

Formation Git

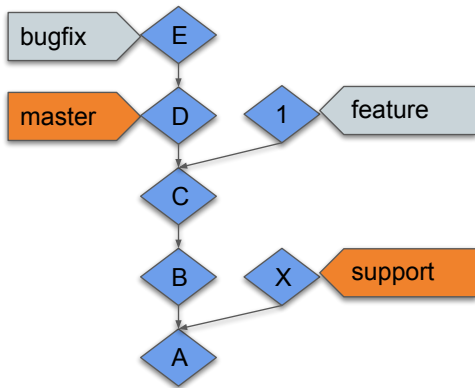
VII - Réécrire l'histoire



Arnaud MERCIER
arnaud.mercier@hexotech.fr



Historiques et branches



Branches temporaires

- branches de travail
- développement de fonctionnalités
- corrections de bug

Branches permanentes

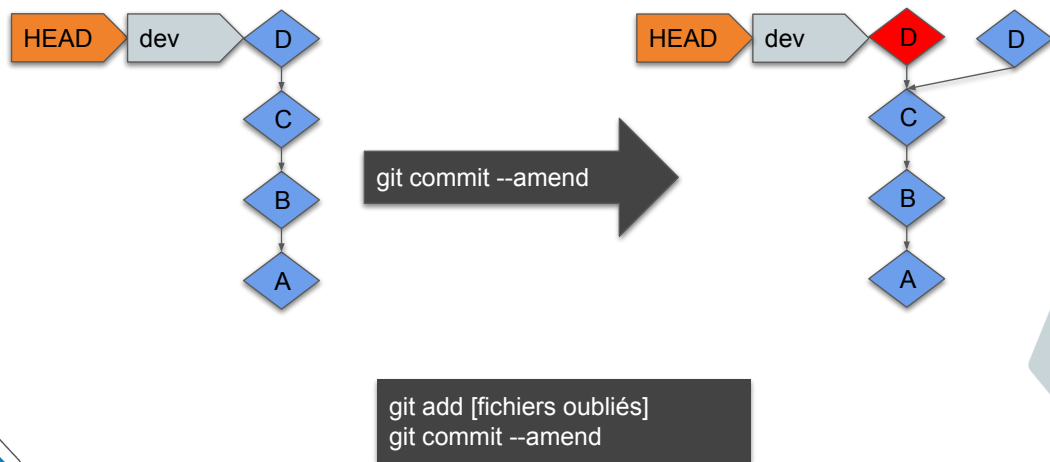
- contient les release
- branche principale
- branche de support de version

les branches permettent d'avoir plusieurs historiques en parallèles. Il existe 2 catégories de branche:

- Branches permanentes; utilisés porter les commits de release soit via la branche principale soit via des branches de support de version
- Branches temporaires; utilisés pour réaliser un développement ou une correction de bug. Cette branche est supprimé une fois mergée

La réécriture d'historique est à réserver de préférence pour les branches de travail

Modifier le dernier commit d'une branche

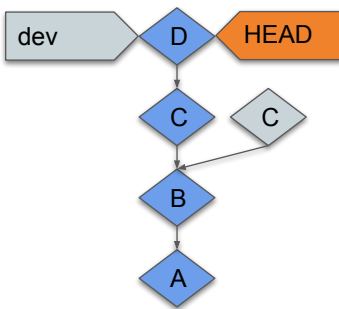


rappel sur l'option --amend de "git commit" qui permet de modifier le dernier commit réalisé (contenue et message)

Git ne supprime et modifie pas les commits.

Lors d'un amend par exemple, git vas créer un nouveau commit et faire pointer la branche sur ce dernier

Git reflog



git reflog -4

```
2154788 HEAD@{0}: checkout: moving from master to D
2154788 HEAD@{1}: commit: message commit D
d17c8d6 HEAD@{2}: commit (amend): message commit C
d17c8d6 HEAD@{3}: commit: C
15cc480 HEAD@{4}: checkout: moving from A to master
```

git reflog -[N] // Affiche l'historique du pointeur HEAD



Purge du reflog après 90 jours ou lors d'un git gc

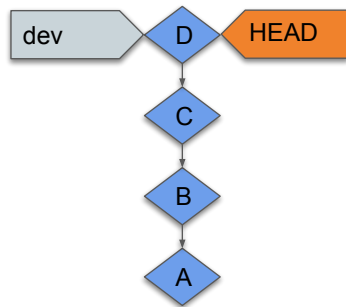
reflog est une commande méconnue mais qui peut s'avérer salvatrice. Elle permet tout simplement de retracer les différents états précédents de HEAD. C'est un peu comme un historique du pointeur HEAD

Attention: le contenu de reflog est purgé après 90 jours ou la commande "git gc"

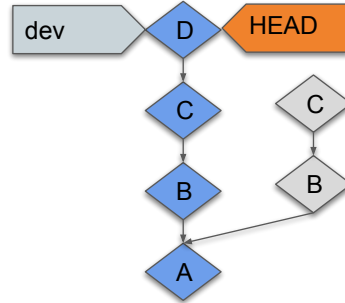
EXEMPLE (DEPOT):

- **git reflog** (retrouver les branches avant rebase)
- **git commit --amend**
- **git reflog** (retrouver le commit orphelin)

Voir les commits orphelins



`git log --graph`



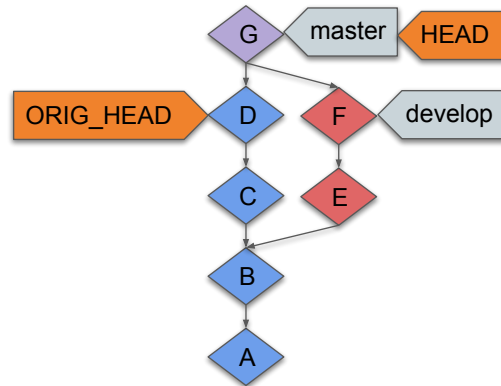
`git log --reflog --graph`

L'option `--reflog` peut être utilisé dans un `git log` ou `gitk` pour enrichir le graph avec les entrées du reflog

EXEMPLE (DEPOT):

- `gitk --all`
- `gitk --all --reflog`

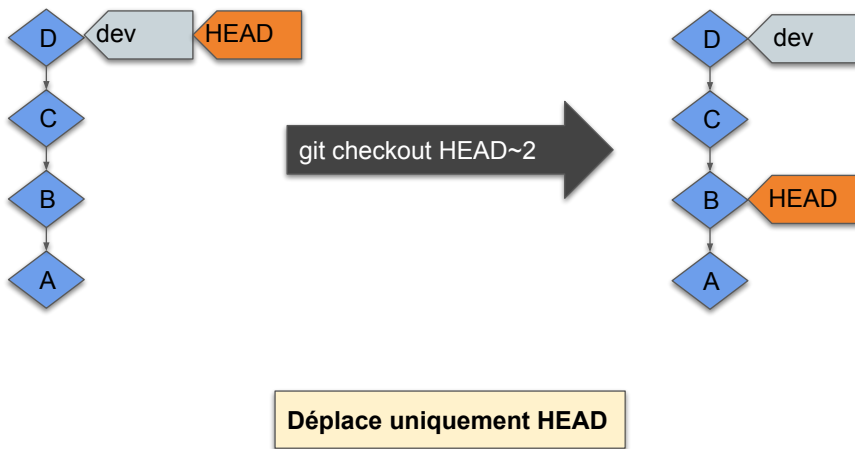
ORIG_HEAD



ORIG_HEAD est la position précédente de HEAD avant un **rebase, merge ou reset**

Le pointeur HEAD indique le commit sur lequel nous nous trouvons actuellement.
ORIG_HEAD est un pointeur qui correspond à la position précédente de HEAD avant un rebase, merge ou reset

Git checkout



Se déplacer dans l'historique d'un dépôt git correspond au fait de déplacer le pointeur HEAD sur le commit souhaité. Pour cela, on utilise la commande "git checkout sha1" ou "git checkout ref": exemple git checkout tag1

Il est également possible de donner une référence relative via:

- `ref^`: on remonte d'un parent par ^ a partir de ref
- `ref~N`: on remonte de N parent a partir de ref

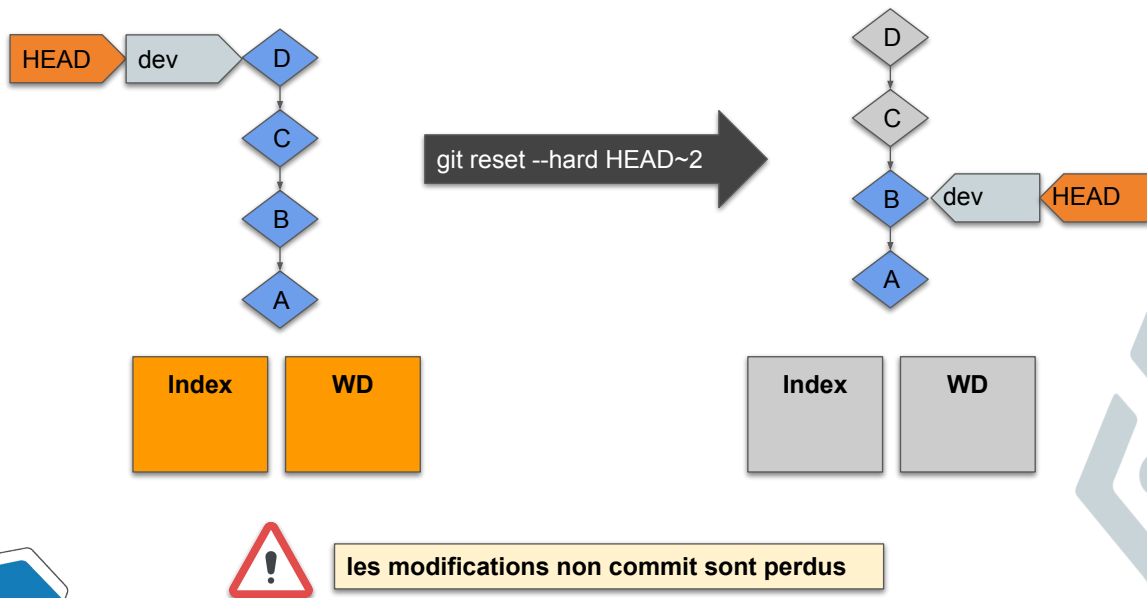
Git reset



`git reset [sha1/ref] [--soft, --mix, --hard] // Déplace HEAD et la branche courante`

la commande git reset déplace la référence de branche courante. il existe plusieurs options a cette commande. les commit ainsi laissés sans références sont dit orphelins.

Git reset HARD



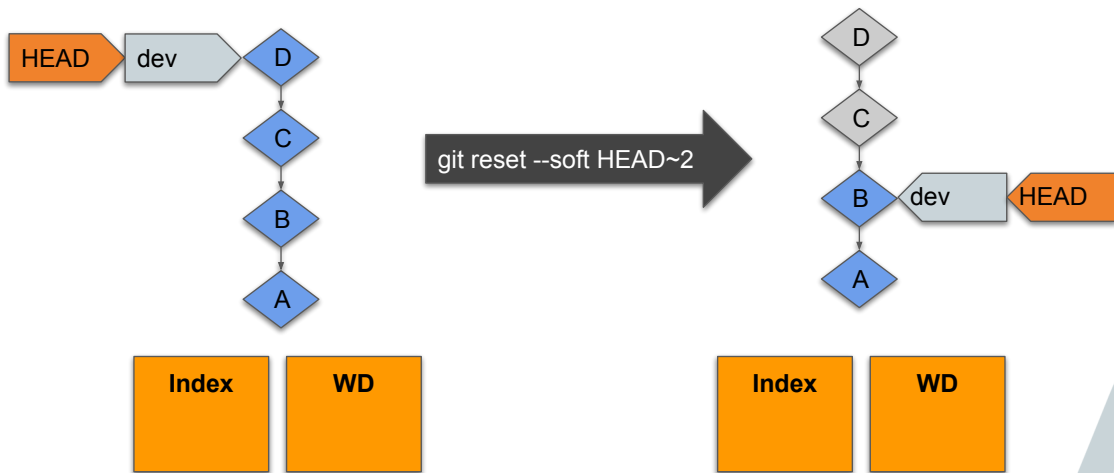
--hard: option la plus simple a comprendre mais aussi la plus risqué. Elle déplace comme toujours la référence de la branche courante et elle reset la zone d'index et le WD.

ATTENTION: les modifications non commits sont perdus

EXEMPLE (DEPOT):

- modification dans le fichier source.txt
- `git reset --hard HEAD^`
- `git status`
- `gitk ORIG_HEAD`
- `git reset --hard ORIG_HEAD`

Git reset SOFT

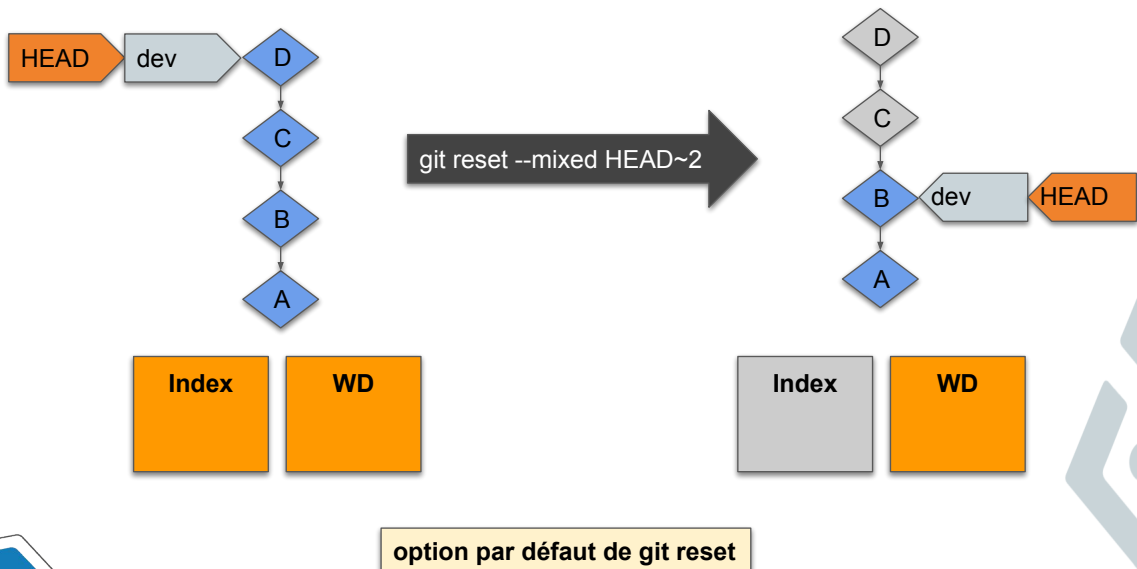


--soft: ne fait que déplacer la référence de la branche courante. l'index et le WD sont inchangés.

EXEMPLE (DEPOT):

- modification dans le fichier source.txt
- git add .
- git reset --soft HEAD^
- git status
- git diff --staged
- gitk ORIG_HEAD
- git reset --hard ORIG_HEAD

Git reset MIXED



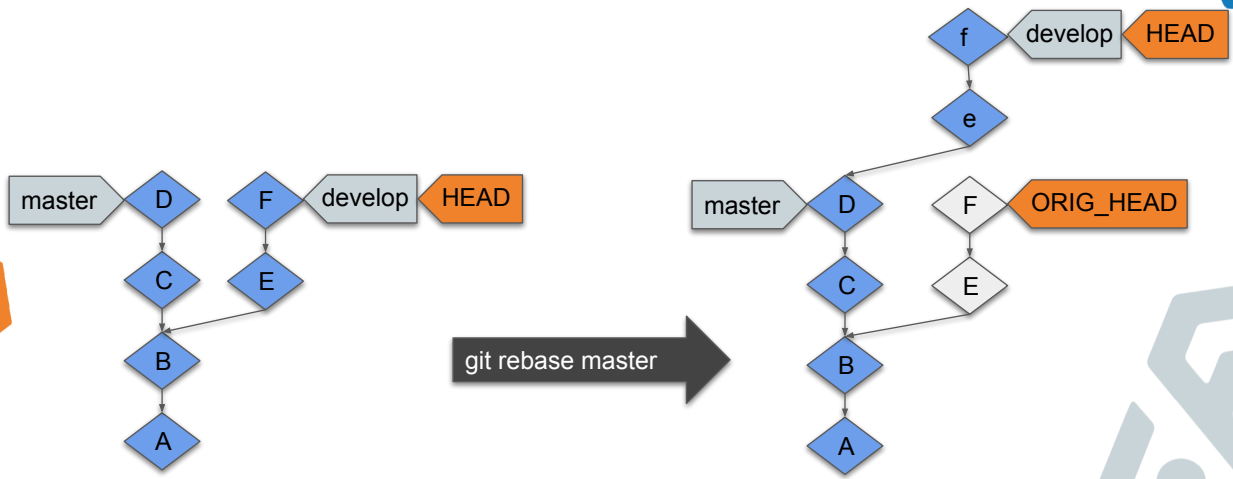
--mixed: Déplace la référence de la branche courante et reset la zone d'index. en revanche le WD reste intacte.

INFO: --mixed est l'option par défaut.

EXEMPLE (DEPOT):

- modification dans le fichier `source.txt`
- `git add .`
- `git reset HEAD^`
- `git status`
- `git diff`
- `gitk ORIG_HEAD`
- `git reset --hard ORIG_HEAD`

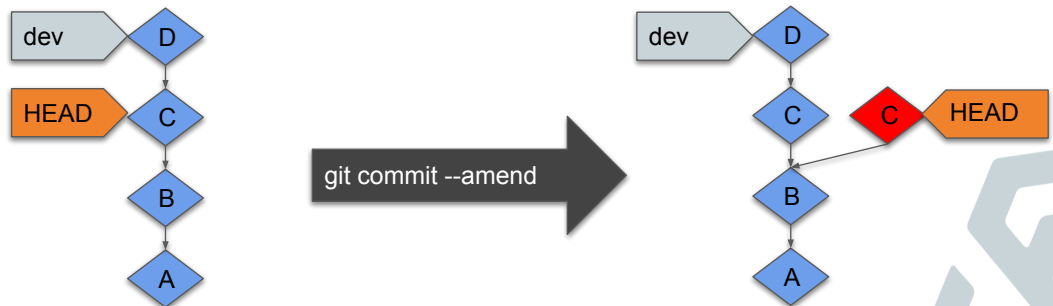
Git rebase



le rebase d'une branche consiste à déplacer le point de départ de la branche vers un autre commit:

- 1- se déplacer sur la branche à rebase: `git checkout [branche_a_rebase]`
- 2- utiliser la commande "`git rebase [branche_destination]`"

Modifier un commit plus ancien



Attention: les modifications sont toujours ajoutées à la branche courante. C'est pourquoi seul le dernier commit d'une branche peut être réécrit pour l'historique de la branche en question. Si on réécrit un commit plus ancien, cela fonctionnera mais le nouveau commit ne sera pas compris dans l'historique de la branche.

Dans ce cas, on pourrait s'en sortir via un cherry-pick du commit D puis un reset de master sur D

Rebase interactif

1 `git rebase -i [sha1 / ref]`

2 Editer le script du rebase

3 Exécution du script

```
git rebase -i HEAD~3
pick commit 7
pick commit 8
pick commit 9

# Commands:
# p, pick <commit> = use commit
# r, reword <commit> = use commit, but edit the commit message
# e, edit <commit> = use commit, but stop for amending
# s, squash <commit> = use commit, but meld into previous commit
# f, fixup <commit> = like "squash", but discard this commit's log message
# x, exec <command> = run command (the rest of the line) using shell
# d, drop <commit> = remove commit
# l, label <label> = label current HEAD with a name
# t, reset <label> = reset HEAD to a label
# m, merge [-C <commit> | -c <commit>] <label> [# <oneline>]
# . create a merge commit using the original merge commit's
# . message (or the oneline, if no original merge commit was
# . specified). Use -c <commit> to reword the commit message.
```



Ordre d'affichage du rebase interactif est l'inverse du git log

le rebase interactif est un outil incroyablement puissant mais malheureusement très peu utilisé. Il permet de complètement réécrire votre historique (modification de commit, réorganisation, suppression, subdivision, ...).

pour faire un rebase interactif:

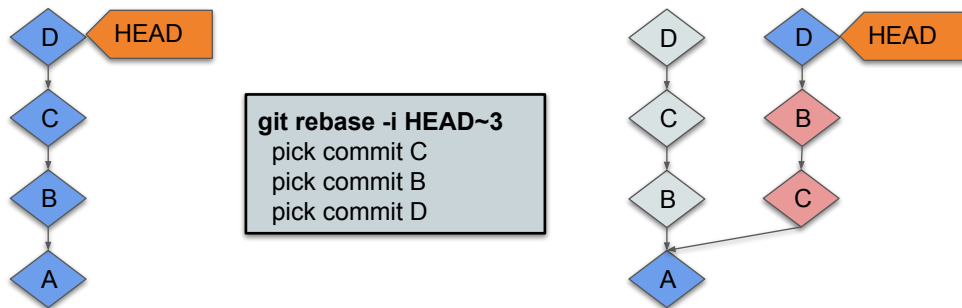
1- utiliser la commande `git rebase -i [sha1 / ref]`

2- un éditeur s'ouvre alors. il vous liste les commit mais dans l'ordre inverse de la commande "git log", cad en haut le commit le plus ancien et en bas le plus récent. Il faut alors éditer les différentes lignes pour indiquer ce que vous souhaitez faire avec chaque commit.

3- Une fois enregistré et quitté, le fichier sera utilisé par git pour le rebase.

Info: il est possible d'abandonner le rebase en cours et revenir dans l'état dans lequel on était avant de lancer la commande "git rebase -i": pour cela il faut utiliser "git rebase --abort"

Rebase interactif: Réordonner



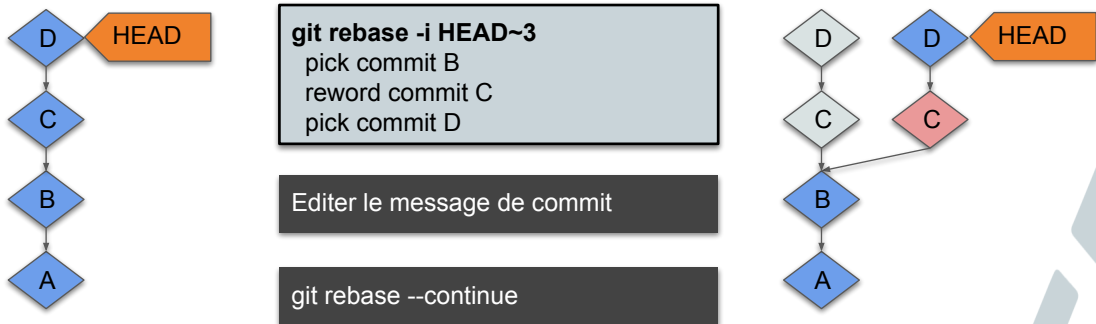
Pour réordonner les commits:

- 1- utiliser la commande "git rebase -i HEAD~N" et réordonner les lignes tout simplement
- 2- utilisation de la commande "git rebase --continue" pour terminer le rebase

EXEMPLE (DEPOT):

- gitk
- git rebase -i HEAD~3
- gitk HEAD ORIG_HEAD
- git reset --hard ORIG_HEAD

Rebase interactif: Changer le message



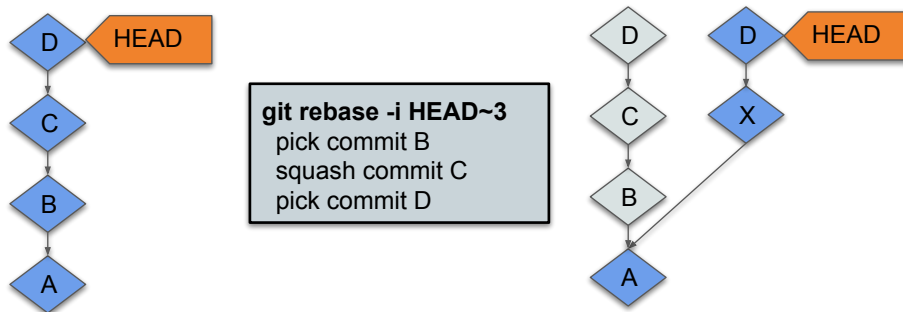
Pour éditer le dernier commit, on utilise “git commit --amend” mais pour éditer un commit plus ancien, il faut utiliser le rebase interactif:

- 1- utiliser la commande “git rebase -i HEAD~N”, il faut alors indiquer “edit” devant les commits a modifier
- 2- modification du commit suivi du “git commit --amend”
- 3- utilisation de la commande “git rebase --continue” pour terminer le rebase

EXEMPLE (DEPOT):

- `gitk`
- `git rebase -i HEAD~3`
- `gitk HEAD ORIG_HEAD`
- `git reset --hard ORIG_HEAD`

Rebase interactif: Fusionner



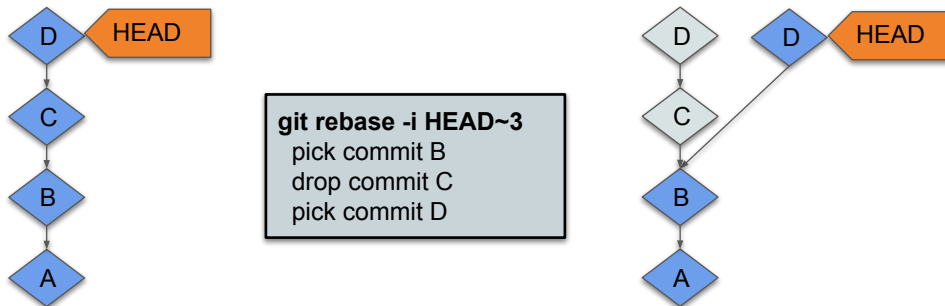
Pour fusionner des commits:

- 1- utiliser la commande "git rebase -i HEAD~N" et indiquer "squash" devant les commits a fusionner avec leur prédécesseur.
- 2- utilisation de la commande "git rebase --continue" pour terminer le rebase

EXEMPLE (DEPOT):

- gitk
- git rebase -i HEAD~3
- gitk HEAD ORIG_HEAD
- git reset --hard ORIG_HEAD

Rebase interactif: Supprimer



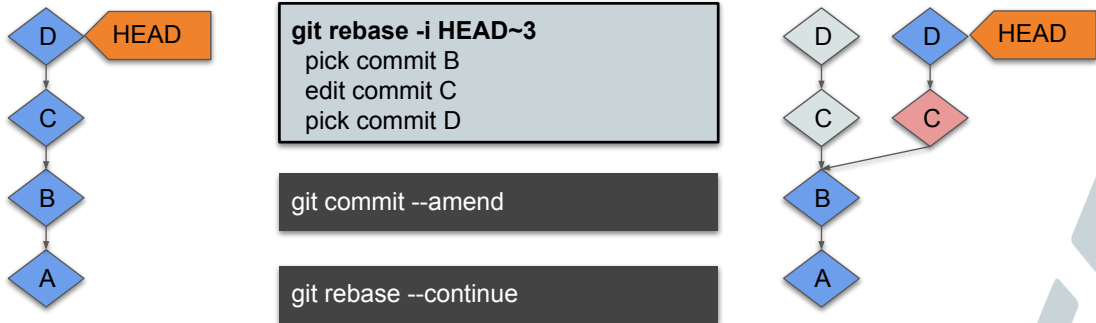
Pour supprimer des commits:

- 1- utiliser la commande “git rebase -i HEAD~N” et indiquer “drop” devant les commits a supprimer.
- 2- utilisation de la commande “git rebase --continue” pour terminer le rebase

EXEMPLE (DEPOT):

- gitk
- git rebase -i HEAD~3
- gitk HEAD ORIG_HEAD
- git reset --hard ORIG_HEAD

Rebase interactif: Editer



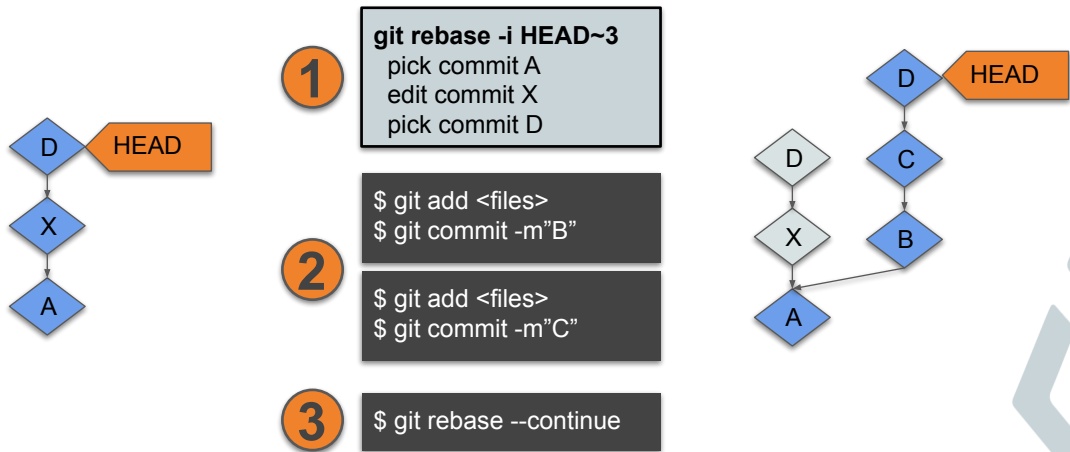
Pour éditer le dernier commit, on utilise "git commit --amend" mais pour éditer un commit plus ancien, il faut utiliser le rebase interactif:

- 1- utiliser la commande "git rebase -i HEAD~N", il faut alors indiquer "edit" devant les commits a modifier
- 2- modification du commit suivi du "git commit --amend"
- 3- utilisation de la commande "git rebase --continue" pour terminer le rebase

EXEMPLE (DEPOT):

- `gitk`
- `git rebase -i HEAD~3`
- `gitk HEAD ORIG_HEAD`
- `git reset --hard ORIG_HEAD`

Rebase interactif: Découper



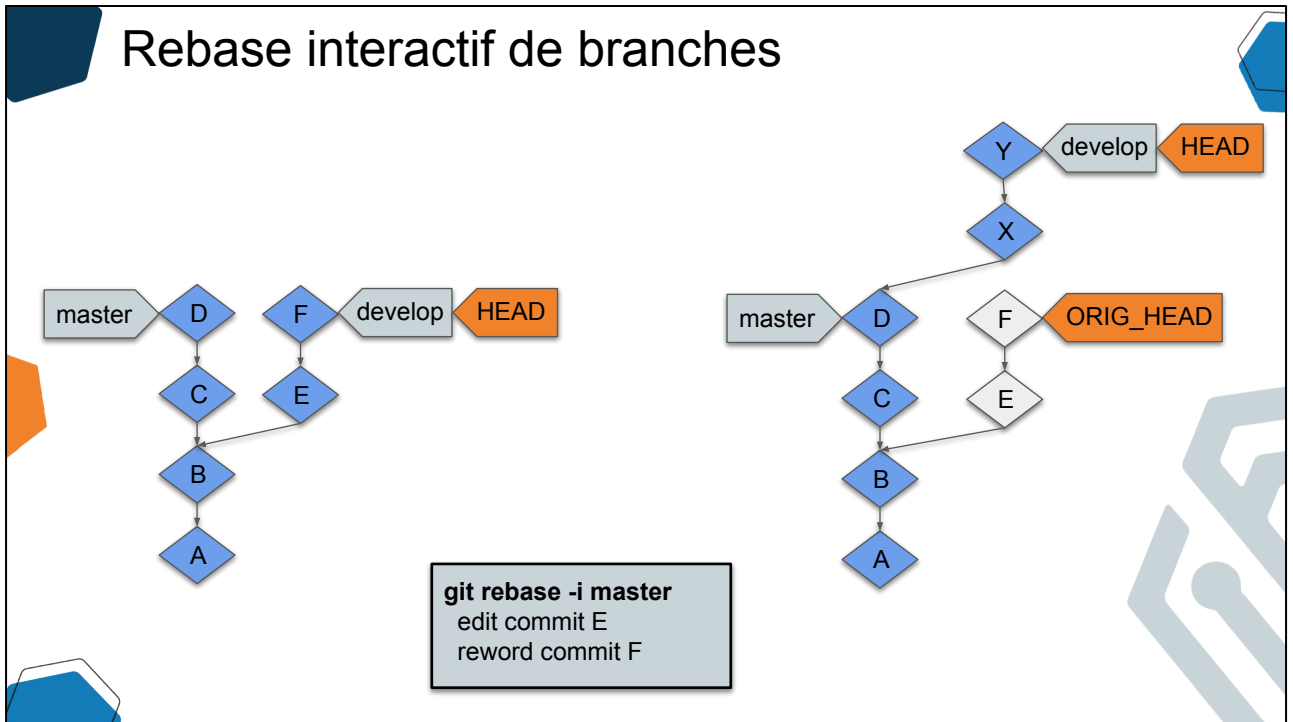
Pour découper un commits en plusieurs commits:

- 1- utiliser la commande "git rebase -i HEAD~N" et indiquer edit devant les commits a découper
- 2- réaliser autant de commit que vous le souhaitez
- 3- utilisation de la commande "git rebase --continue" pour terminer le rebase

EXEMPLE (DEPOT):

- gitk
 - git rebase -i HEAD~3
 - gitk HEAD ORIG_HEAD
- git reset --hard ORIG_HEAD

Rebase interactif de branches



Le rebase interactif, comme un rebase classique, peut se faire d'une branche sur une autre. A ce moment là, la liste des commits pris dans le rebase correspond au commits spécifiques à la branche rebase

Il est bien sûr possible de réaliser plusieurs actions différentes dans un rebase interactif. En revanche, une seule action possible par commits. Dans le cas où on souhaite faire plusieurs actions sur un commit, il faut utiliser l'action la plus englobante.

TP 7.1

Scénario

Votre collègue a tenté de de revenir à l'état de la version SITE_V02 du site. Tout semble bon car il a bien la version souhaité (il n'y a que la médaille de bronze sur la page du site). Mais lorsqu'il fait un "git checkout master", rien de ne passe et il reste sur le commit pointé par SITE_V02 (les médailles d'argent et or ne sont pas sur la page du site)

Objectifs

1- Trouvez la source du problème et régler la situation

SOLUTION:

Quand vous aidez une personne, pensez à ces commandes pour avoir du contexte fiable:

- history
- git status
- git log --oneline
- git reflog

Ici on peut voir qu'il a fait comme dernière action un "git reset" sur SITE_V02 de la branche master. Si on affiche le log depuis ORIG_HEAD, on retrouve bien le commit précédent de master. Un simple "git reset --hard ORIG_HEAD" permet de résoudre le problème de référence. En revanches les modifications non commits sont perdus.

TP 7.2

Scénario

Aidez Arya dans la gestion de sa liste. Elle souhaite revoir son historique pour le modifier légèrement. Elle n'a pas beaucoup de temps, il faut alors faire ces retouches en une seule fois.

Objectifs

- 1- Fusionnez les commits qui ajoutent le limier et la montagne en un seul. Pour ce commit, indiquez le message "ajout des frères Clegane"
- 2- Modifiez le message du dernier commit "Revert 'ajout du limier' " pour "retirer le limier"

SOLUTIONS:

- `git log --oneline`
- `git rebase -i HEAD~3`
- éditer le fichier
 - `pick 3dafd7e` ajout de la montagne
 - `squash f8bb732` ajout du limier
 - `reword 19c61e8` Revert "ajout du limier"
- enregistrer et fermer le fichier
- éditer les messages pour terminer le rebase
- `gitk HEAD ORIG_HEAD`

TP 7.3

Scénario

Aidez Arya dans la gestion de sa liste. Elle souhaite rebase et merge sa branche NORD dans la branche master pour prendre en compte les mord au delà du mur. mais elle souhaite en plus diviser le commit de la branche NORD en 2 sous commits

- 1- mort de Ben
- 2- mort du lord commander

Objectifs

- 1- Faire le rebase en mode interactif pour minimiser le nombre d'actions
- 2- merge NORD dans master

SOLUTIONS:

- **git log --oneline --all**
- **git switch NORD**
- **git rebase -i master**
- **editer le fichier**
 - `edit 944188b lord commander dead and oncle ben dead?`
- **enregistrer et fermer le fichier**
- **corriger les conflits dans done.txt**
- **git add done.txt**
- **git restore --staged --patch ou git gui (désindexer lord commander)**
- **git commit -m "mort de Ben"**
- **git add .**
- **git commit -m "mort du lord commander"**
- **git rebase --continue**
- **gitk HEAD ORIG_HEAD**
- **git switch master**
- **git merge NORD**

Bilan

- 1 `git rebase -i [sha1 / ref]`
- 2 Editer le script du rebase
- 3 Exécution du script

rebase interactif

```
# Commands:
# p, pick <commit> = use commit
# r, reword <commit> = use commit, but edit the commit message
# e, edit <commit> = use commit, but stop for amending
# s, squash <commit> = use commit, but meld into previous commit
# f, fixup <commit> = like "squash", but discard this commit's log message
# x, exec <command> = run command (the rest of the line) using shell
# d, drop <commit> = remove commit
# l, label <label> = label current HEAD with a name
# t, reset <label> = reset HEAD to a label
# m, merge [-C <commit> | -c <commit>] <label> [# <oneline>]
# . create a merge commit using the original merge commit's
# . message (or the oneline, if no original merge commit was
# . specified). Use -c <commit> to reword the commit message.
```

git reset

Mode name	HEAD position (position of the HEAD)	Index	Work tree
soft	change	unchanged	unchanged
mixed	change	change	unchanged
hard	change	change	change

Nous avons vu dans ce chapitre différentes méthodes pour réécrire un historique de branche:

- `git commit --amend`: dernier commit de la branche
- `git reset`: déplace la branche courante sur le commit souhaité
- `git rebase [-i]`: permet de rebase une branche tout en modifiant l'historique rebase