dancarastan dancarastan 1,2K juil.  5 16:36 1-git.nav
-rw-rw-r-- 1 dancarastan dancarastan    0 juil.  5 16:36 1-git.out
-rw-rw-r-- 1 dancarastan dancarastan 178K juil.  5 16:36 1-git.pdf
-rw-rw-r-- 1 dancarastan dancarastan 2,2K juil.  5 16:36 1-git.run.xml
-rw-rw-r-- 1 dancarastan dancarastan    0 juil.  5 16:36 1-git.snm
-rw-rw-r-- 1 dancarastan dancarastan  23K juil.  5 16:36 1-git.synctex.gz
-rw-rw-r-- 1 dancarastan dancarastan  13K juil.  5 16:36 1-git.tex
-rw-rw-r-- 1 dancarastan dancarastan    0 juil.  5 16:36 1-git.toc
-rw-rw-r-- 1 dancarastan dancarastan  608 juil.  5 16:36 1-git.vrb
drwxrwxr-x 2 dancarastan dancarastan 4,0K juil.  5 10:40 Figures
-rw-rw-r-- 1 dancarastan dancarastan 7,3K juil.  5 10:45 gitdags.sty
-rw-rw-r-- 1 dancarastan dancarastan  273 juil.  5 10:06 Makefile
```

Je décide d'utiliser `git add .` et puis `git commit -m "My commit"`. Que se passera-t-il ?

Réponse

Réponse 1 : Il ne se passera rien, le dépôt n'est pas initialisé. On devrait initialiser le dépôt avec

```
$ git init .
```

Réponse 2 : Supposons que le dépôt est déjà initialisé, tous les fichiers temporaires (.aux, .log, etc) et pdf générés par la compilation LaTeX(1-git.pdf) seront commités.

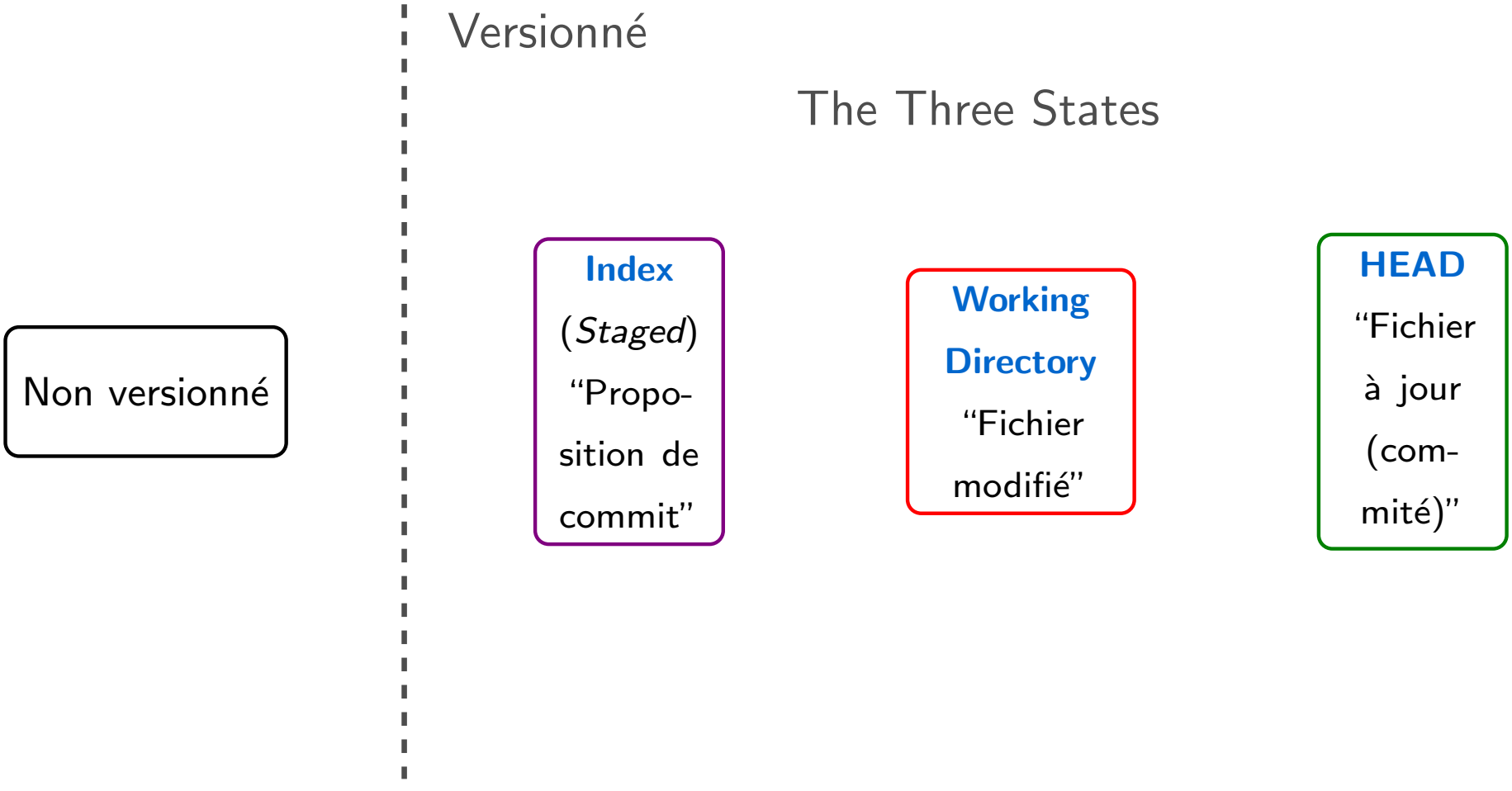
Solutions :

pénible : Ajouter (`git add`) chaque fichier individuellement

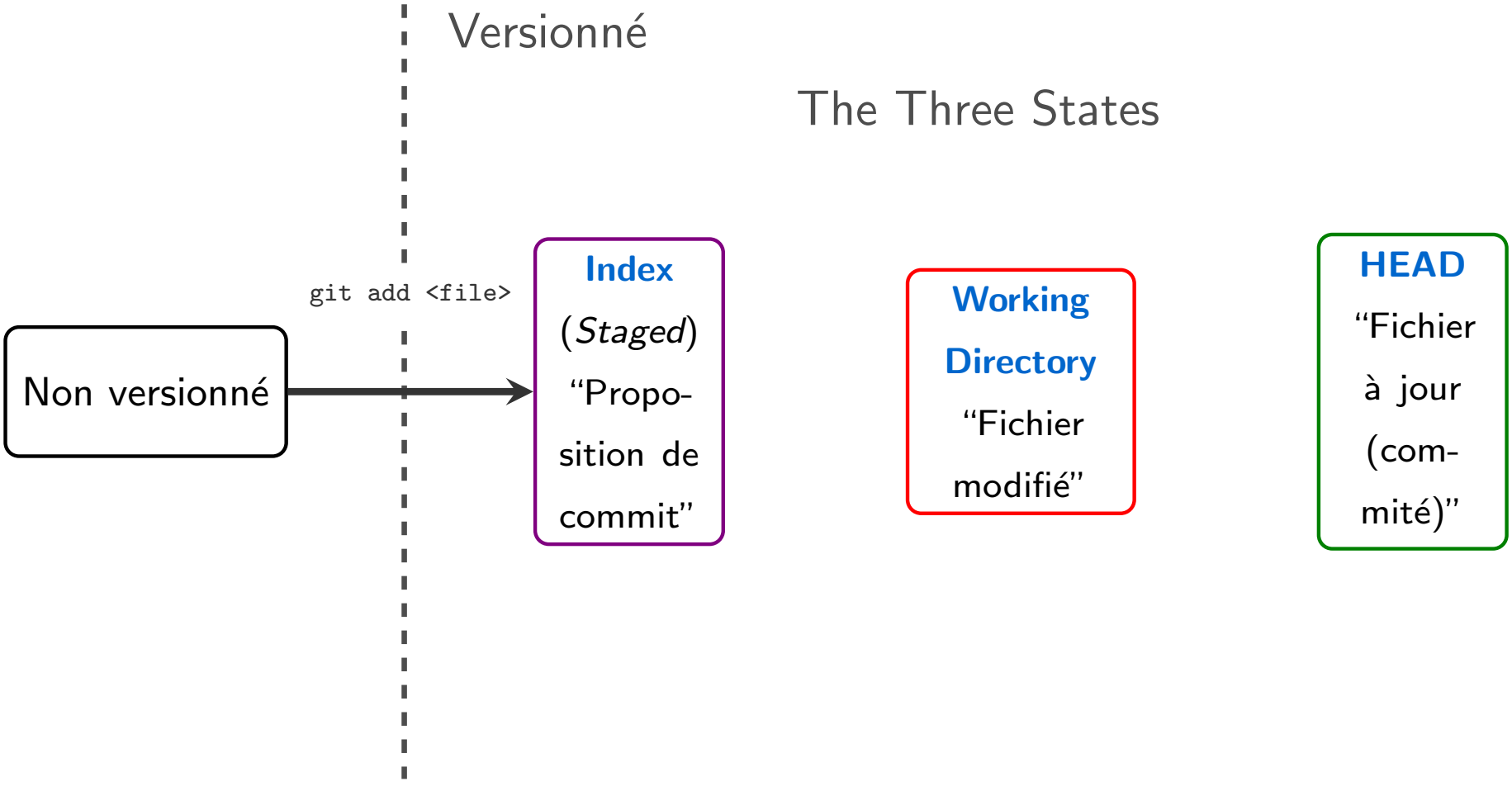
recommandé : Utiliser un fichier `.gitignore`[1]

[1] https://git-scm.com/book/fr/v2/Les-bases-de-Git-Enregistrer-des-modifications-dans-le-dépôt. Partie "Ignorer des fichiers"

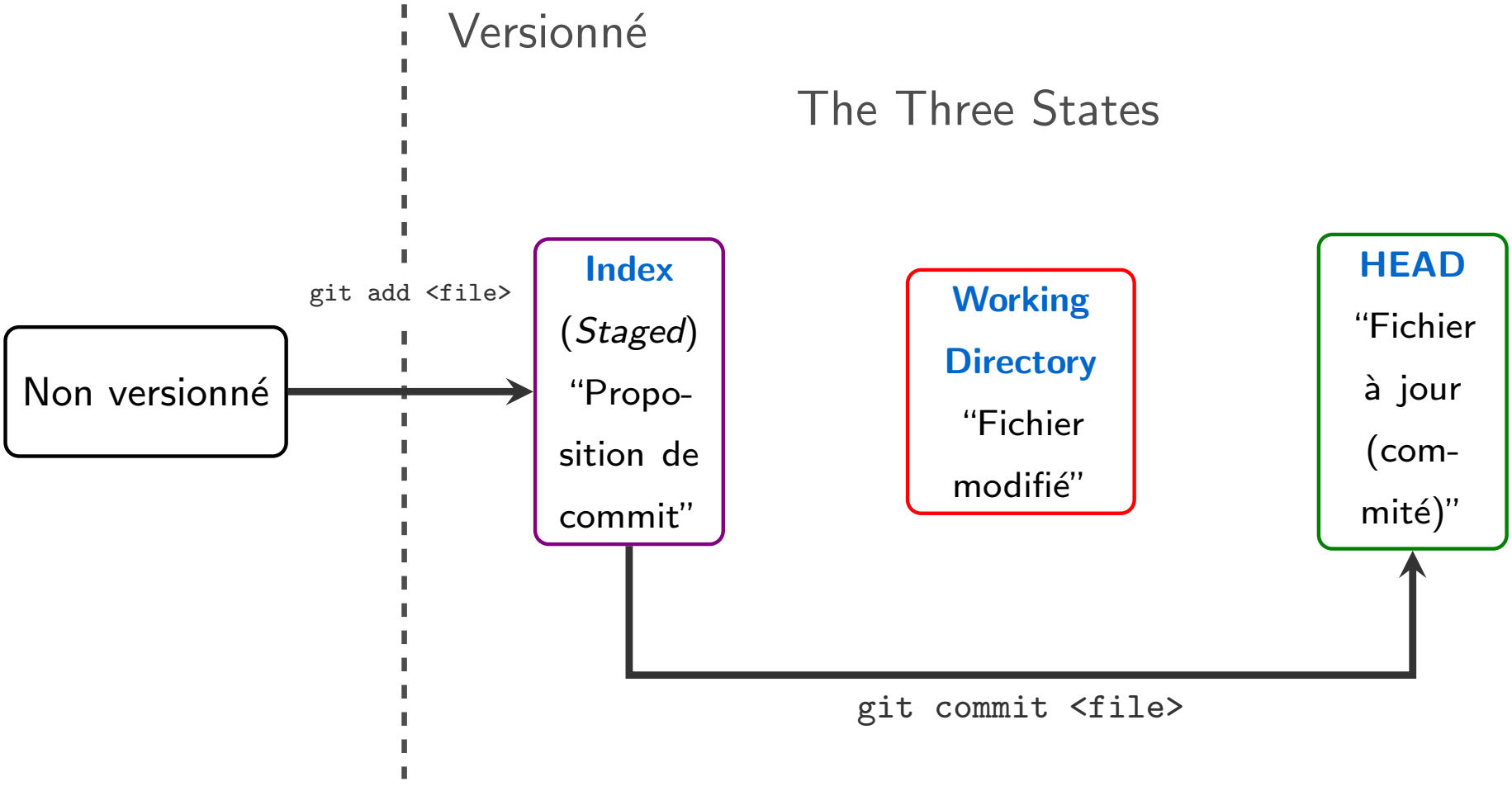
Le cycle de vie d'un fichier



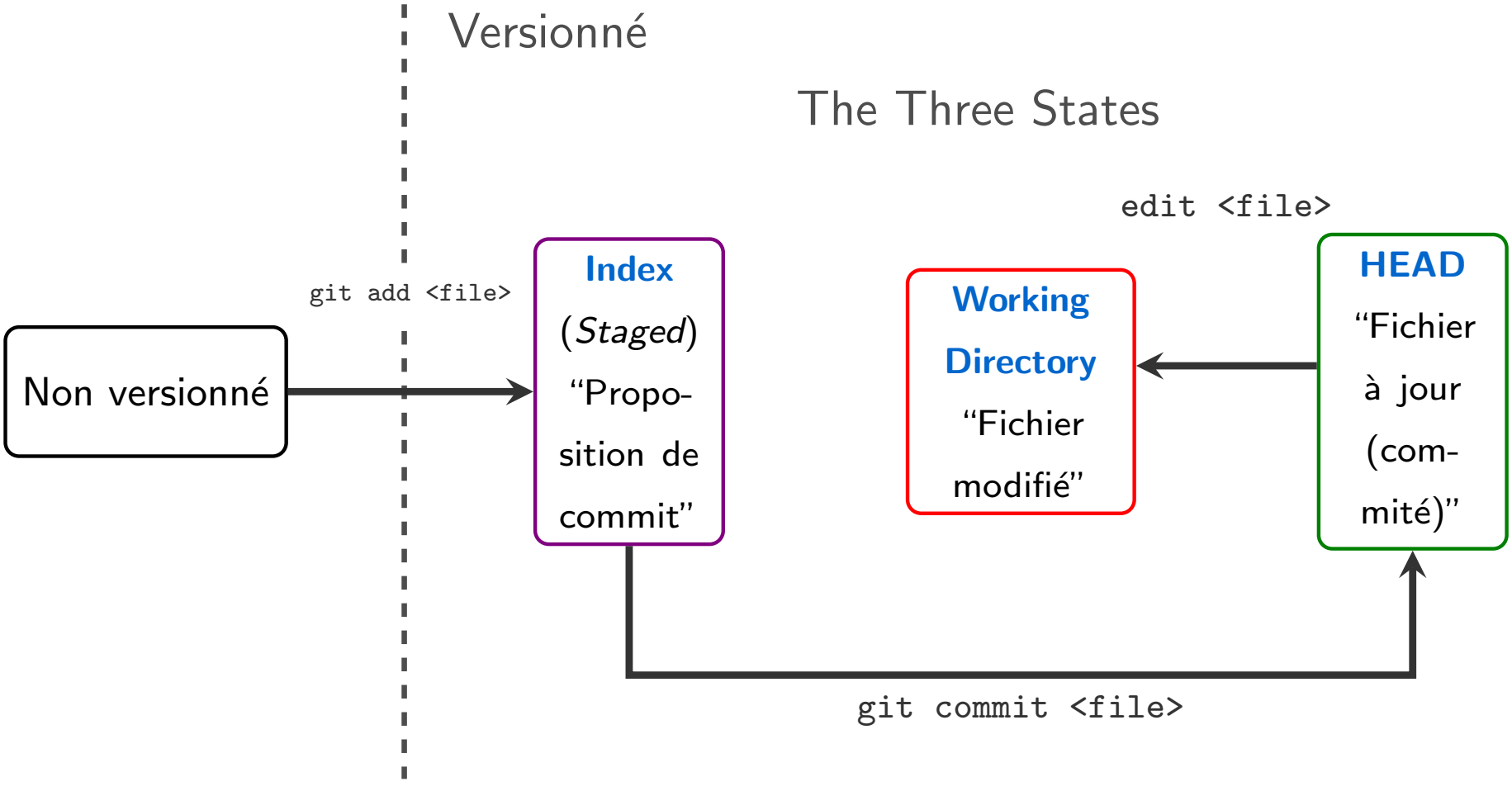
Le cycle de vie d'un fichier



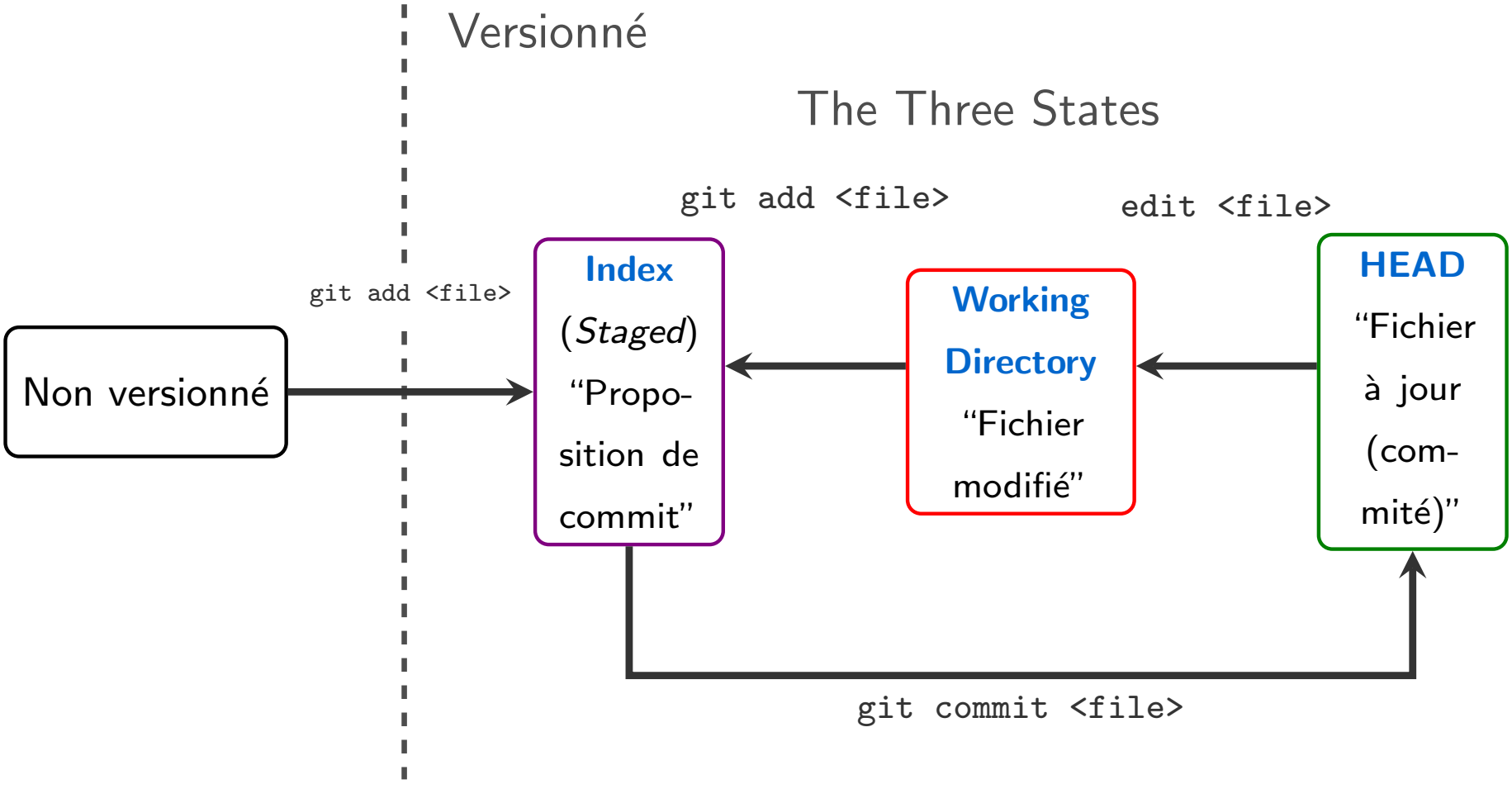
Le cycle de vie d'un fichier



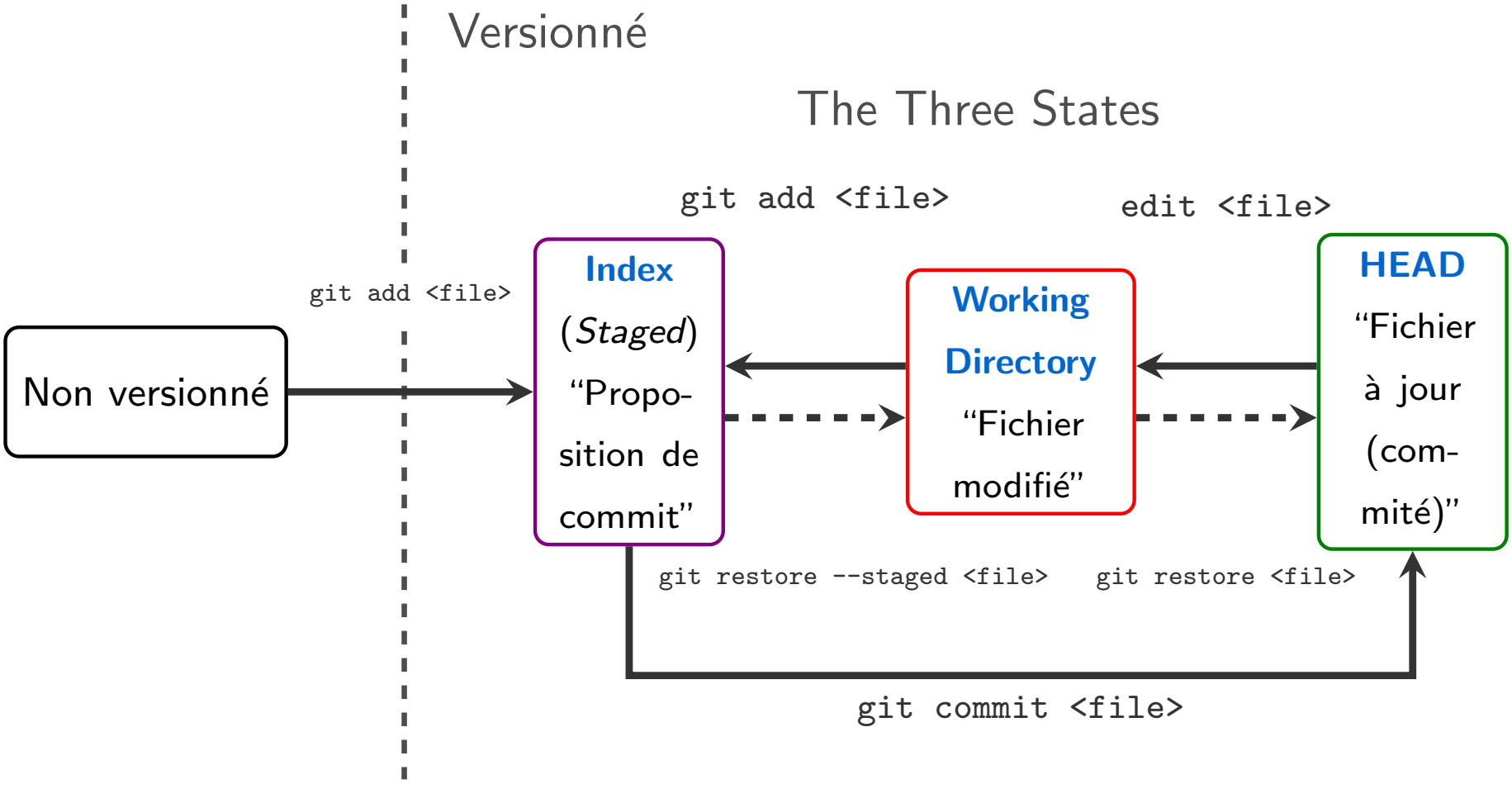
Le cycle de vie d'un fichier



Le cycle de vie d'un fichier

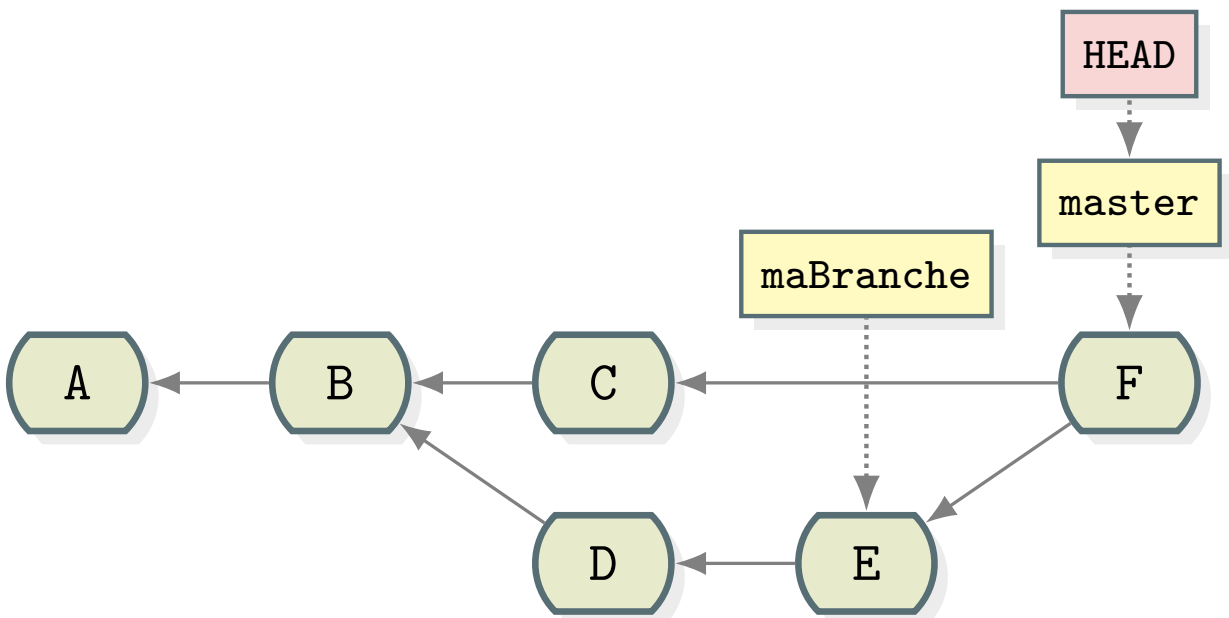


Le cycle de vie d'un fichier



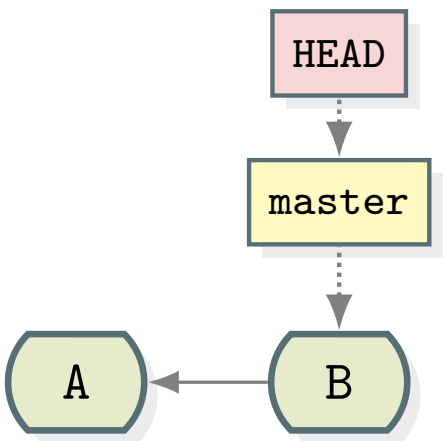
La notion d'historique

- Un commmit enregistre les modifications indexées (par une commande add) à un instant donné
- Une branche est un pointeur sur un commit
- Chaque commit pointe vers son prédécesseur
- La variable HEAD pointe sur la branche sur laquelle on travaille actuellement.



Historique : les branches

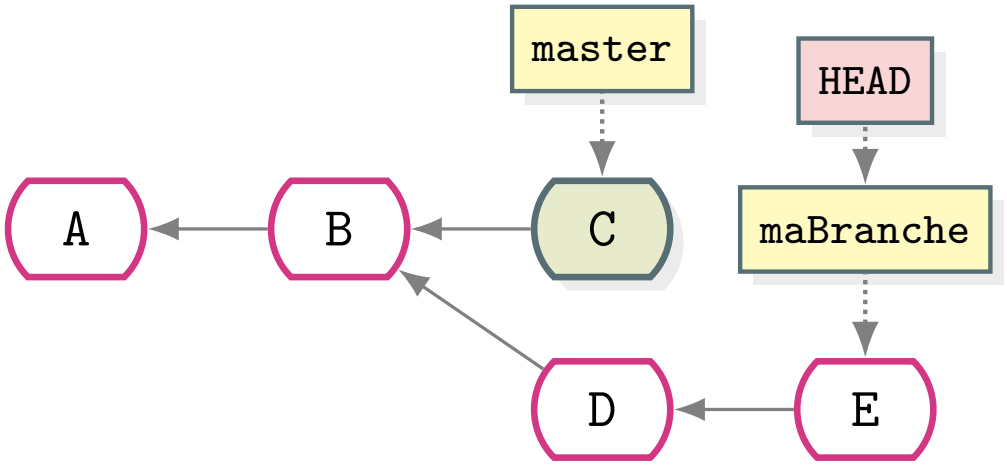
- L'historique peut inclure plusieurs branches, c'est-à-dire des sous-graphes qui évoluent en parallèle.




```
echo modif A > fichier1.txt
git add fichier1.txt
git commit -m "commit A"
echo modif B > fichier1.txt
git commit -am "commit B"
```

Historique : les branches

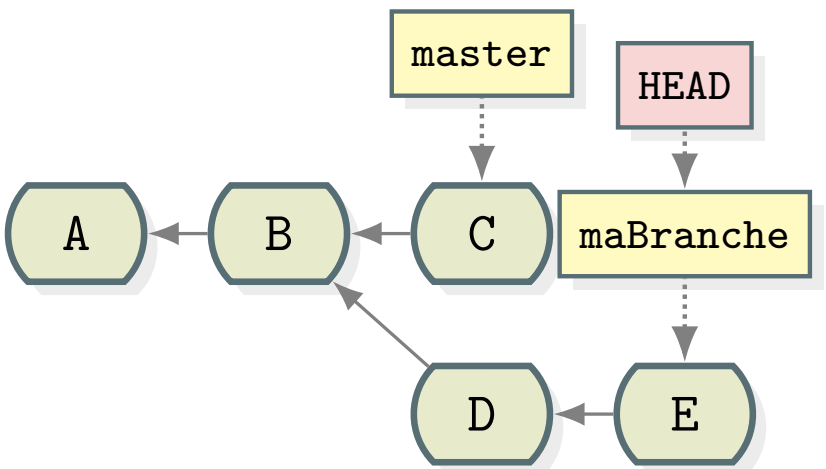
- L'historique peut inclure plusieurs branches, c'est-à-dire des sous-graphes qui évoluent en parallèle.




```
echo modif A > fichier1.txt
git add fichier1.txt
git commit -m "commit A"
echo modif B > fichier1.txt
git commit -am "commit B"
git branch maBranche
echo modif C > fichier1.txt
git commit -am "commit C"
git checkout maBranche
echo modif D > fichier2.txt
git add fichier2.txt
git commit -m "commit D"
echo modif E > fichier2.txt
git commit -am "commit E"
```

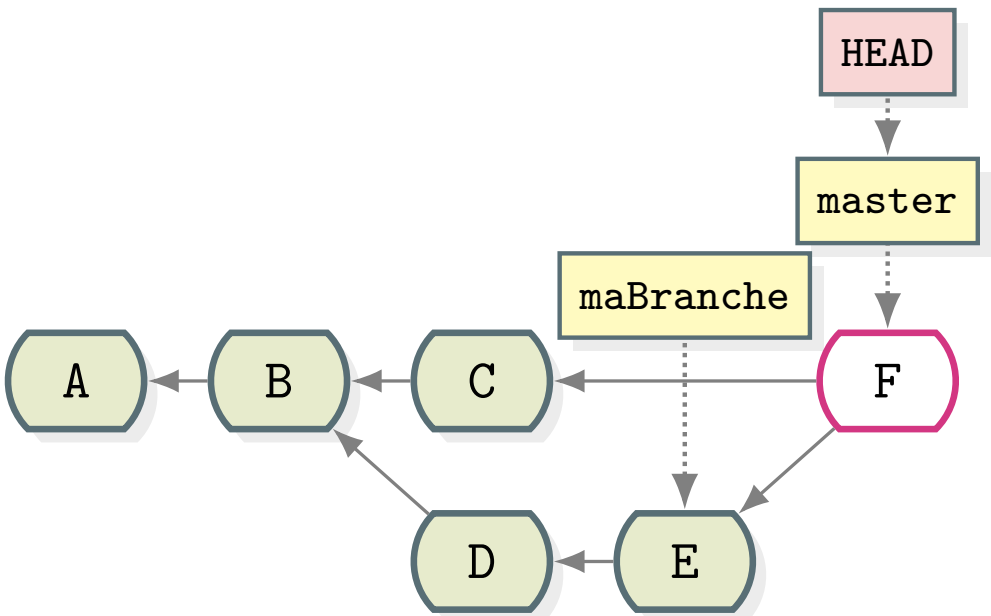
Historique : les merges

On appelle merge toute version ayant un degré sortant strictement supérieur à 1. Cette version correspond alors à la fusion des commits de plusieurs branches.



Historique : les merges

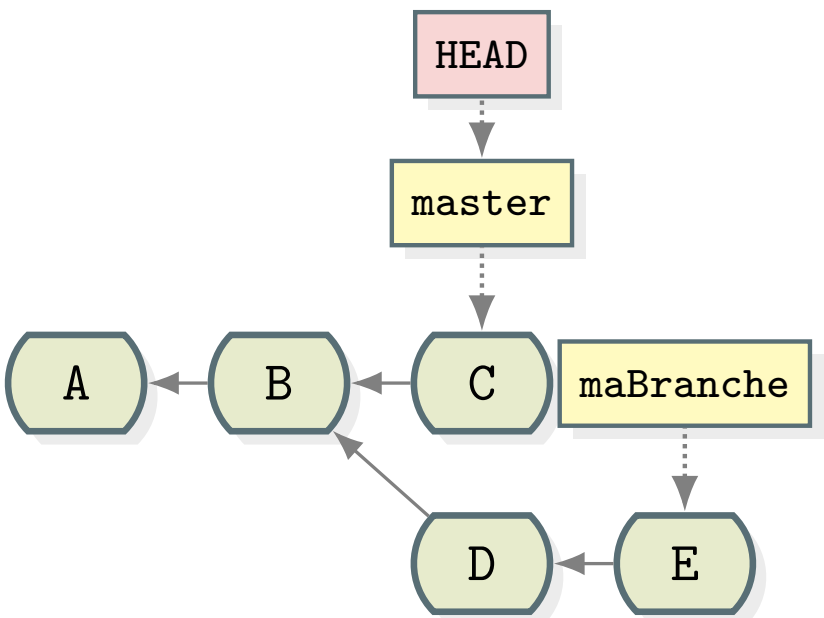
On appelle merge toute version ayant un degré sortant strictement supérieur à 1. Cette version correspond alors à la fusion des commits de plusieurs branches.




```
git checkout master
git merge maBranche
```

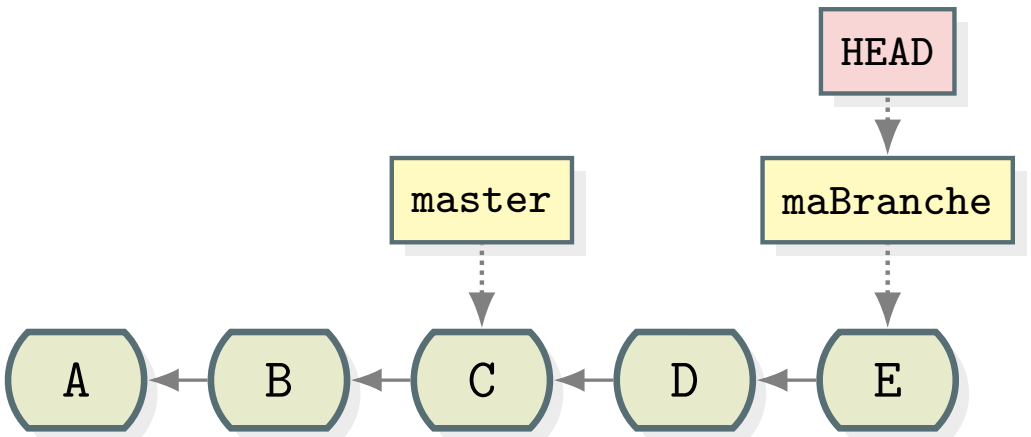
Historique : les rebases

- Autre manière de fusionner 2 branches
- Fusionne entièrement la branche source dans la branche destination
- Permet de simplifier l'historique



Historique : les rebases

- Autre manière de fusionner 2 branches
- Fusionne entièrement la branche source dans la branche destination
- Permet de simplifier l'historique
- **Ne jamais rebaser des commits qui ont déjà été poussés sur un dépôt public**

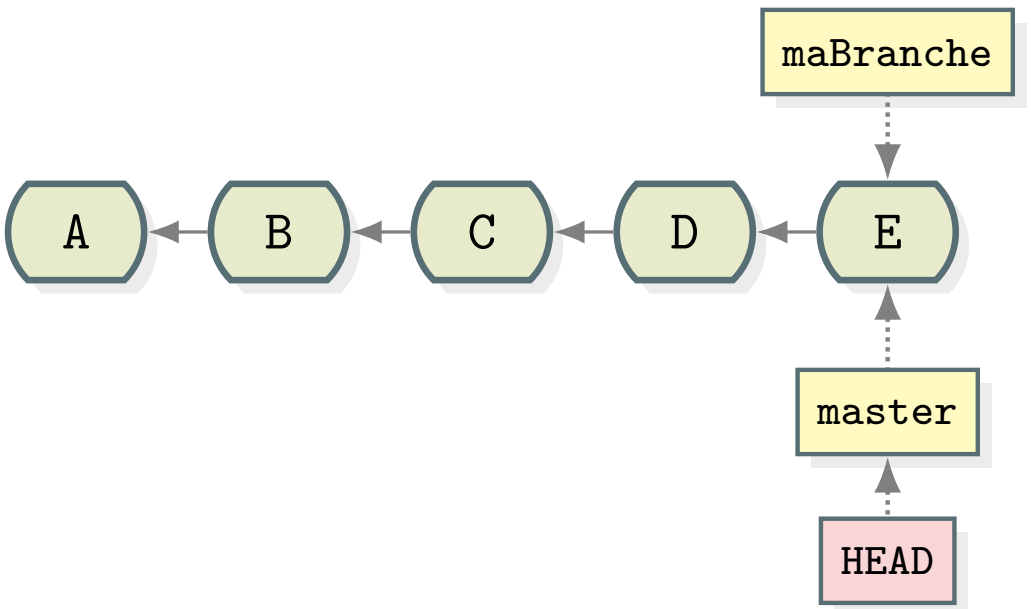



```
git checkout maBranche
git rebase master
```

Historique : les rebases

- Autre manière de fusionner 2 branches
- Fusionne entièrement la branche source dans la branche destination
- Permet de simplifier l'historique

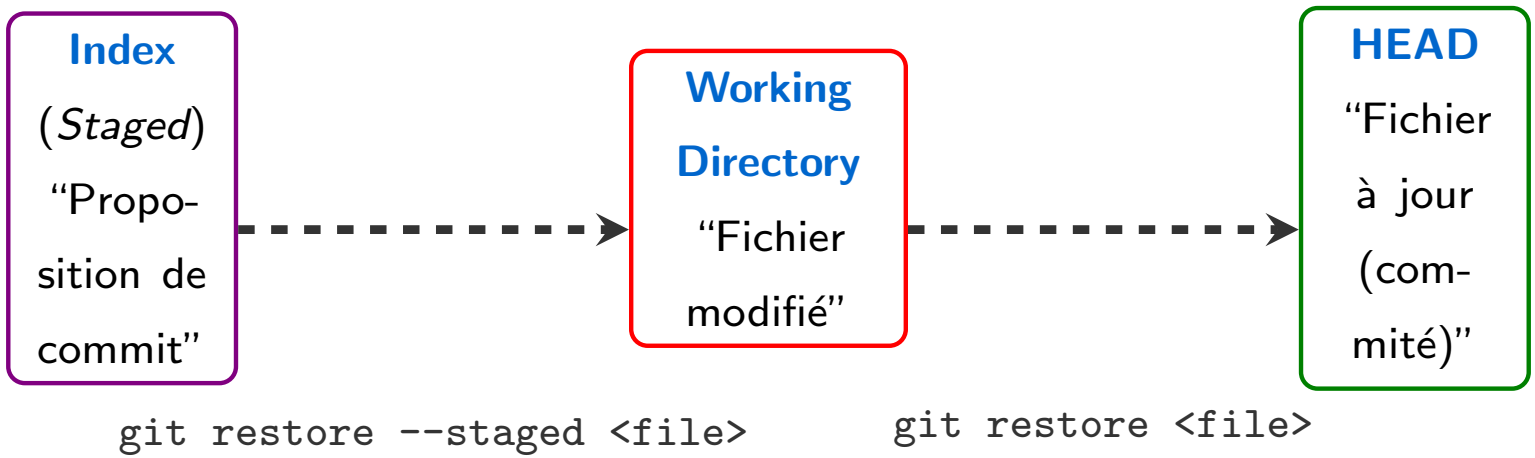
fast-forward merge lorsque le commit de destination est accessible en suivant l'historique des commits




```
git checkout maBranche
git rebase master
git checkout master
git merge maBranche
```

Revenir en arrière[1]

Cas de modifications non commitées



[1] Plus d'infos : `https://git-scm.com/book/fr/v2/Les-bases-de-Git-Annuler-des-actions`

A propos de git checkout

Une commande qui peut prêter à confusion

- Permet de naviguer dans les branches
- Permet de modifier le contenu de fichiers

Nouvelles commandes depuis git 2.23

- git switch pour changer de branche
- git restore pour modifier le contenu d'un ficher
- A utiliser en remplacement de `git checkout`

Revenir en arrière

Cas de modifications commitées

Trois commandes disponibles:

- amend : modifier le dernier commit
  - Ajoute des fichiers au commit
  - Changer le message de commit
- revert : annuler un commit par un autre commit
- reset : rétablir la situation d'un ancien commit

Si l'erreur a été rendue publique, la seule bonne pratique est `revert`.

La commande reset[1]

- Restaurer un ancien commit (mais en conservant toutes les modifications dans le Working Directory et l'Index)

```
git reset --soft commitID
```

- Restaurer un ancien commit et l'Index (mais en conservant toutes les modifications Working Directory)

```
git reset commitID
```

- Restaurer un ancien commit, l'Index, et le Working Directory

```
git reset --hard commitID
```

Question : quelle est la commande qui restaure le contenu des fichiers correspondants ?

[1] Plus d'infos : https://git-scm.com/book/fr/v2/Utilitaires-Git-Reset-démystifié

Pour aller plus loin

Pro Git book `https://git-scm.com/book/fr/v2`

GitHub Git Cheat Sheet `https://education.github.com/git-cheat-sheet-education.pdf`

Cours DevOps de Thomas ROPARS `https://tropars.github.io/teaching/#devops`