Parcourir et ré-écrire l'history de Git

Pouvoir parcourir et agir sur l'index de Git est extrêment intéressant dans 2 cas:

- la résolution d'un bug;
- nettoyer les messages de commit;

> Ce TP sera volontairement peu dirigé, il est obligatoire de rechercher quelques réponses en autonomie pour le terminer.

Récupération du projet

- Récupérer le dépot "https://github.com/mickaelandrieu/play-with-git-history" sur votre machine.

Découverte de l'historique et du projet

- Consulter l'historique du projet à l'aide de la commande git log: quel curieux historique, n'est-ce pas?
- Notez le Hash/SHA-1 du commit correspondant à l'ajout du fichier "file7.md".
- A l'aide de la commande git checkout, revenez à ce commit: il ne devrait y avoir que 7 fichiers disponibles.
- Revenez à l'état actuel du projet. Indice: comment reviens-t-on sur une branche git?

Altérons l'historique maintenant

- Se mettre sur la branche "test1" que l'on aura précedemment créée.

- A l'aide de la commande git log identifier le SHA-1 du tout

premier commit d'ajout: c'est celui du fichier "file1.md".

- Exécuter la commande git rebase -i <le-hash> et lire ce qui suit.

La théorie

Lors d'un rebase interactif, on peut intervenir sur tous les commits suivants le commit que l'on a identifié jusqu'au commit actuel. Il y a alors 6 possibilités:

- pick ou p: on laisse tel quel, le commit reste inchangé;
- reword ou r: on peut ré-éditer le message du commit;
- squash ou s: rassemble le commit avec le commit précédent: les 2 messages de commits sont préservés;
- fixup ou f: pareil que squash mais cette fois le message du commit "fixupé" n'est pas conservé;
- delete ou d: supprime le commit de l'index et donc de l'historique: les modifications sont perdues. Il est possible également de supprimer la ligne pour parvenir au même résultat;
- exec ou e: execute une commande à définir sur le commit en question. (Commande particulière)

La pratique

- Supprimer le commit nommé "adding file 14".
- Assembler les commits de "adding file 1" à "adding file 10"
- Renommer le commit qui en résulte "adding file 1 to 10".

> Vous pouvez faire un rebase interactif autant de fois que nécessaire

A quoi peut bien servir "git rebase interactif exec"?

On peut imaginer que l'on recherche dans l'historique quel commit a introduit le bug dans un projet. Une façon efficace de faire est de créer

un petit script qui teste si le bug est présent dans le projet, imaginons que ce script est un fichier bash qui s'appelle "bug-is-here.sh" et qu'il retourne "OK" ou "BUG" si le bug est présent.

On peut alors en revenant suffisamment loin dans l'historique trouver le commit coupable à l'aide de la commande git rebase --interactive --exec "bug-is-here.bash" <specific-sha1>: utile non?

C'est pas moi!

Cherchez dans la documentation de Git comment changer l'auteur d'un commit et faites-vous passer pour l'auteur du commit "adding file 11"!