



FORMATION GIT

2 jours



Disposition de titre et de contenu avec liste

- **Module 1: Introduction**
- **Module 2: Installation et configuration**
- **Module 3: Premier pas**
- **Module 4: GitHub**
- **Module 5: Commandes GIT de base**
- **Module 6: Gestion des branches et Merge**
- **Module 7: Rebase**
- **Module 8: Git avec un dépôt distant**



MODULE 1

Introduction

Présentation

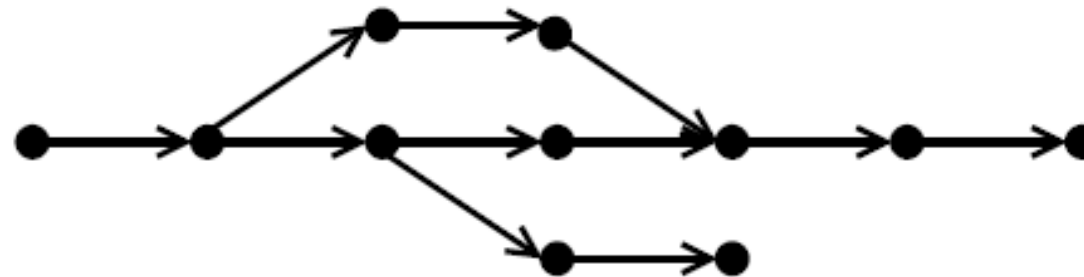
- Source Code Management
- Buts d'un gestionnaire de versions
- Historique des gestionnaires de versions
- Subversion
- Historique – Git
- VCS distribué (Git, Mercurial...)
- Serveur central avec Git
- Git est décentralisé

Source Code Management

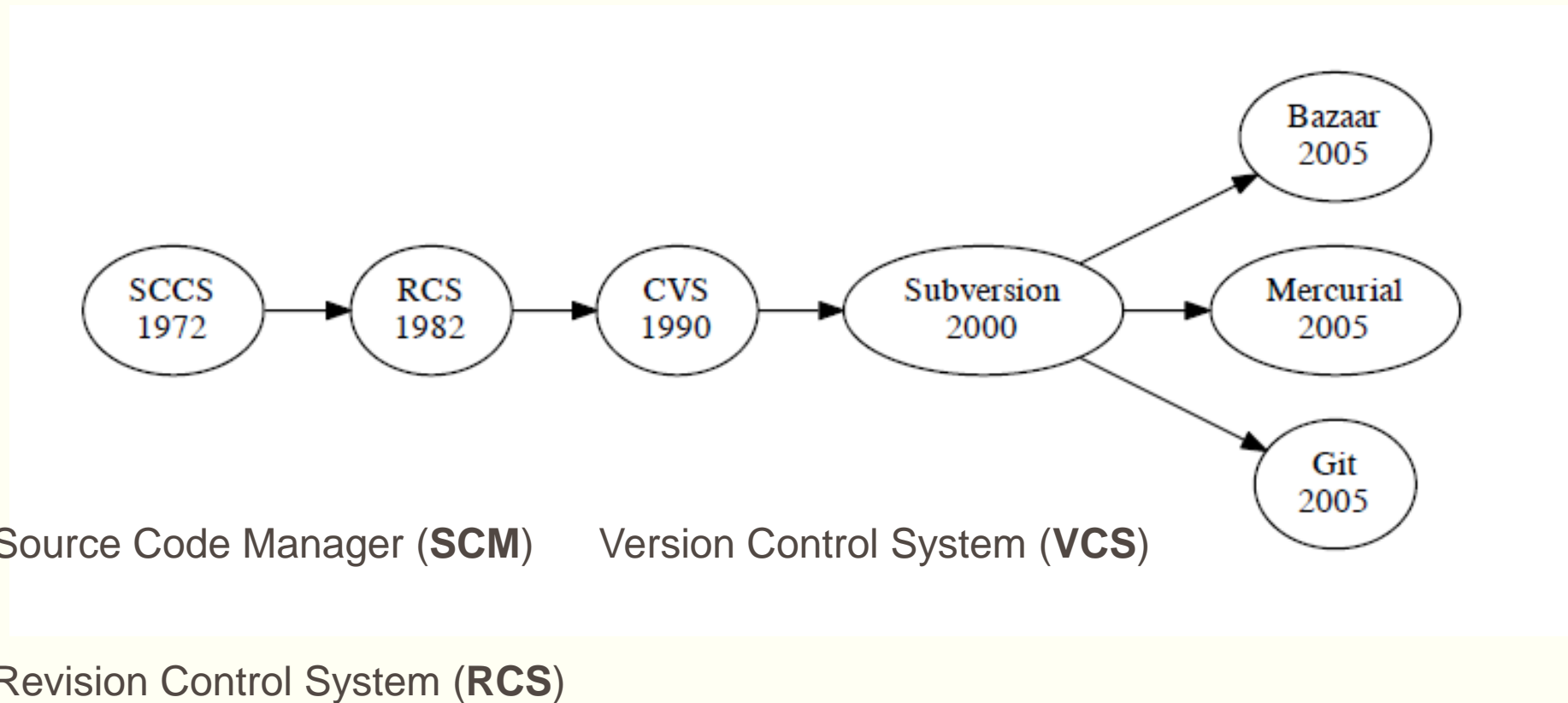
- Pour tout projet informatique, il faut une **stratégie de backup**
- On ajoute souvent une **gestion des versions**
- Un développeur peut proposer plusieurs **révisions** par jour

Buts d'un gestionnaire de versions

- Faciliter la gestion d'un projet de programmation ;
- Garder l'historique de toutes les modifications (commits) ;
- Travailler en équipe ;
- Avoir des branches de développement :
 - ✓ pour développer une fonctionnalité séparément ;
 - ✓ pour une certaine version (2.4.0 ! 2.4.1 ! ...).



Historique des gestionnaires de versions

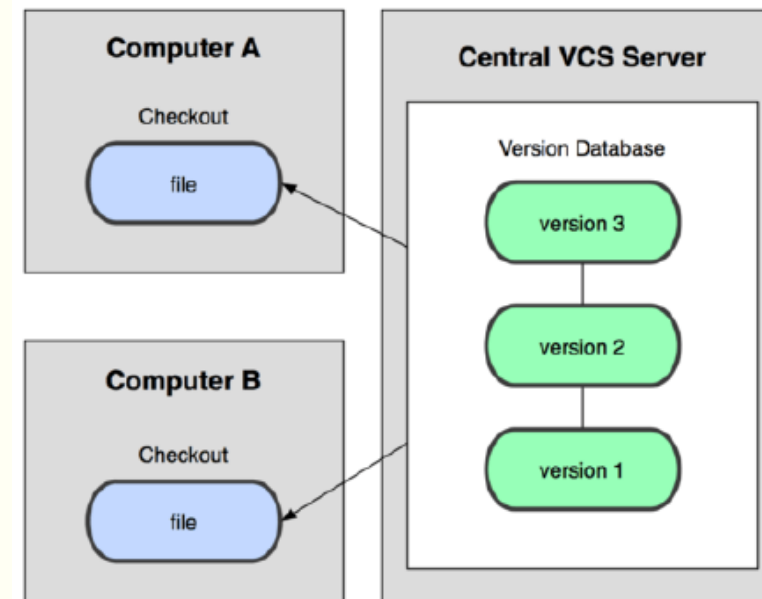


Subversion

Subversion (SVN) est **centralisé** :

- Un serveur central contient toutes les données ;
- Beaucoup de requêtes entre le client et le serveur (assez lent) ;
- Besoin d'une connexion internet pour travailler.

- **Centralisé** == *repository* (dépôt) central
- On “**emprunte**” et on travaille sur des *working copies* (copies de travail)



Historique - Git

- Git Système inventé par **Linus Torvalds** pour le kernel Linux
-
- Git a vu le jour en **avril 2005**
 - ✓ Premier commit le 8 avril
- Logiciel de gestion de versions décentralisé
 - ✓ Connexion internet uniquement pour les pull et push



Git

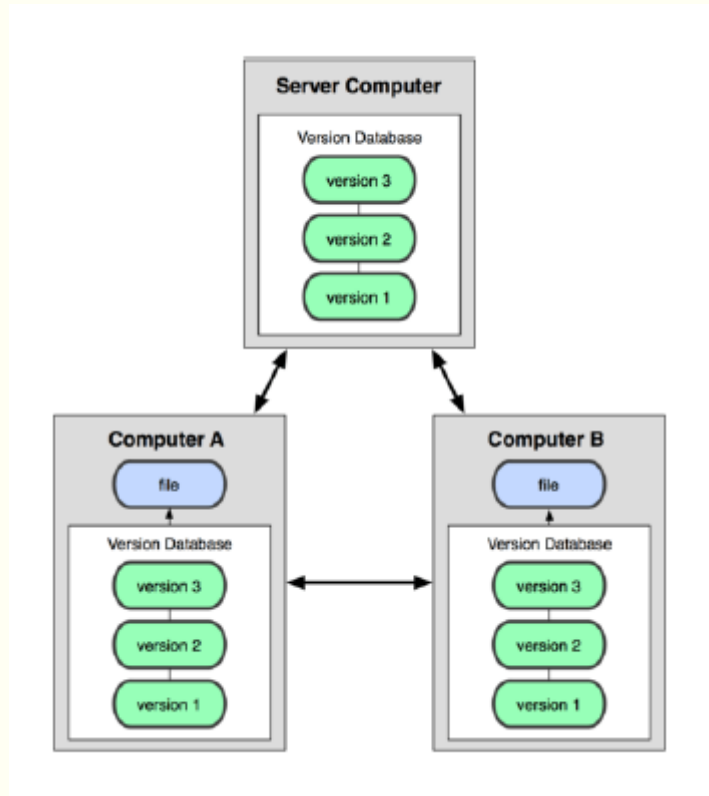
Git est **décentralisé/distribué** :

- Toutes les données sont sur notre machine ;
- Les opérations sont très rapides ;
- Connexion internet seulement pour les *pull* et *push*.

Git est aussi **plus puissant** et **plus flexible** :

- Pour la gestion des branches ;
- Possède de nombreuses fonctionnalités plus avancées.

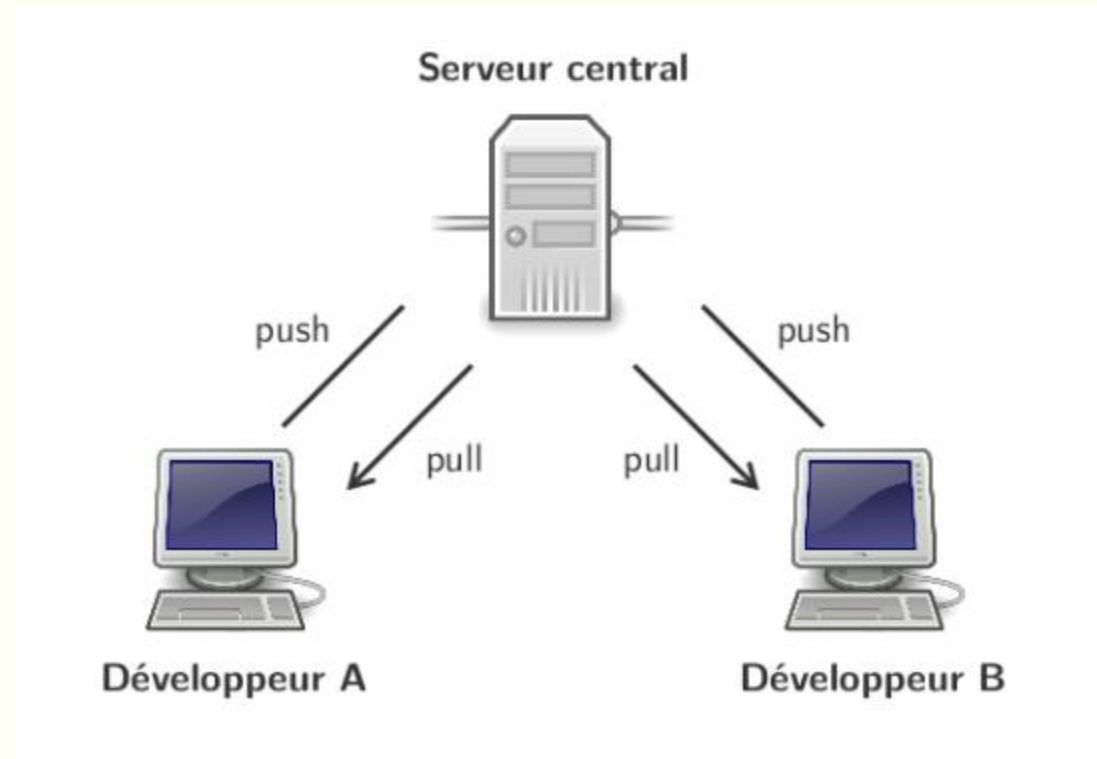
VCS distribué (Git, Mercurial...)



- **Décentralisé** : Les versions / branches / tags sont en local
- On travaille sur son *repository* local et on publie sur les autres *repositories*
- Possibilité d'avoir un *repository* central (mais pas obligé)

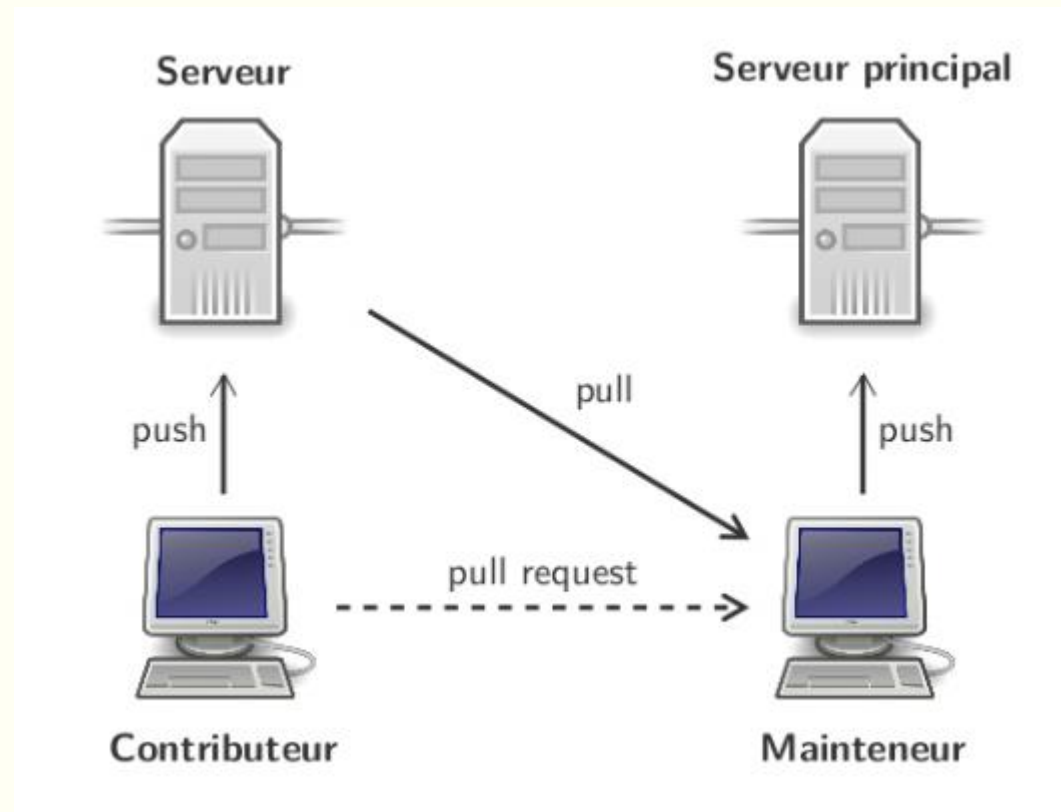
Serveur central avec Git

- Une manière simple de travailler en équipe ;
- Accès en écriture pour **tous les développeurs**.



Git est décentralisé

- Seul les **mainteneurs** ont accès en écriture ;
- Les **contributeurs** font des pull requests.



Disposition de titre et de contenu avec liste

Question ?



MODULE 2

Installation et configuration

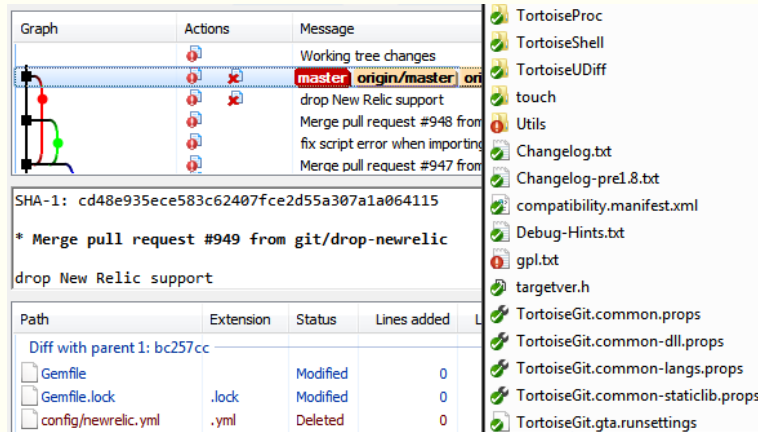
Présentation

- Installation
- Clients graphique
- Configuration : Votre identité
- Configuration : Votre éditeur de texte
- Configuration : Votre outil de différences
- Vérifier vos paramètres

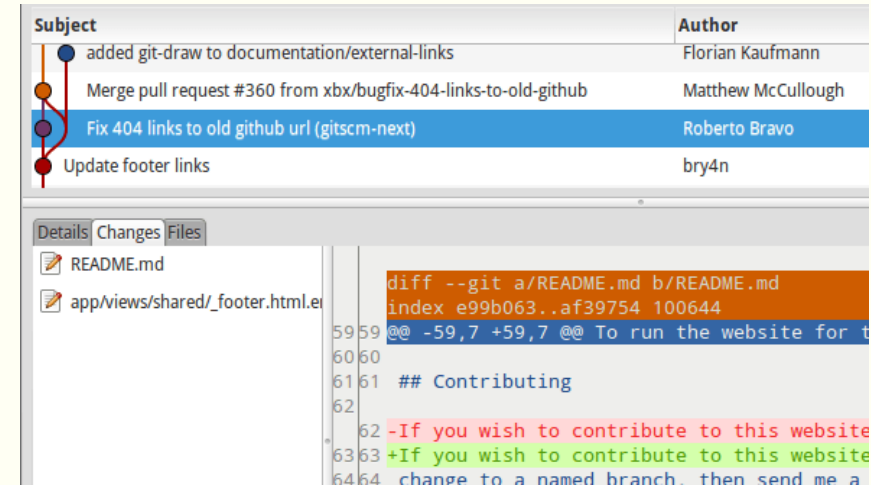
Installation

- **Sous Linux:** via le gestionnaire de paquet
 - Ubuntu : apt-get install git
 - Centos: sudo yum install git
- **Sous OSX :** via homebrew (brew install git)
- **Sous Windows :** <https://gitforwindows.org/>

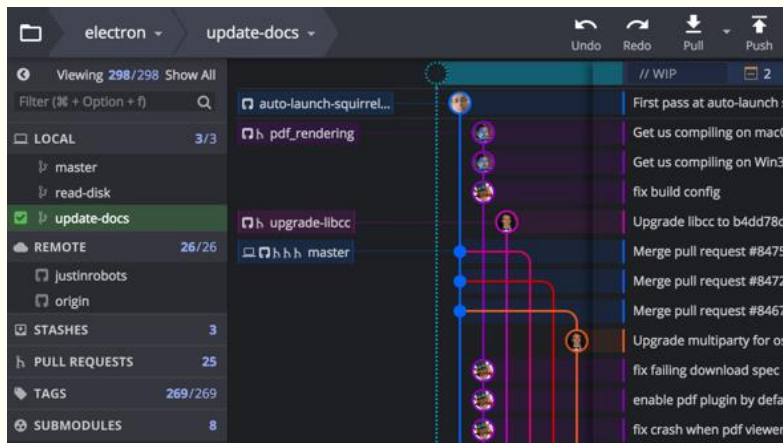
Clients graphique : Windows



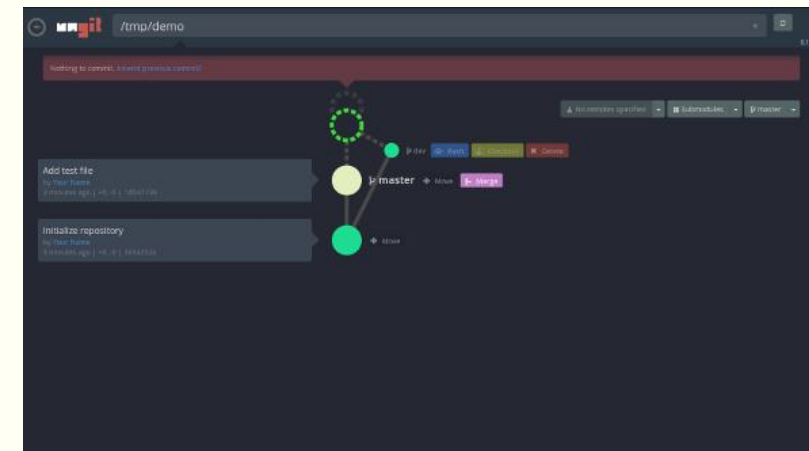
• TortoiseGit : Free



Gitg : Free

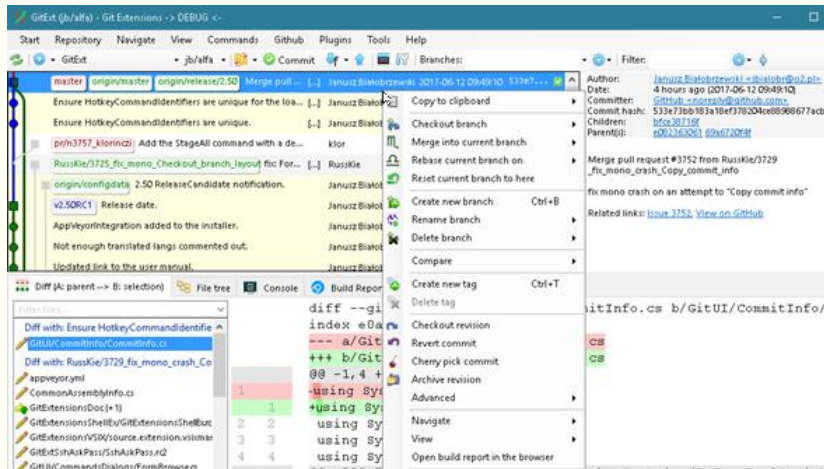


• GitKraken : Free for non-commercial use

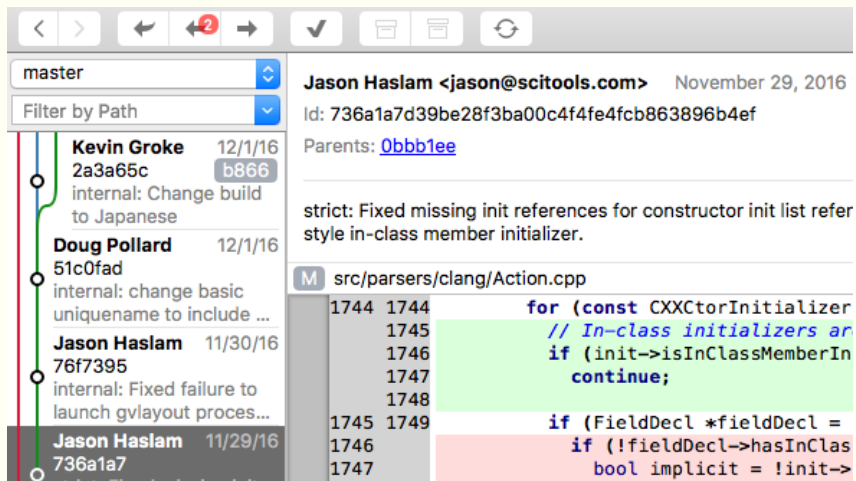


Ungit : Free

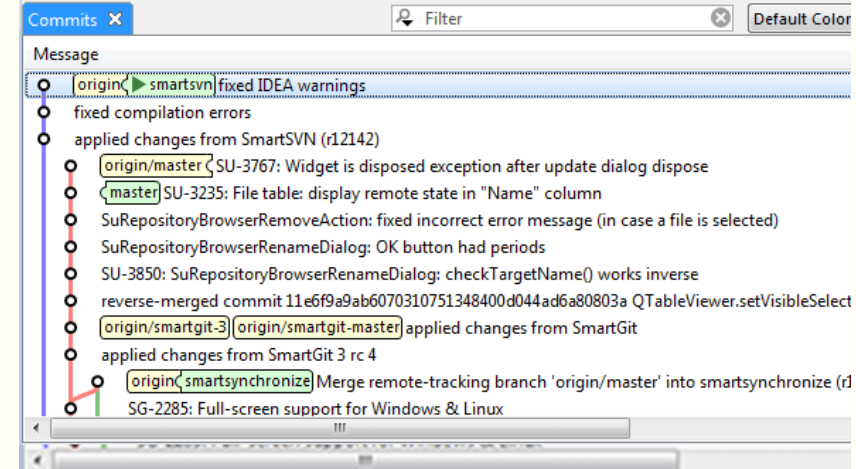
Clients graphique : Linux



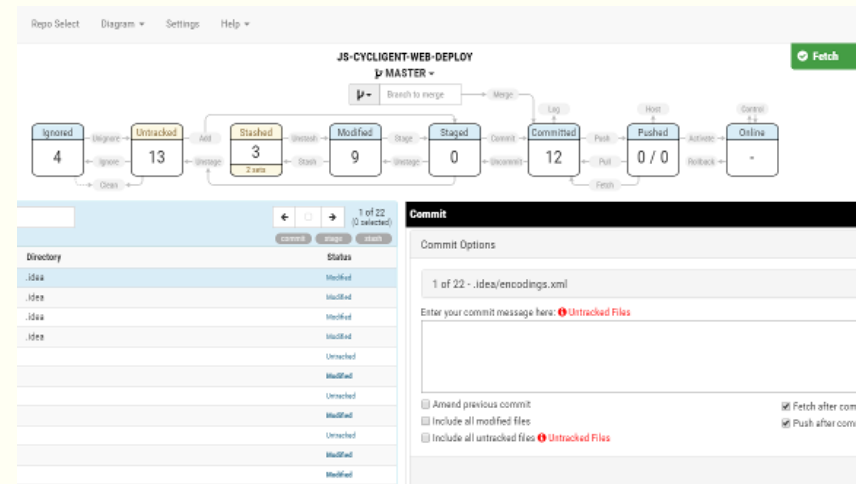
• Git Extensions : Free



• GitAhead : Free



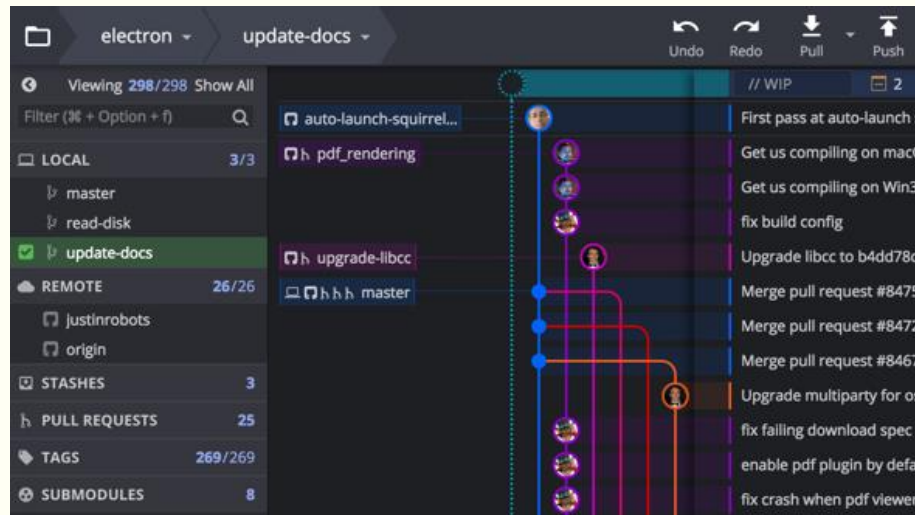
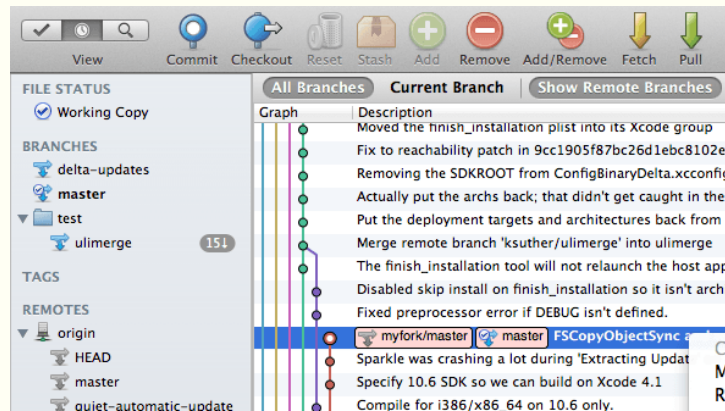
SmartGit : Proprietary



Cycligent Git Tool

Clients graphique : Macosx

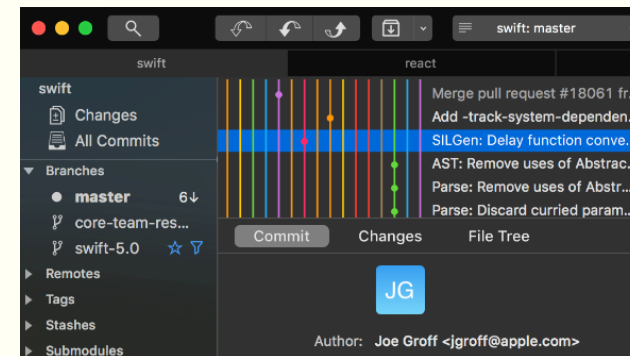
•SourceTree : Free



•GitKraken :Free for non-commercial use



•GitUp Free



•Fork : Free

Configuration : Votre identité

- La configuration globale de Git est située dans **~/.gitconfig**
- La configuration propre à chaque *repository* Git est située dans **<repository>/.git/config**

Plusieurs **niveaux** de configuration

- `.git/config` : pour le dépôt (`--file`)
- `~/.gitconfig` : pour l'utilisateur (`--global`)
- `/etc/gitconfig` : pour toute la machine (`--system`)

- A minima, il faut configurer son nom d'utilisateur et son adresse *email* (informations qui apparaîtront dans chaque `commit`):
 - ✓ `git config --global user.name "JohnDoe"`
 - ✓ `git config --global user.email johndoe@example.com`

Configuration : Votre éditeur de texte

- Vous pouvez configurer l'éditeur de texte qui sera utilisé quand Git vous demande de saisir un message.
- Par défaut, Git utilise l'éditeur configuré au niveau système, qui est généralement Vi ou Vim sous Linux, notepad sous windows ...
- Si vous souhaitez utiliser un éditeur de texte différent, comme Emacs, vous pouvez entrer ce qui suit :

```
$ git config --global core.editor emacs
```

Configuration : Votre outil de différences

- Une autre option utile est le paramétrage de l'outil de différences à utiliser pour la résolution des conflits de fusion.
- Supposons que vous souhaitiez utiliser vimdiff :

```
$ git config --global merge.tool vimdiff
```

Vérifier vos paramètres

- Si vous souhaitez vérifier vos réglages, vous pouvez utiliser la commande *git config --list*
- pour lister tous les réglages que Git a pu trouver jusqu'ici :

```
$ git config --list
user.name=Scott Chacon
user.email=schacon@gmail.com
color.status=auto
color.branch=auto
color.interactive=auto
color.diff=auto
...
```


Disposition de titre et de contenu avec liste

Question ?

Disposition de titre et de contenu avec liste

TP

Installation de Git sous Windows

TP

Configuration de Git



MODULE 3

Premier pas

Présentation

- Créer le dépôt Git
- Répertoire caché .git/
- Les trois états
- Ce cycle de vie d'un fichier*
- Créer un nouveau fichier
- Créer le commit
- Modifier un fichier
- Historique des versions
- Ancêtres et références

Définitions

- ***Commit*** : ensemble cohérent de modifications
- ***Repository*** : ensemble des *commits* du projet (et les branches, les *tags* (ou libellés),...)
- ***Working copy*** (ou copie de travail) : contient les modifications en cours (c'est le répertoire courant)
- ***Staging area*** (ou index) : liste des modifications effectuées dans la *working copy* qu'on veut inclure dans le prochain *commit*

Repository(dépôt)

- C'est l'endroit où Git va stocker tous ses objets : versions, branches, *tags*...
- Situé dans le sous-répertoire `.git` de l'emplacement où on a initialisé le dépôt
- Organisé comme un *filesystem* versionné, contenant l'intégralité des fichiers de chaque version (ou *commit*)

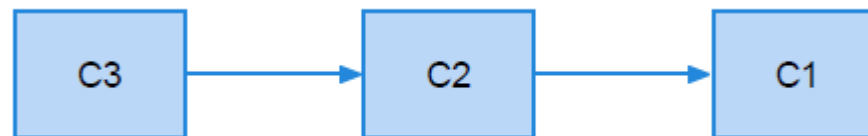
Commit

Fonctionnellement : **Unité d'œuvre**

- Doit compiler
- Doit fonctionner
- Doit signifier quelque chose (correction d'anomalie, développement d'une fonctionnalité / fragment de fonctionnalité)

Commit

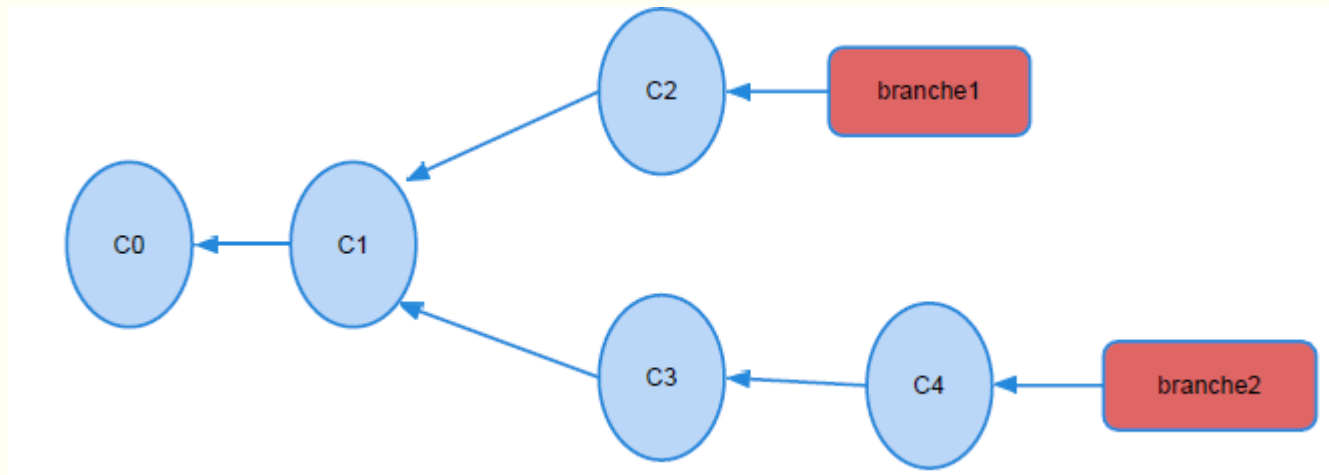
- Techniquement : **Pointeur**
vers un *snapshot* du *filesystem* dans son ensemble
- Connaît son ou ses parents
- Possède un identifiant unique



Le repository

Le *repository* contient l'ensemble des *commits* organisés sous forme de graphe acyclique direct:

- Depuis un *commit*, on peut accéder à tous ses ancêtres
- Un *commit* ne peut pas connaître ses descendants
- On peut accéder à un *commit* via son ID unique
- Des pointeurs vers les *commits* permettent d'y accéder facilement (branches, *tags*)



Créer le dépôt Git

Pour un **nouveau projet** :

```
$ mkdir project
```

```
$ cd project/
```

```
$ git init
```

Pour un projet **existant** :

```
$ git clone git://example.net/project
```

```
$ cd project/
```

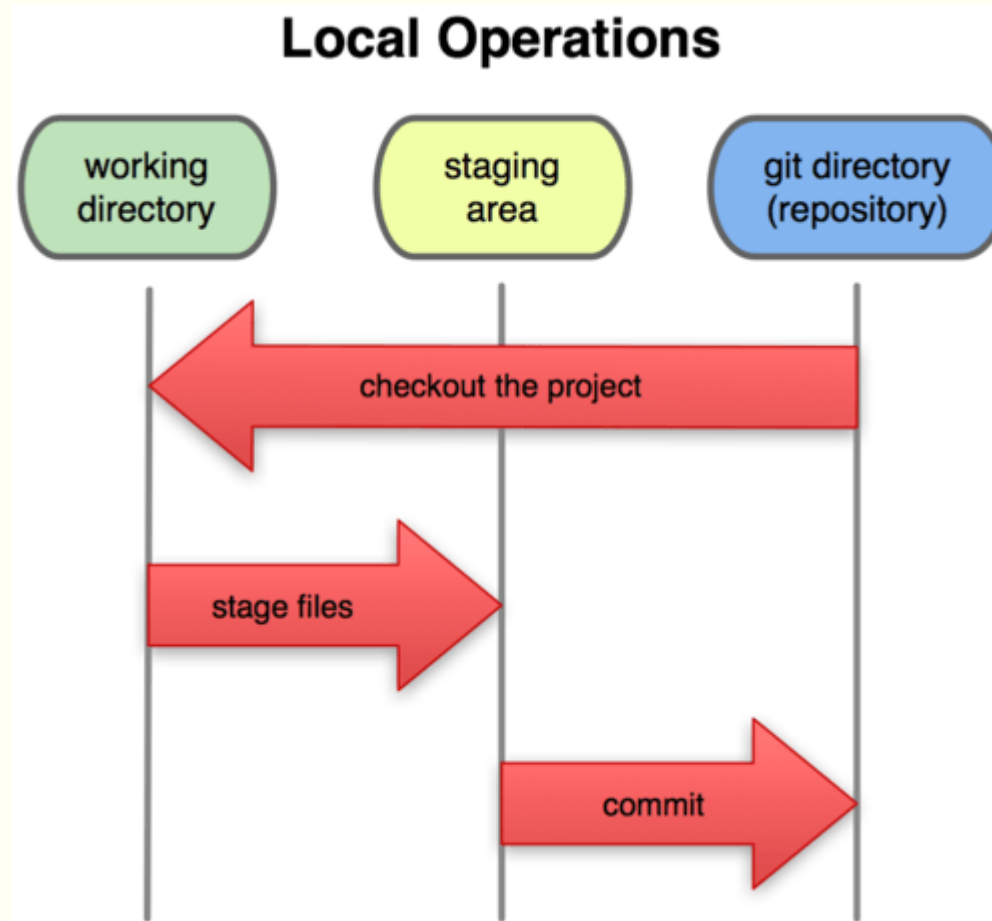
Répertoire caché .git/

- Répertoire .git Git maintient un répertoire .git unique à la racine du projet

```
$ ls .git/  
config  description  HEAD  hooks/  info/  objects/  refs/
```

- Copie intégrale des données en local
- Accès à tout l'**historique des modifications**

Les trois états

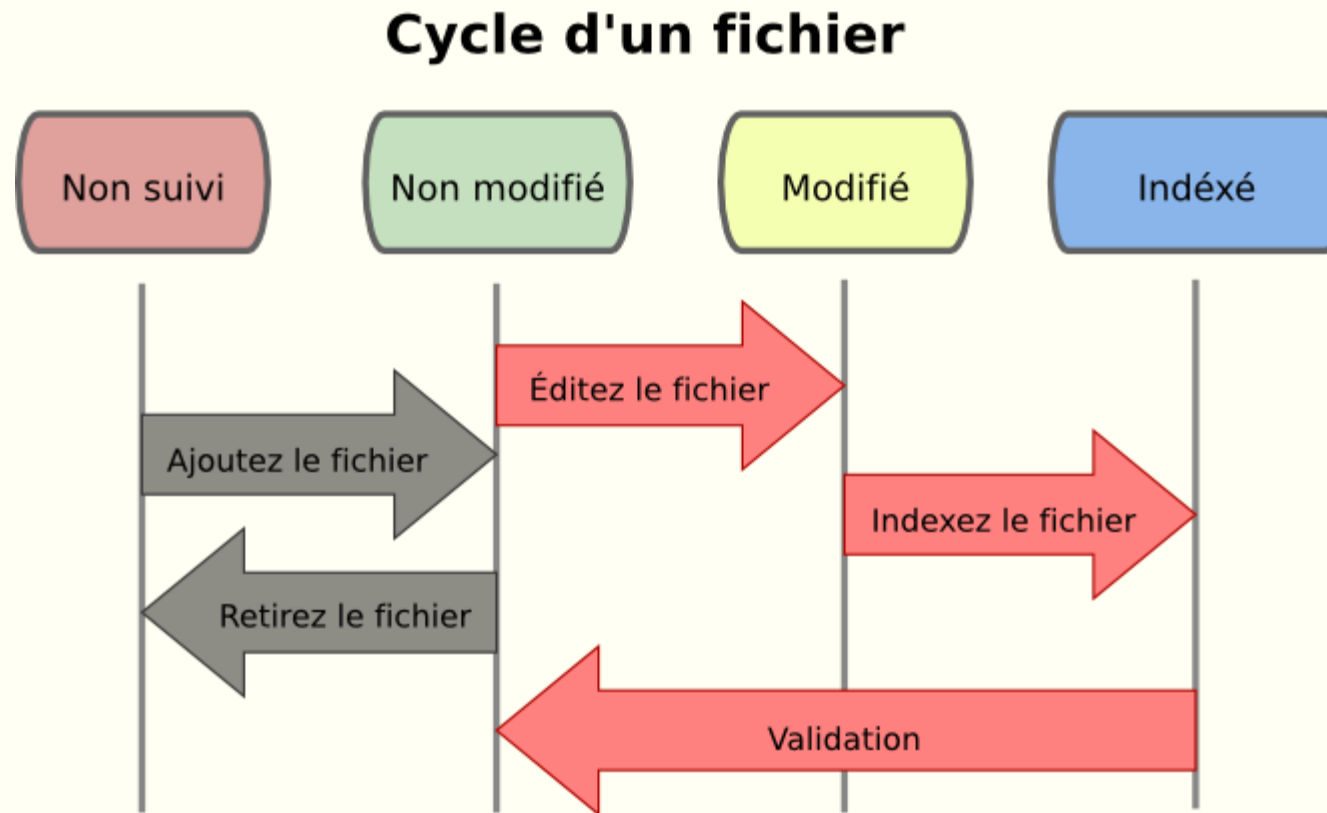


Le répertoire Git, le répertoire de travail et la zone d'index.

Les trois états

- **Le répertoire Git** est l'endroit où Git stocke les méta-données et la base de données des objets de votre projet.
- C'est la partie la plus importante de Git, et c'est ce qui est copié lorsque vous clonez un dépôt depuis un autre ordinateur.
- **Le répertoire de travail** est une extraction unique d'une version du projet.
- Ces fichiers sont extraits depuis la base de données compressée dans le répertoire Git et placés sur le disque pour pouvoir être utilisés ou modifiés.
- **La zone d'index** est un simple fichier, généralement situé dans le répertoire Git, qui stocke les informations concernant ce qui fera partie du prochain instantané.

Ce cycle de vie d'un fichier



Ce cycle de vie d'un fichier

- **Untracked**

- ✓ non pris en compte par Git

- **Unmodified/Committed**

- ✓ aucune modification

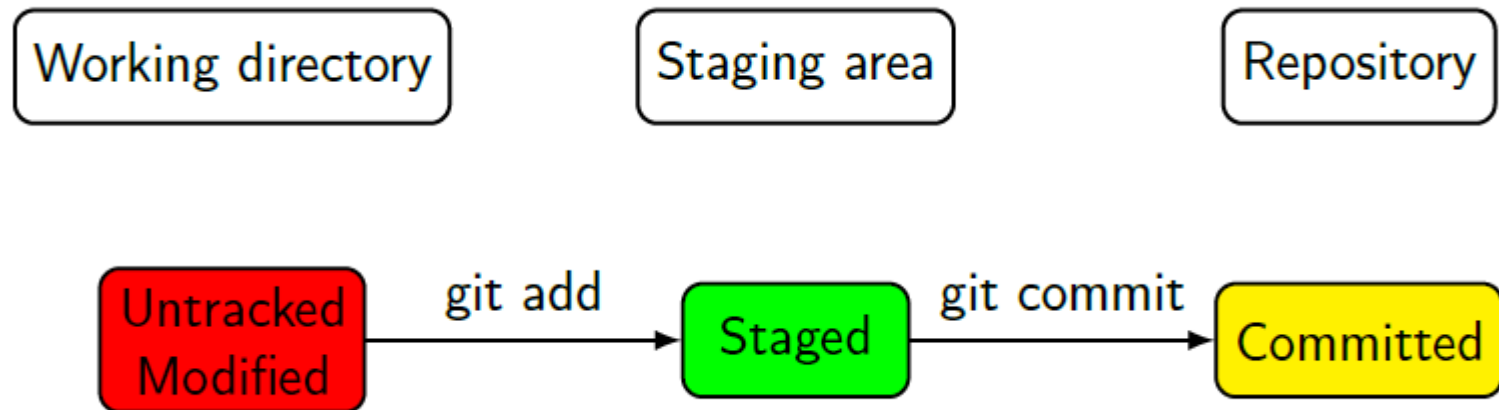
- **Modified**

- ✓ fichier modifié
- ✓ pas pris en compte pour le prochain commit

- **Staged**

- fichier ajouté, modifié, supprimé ou déplacé
- pris en compte pour le prochain commit

États d'un fichier



Créer un nouveau fichier

\$ echo hello > README

État du fichier : **untracked**

\$ git status

```
# On branch master
#
# Initial commit
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       README
nothing added to commit but untracked files present
```

Créer un nouveau fichier

\$ git add README

État du fichier : **untracked** → **staged**

\$ git status

```
# On branch master
#
# Initial commit
#
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
#
#       new file:   README
#
```

Créer le commit

\$ git commit

[écrire le message du commit]

\$ git log

```
commit c3aab8bb6cca644162a4fa82df283682717da3d4
Author: Your Name <foo@bar.be>
Date:   Wed Feb 1 15:19:03 2012 +0100
```

Titre du commit (pas trop long)

Plus longue description.

Ligne vide après le titre.

Commit

- `git commit -m "mon commentaire de commit"`
 - génère un *commit* avec les modifications contenues dans la *staging area*
- `git commit -a -m "mon commentaire de commit"`
 - ajoute tous les fichiers modifiés (pas les ajouts / suppressions)
à la *staging area* et commite
- `git commit--amend`
 - corrige le *commit* précédent

Modifier un fichier

État du fichier README : **unmodified**

```
$ echo world >> README
```

État : **modified**

```
$ git add README
```

État : **staged**

```
$ git commit
```

État : **committed**

Historique des versions

- `git log [-n] [-p] [--oneline]: historique`
 - affiche les ID des *commits*, les messages, les modifications
 - `-n` : limite à *n commits*
 - `-p` : affiche le diff avec le *commit* précédent
 - `--oneline` : affiche uniquement le début de l'ID du *commit* et le commentaire sur une seule ligne pour chaque *commit*
- `git show [--stat] : branche, tag, commit-id ...`
 - montre le contenu d'un objet
- `git diff :`
 - `git diff id_commit` : diff entre *working copy* et *commit*
 - `git diff id_commit1 id_commit2` : diff entre deux *commits*

Ancêtres et références

- `id_commit^` : parent du *commit*
- `id_commit^^` : grand-père du *commit*...
- `id_commit~n` : n-ième ancêtre du *commit*
- `id_commit^2` : deuxième parent du *commit* (*merge*)
- `id_commit1..id_commit2` :
variations entre le *commit* 1 et le *commit* 2
(ex. `git log id_commit1..id_commit2` : tous les commits accessibles depuis *commit2* sans ceux accessibles depuis *commit1*)

Disposition de titre et de contenu avec liste

Question ?

Disposition de titre et de contenu avec liste

TP

Premier Pas



MODULE 4

GitHub

Présentation

- Introduction
- Les dépôts
- Créer un compte GitHub
- Créer un nouveau dépôt

Introduction

- GitHub est un service web d'hébergement et de gestion collaborative de développement de logiciels basé sur le programme Git.
- De manière résumée, à travers GitHub vous pouvez accéder aux fichiers d'un projet que quelqu'un d'autre a créé ou bien créer votre projet et permettre à d'autres utilisateurs d'y accéder.
- L'historique de l'évolution des projets dans github est également conservé, ce qui, au besoin, permet de pouvoir revenir à des versions antérieures.

Les dépôts

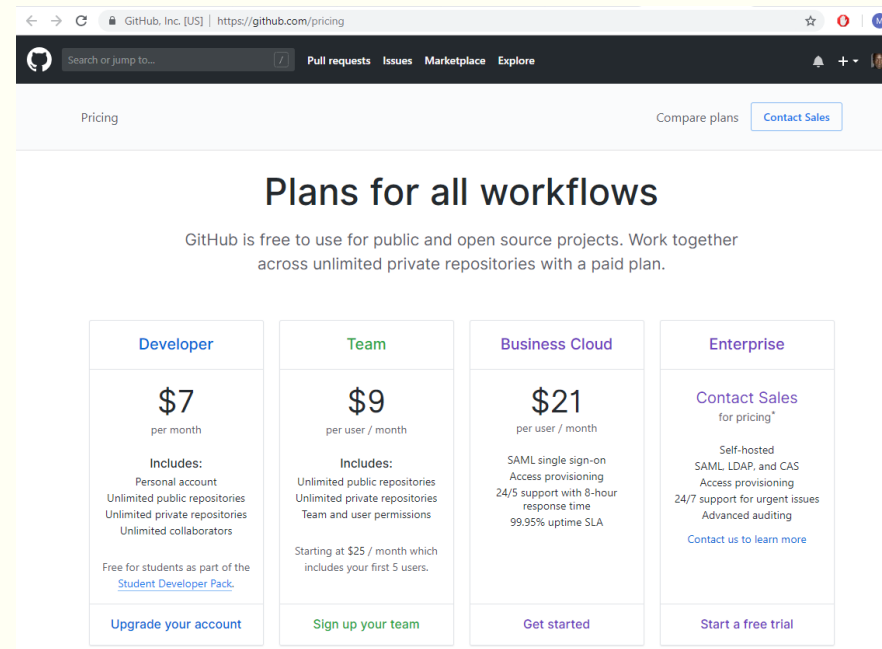
- **GitHub permet à ses utilisateurs de créer des dépôts** qui peuvent contenir de multiples fichiers, dossiers et sous-dossiers.
- Un dépôt est l'équivalent d'un dossier de projet.
- Quand on regarde un dépôt sur GitHub, on voit la version la plus récente des fichiers.
- Les dépôts peuvent être publics ou privés.
- Pour rendre un dépôt privé sous GitHub il faut un compte payant. Dans un cadre open-source, nous travaillerons uniquement avec des dépôts publics.
- Quand un dépôt est public, on peut facilement télécharger son contenu et l'utiliser.
- Si vous avez les permissions, vous pouvez également modifier les fichiers et sauvegarder vos changements dans GitHub

Créer un compte GitHub

- Pour créer un compte GitHub, rendez-vous à cette adresse.

<https://github.com/pricing>

- Si vous prévoyez de travailler uniquement avec des dépôts publics, vous pouvez simplement choisir de créer un compte gratuit.

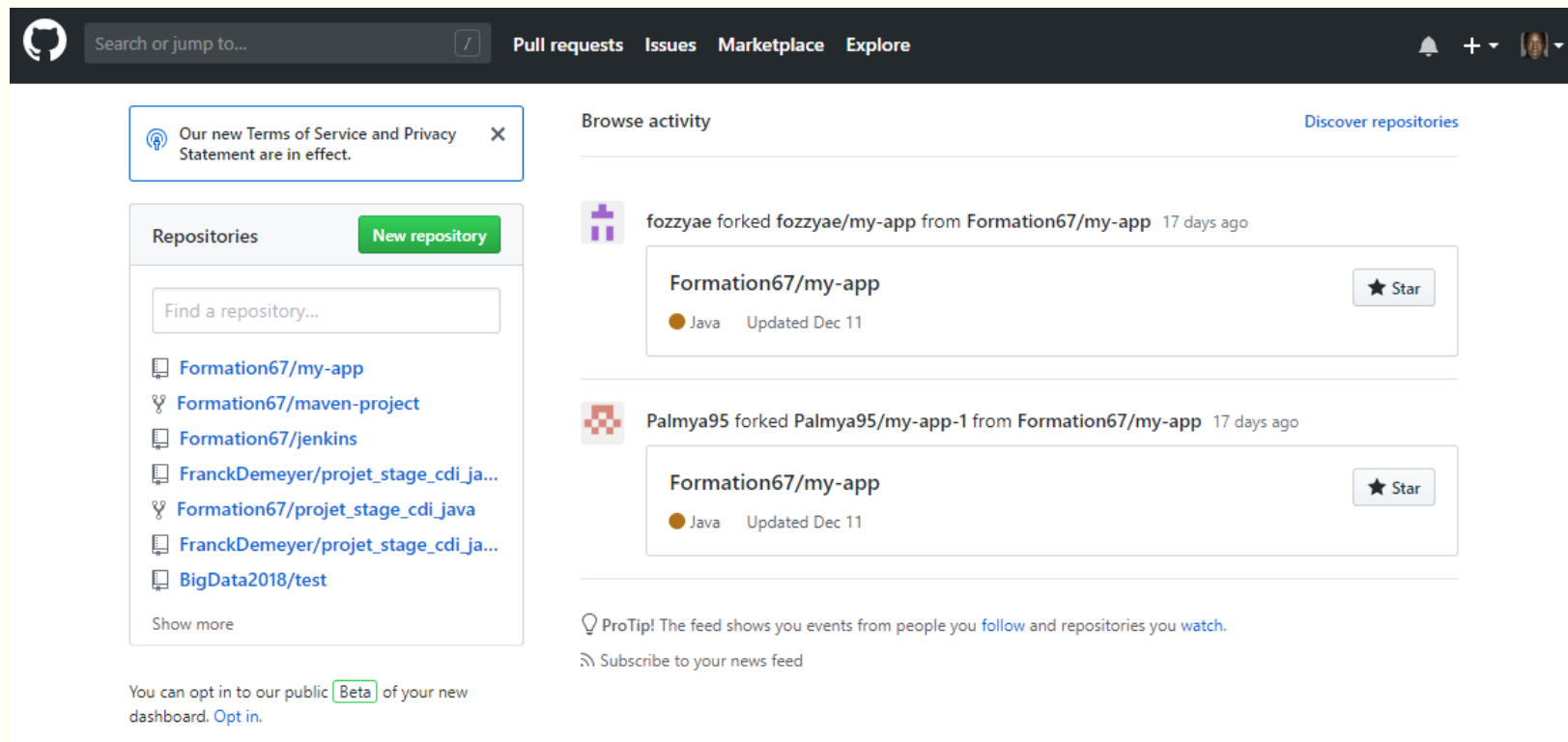


The screenshot shows the GitHub Pricing page. At the top, there's a navigation bar with links for Pull requests, Issues, Marketplace, and Explore. Below this, the main heading is "Plans for all workflows". A subtext states: "GitHub is free to use for public and open source projects. Work together across unlimited private repositories with a paid plan." There are four plan cards displayed:

Developer	Team	Business Cloud	Enterprise
\$7 per month	\$9 per user / month	\$21 per user / month	Contact Sales for pricing*
Includes: Personal account Unlimited public repositories Unlimited private repositories Unlimited collaborators Free for students as part of the Student Developer Pack .	Includes: Unlimited public repositories Unlimited private repositories Team and user permissions Starting at \$25 / month which includes your first 5 users.	SAML single sign-on Access provisioning 24/5 support with 8-hour response time 99.95% uptime SLA	Self-hosted SAML, LDAP, and CAS Access provisioning 24/7 support for urgent issues Advanced auditing Contact us to learn more
Upgrade your account	Sign up your team	Get started	Start a free trial

Créer un nouveau dépôt

- Cliquez sur “**New Repository**” dans la section en haut du site.



Créer un nouveau dépôt


- Donnez un nom et une description à votre projet.
- Par défaut vous aurez un dépôt public et le projet sera vide à sa création.
- Nous pouvons y placer un fichier readme, en cochant la case “***Initialize this project with a README***”.

Create a new repository

A repository contains all the files for your project, including the revision history.

Owner

Repository name


 Formation67 ▾

 /


first-github ✓

Great repository names are short and memorable. Need inspiration? How about [psychic-meme](#).

Description (optional)

☒  Public

Anyone can see this repository. You choose who can commit.

☐  Private


You choose who can see and commit to this repository.

☒ Initialize this repository with a README

This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

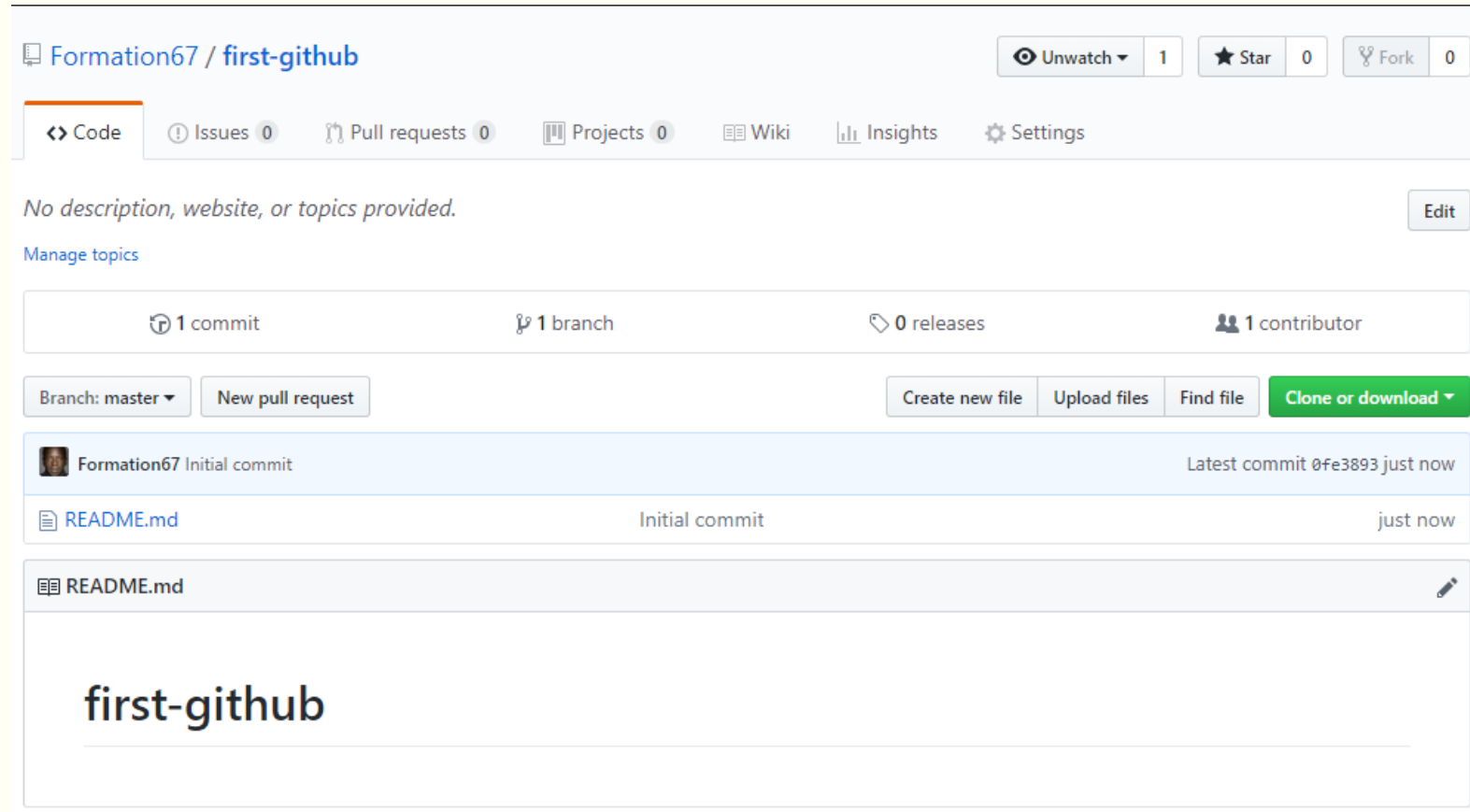
Add .gitignore: None ▾

 |

Add a license: None ▾ 

Create repository

Créer un nouveau dépôt



Disposition de titre et de contenu avec liste

Question ?

Disposition de titre et de contenu avec liste

TP

GitHub - Basics



MODULE 5

Commandes GIT de base

Présentation

- Configuration des outils
- Créer des dépôts
- Changements au niveau des noms de fichiers
- Modifications locales
- Vérifier l'historique des versions
- Exclure du suivi de version
- Effectuer des changements

Configuration des outils

Git config

L'une des commandes git les plus utilisées est **git config**.

On l'utilise pour configurer les préférences de l'utilisateur : son mail, l'algorithme utilisé pour diff, le nom d'utilisateur et le format de fichier etc.

```
$ git config --global user.name "[nom]"
```

Définit le nom que vous voulez associer à toutes vos opérations de commit

```
$ git config --global user.email "[adresse email]"
```

Définit l'email que vous voulez associer à toutes vos opérations de commit

```
$ git config --global color.ui auto
```

Active la colorisation de la sortie en ligne de commande

Configuration des outils

- Vérifier la configuration

git config --global -l

Créer des dépôts

- Démarrer un nouveau dépôt ou en obtenir un depuis une URL existante

```
$ git init [nom-du-projet]
```

Crée un dépôt local à partir du nom spécifié

```
$ git clone [url]
```

Télécharge un projet et tout son historique de versions

- Créer un dépôt local dans un répertoire local existant

```
cd repertoire_projet
```

```
git init
```

```
git add -A
```


Changements au niveau des noms de fichiers

- Enlever un fichier du prochain commit

`git rm --cached fichier`

- Supprimer un fichier

`git rm fichier`

- Supprimer récursivement les fichiers d'un répertoire

`git rm repertoire/ -r`

- Renommer un fichier

`git mv fichier nouveau_nom`

- Déplacer un fichier

`git mv fichier destination/`

Modifications locales

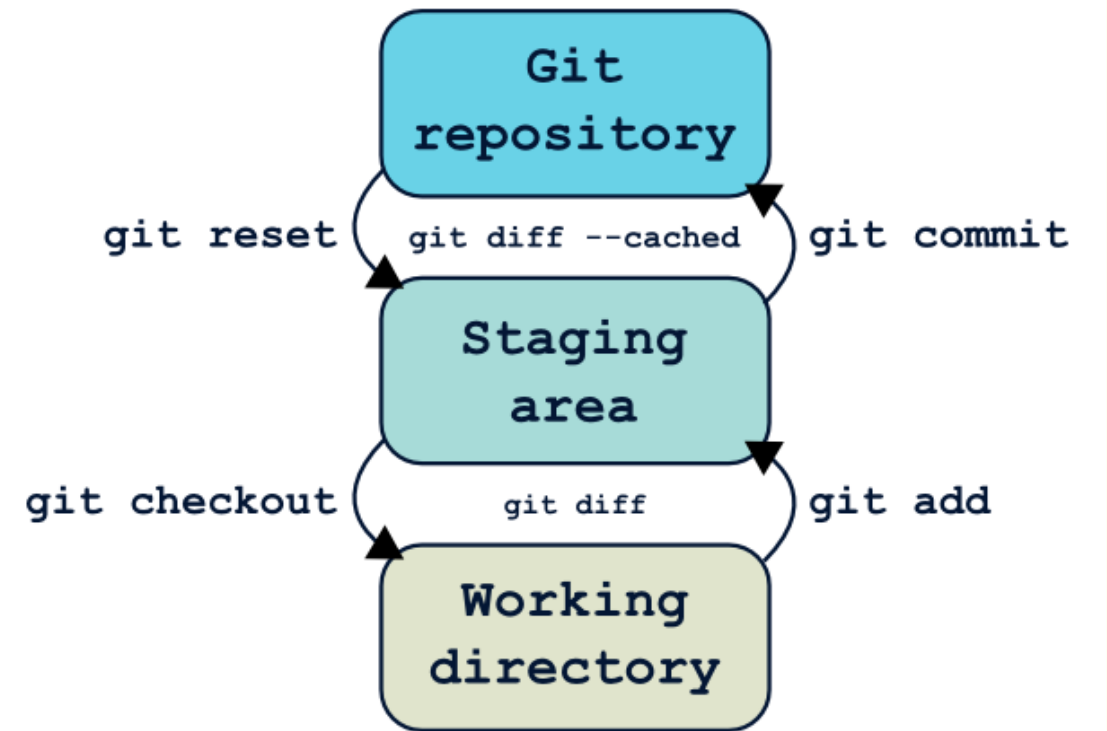
- Annuler les modifications réalisées dans un fichier

`git checkout -- fichier`

`git reset [--mixed] HEAD fichier`

- Ajouter des fichiers au prochain commit

`git add fichier1 fichier2 fichier3`



Effectuer des changements

- **Afficher l'état des fichiers nouveaux ou modifiés**

`git status`

- **Afficher les modifications des fichiers suivis modifiés**

`git diff`

- **Afficher les modifications du prochain commit**

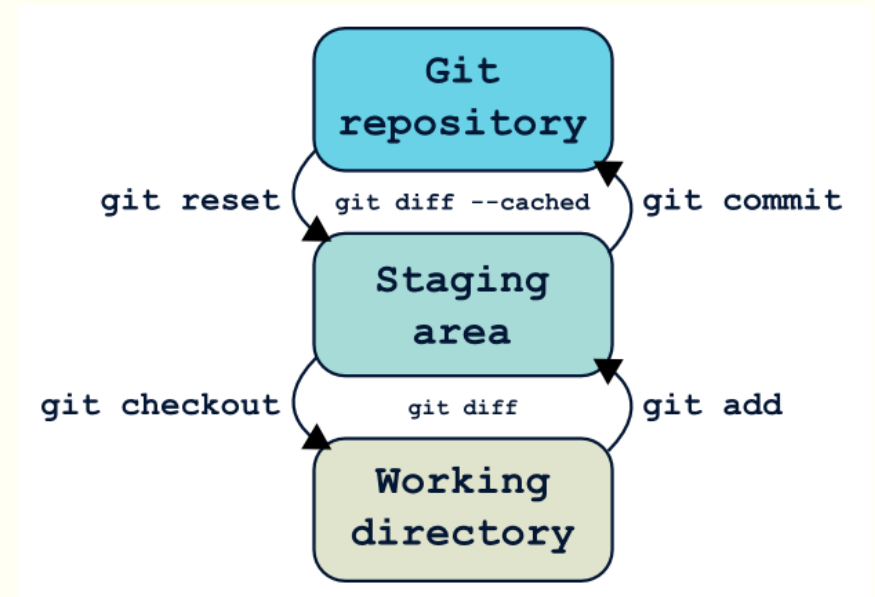
`git diff --cached`

- **Effectuer un commit**

`git commit`

`git commit -a` (ajouter automatiquement les fichiers)

`git commit -m "Message du commit"`



Effectuer des changements

- **Modifier le dernier commit**

`git commit --amend`

- **Etiqueter le dernier commit**

`git tag nom_tag`

- **Annuler les n derniers commit**

`git revert HEAD (dernier commit)`

`git revert HEAD~ (2 derniers commit)`

`git revert HEAD~2 (3 derniers commit)`

Effectuer des changements

- Retourner à la version du dernier commit

(Supprime les nouveaux fichiers et les modifications)

ATTENTION : Cette opération ne peut pas être annulée

git reset --hard HEAD

Vérifier l'historique des versions

- **Afficher tous les commits (format par défaut ou court)**

`git log`

`git log --pretty=-short`

- **Afficher les x derniers commits**

`git log -n x`

- **Afficher les commits d'un fichier ou d'un répertoire**

`git log fichier`

`git log repertoire/`

Vérifier l'historique des versions

- **Afficher des statistiques pour chaque fichier modifié**

`git log --stat`

- **Afficher le contenu d'un commit**

`git show id_commit`

Exclure du suivi de version

- **Créer la liste des fichiers à ignorer et la publier**

```
git config --global core.excludefiles **/*.log
```

- **Modifier le fichier .gitignore**

```
git add .gitignore
```

```
git commit -m "Partage des fichiers à ignorer"
```

- **Afficher la liste de tous les fichiers ignorés**

```
git ls-files --other --ignored --exclude-standard
```


Disposition de titre et de contenu avec liste

Question ?

Disposition de titre et de contenu avec liste

TP

Les bases de Git

TP

Afficher les modifications



MODULE 6

Gestion des branches et Merge

Présentation

- Introduction
- Le concept de branche
- Création d'une nouvelle branche
- Branche courante
- Changer de branche
- Commit sur une branche
- Opérations de base sur une branche
- Fusion de branches
- Gestion de conflits

Introduction

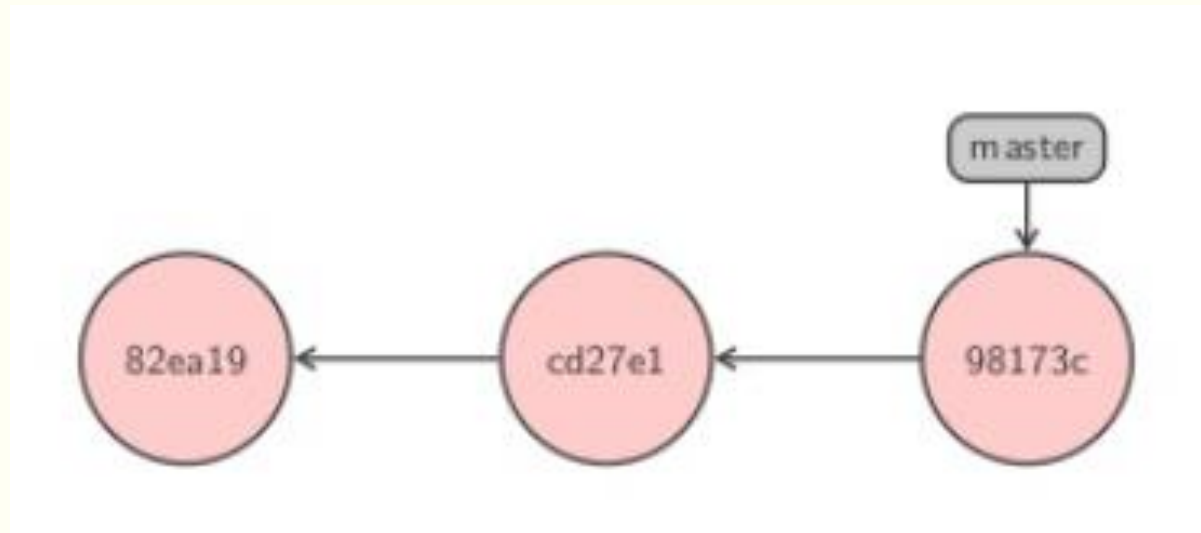
- Déviation par rapport à la route principale
- Permet le développement de différentes versions en parallèle
 - Version en cours de développement
 - Version en production (correction de bugs)
 - Version en recette
 - ...
- On parle de “**merge**” lorsque tout ou partie des modifications d'une branche sont rapatriées dans une autre
- On parle de “feature branch” pour une branche dédiée au développement d'une fonctionnalité (ex : gestion des contrats...)

Introduction

- **branch** == pointeur sur le dernier *commit* (sommet) de la branche
 - les branches sont des références
- **master** == branche principale(*trunk*)
- **HEAD** == pointeur sur la position actuelle de la working copy

Le concept de branche

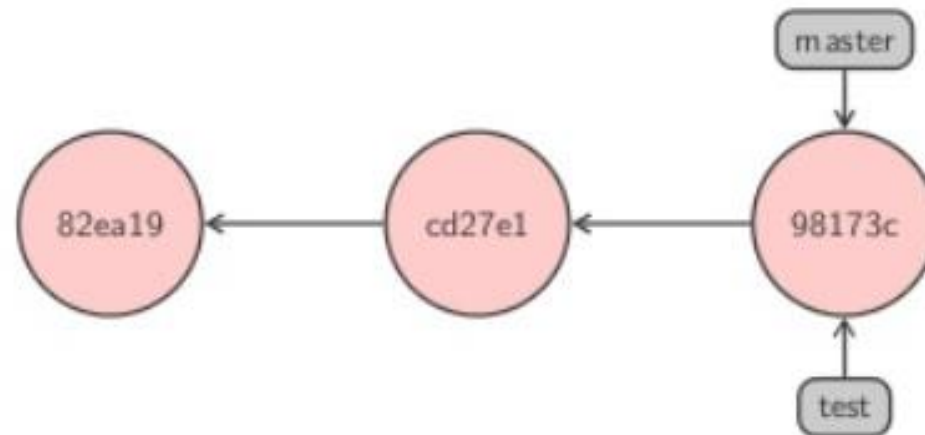
- Une **branche** pointe vers un commit
- À chaque nouveau commit, le **pointeur de branche** avance
- Un commit pointe vers le commit parent



Création d'une nouvelle branche

- Une nouvelle **branche** est créée avec « git branch name »

```
$ git branch test
```

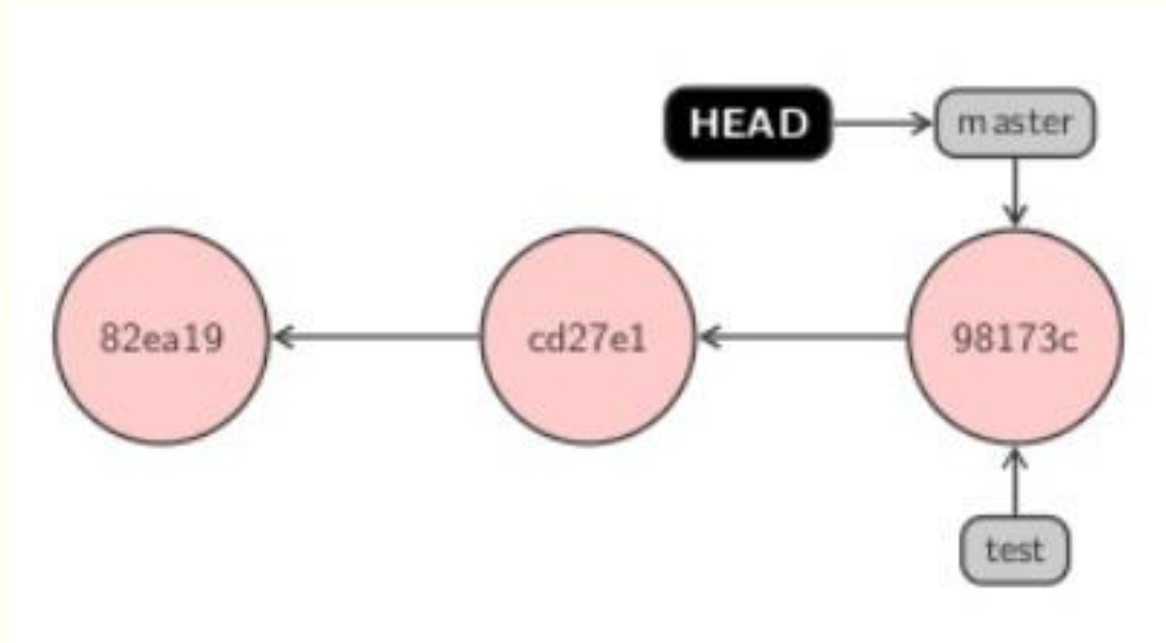


Branche courante

- La commande « git branch » liste les branches existantes

```
$ git branch
* master
  test
```

- La branche courante est identifiée par HEAD

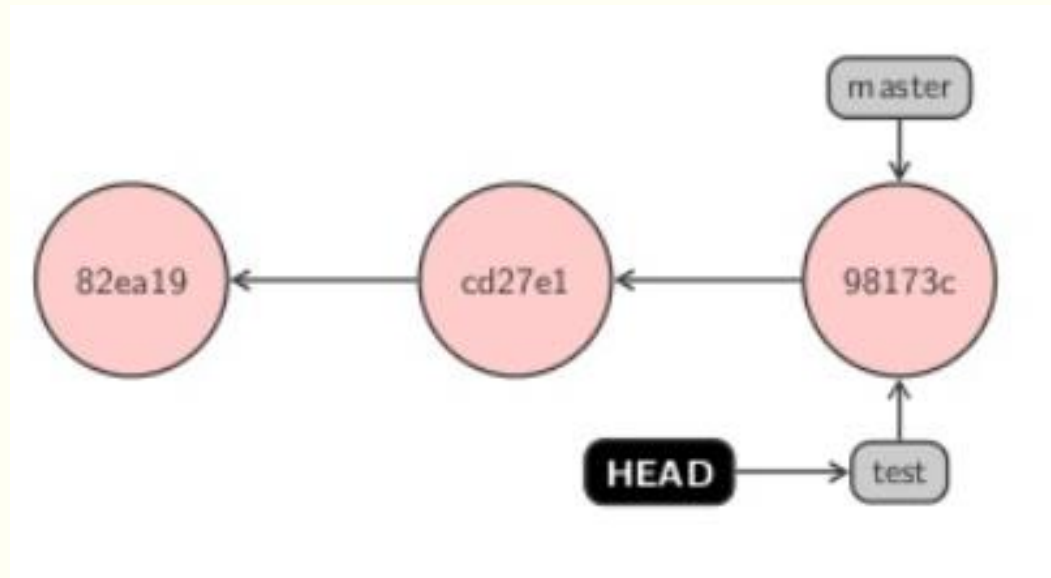


Changer de branche

- La commande « git checkout name » change de branche

```
$ git checkout test  
Switched to branch 'test'
```

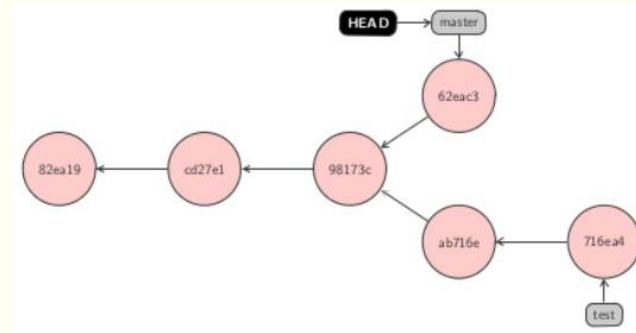
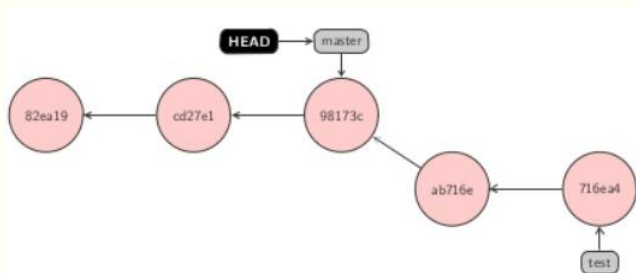
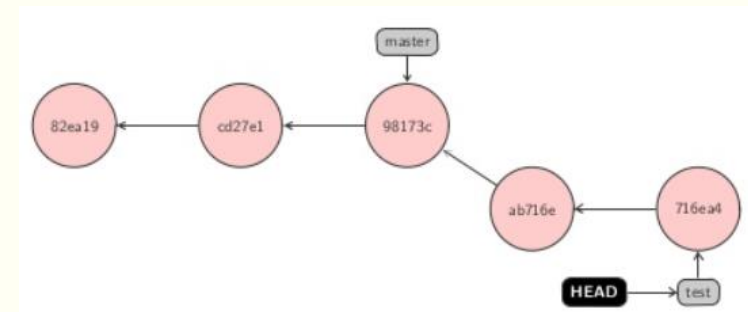
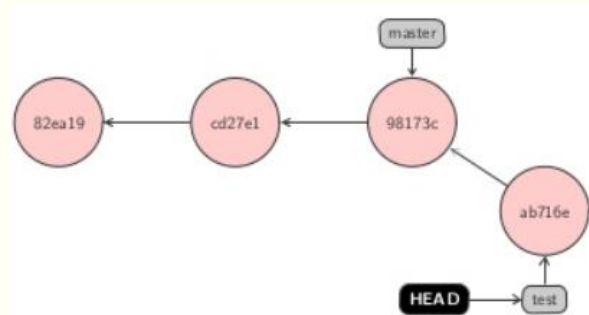
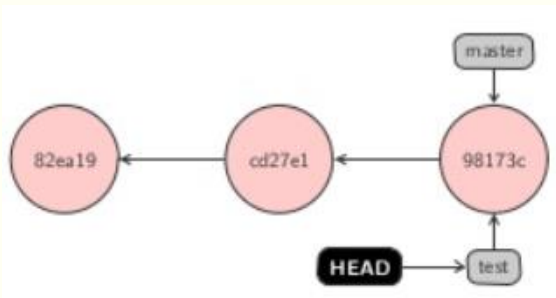
- La branche courante est identifiée par HEAD



Commit sur une branche

- Un commit va toujours se faire sur la branche courante

```
...  
$ git commit ...  
$ git checkout master  
...  
$ git commit ...
```



Opérations de base sur une branche

- On peut supprimer une branche avec l'option -d

```
$ git branch -d test  
Deleted branch test (was 617a041).
```

- On peut **renommer** une branche avec l'option -m

```
$ git branch  
* master  
  test  
$ git branch -m test alternative  
$ git branch  
alternative  
* master
```

Fusion de branches

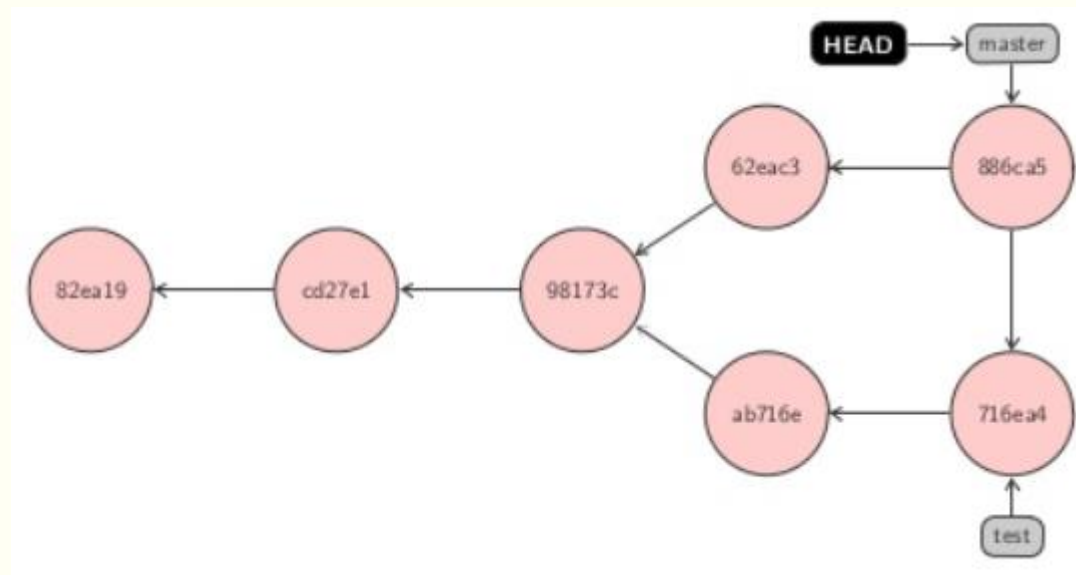
- On peut fusionner deux branches pour en combiner les modifications
- La fusion se fait vers la branche courante
- On doit associer un message lors d'une fusion

Une fusion est un commit qui agrège plusieurs modifications

```
$ git merge test
Merge made by the 'recursive' strategy.
 ABOUT | 1 +
AUTHORS | 1 +
2 files changed, 2 insertions(+)
create mode 100644 ABOUT
create mode 100644 AUTHORS
```

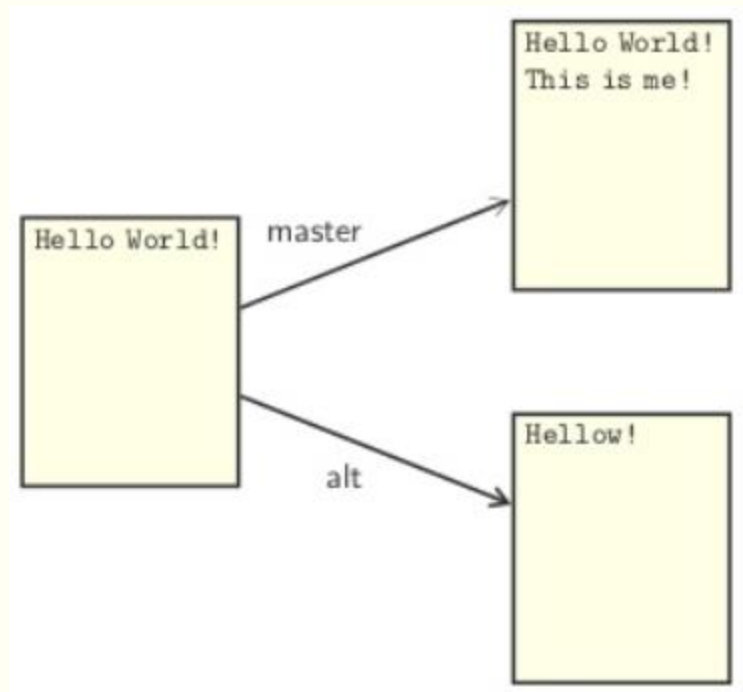
Fusion de branches

- La branche courante ne change pas après une fusion
- La branche fusionnée continue d'exister



Gestion de conflits

- Conflit lorsque deux branches à fusionner contiennent des modifications sur le même fichier



Gestion de conflits II

- Conflit détecté lors d'une demande de merge

```
$ git merge alt
Auto-merging file.txt
CONFLICT (content): Merge conflict in file.txt
Automatic merge failed; fix conflicts and then
commit the result.

$ git status
# On branch master
# Unmerged paths:
#   (use "git add/rm <file>..." as appropriate to
#   mark resolution)
#
#       both modified :   file.txt
#
```

- Résolution manuelle du conflit suivie d'un commit

```
$ cat file.txt
<<<<<<< HEAD
Hello World!
This is me!
=====
Hellow!
>>>>>> alt
```


-
- Checkout interdit On ne peut pas changer de branche n'importe quand

```
$ echo 'Hello!' > me.txt
$ git add me.txt
$ git commit -m "Initial commit"

$ git branch alt
$ echo 'I am God' >> me.txt
$ git add me.txt
$ git commit -m "God is there"

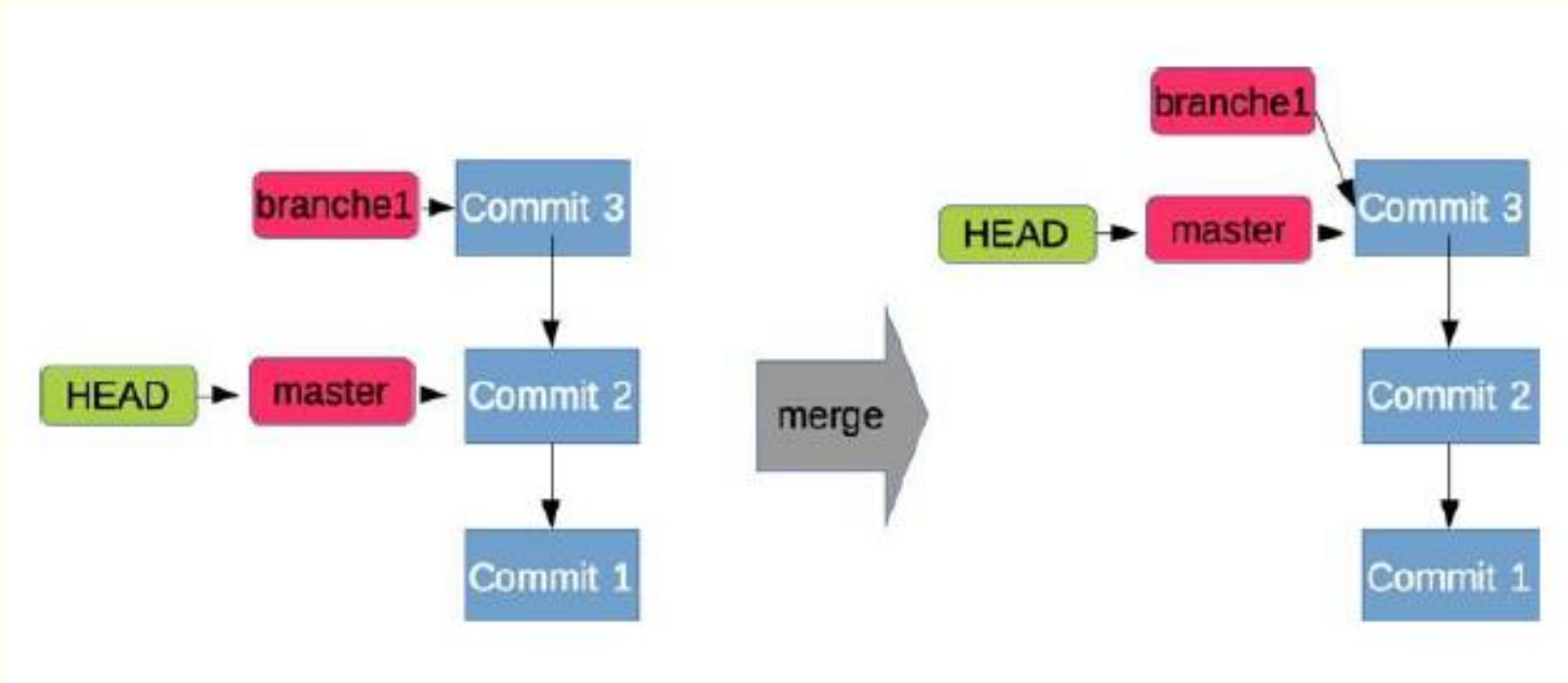
$ git checkout alt
$ echo 'I am devil' >> me.txt
$ git add me.txt
$ git checkout master
error: Your local changes to the following files
would be overwritten by checkout:
    me.txt
Please, commit your changes or stash them before
you can switch branches.
Aborting
```

Merge

- Fusionner 2 branches / Réconcilier 2 historiques
- Rapatrier les modifications d'une branche dans une autre
- ATTENTION: par défaut le *merge* concerne tous les *commits* depuis le dernier *merge* / création de la branche
- Depuis la branche de destination :
git merge nom_branche_a_merger
- On peut aussi spécifier un ID de *commit* ou un *tag*, plutôt qu'une branche
- 2 cas : *fast forward* et *non fastforward*

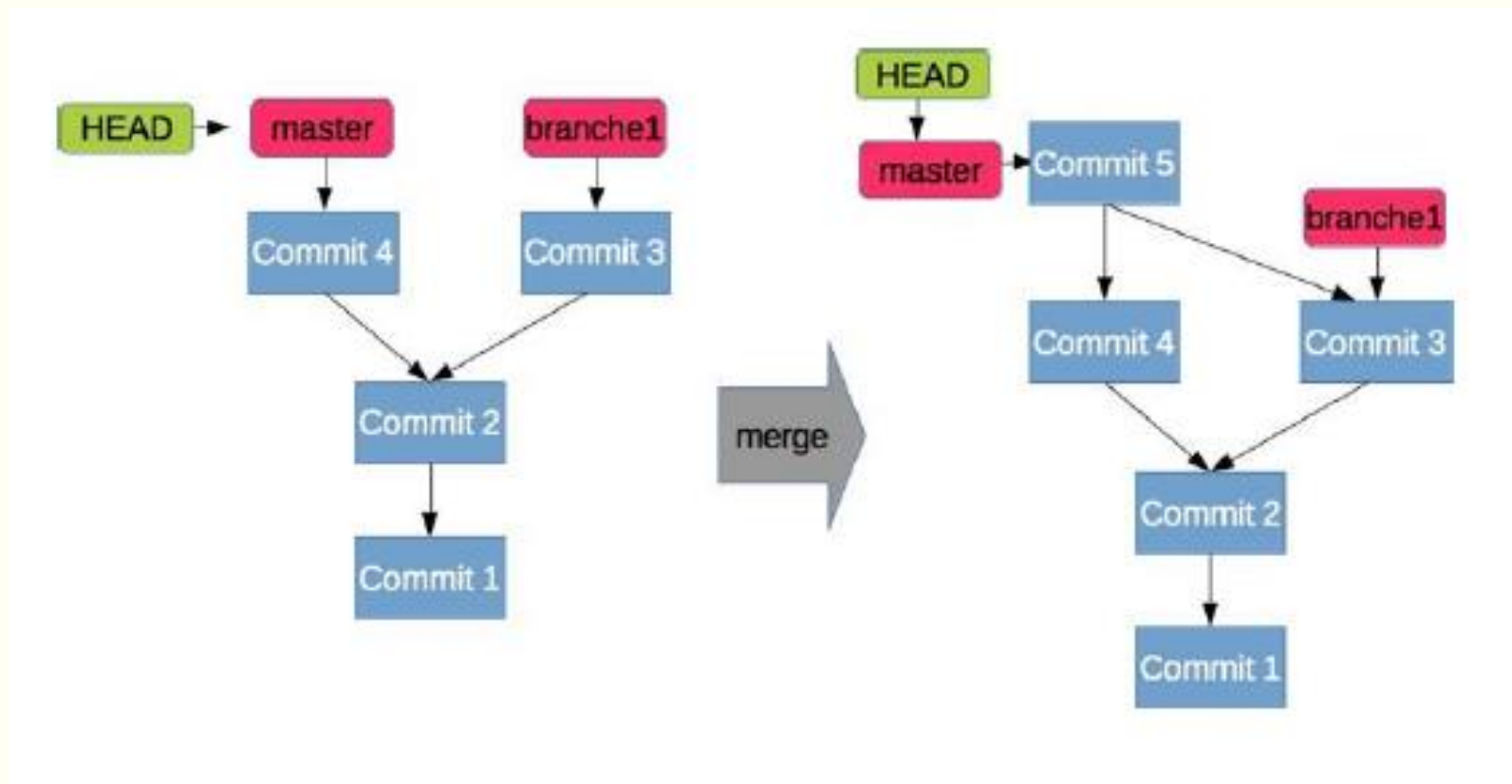
Fast-forward

- Cas simple /automatique
- Quand il n'y a pas d'ambiguïté sur l'historique



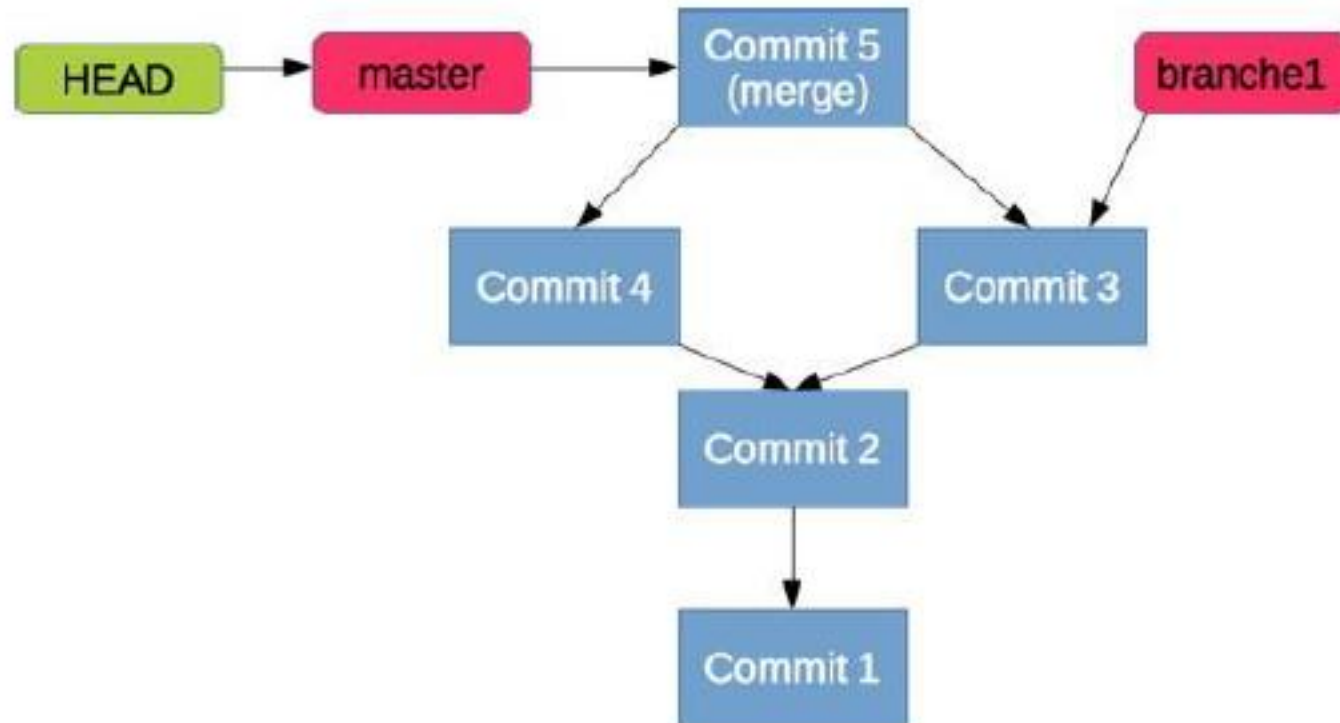
Non fast-forward

- Quand il y a ambiguïté sur l'historique
- Création d'un *commit* de *merge*



Conflit

- On souhaite merger la branche `branche1` sur `master` pour obtenir:



Conflit

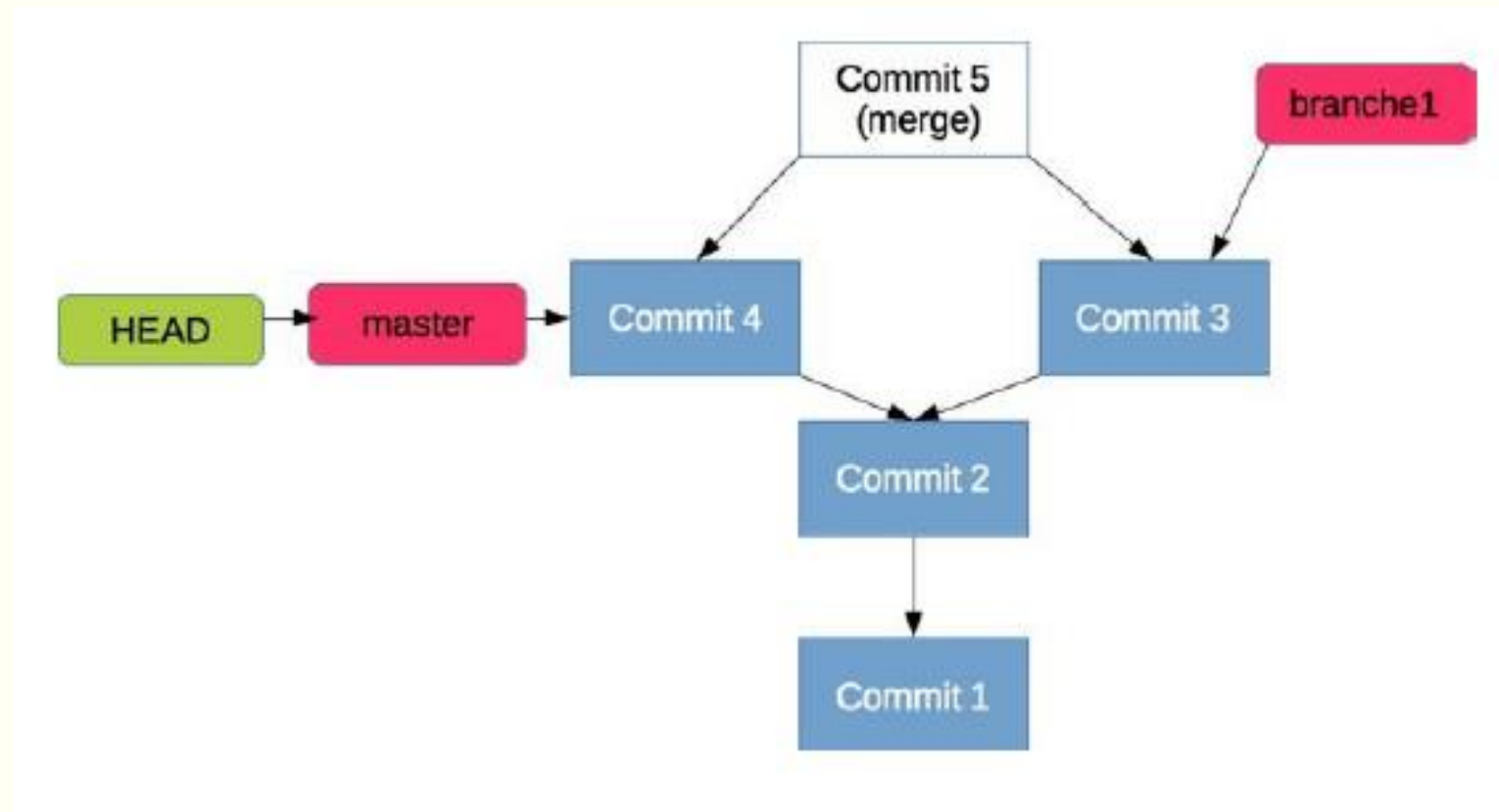
- Commit 4 et commit 3 modifient la même ligne du fichier
- Git ne sait pas quoi choisir
 - conflit
 - suspension **avant** le commit de merge
- git mergetool/ Résolution du conflit / git commit
 - ou
 - gitmerge --abort ou git reset --merge
 - ou
 - git reset --hard HEAD pour annuler
- NB : branche1 ne bougera pas

Merge

- Si on veut éviter le *fast forward* (*merge* d'une *feature branch*) on utilise le flag-no-ff
- Ex : `git merge branche1 --no-ff`

Annulation (après merge)

- `git reset --hard HEAD^`



Disposition de titre et de contenu avec liste

TP

Branche et merge

Disposition de titre et de contenu avec liste

Question ?



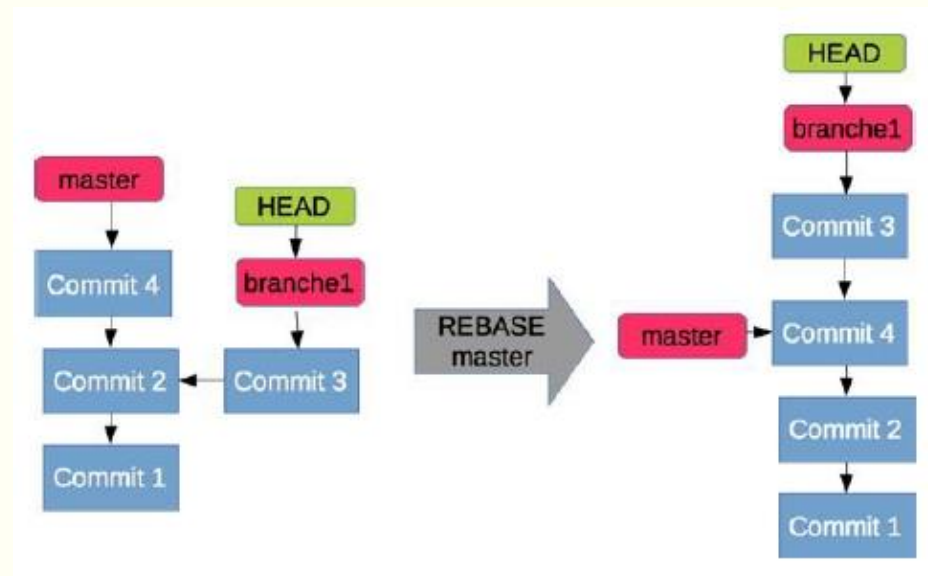
MODULE 7

Rebase

Présentation

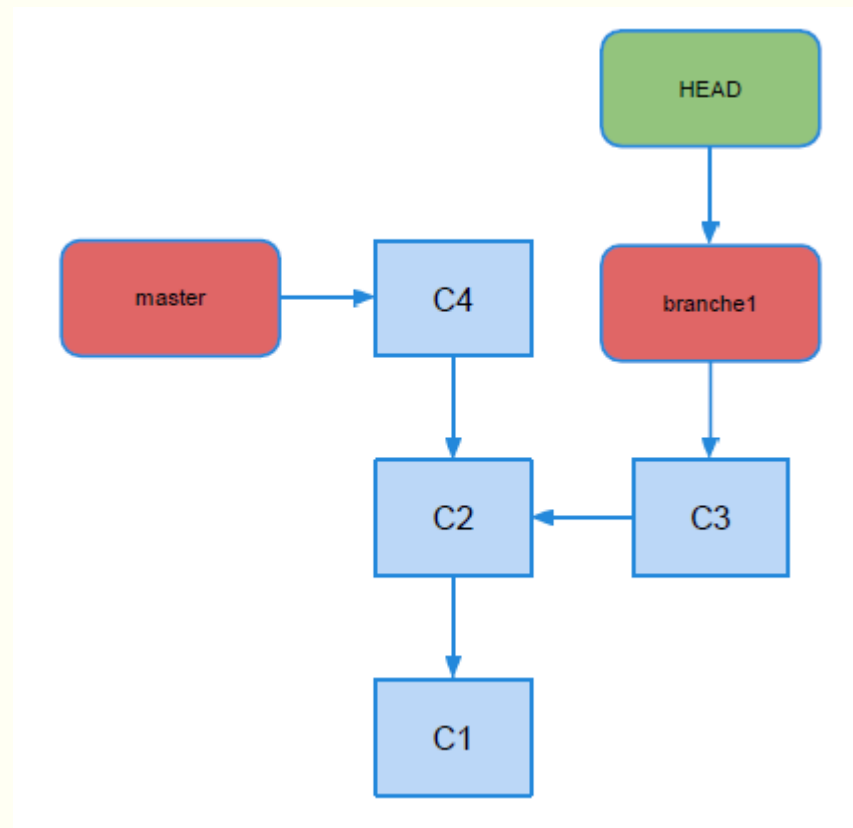
Rebase

- Modifie / réécrit l'historique
- Modifie / actualise le point de départ de la branche
- Remet nos commits au dessus de la branche contre laquelle on rebase
- Linéarise (évite de polluer l'historique avec des commits de merge inutiles)



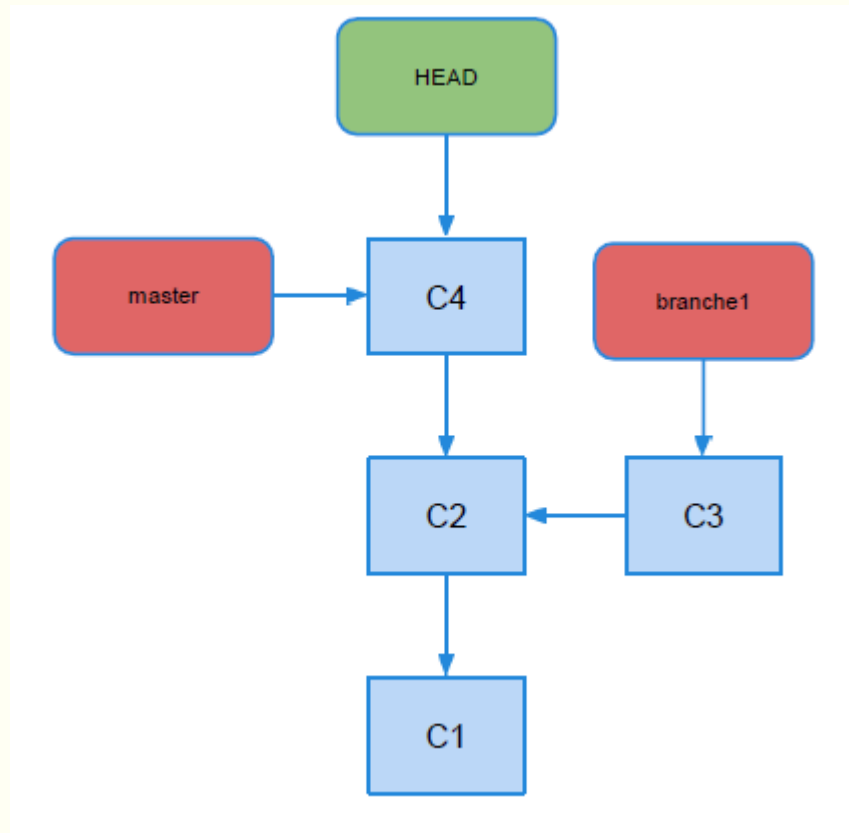
Rebase

- Situation de départ : 3 commits sur master (C1,C2 et C4) , 3 commits sur branche1 (C1, C2 et C3) , création de branche 1 à partir de C2



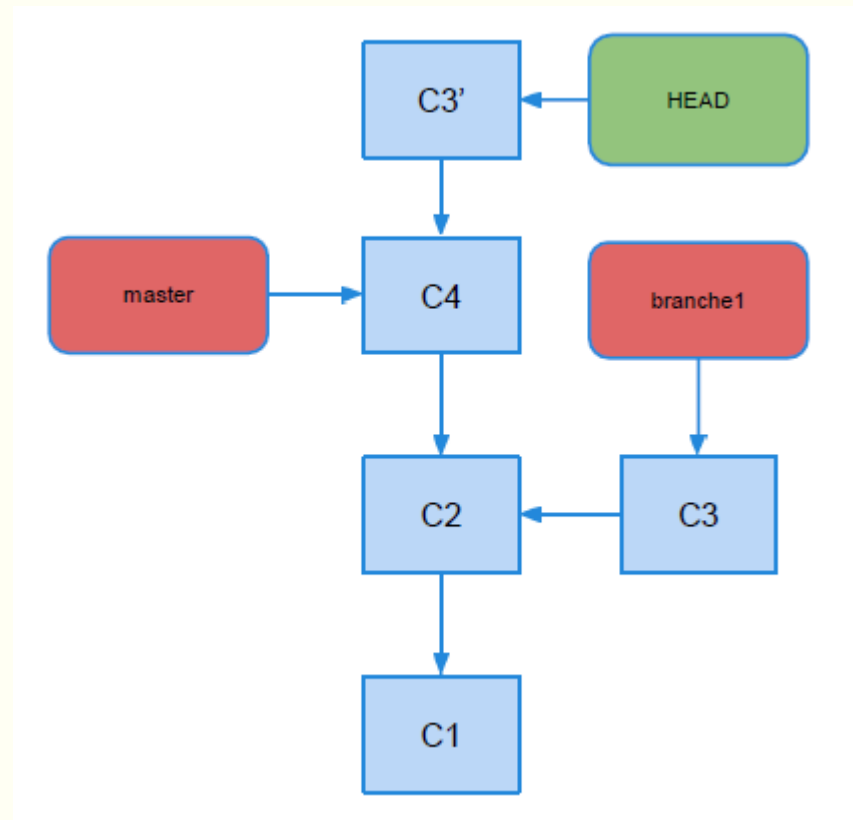
Rebase

- Depuis branche 1 on fait un git rebase master
- HEAD est déplacé sur C4



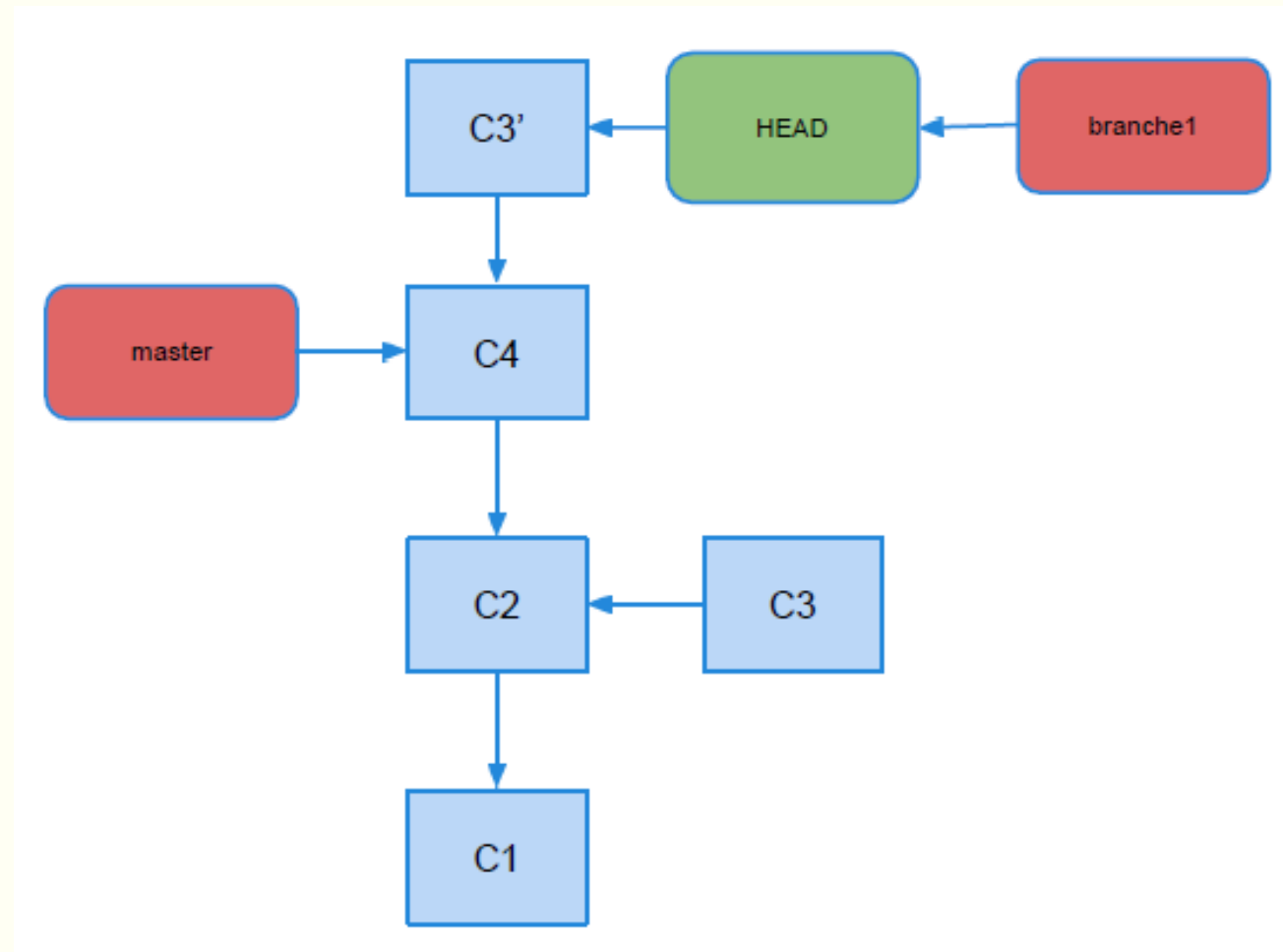
Rebase

- Git fait un diff entre C3 et C2 et l'applique à C4 pour “recréer” un nouveau C3 (C3') dont le père est C4



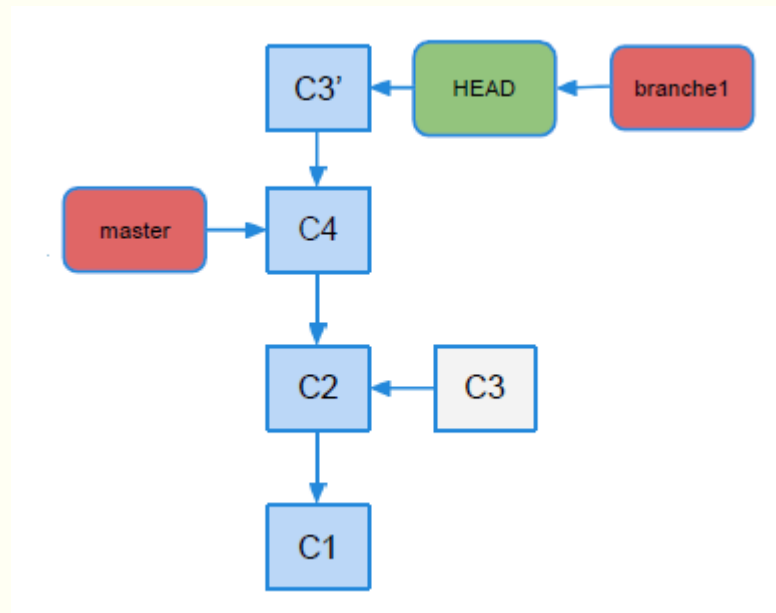
Rebase

- Git reset branche 1 sur la HEAD , le rebase est terminé



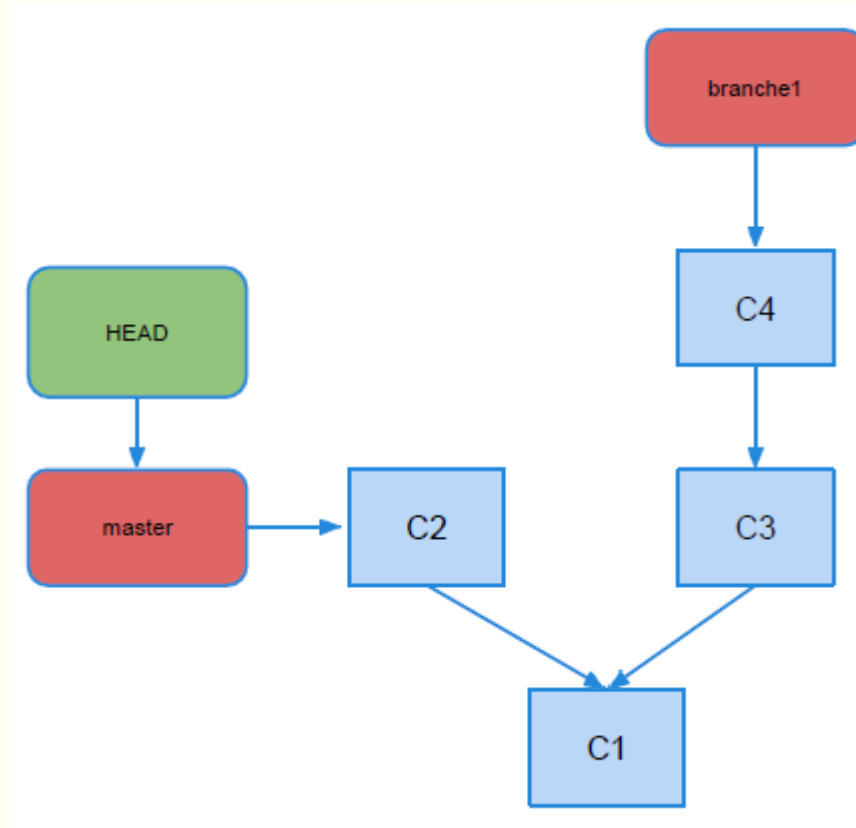
Rebase

- Rebase modifie / réécrit l'historique
- Les commits de branche1 deviennent des descendants de ceux de master (la hiérarchie devient linéaire)
- **On ne modifie pas les commits : de nouveaux commits sont créés à partir de ceux qu'on rebase (on peut toujours les récupérer via id ou reflog)**



Merge VS Rebase

- Situation de départ
- 2 branches
- Ancêtre commun C1

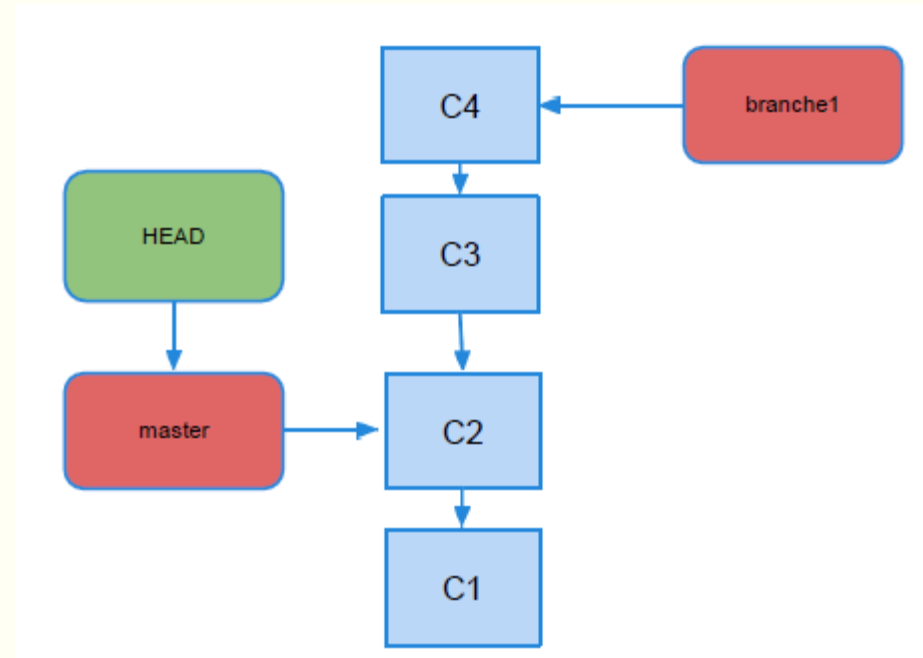


Merge VS Rebase

- **Rebase** : pour la mise à jour des branches avant merge linéaire (commits indépendants) ex : corrections d'anomalies → on ne veut pas de commit de merge
- **Merge sans rebase** : pour la réintégration des feature branches (on veut garder l'historique des commits indépendants sans polluer l'historique de la branche principale)

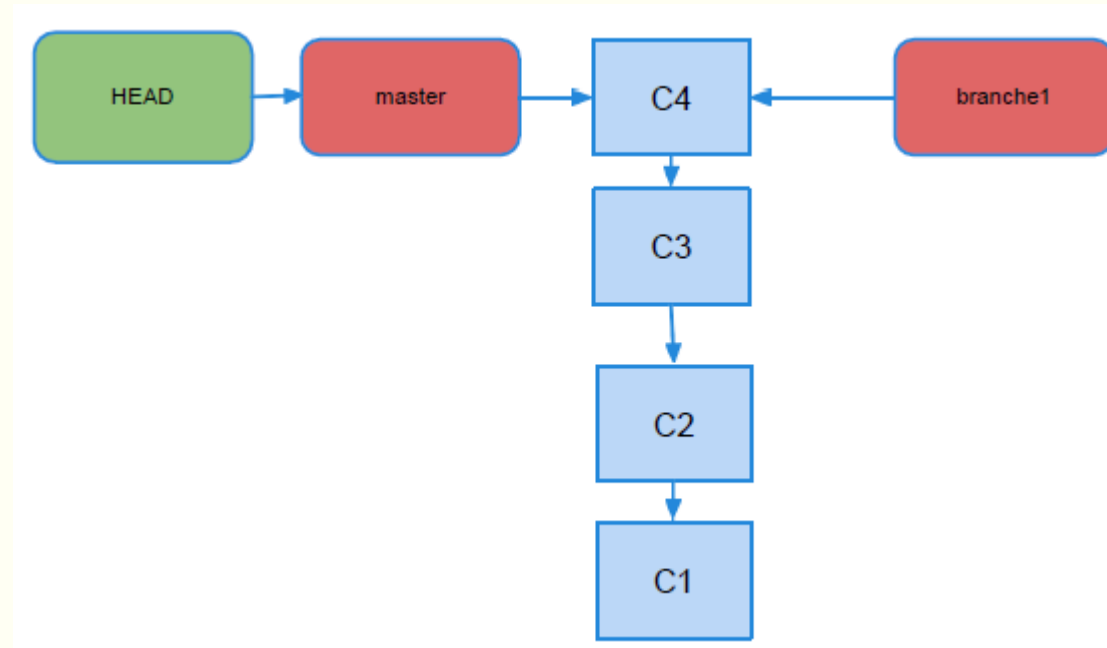
Merge VS Rebase

- Rebase de branche1 sur master



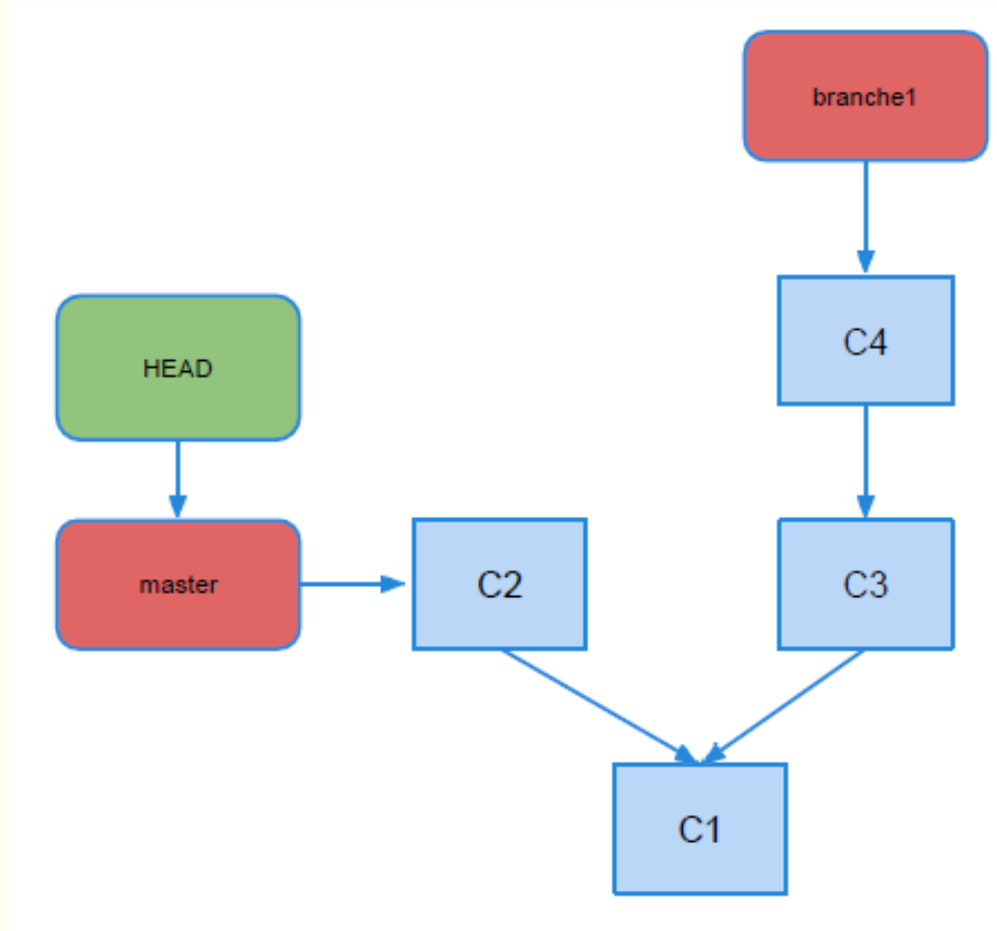
Merge VS Rebase

- Merge de branche 1 dans master
- Fast forward



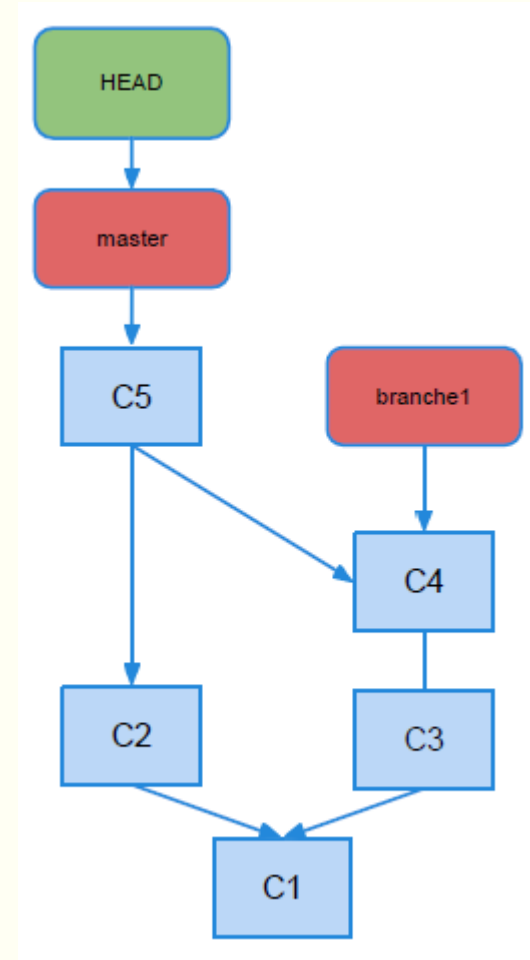
Merge sans Rebase(1/2)

- Situation dedépart
- 2 branches
- Ancêtre commun C1



Merge sans Rebase(2/2)

- Merge de branche 1 dans master
- Non fast forward
- Création d'un commit de merge(C5)



Disposition de titre et de contenu avec liste

Question ?

Disposition de titre et de contenu avec liste

TP

Rebasing



MODULE 8

Git avec un dépôt distant

Présentation

Git avec un dépôt distant
