Clé SSH – relier mon ordinateur à mon compte git hub dans un objectif de sécurisation – mais honnêtement j’ai pas tout compris

Exercices <https://learngitbranching.js.org/?locale=fr_FR>

Git commit : je sauvegarde mes modifications et j’ajoute un commentaire. Ca me permet de me repérer dans mon dev et de travailler en équipe

Git branch : je crée une nouvelle branche, en quelque sorte un nouveau doc pour coder qui me permet d’avancer dans mon code sans toucher à la branche principale. Soit pour découper mon travail en étapes claires, soit pour travailler en équipe autour d’un doc principal : le main

Git merge : pour fusionner une branch et un main ou fusionner plusieurs branches et un main

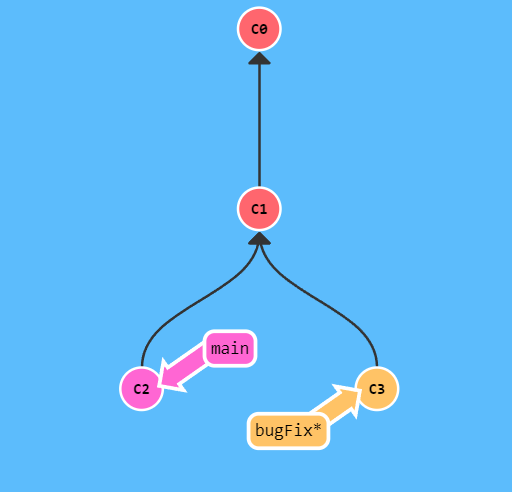
Git checkout : pur savoir où je me trouve dans mes commits

Git rebase : me permet de combiner des contenus. Rebase reprend un ensemble de commits, les recopie et les colle ailleurs, en bout de chaîne. Rebase a l’avantage de conserver de manière plus claire et linéaire toutes les modifications déjà apportées. Si je crée une branche bugFix et que je veux la fusionner avec mon doc de travail principal main. Je peux écrire

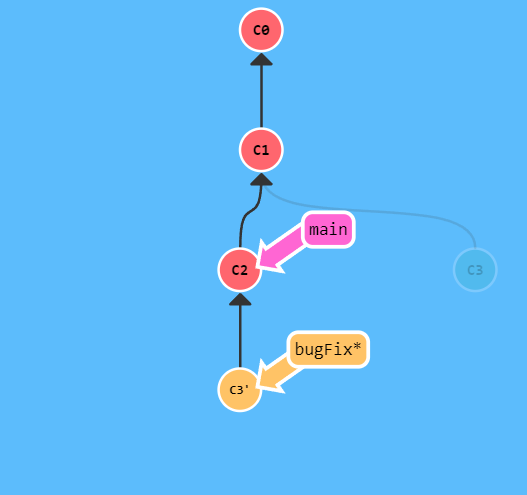
Git rebase main

bugFix vient alors s’ajouter à la suite de main.

Voici ce que ça donne en deux étapes



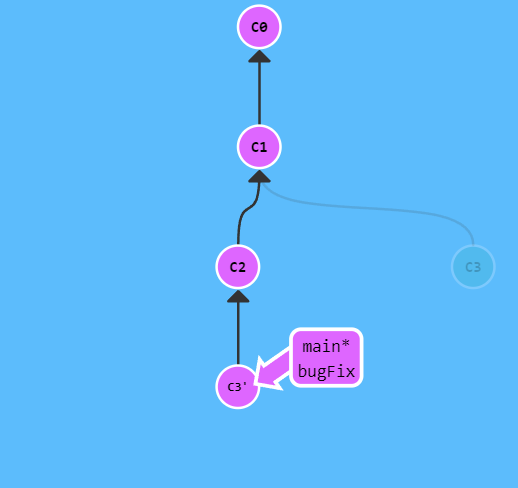
A noter que nous n’allons pas supprimer totalement la branche C3 de bugFix, elle existera toujours quelque part.



A ce stade en revanche main n’a pas été mis à jour.

Positionnons-nous sur la branche main.

Git rebase bugFix



C’est tout bon, j’ai tout mergé et tout mis au bon endroit

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

HEAD : c’est un point qui indique où nous nous trouvons et c’est important puisque c’est là que nous agissons.

Nous pouvons détacher HEAD pour l’attacher à un commit au lieu d’une branche.

En mode débutant, pour bouger HEAD je peux utiliser la commande checkout qui me fait me positionner quelque part. Je peux utiliser checkout avec le nom d’une branche ou l’identifiant d’un commit.

Git log pour lister tous les commits qui ont été faits – journal d’activité –

Git log --oneline en option pour avoir ça en plus succinct

Pour quitter le log touche Q

Se déplacer dans les commits. On peut utiliser

^ pour revenir d’un commit en arrière. Et ^^^ pour revenir de 3 commits en arrière

Demander main^ revient à dire « je veux aller sur le premier parent de main »

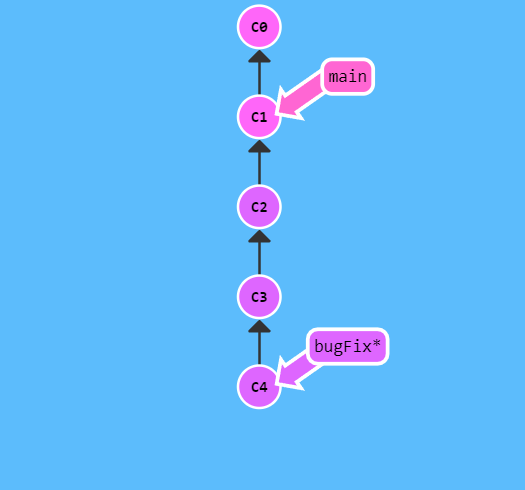
~1 pour revenir d’un commit en arrière et ~3 pour revenir de 3 commits en arrière

Demander HEAD~4 revient à demander « je veux aller 4 parents avant HEAD »

* Il peut être tout aussi intéressant d’utiliser checkout pour positionner HEAD où on le souhaite

Réassigner les branches à un commit avec -f : le forçage

Ainsi la commande git branch -f main HEAD~3 fait bouger de force la branche main à 3 parents avant HEAD 🡪 résultat ci-dessous



Git revert et git reset : annuler des changements avec Git

git reset annule des changements en déplaçant la référence en arrière dans le temps sur un commit plus ancien. En ce sens, on peut considérer cela comme une façon de "réécrire l'histoire"; git reset fait remonter une branche en arrière comme si le(s) commit(s) n'avai(en)t jamais eu lieu.

Bien que le reset marche parfaitement pour les branches locales sur notre propre machine, cette façon de "réécrire l'histoire" ne marche pas avec les banches distantes (remote) que d'autres personnes utilisent.

Pour pouvoir annuler des changements et partager ces annulations avec d'autres, nous devons utiliser git revert. Regardons comment cela fonctionne.