



# **GIT – Plan du cours**

Said AKHROUF – ETM Ibn Rochd

## **MODULE 1 : Versionnage du code source**

Généralités

Git

Commandes de base

Métadonnées

Annuler les modifications

Branches

Bonnes pratiques

## **MODULE 2 : Fonctions Avancées**

Gitignore

Plateforme d'hébergement

pull request

# **MODULE 1 : Versionnage du code**

# Généralités

## **MODULE 1 : Versionnage du code source > Généralités**

Lorsque le développement était assuré par une seule personne, le développeur se contentait de compresser le dossier contenant le code source. Auquel il ajoutera soit la date soit la version 1, 2, ....

Si le développement collaboratif (en équipe), la tâche devient beaucoup plus difficile surtout lors de la maintenance (*régressive*) des applications. Nous sommes amenés à poser plusieurs questions :

- Qui a fait ça ? (création, modification ou suppression)
- Quand il a fait ça ?
- Pourquoi il a fait ça ?
- Quelles sont les ressources qui ont été modifiées pour le même besoin ?

Toujours lors d'un développement collaboratif, nous devons :

- Éviter (ou tracer) qu'un développeur a écraser la modification d'un autre (*conflit*)
- Repérer la(les) version(s) opérationnelle(s)
- Mettre en place un(des) processus qualité

## ***MODULE 1 : Versionnage du code source > Généralités***

La plupart des IDE disposer d'un mécanisme de versionnage local

Pour les solutions en client/serveur, nous trouvons :

- CVS (Control Version System)
- SVN (Subversion)
- Git
- Mercurial

Il existe également plusieurs solutions propriétaires : Microsoft Azure DevOps (ex-TFS); .....

# Git



## ***MODULE 1 : Versionnage du code source > Git***

Créé par **Linus Torward** en **2005** pour **mieux** gérer le code source de linux

N° 1 actuellement

+ Souple et puissant

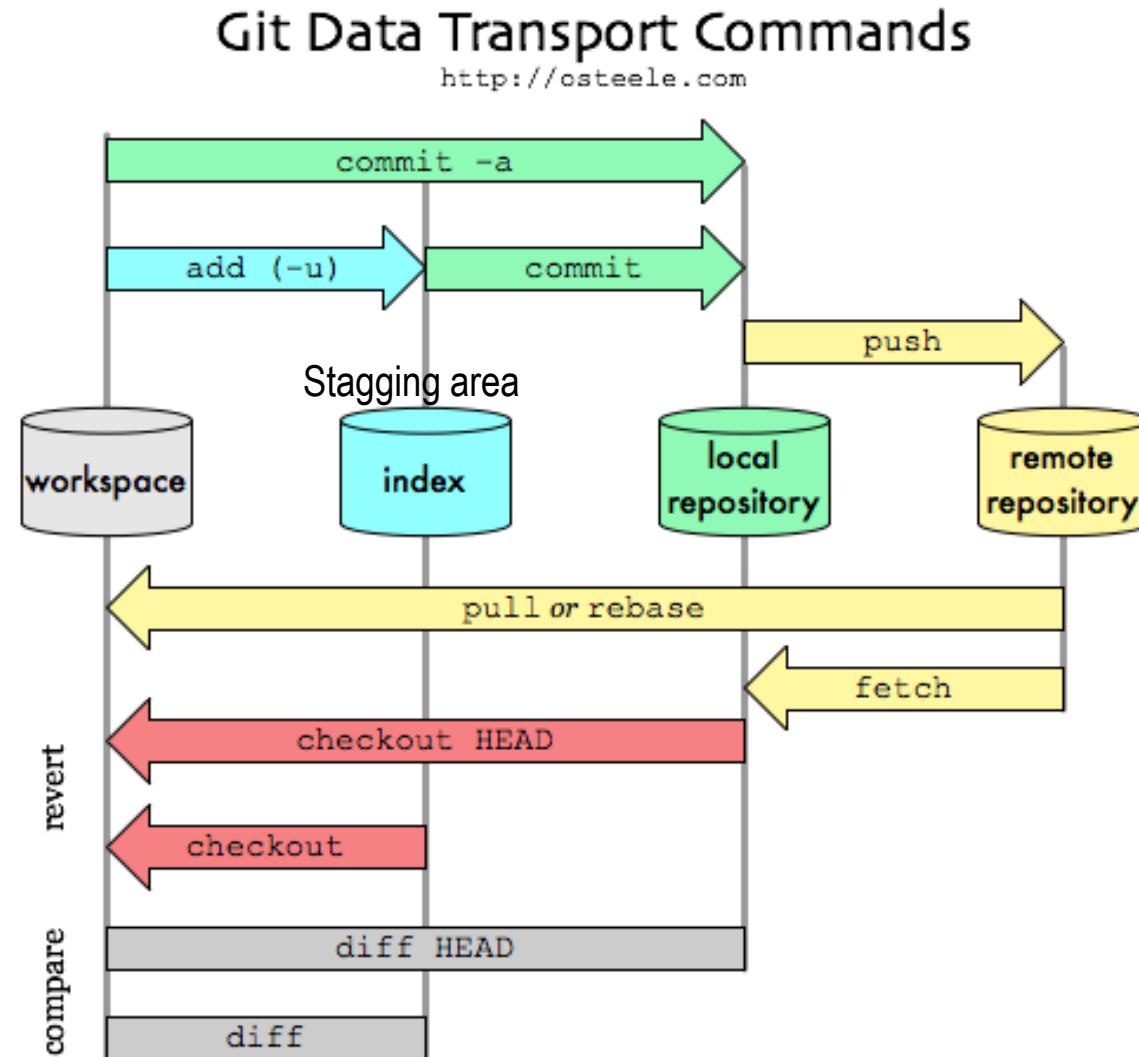
- Complexe

Par rapport au concurrent direct (dans l'open source) :

- Pas d'obligation d'un serveur central : chaque client travaille sur son dépôt
- Accessible même déconnecté : tout l'historique (full-mirroring)
- Organisation plus flexible (branches)



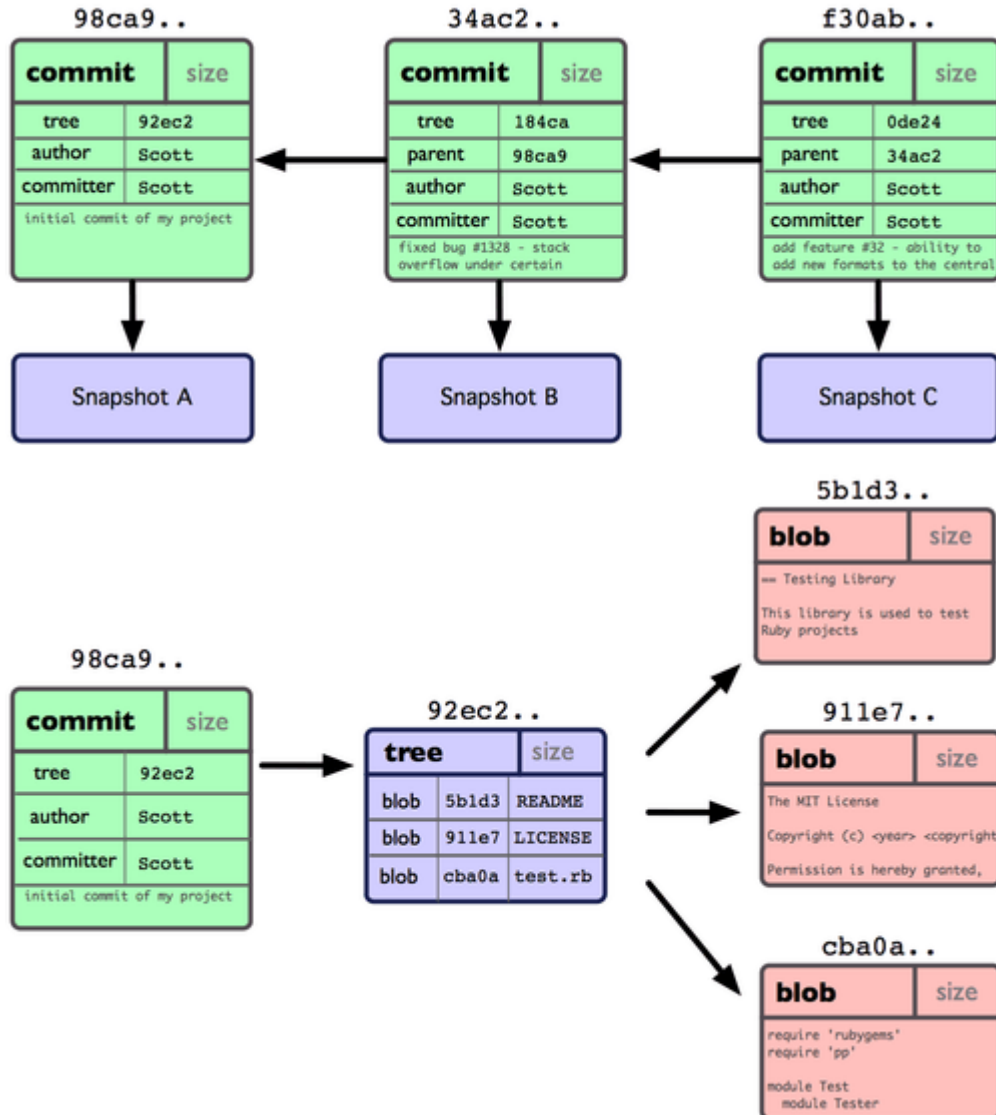
## MODULE 1 : Versionnage du code source > Git



## MODULE 1 : Versionnage du code source > Git

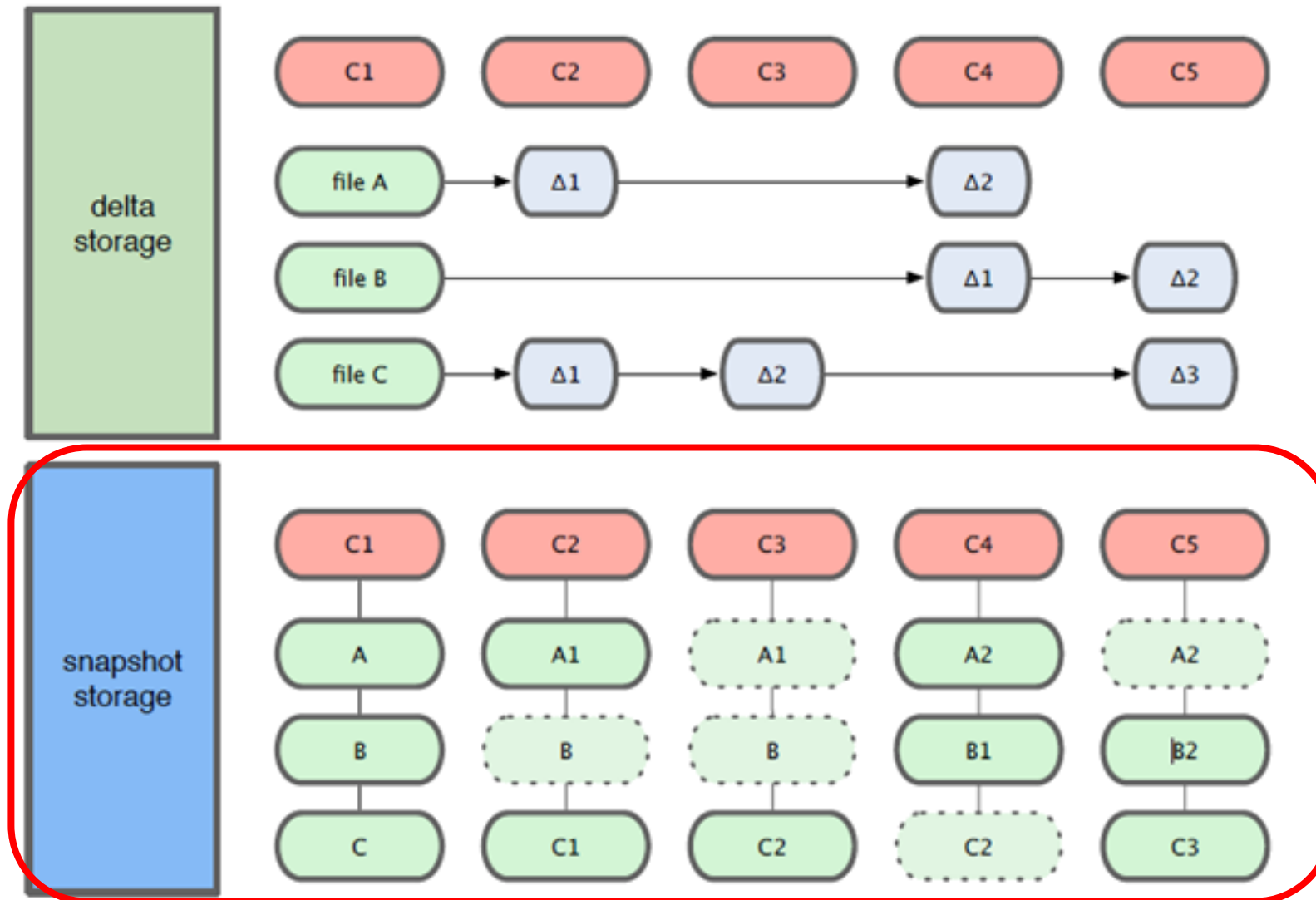
Terme	Description
<b>Repository</b> (dépôt)	dossier versionné (local ou distant)
<b>Commit</b>	Enregistrer les dernières modifications dans le dépôt
<b>Version</b> (révision)	état du code source arrêté par un commit
<b>Branche</b>	version alternative du code source liée à une tentative de développement spécifique
<b>Trunk ou master</b>	branche principale du code source
<b>Head</b>	Branche actuelle
<b>Merge</b>	tentative d'unification de deux branches
<b>Conflit</b>	problème de merge nécessitant une prise de décision

## MODULE 1 : Versionnage du code source > Git



Terme	Description
<b>Blob</b>	Contenu d'une version d'un fichier
<b>Tree</b>	Arborescence de références
<b>Commit</b>	Pointe sur un <b>Tree</b> sur l'envoi et contient les métadonnées
<b>Tag</b>	Annotation manuelle d'un <b>Commit</b>
<b>Hash SHA1</b>	Identifiant unique pour tout objet

## MODULE 1 : Versionnage du code source > Git



# Commandes de base

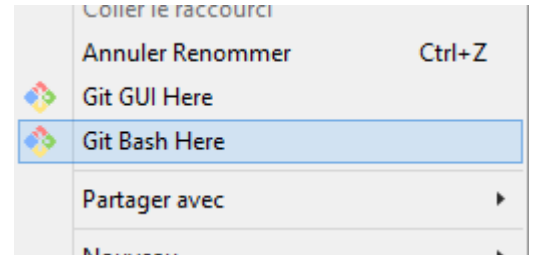
## **MODULE 1 : Versionnage du code source > Commandes de base**

Installation : accéder à [git-scm.com/downloads](https://git-scm.com/downloads) , télécharger la version correspondante à votre système d'exploitation puis installer en suivant l'assistant

Vérification : créer un nouveau dossier puis faites un click-droit

Une fenêtre console sera affichée. Tapez la commande :

```
git --version
```



Configuration : les informations de l'utilisateur actuel sont défini dans le fichier **C:\Users\xxxx\gitconfig**

Ces informations peuvent être modifiées avec les commandes suivantes sur un dépôt

```
git config --global user.name "Said Akhrouf"
```

```
git config --global user.email "Akhrouf@gmail.com"
```

Documentation : une documentation des commandes est disponible dans le dossier

**C:\Programmes\Git\mingw64\share\doc\git-doc** qui est disponible également via la commande

```
git help xxxxx
```

## MODULE 1 : Versionnage du code source > Commandes de base

Créer un dépôt distant : créer un dépôt (**repository**) public ou privé dans l'un des sites :

[github.com](https://github.com)

[bitbucket.com](https://bitbucket.com)

[devops.azure.com](https://devops.azure.com)

Après deux options possibles :

1. Copier depuis un serveur : exécuter la commande suivante (fournie sur le site utilisé)

```
git clone https://github.com/akhrouf/master14.git
```

```
git clone https://github.com/akhrouf/master14.git .
```

```
git clone --mirror https://github.com/akhrouf/master14.git
```

2.a. Initialisation : pour un nouveau projet, vous pouvez d'abord initialiser le dépôt local

```
git init
```

2.b. Indiquer le dépôt distant : afin de pouvoir relier le dépôt local avec le dépôt distant, exécuter la commande

```
git remote add origin https://github.com/akhrouf/master14.git
```



## MODULE 1 : Versionnage du code source > Commandes de base

Ajouter un(des) fichier(s) au versionnage : indexer le(s) nouveau(x) fichier(s) dans le « **staging area** »

```
git add Carre.java  
git add .
```

Enregistrement des modifications dans le dépôt local :

```
git commit Carre.java  
git commit -m "premier commit"  
git commit -a          du Workspace au Local Repo.
```

Corriger le dernier commit avec les modifications actuelles : en cas d'oubli de fichiers

```
git commit --amend
```

Il faut passer par le Git lors de la manipulation du système de fichier sinon les modification seront ignorées

Renommer un fichier :

```
git mv carre.java Carre.java
```

Supprimer un fichier :

```
git rm Carre.java
```

## ***MODULE 1 : Versionnage du code source > Commandes de base***

### ***ECHANGES AVEC LE SERVEUR DISTANT :***

Envoyer les modifications vers le dépôt distant :

```
git push
```

Récupérer le dépôt distant et l'appliquer sur l'espace de travail :

```
git pull
```

Récupérer le dépôt distant et l'appliquer sur le dépôt local sans affecter l'espace de travail :

```
git fetch
```

# Métadonnées

## ***MODULE 1 : Versionnage du code source > Métadonnées***

Les modifications actuelles dans l'espace de travail (ajouté, modifié, supprimé) :

```
git status
```

Voir les modifications actuelle :

```
git diff
```

modifications entre Workspace et Staging Area

```
git diff sha1_1 sha1_2
```

modifications entre deux commits

Historique des commits :

```
git log
```

Historique des opérations exécutées :

```
git reflog
```

Afficher qui a modifié un fichier :

```
git blame Carre.java
```

Annuler les modifications

## MODULE 1 : Versionnage du code source > Annuler les modifications

Retirer un fichier indexé du « staging area » : inverse de « git add »

```
git reset Carre.java
```

Annuler les modifications sur l'espace de travail :

```
git checkout Carre.java  
git checkout sha1 Carra.java
```

mettre le contenu du « staging area »  
mettre le contenu du commit **sha1**

Changer la HEAD (pointeur actuelle) :

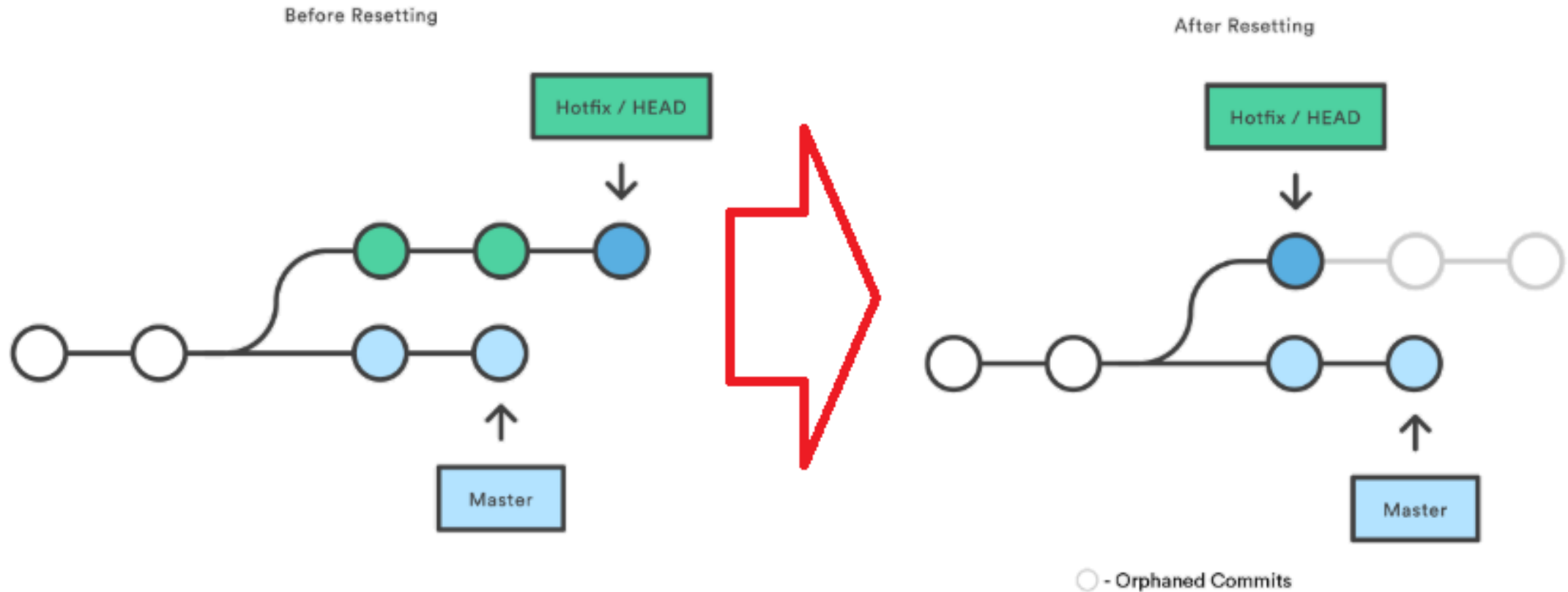
```
git reset sha1 --soft  
git reset sha1 --mixed  
git reset sha1 --hard  
git reset 'HEAD@{1}'
```

écrase le dépôt local seulement  
n'écrase pas l'espace de travail  
écrase le tout  
annuler le dernier reset

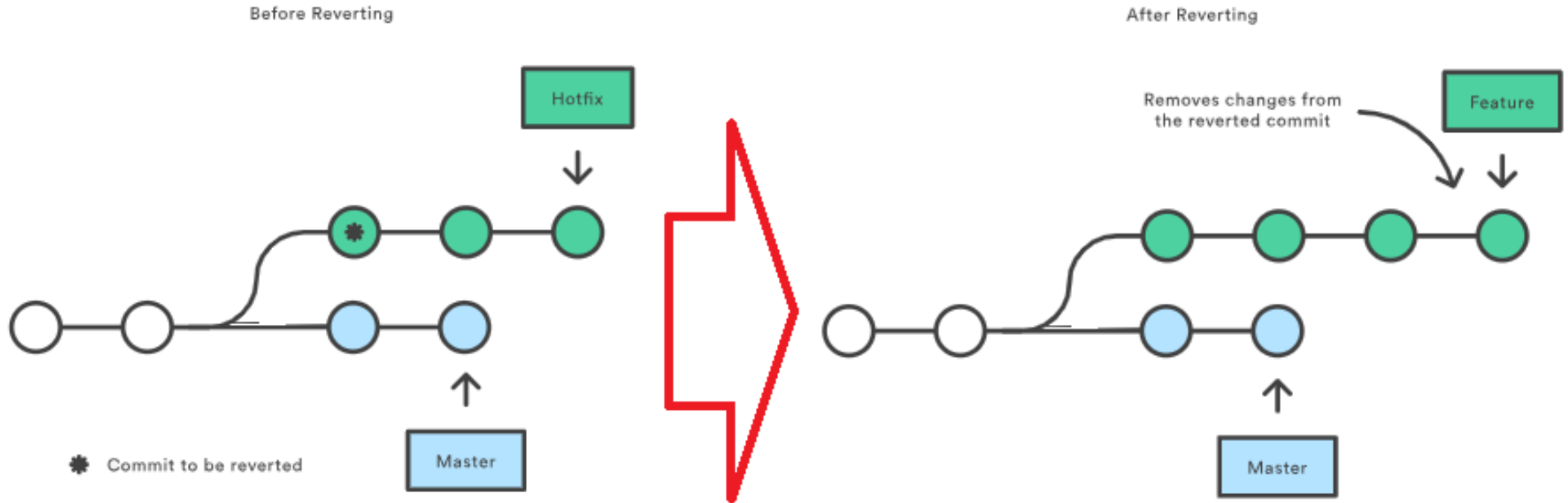
Annuler le commit en laissant l'historique :

```
git revert sha1
```

## MODULE 1 : Versionnage du code source > Annuler les modifications

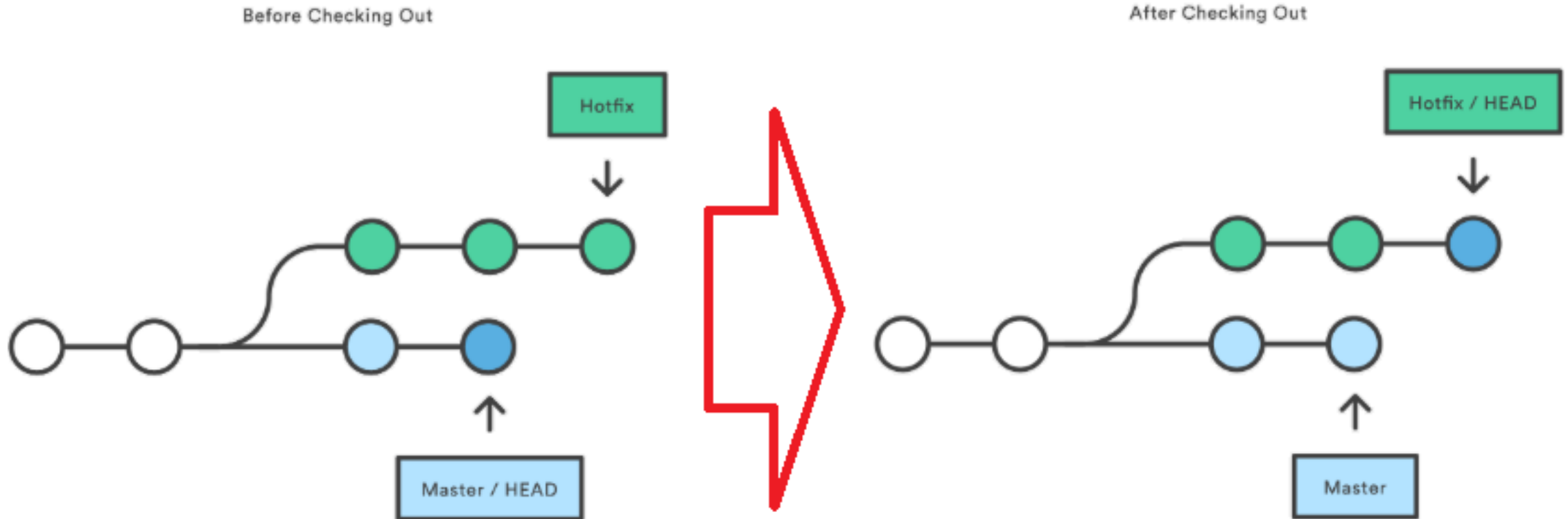


## MODULE 1 : Versionnage du code source > Annuler les modifications





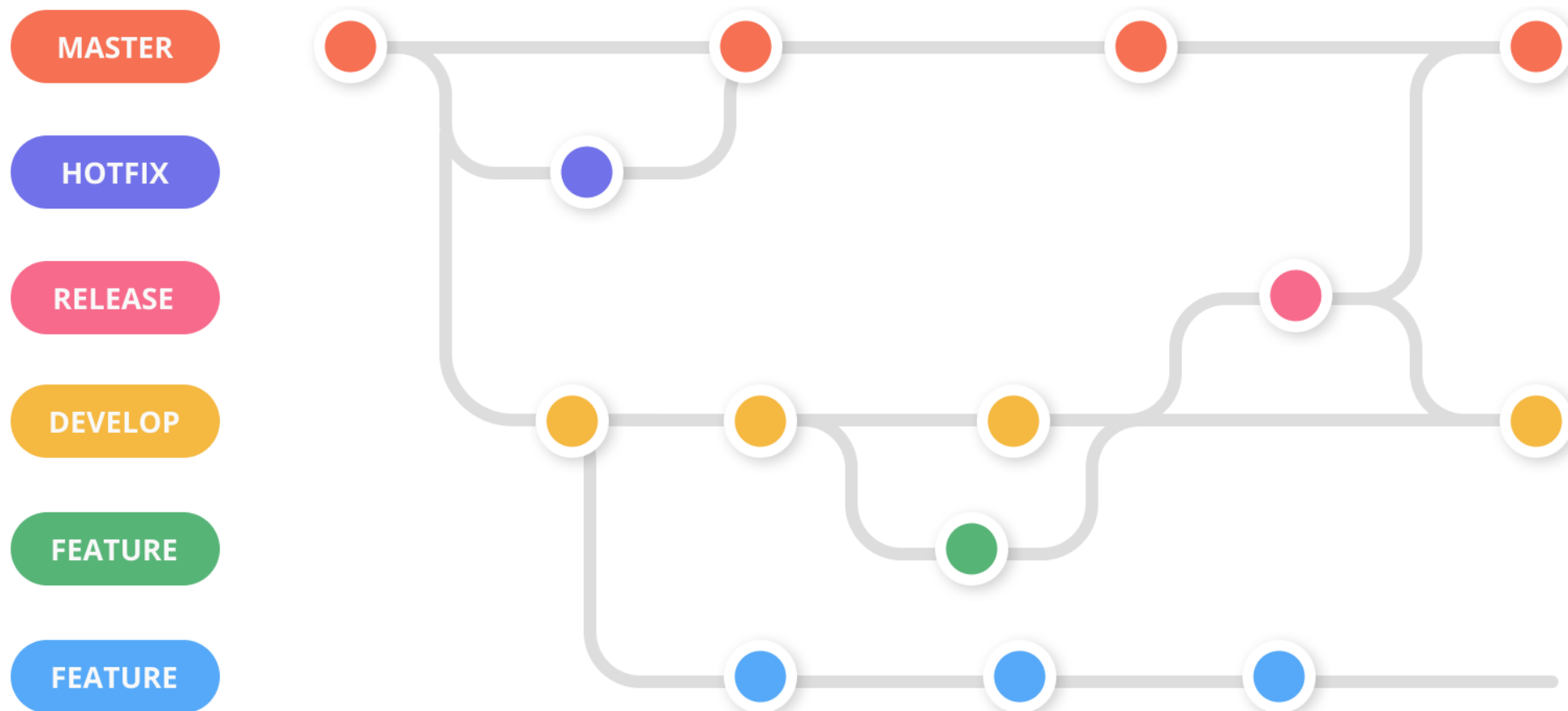
## MODULE 1 : Versionnage du code source > Annuler les modifications



# Branches

## MODULE 1 : Versionnage du code source > Branches

L'objectif principal est de faire des développements en parallèle de la branche principale (MASTER) afin d'avoir toujours un code opérationnel (qu'on peut déployer)



## MODULE 1 : Versionnage du code source > Branches

### Lister les branches existantes :

```
git branch
```

### Créer une nouvelle branche :

```
git branch créer_patient  
git branch créer_patient develop  
git branch créer_patient sha1
```

### Changer de branche (pointer le HEAD vers le commit le plus récent) :

```
git checkout créer_patient  
git checkout -b créer_patient  
git checkout master  
git checkout sha1
```

créer et se positionner une branche  
se positionner sur la branche principale  
se positionner sur un commit particulier

### Supprimer une branche :

```
git branch -d créer_patient
```

## MODULE 1 : Versionnage du code source > Branches

### Fusionner deux branches :

```
git checkout develop  
git merge créer_patient
```

**Il faut d'abord se positionner sur la destination  
Puis préciser la source du « merge »**

La fusion peut ne pas s'exécuter à cause des conflits. Dans ce cas, on peut :

- Soit annuler la fusion :

```
git merge --abort
```

- Soit résoudre les conflits en éditant les fichiers puis finaliser la fusion avec :

```
git add .
```

Puis :

```
git commit -m ".."
```

ou :

```
git merge --continue
```

### Affichage des conflits :

```
git status
```

## ***MODULE 1 : Versionnage du code source > Branches***

### Fusion avec une stratégie de résolution de conflits : (à éviter)

```
git merge -s ours
```

Priorité à nos fichiers

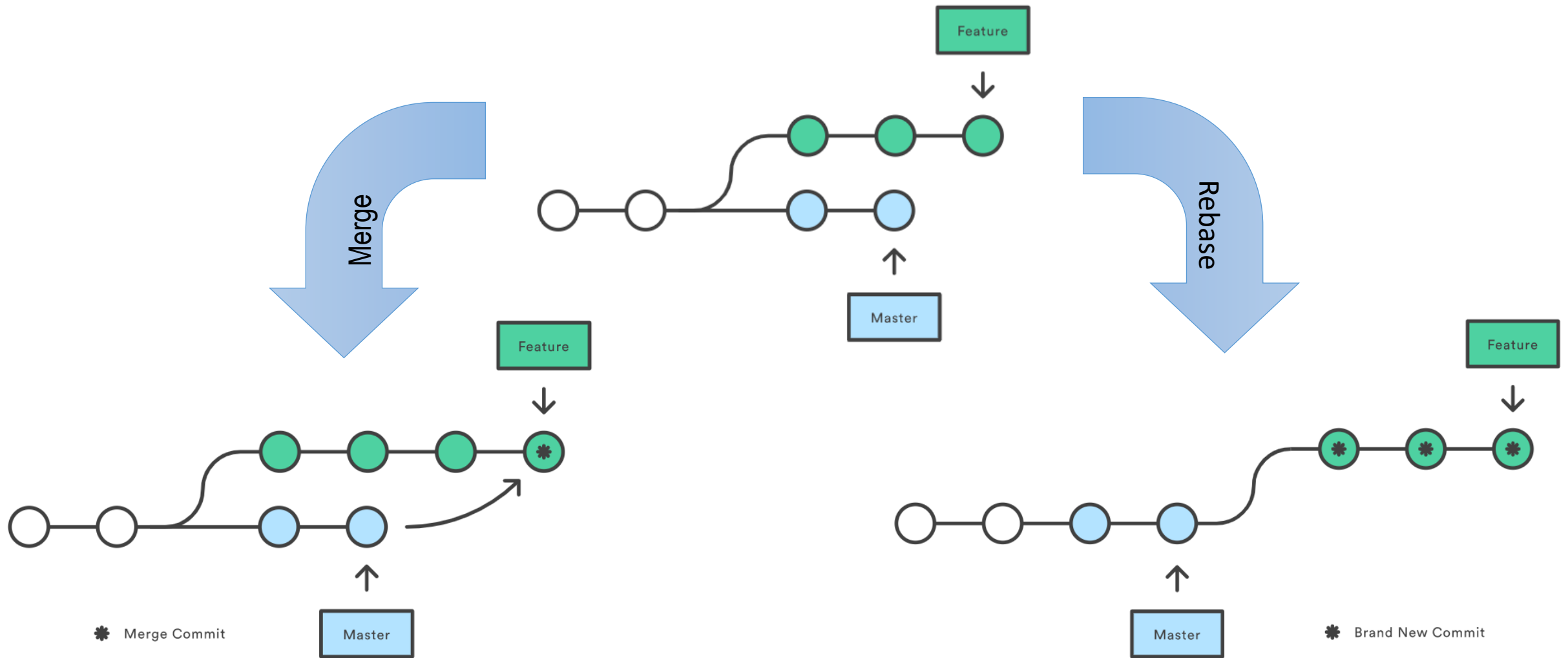
```
git merge -s theirs
```

Priorité aux fichiers existants (des autres)

### Intégrer toutes les modifications d'une branche sur une autre branch :

```
git rebase créer_patient
```

## MODULE 1 : Versionnage du code source > Branches



Bonnes pratiques



## ***MODULE 1 : Versionnage du code source > Bonnes pratiques***

Ne pas versionner de fichiers générés automatiquement (logs, PDF, exécutables, etc.) ou personnels

La branche MASTER doit toujours contenir un code déployable

Faire de petits commits réguliers et facile à intégrer, leur donner un nom explicite

Utiliser les branches pour :

- Les développements à plusieurs ;
- Chaque développement conséquent d'une nouvelle fonctionnalité

Ne pas développer sur la branche master à plusieurs pour éviter les conflits lors des pull

Faire de petits commits locaux, et pusher des commits plus conséquents, toujours testés et fonctionnels !

Faire des pull régulièrement

# **MODULE 2 : Fonctions Avancées**

# GitIgnore

## **MODULE 2 : Fonctions Avancées > Gitignore**

Git ne voit que des lignes de texte

Éviter de versionner les fichiers non textuels, images, binaires, logs ...

Pour les projets publiques, ne pas versionner les fichiers de paramétrage (infos serveur BDD, messagerie, ...)

Créer un fichier **.gitignore** pour définir quels fichiers ne seront pas versionnés

Des modèles existent sur : [gitignore.io](https://gitignore.io) ou [github.com/github/gitignore](https://github.com/github/gitignore)

## **MODULE 2 : Fonctions Avancées > GitIgnore**

### **Exemple 1 :**

```
# gitignore template for InforCRM (formerly SalesLogix)
# website: https://www.infor.com/product-summary/cx/infor-crm/
#
# Recommended: VisualStudio.gitignore

# Ignore model files that are auto-generated
ModelIndex.xml
ExportedFiles.xml

# Ignore deployment files
[Mm]odel/[Dd]eployment

# Force include portal SupportFiles
!Model/Portal/*/SupportFiles/[Bb]in/
!Model/Portal/PortalTemplates/*/SupportFiles/[Bb]in
```

## ***MODULE 2 : Fonctions Avancées > GitIgnore***

### **Exemple 2 : (partie pour ASP.NET Core)**

```
# User-specific files
*.suo
*.user
*.userosscache
*.sln.docstates
```

```
# Build results
[Dd]ebug/
[Rr]elease/
x64/
x86/
bld/
[Bb]in/
[Oo]bj/
[Ll]og/
```

# Plateformes d'hébergement

## **MODULE 2 : Fonctions Avancées > Plateformes d'hébergement**

Déployer chez soi une plateforme Git telle que : **GitLab** (**linux seulement**), **Gogs**, ...

Télécharger et décompresser Gogs depuis [dl.gogs.io/](https://dl.gogs.io/)

A l'intérieur du dossier décompressé, lancer la commande suivante :

```
gogs web
```

La console affichera alors le numéro de port d'écoute, par défaut 3000.

Pour l'installation, taper dans le navigateur [127.0.0.1:3000](http://127.0.0.1:3000)

Remplir les informations relatives au serveur de BDD et au serveur de messagerie.

Faute d'avoir **MySQL**, **SQL Server** ou **PostgreSQL**, sinon Gogs utilisera **SQLite3**

**Il faut remplir également, les paramètres du compte administrateur (dernière section)**





## MODULE 2 : Fonctions Avancées > Plateformes d'hébergement

Gogs permet de gérer les tickets qui peuvent être associés à des jalons

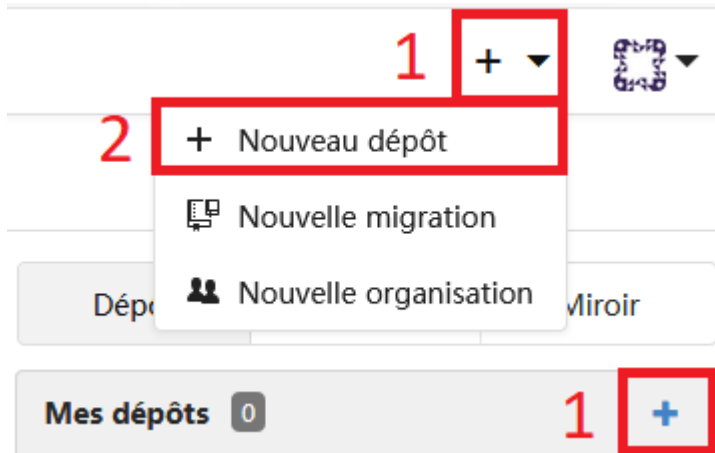
Commencer par définir les utilisateurs et les organisations (équipes, département, ...)

The screenshot shows the Gogs web interface. On the left, a user menu is open, showing options like 'Votre profil', 'Vos paramètres', 'Aide', 'Administration' (highlighted with a red box and labeled '2'), and 'Déconnexion'. Above this menu, a red box highlights a user icon with a dropdown arrow, labeled '1'. In the center, a sidebar menu lists 'Administration' (highlighted with a red box and labeled '3'), 'Tableau de bord', 'Utilisateurs', 'Organisations', 'Dépôts', 'Authentifications', 'Configuration', and 'Notes Systèmes'. On the right, the 'Gestion des Utilisateurs (Total : 1)' section is visible, with a red box highlighting the 'Créer un nouveau compte' button, labeled '4'. Below this is a search bar and a table of users.


ID	Nom	E-mail	Activés	Administrateur	Dépôts	Créés	Éditer
1	akhrouf	akhrouf@gmail.com	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	0	Sep 03, 2020	<a href="#">✎</a>

## MODULE 2 : Fonctions Avancées > Plateforme

Maintenant, créer un dépôt (distant)



### Nouveau dépôt

**Propriétaire \***  akhrouf

**Nom du dépôt \*** master14

Idéalement, le nom d'un dépôt devrait être court, mémorable et **unique**.

**Visibilité** ☒ Ce dépôt est **privé**

**Description**

Description du dépôt. 512 caractères maximum.

Caractères disponibles: 512

**.gitignore** Choisissez un modèle de fichier .gitignore

**Licence** Sélectionner un fichier de licence

**Fichier Readme ?** Default

☐ Initialiser ce dépôt avec le modèle et les fichiers sélectionnés

**Créer un dépôt** Annuler



## Introduction rapide

### Cloner ce dépôt

Besoin d'aide pour dupliquer ? Visitez [l'aide](#) !

HTTP

SSH



### Créer un nouveau dépôt en ligne de commande

```
touch README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin http://localhost:3000/akhrouf/master14.git
git push -u origin master
```

### Soumettre un dépôt existant par ligne de commande

```
git remote add origin http://localhost:3000/akhrouf/master14.git
```

Pull request

## MODULE 2 : Fonctions Avancées > Pull Request

Cette fonctionnalité n'existe pas dans Git. Elle a été créée initialement par Github.  
Peut avoir un autre nom : GitLab (merge request)

Généralement, pas tout développeur peut fusionner sa branche (develop, créer\_patient ou autre) avec la branche MASTER (ou autre). L'objectif étant d'appliquer un processus Qualité.

Ce dernier va demander au responsable de la branche de faire un « pull » depuis la branche du développeur

Le responsable a la possibilité de rejeter le « pull request » afin de permettre au développeur de prendre en charge les remarques signalées ou d'accepter la fusion

