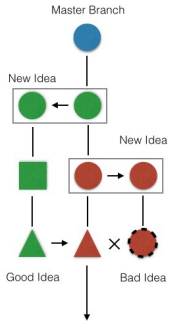
# Séance 4

# Git : les branches

## Concepts

### La branche master

Par défaut, la branche sur laquelle se trouvent tous les commits est la branche « master ».

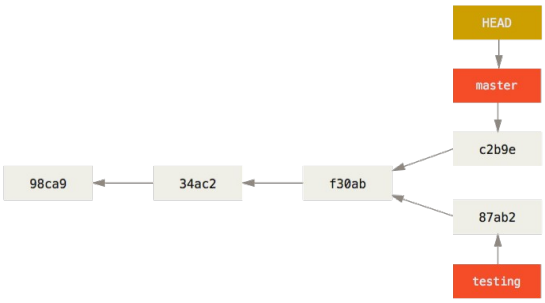



### A quoi servent les branches ?

On crée une branche lorsque l'on veut maintenir 2 versions du code. Une version restera inchangée, alors que l'autre version pourra évoluer. Lorsque les évolutions seront validées, l'ancienne version pourra être abandonnée.

### Workflow

En général, on crée une branche « dev » à partir de « master ». Les nouveaux commits sont enregistrés sur cette branche « dev ».




A terme, on fusionnera la branche « dev » sur « master », pour rapatrier tous ces commits.

**Bonnes pratiques :**

Il est recommandéde supprimer la branche « dev » après sa fusion dans « master »

Il est également possible de fusionner les commits de dev au moment de la fusion

# Branches locales

## Commandes de base

Exercice

Créer un dépôt vide et executer les commandes présentées ci-après

### Création

Syntaxe

```
git checkout -b <branch>
ou bien
git switch -c <branch>
```

Crée une branche **dans le dépôt local.**
Cette branche est similaire à la branche courante (mêmes commits)
Elle déplace le HEAD du dépôt local sur cette branche (Schéma)

### Lister les branches

Syntaxe :

```
git branch
```

La branche courante est représentée avec un asterisk

### Changer de branche

```
git checkout <branchname>
ou bien
git switch <branch>
```

Lorsqu'on change de branche, on déplacer la tête HEAD de git
**NB : Les modifications faites dans le répertoire de travail sont indifférentes à la branche sur laquelle se trouve le HEAD du dépôt local.**

## Connaître la branche sur laquelle on se trouve

```
git status
```

## Renommer

Syntaxe

```
git branch -m new-branchname
```

## Historiques

**D’une seule branche :**

```
git log contactForm
```

```
4798a1e (HEAD -> contactForm, origin/contactForm) Contact Form
b445176 (origin/feature-priority, feature-priority) Install test pack
e4ae8ca File Upload
e4ca10f Webpack encore
8e1f89f Category : fixtures
fb66cd8 Priority : Fixtures and twig display
0d266eb update todo
975b65a Entity Creation
08fd4b6 (origin/feature-filtre-tri, feature-filtre-tri) secure app_secret
7b957b4 fix coding style
b9bd6f0 filtres-et-tri
0111142 (origin/master, master) filtres
eeadb9a update query builer for dynamic ordering
693e88e todo categories
6df5842 cleaning...
98a36c7 Order Todos
1ac4660 Assets et Crud Todo
88ae6f1 Entité Todo
044b130 first commit
```

**Sous forme de graphe :**

```
git log --oneline --graph --all
```

https://devhints.io/git-log

### Comparer les branches

Syntaxe

```
git diff <source_branch> <target_branch>
```

### Suppression

Syntaxe

```
git branch -d <branch>
```

## Fusion de branches

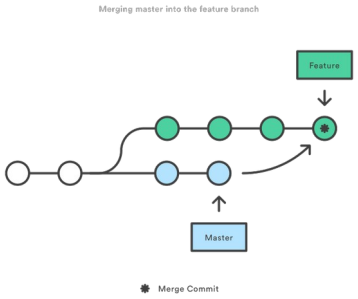
### merge

La commande « merge » crée un commit spécial qui a deux parents. Elle ajoute les commits d'une branche dans une autre. Par exemple, lorsqu'on a travaillé sur des nouvelles fonctionnalités qu'on veut livrer en production, on fusionne la branche de dev dans la branche de prod.
Se positionner sur la branche qui va recevoir les modifications

```
git checkout <nom-de-branche-qui-va-recevoir-les-commits>
```

puis :

```
git merge <branch>
```



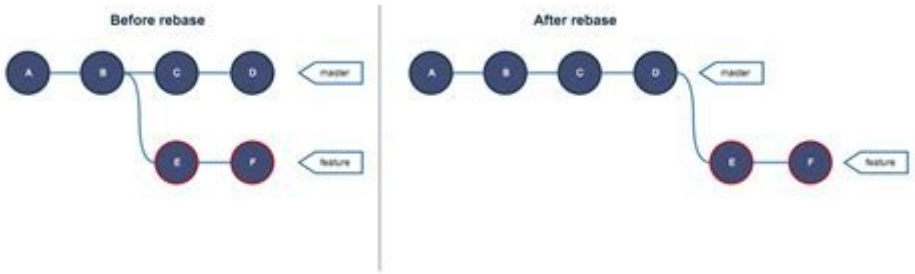

NB : Une fois que la branche A est fusionnée dans une branche B, on peut se rendre dans sur branche A et fusionner B dedans. Ainsi les deux branches auront le même historique

Exercice

Créer un dépôt vide
Faire un premier commit sur master
Créer une branche dev. Faire un commit.
Revenir sur la branch master, lancer un merge
Visualiser l'historique

## rebase

Lorsqu'on a démarré il y a un bail un travail parallèle (une expérience, un chantier de recherche…) mais qu'on n'a plus eu de temps à y consacrer depuis longtemps, et qu'entre-temps la branche de départ, celle dont notre expérience est partie, a beaucoup bougé. Lorsqu'on peut enfin reprendre le boulot sur la branche « annexe », on aimerait faire repartir son travail d'une version récente, à jour, de la branche de base, histoire de profiter des évolutions de celle-ci.
Avantage : l'historique est plus linéraire après un rebase...




**Syntaxe :** Se positionner sur la branche « annexe »

```
git rebase <basebranch>
```

Exercice

Créer un dépôt vide
Créer une branche dev. Faire un commit.
Revenir sur la branche master. Faire un commit
Revenir sur la branche dev. Faire un commit
Visualiser l'historique
Faire un rebase

Visualiser l'historique

NB : Une fois que la branche A est rebase à partir d'une branche B, on peut se rendre dans sur branche B et la rebase sur A. Ainsi les deux branches auront le même historique

**Pour aller plus loin**

https://delicious-insights.com/fr/articles/bien-utiliser-git-merge-et-rebase/

Exercice : Hotfix + rebase

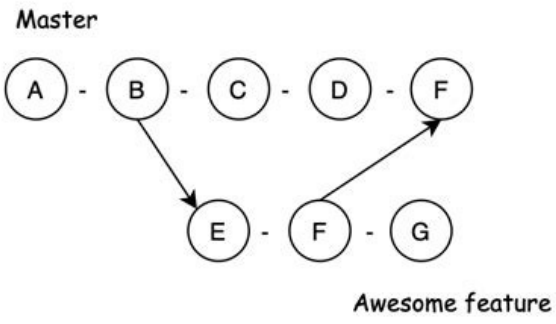
TODO

**Attention : Le rebase modifie l'historique**

Gérer les conflit avec les branches distantes : https://betterstack.com/community/questions/git-push-rejected-after-feature-branch-rebase/

## Cherry-picking

Le cherry-picking permet d'ajouter un commit d'une branche dans l'historique d'une autre branche.




Après un commit :

```
git checkout dev
git cherry-pick <hash>
git checkout master
```

# Branches distantes

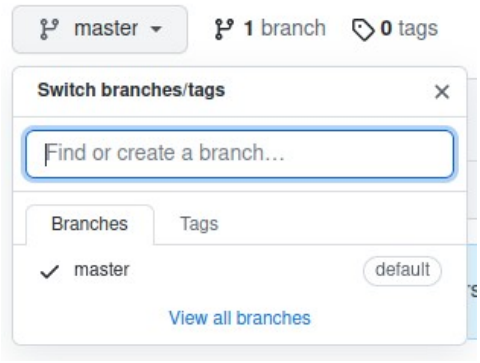
## Commmandes

### Pousser une branche locale

```
git push origin <branch>
```

### Créer une branche sur GitHub

Il est possible de créer une branche directement sur votre plateforme




### Lister les branches distantes

Avant de pousser une branche, on veut savoir si la branche n'existe pas sur le dépôt distant.

```
git branch -r
git branch -a
```

### Tirer une branche distante qui n'existe pas localement

```
git fetch origin
git checkout --track origin/<branch-name>
```

```
Exercice en binôme
```

### Tirer une branche distante

La commande git pull fusionne la branche distante spécifiée **dans la branche courante.** C'est la combinaison de deux autres commandes, git fetch suivie de git merge.

```
git pull <remote-name>[/<branch-name>]
```

https://www.atlassian.com/fr/git/tutorials/syncing/git-pull

### Suppression locale et distante

```
git branch -d <branch-name>
git push --delete <remote-name> <branch-name>
```

https://www.zendevs.xyz/comment-supprimer-une-branche-git-en-local-et-a-distance/

### Ecraser une branche locale

Se placer sur la branche locale, puis :

```
git reset --hard origin/master
```

## `pull request`

TODO

## `Protéger une branche dans un dépôt privé`

TODO

# Commandes avancées

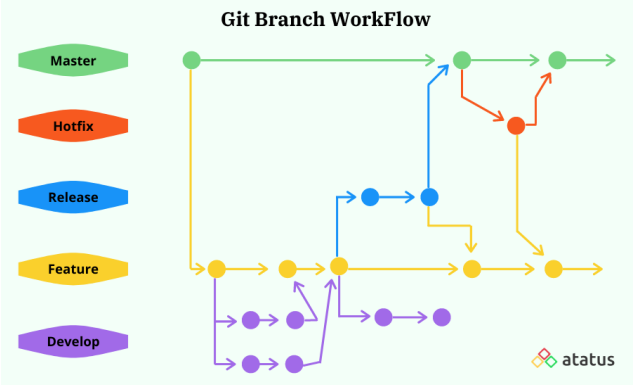
## Changer d'éditeur par défaut

Pour les fusions, l'éditeur par défaut est Vim. Pour utiliser Nano :

```
git config --global core.editor "nano"
```

# Git Flow

## Schéma




## Tutoriel

https://mindsers.blog/fr/gitflow-la-methodologie-et-la-pratique/

## Git-flow ou Trunk-Based ?

https://www.abtasty.com/fr/glossary/trunk-based-development/

# Récupérer un fichier d'une autre branche

```
git checkout <file> <branch>
```

Attention Rappel : checkout est une commande a plusieurs usages

1. Changer de branche
2. Récupérer des fichiers
3. Se déplacer dans l'historique

# Créer une branche à partir d'un commit particulier

## Etape 1 : se déplacer dans l'historique

Syntaxe :

```
git checkout <hash>
ou
git checkout tags/<tagname>
```

## Etape 2 : créer une branche

Syntaxe :

```
git branch <nom-de-la-nouvelle-branche>
```

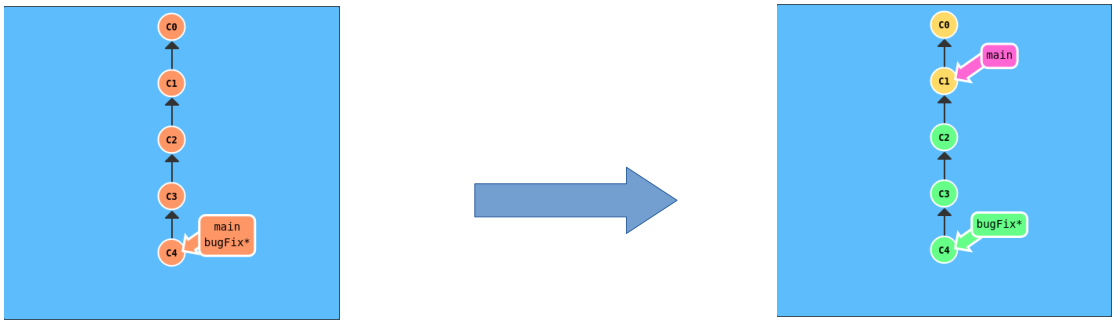
## Etape 3 : repositionner le HEAD sur la branche d'origine

Syntaxe

```
git checkout master
```

# Déplacer une branche sur un commit particulier

```
git branch -f <branch> <commit>
```




Equivalent à

```
git checkout main
git rebase c1
```

## Pour aller plus loin

### Documentation

https://docs.github.com/en/github

### Apprendre

https://learngitbranching.js.org/?locale=fr_FR