Rebasing

Dans Git, il y a deux façons d'intégrer les modifications d'une branche dans une autre : en fusionnant (merge) et en rebasant (rebase).

Avec la commande rebase, vous pouvez prendre toutes les modifications qui ont été validées sur une branche et les rejouer sur une autre.Cela fonctionne en cherchant l'ancêtre commun le plus récent des deux branches (celle sur laquelle vous vous trouvez et celle sur laquelle vous rebasez), en récupérant toutes les différences introduites par chaque commit de la branche courante, en les sauvant dans des fichiers temporaires, en réinitialisant la branche courante sur le même commit que la branche de destination et en appliquant finalement chaque modification dans le même ordre

Vous aurez souvent à faire cela pour vous assurer que vos commits s'appliquent proprement sur une branche distante — par exemple, sur un projet où vous souhaitez contribuer mais que vous ne maintenez pas. Dans ce cas, vous réaliseriez votre travail dans une branche puis vous rebaseriez votre travail sur origin/master quand vous êtes prêt à soumettre vos patchs au projet principal. De cette manière, le mainteneur n'a pas à réaliser de travail d'intégration — juste une avance rapide ou simplement une application propre.

Rebaser rejoue les modifications d'une ligne de commits sur une autre dans l'ordre d'apparition, alors que la fusion joint et fusionne les deux têtes. **L'historique des commits se retrouve ainsi linéaire** car tous les commits de fusion sont éliminés.

Bonnes pratiques :

Si le master doit être rebasé à partir de plusieurs branches, chaque rebase doit être effectué l'un après l'autre , branche par branche.

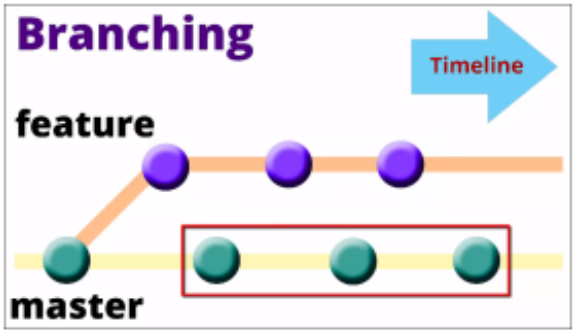
Il faut éviter de rebaser des commits qui ont déjà été poussés sur un dépôt public. : quand on rebase , les commits existants sont abandonnés et de nouveaux sont recréés. Si on les repoussent vers le dépôt public, les autres utilisateurs du dépôt devront re-fusionner . Et ainsi de suite lorsque chacun fera un git pull.

La conséquence est que dans le git log (historique), il y aura une multiplication des historiques de commit.

Pour se prémunir de ce problème, rebaser automatiquement lors du pull du projet avec l'option --rebase

git pull --rebase

Simple Rebase Example




```
git checkout master
git status
git checkout -b myfeaturerb # Crée la branche myfeaturerb à partir de master
npp hellorb.txt # ajouter du texte
git status
git add .
git commit -m "Creation de hellorb.txt"

git checkout master
git rebase myfeaturerb  # met à jour la branche master en rejouant tous les commits de master et ceux de myfeature dans l'ordre. Tous les commits de myfeaturerb se retrouvent également dans master de manière linéarisée
git log --oneline --all --graph

```

Setup for rebasing conflict

A l'instar de merge, il peut exister des situations de conflit entre branches avec rebase.

Il y a 2 manières de résoudre le conflit avec rebase :

- abort : annule les effets du dernier rebase

- effectuer une résolution de conflit

Objectif : créer une situation de conflit entre 2 branches

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

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

git add .
git commit -m "Creation de conflict-rb.txt"

***** Etape 2 *****
git checkout -b rbproblem # Creation et checkout vers la branche
git branch

npp conflict-rb.txt
# Changer la ligne 3 comme suit
Je modifie la ligne 3

git add .
git commit -m "Modification de la ligne 3 pour rb"


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

npp conflict-rb.txt
# Remplacer la ligne 3 :
Je mofifie la ligne 3 dans master

git add .
git commit -m "Generation du conflit : Modification de la ligne 3 de conflict-rb.txt dans master"

git status

git hist
```

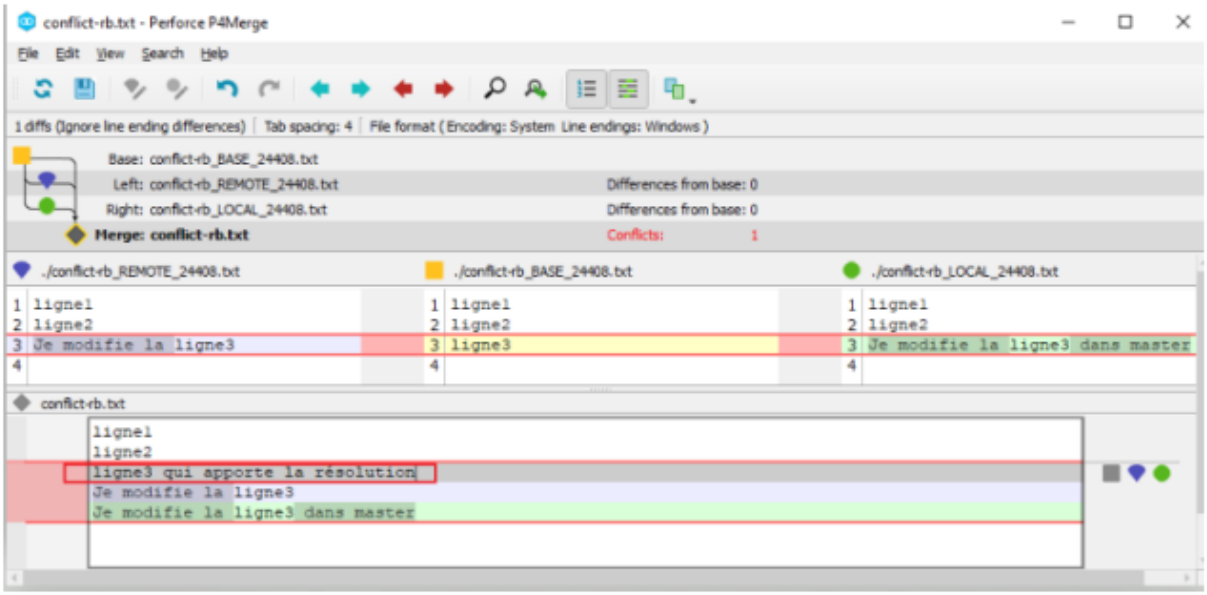
Abort a Rebase

Une fois la situation de conflit en place , on va tenter d'effectuer un rebase sur master

```
1 git branch
2 git checkout master
3 git difftool master rbproblem
4 git rebase rbproblem # Le conflit est détecté.
5 git status # permet de consulter les différentes options possibles;
6
7 # On choisit abort pour annuler les effets de la demande de rebase et revenir à
  l'etat initial
8 git rebase --abort
9
10
```

Rebase Conflict and Resolution

```
1 git checkout master
2 git rebase rbproblem # conflit
3
4 *** Resoudre le conflit ***
5 # Meme démarche que pour résoudre un conflit avec merge
6 git mergetool
7 # cf. illustration ci-dessous
8 # Effectuer les modifications dans la partie basse de l'écran et enregistrer.
9 # local représente le contenu du fichier dans la branche master.
10 # remote représente le contenu du fichier dans la branche rbproblem
11 # On va choisir d'appliquer la résolution des 2 cotés : Ligne3 qui apporte la
   résolution
12 # sauvegarder la résolution dans l'outil
13
14 ***** Finir le rebase *****
15 git status   # On voit que le rebase n'est pas encore terminé
16 git rebase --continue
17
18 git log --oneline --all --graph
19
20
```


conflict-rb.txt - Perforce P4Merge
File Edit View Search Help
1 diffs (Ignore line ending differences) | Tab spacing: 4 | File format (Encoding: System Line endings: Windows)
Base: conflict-rb_BASE_24408.txt
Left: conflict-rb_REMOTE_24408.txt
Differences from base: 0
Right: conflict-rb_LOCAL_24408.txt
Differences from base: 0
Merge: conflict-rb.txt
Conflicts: 1
./conflict-rb_REMOTE_24408.txt
./conflict-rb_BASE_24408.txt
./conflict-rb_LOCAL_24408.txt
1 ligne1
2 ligne2
3 Je modifie la ligne3
4
1 ligne1
2 ligne2
3 ligne3
4
1 ligne1
2 ligne2
3 Je modifie la ligne3 dans master
4
conflict-rb.txt
ligne1
ligne2
ligne3 qui apporte la résolution
Je modifie la ligne3
Je modifie la ligne3 dans master