Branching and Merging

Branching Basics

```
1 git branch # list local branch
2 git branch -a # list local and remote branch (*master = local & remotes/origine
  /master = remote)
3 git branch mynewbranch # new branch is created
4 git branch -a # new branch listed
5 git checkout mynewbranch # pour changer de branche
6 git branch -a
7 git log --oneline --decorate
8 git checkout master
9 git branch -m mynewbranch newbranch # renommer une branche
10 git branch -a
11 git branch -d newbranch
12 git branch -a
```

Happy Path / Fast Forward Merges

Merge se fait toujours d'une branche sur une autre :

exemple :

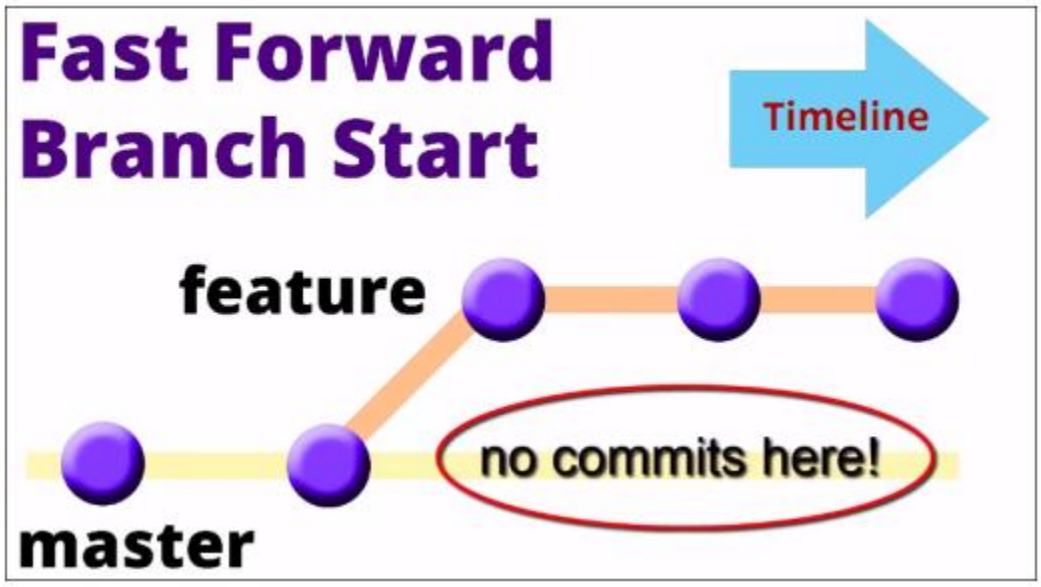
La branche courante est master :

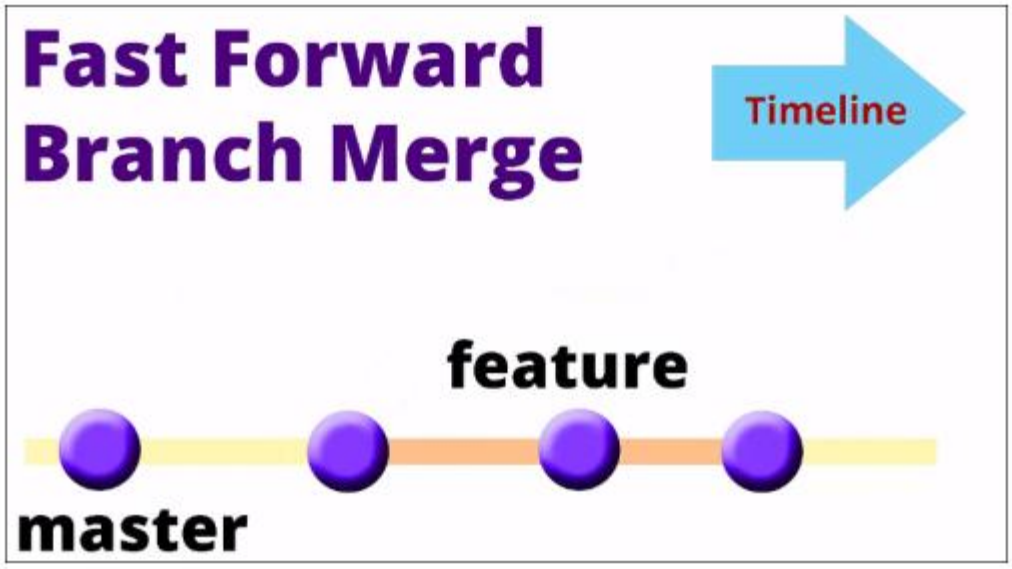
git merge title-change

va provoquer le merge de **title_change sur master**.

master sera modifiée pour intégrer les changements que title_change apporte.

title_change ne sera pas modifiée et pourra être supprimée après si elle n'est plus utile.


Fast Forward
Branch Start
Timeline
feature
no commits here!
master


Fast Forward
Branch Merge
Timeline
feature
master

git hist : correspond à l'alias hist que nous avons créé sur la commande git log dans le chapitre basics

```
git checkout -b  title-change # crée une branche et fait un checkout sur la branche
git status
npp simple.html # changer le texte
git status
git add .
git commit -m "Changing title of HTML"
git log --oneline
git checkout master  # On se replace dans master
git diff master title-change
git difftool master title-change
git merge title-change # master intègre les changements de title_change
git hist  # Correspond à l'alias hist que nous avons créé sur la commande git log dans le chapitre basics
git branch
git branch -d title-change # destruction de de title_change
git hist
```

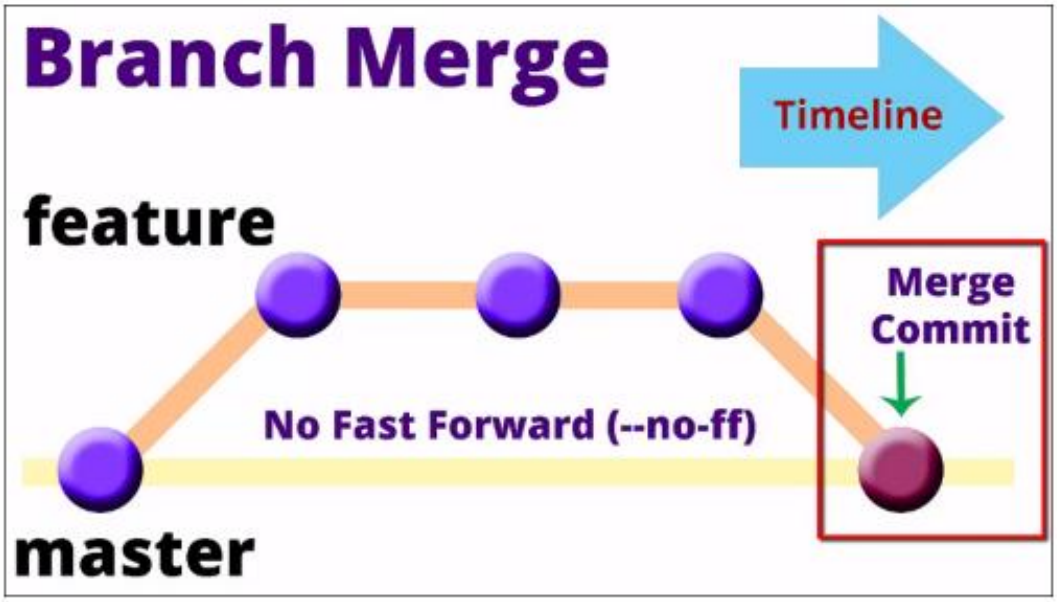
Happy Path / Disable Fast Forward Merges

merge va appliquer tous les commits historiques les uns après les autres jusqu'au dernier.

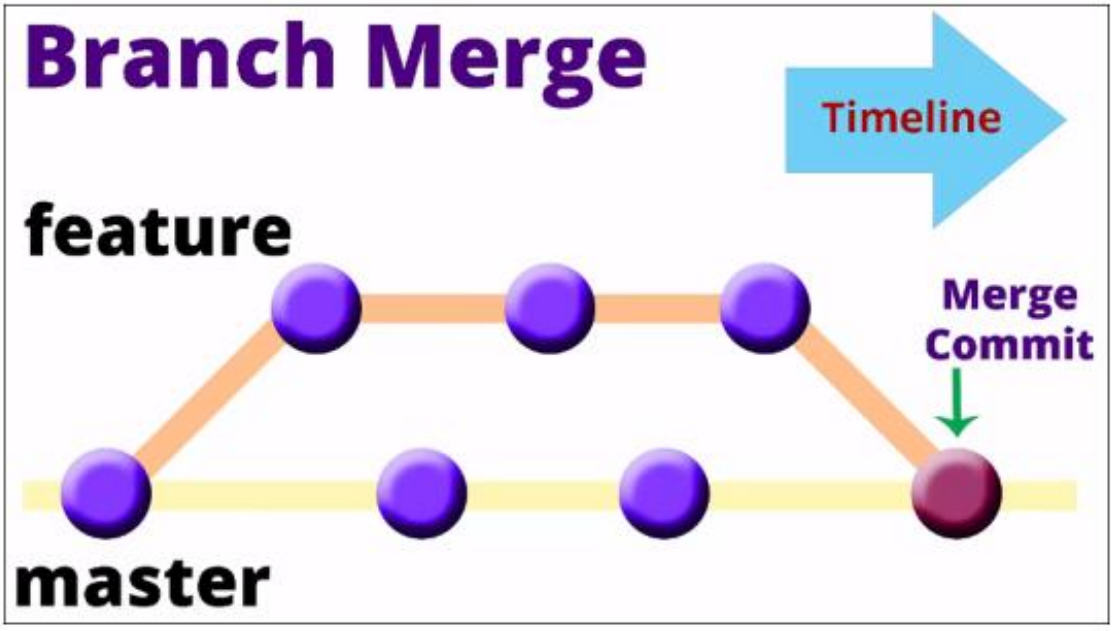
- Si entre les 2 branches , les commits historiques sont cohérents => pas de pb , on applique **Fast Forward** : pas de création de nouveau commit , master reprend le commit de la branche fusionnée.
- Si il y a ambiguité entre les commits historiques des branches => on applique **No Fast Forward** : Il y a creation d'un nouveau commit, le commit de merge

git hist : correspond à l'alias hist que nous avons créé sur la commande git log dans le chapitre basics

```
git checkout -b add-copyright
git branch
npp simple.html # ajouter du code pour le copyright
git status
git add .
git commit -m "Adding copyright notice"
npp README.md # Ajouter du texte pour le copyright
git add .
git commit -m "Adding copyright notice to readme"
git hist
git checkout master
git merge add-copyright --no-ff
git hist
git branch -d add-copyright
git hist
```



Automatic Merges

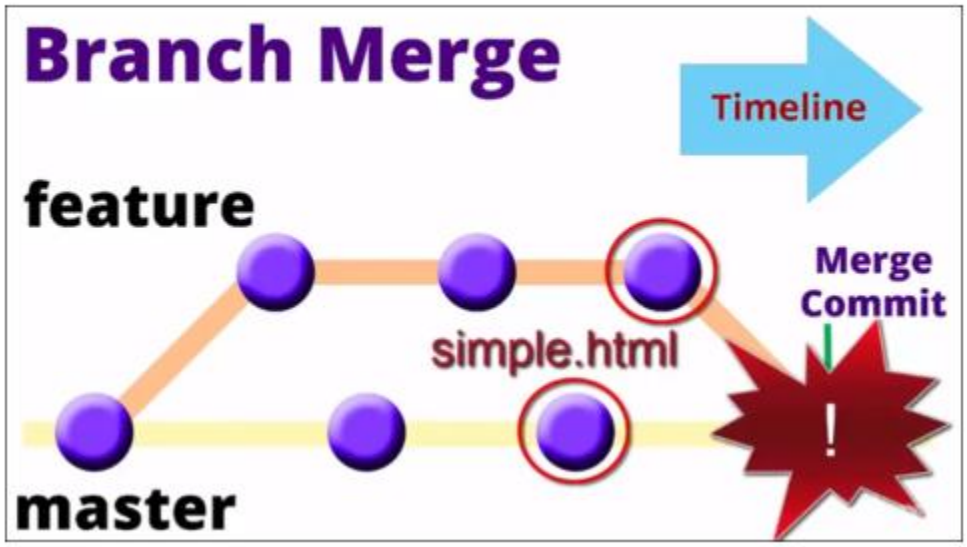


Dans le cas où le master et la branche ont effectué des commits chacun de leur côté, au moment du merge , git va automatiquement créer un commit de merge, sans qu'on ait besoin de préciser le paramètre no-ff.

git hist : correspond à l'alias hist que nous avons créé sur la commande git log dans le chapitre basics

```
git checkout -b simple-changes
git branch
npp humans.txt # ajouter du code
git add .
git commit -m "Changing humans.txt"
git checkout master
npp index.html # ajouter du code
git add .
git commit -m "Changing footer on index.html"
git hist
git merge simple-changes -m "Merging changes from simple-changes branch"
git branch
git branch -d simple-changes
git hist
```

Conflicting Merges and Resolution



Objectif : créer une situation de conflit entre branches et le résoudre au moment du merge.

- Le dernier commit de chaque branche modifie la même ligne dans un fichier

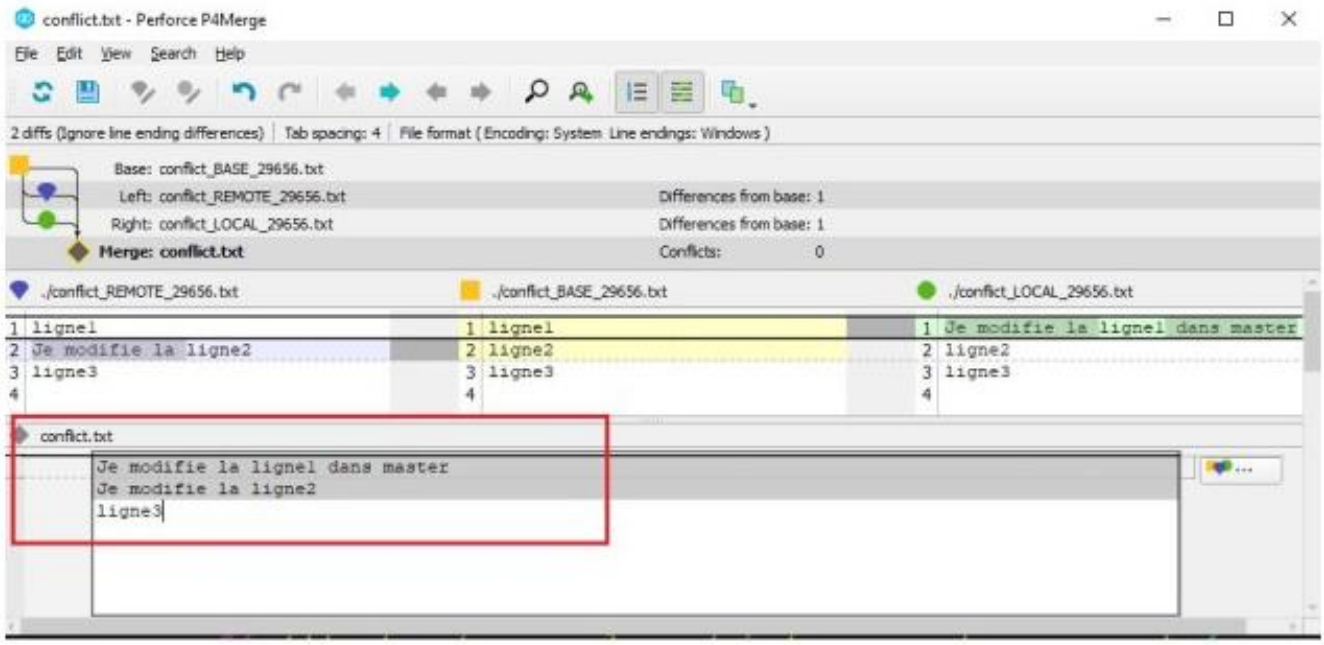
```
***** Etape 1 *****
git checkout master

npp conflict.txt
#ajouter les lignes suivantes:
ligne1
ligne2
ligne3

git add .
git commit -m "Creation de confict.txt"

```

```
***** Etape 2 *****
git checkout -b realwork
git branch

npp conflict.txt
# Changer la ligne 2 comme suit
Je modifie la ligne 2

git add .
git commit -m "Modification de la ligne 2"


```

```
***** Etape 3 *****
git checkout master

npp conflict.txt
# Remplacer la ligne 1 :
Je mofifie la ligne 1 dans master

git add .
git commit -m "Modification de conflict.txt dans master"

git hist

```

```
***** Etape 4  *****
***** Commencer le merge ****
git merge realwork -m "Merging changes from realwork branch"


git diff master realwork  # Constater les différences de manière texteuelle
git difftool master realwork # Constater les différences avec l'outil graphique

*** Resoudre le conflit ***
git mergetool
# cf. illustration ci-dessous
# Effectuer les modifications dans la partie basse de l'écran et enregistrer.
# local représente le contenu du fichier dans la baranche master.
# remote représente le contenu du fichier dans la branche realwork


***** Etape 5 *****
***** Finir le commit *****
git status
git add.
git commit -m "Resolution du conflit merge avec realwork"
git status

git branch
git branch -d realwork # Si vous souhaitez supprimer la branche realwork
git hist
```

illustration de git mergetool



Puisque la commande git merge realwork est exécutée depuis la branche master :

- LOCAL représente master
- REMOTE représente realwork
- BASE représente le fichier la dernière fois que les 2 branches étaient en accord

La branche master va "évoluer" pour intégrer le contenu de la branche realwork. La branche realwork reste intacte

Complément à effectuer si vous ne souhaitez pas que git suive les fichiers .orig

```
npp .gitignore # ajouter *.orig
git status
git add .gitignore
git commit -m "creating ignore file"
```

Cleanup and Push back to GitHub

```
git pull origin master
git push origin master

```