Consultation de l'historique et des modifcations

Nous allons repartir du dépôt suivant pour la suite du cours (pizza.zip) : Pizza

Dans le dossier Pizza identifiez-vous, il y a déjà un dossier .git/ et un auteur : Antoine07. Tapez dans le dossier Pizza les lignes de code suivantes :

```
# Mettez votre pseudo
git config --local user.name Tony
git config --local user.email tony@tony.fr
```

git log

Nous vous avons déjà présenté dans la première partie les commandes pour consulter l'historique : git log. Nous allons revoir quelques commandes pratiques.

Astuce si vous souhaitez afficher les logs et sortir de la commande (retrouver le prompt de votre terminale), tapez la ligne de code suivante :

```
# sans pagination pas de console bloque
git --no-pager log --oneline

# 3 commits seulement
git --no-pager log -3 --oneline
```

Affiche les 3 derniers commits et sort de la commande.

Placez vous dans votre projet, nous allons consulter les deux derniers commit, n'oubliez pas pour sortir de l'historique il faut taper la lettre "q" de votre clavier.

```
git log --oneline -2
```

Vous pouvez également consulter les commits d'un auteur en particulier :

```
git log --author "Antoine07"
```

Mais vous pouvez consulter l'historique à l'aide des commandes suivantes également :

```
git log --since=1.day
git log --before="2019-09-01"
```

Vous pouvez également consulter l'historique d'un fichier uniquement :

```
git log calzone.txt
```

Vous pouvez également visualiser les modifications au sein d'un fichier ou de l'ensemble/d'un ensemble des fichier(s) :

```
git log -p calzone.txt

# Dans l'ensemble des commits
```

```
git log -p

# Ou pour les 2 derniers commits
git log -2 -p
```

01 Exercice d'application

Dans le dossier Pizza modifiez la ligne "Coulis de tomate" comme suit :

```
Pizza : Calzone
Coulis de tomate  -->[modifiez en]--> Coulis de tomates cerises
mozzarella
jambon supérieur
champignons de Paris frais
oeuf
```

Puis faite un commit et visualisez les changements à l'aide de la commande git log suivante :

```
git log -p calzone.txt
```

Vous devriez voir maintenant des + et - qui correspondent respectivement à ajouté et retiré des/une ligne(s) :

```
Pizza : Calzone
-Coulis de tomate
+Coulis de tomates cerises
 mozzarella
 jambon supérieur
 champignons de Paris frais
```

Une autre commande Git permet également de visualiser les statistiques des modifications des fichiers, pratique pour savoir combien de ligne de code ont été modifiées ou ajoutées sur un fichier ou projet :

```
git log --stat calzone.txt
```

Rechercher dans les logs

La commande suivante permet de rechercher des logs (messages des commits) : E expression pour rechercher les/l'occurence(s) et i pour insensible à la casse (majuscule/minuscule) :

```
git log -E -i --grep='Pizza'
```

02 Exercice recherche dans les logs

Recherchez les logs dans lequel le mot “Pizza” apparaît.

git blame

Il parfoit utile de savoir qui a fait une/des modification(s) lorsqu'on travaille à plusieurs sur un même projet. On peut alors la contacter afin qu'elle explique ou corrige la/les modification(s) dans l'application :

```
git blame calzone.tx

# b058c4f1 (Antoine07 2019-10-06 09:14:14 +0200 1) Pizza : Calzone
# b058c4f1 (Antoine07 2019-10-06 09:14:14 +0200 2) Coulis de tomates cerises
# ...
```

git diff (introduction)

Vous pouvez lorsque vous faite une modification dans un fichier avant de l'indexé visualiser les modifications à l'aide de la commande suivante :

```
git diff
```

On peut ainsi se demander si on doit ou non l'indexer.

03 Exercice d'application sur les différences dans un fichier

Modifiez sans indexé le fichier calzone.txt en ajoutant la ligne suivante et tapez la commande git diff dans le terminal :

```
supplément : gruyère + 5 €
```

La commande ci-dessous affichera les modifications dans le WD & l'index

```
git diff
```

La commande ci-dessous affichera les différences entre le dernier commit et l'index :

```
git diff --cached
```

Ajoutez dans l'index les modifications et retapez la commande ci-dessus vous verrez maintenant les modifications :

```
git add calzone.txt
# git diff n'affiche plus rien
```

```
# par contre avec l'option --cached on affiche les diff dans l'index
git diff --cached
```

Voir modification après avoir commité

Vous pouvez voir les différence après avoir commité les changements, supposons que l'on veuille voir les modifications un commit en arrière :

```
git diff HEAD~1
```

HEADˆ1 pour reculé d'un commit en arrière.

Visualiser entre deux positions du HEAD

Si on souhaite visualiser les différences entre le HEAD et un état antérieur du dépôt, par exemple 3 commits avant vous taperez :

```
git diff HEAD~3 HEAD
```

Visualiser les modifications entre deux commits donnés

Mais vous pouvez également voir les modifications entre deux commits :

```
# Récupérez les haches des deux derniers commits
git log --oneline

# Et les logs entre deux commits
git log f5541d6 b058c4f
```

diff stat

Dernière commande : faire des stat sur les différences opérées dans les fichiers:

```
git diff --stat
# calzone.txt | 2 +-
```

git restore (nouvelle commande version 2.23)

Annuler les modifications dans l'espace de travail (WD : working directory)

Modifiez le fichier calzone.txt comme suit, on met 300 euros pour le supplément gruyère quelque chose d'absurde (. . .)

```
Pizza : Calzone
Coulis de tomates cerises
mozzarella
jambon supérieur
champignons de Paris frais
oeuf
supplément : gruyère + 300 €
```

Pour remettre le dépôt dans l'état dans lequel il se trouvait juste avant cette modification absurde il suffit de tapez :

```
git restore calzone.txt
```

Par contre si on refait la modification et que l'on ajoute le fichier dans l'index il faudra alors taper :

```
git restore --staged
```

04 Exercice restore

- 1/ Créez le fichier **super_pizz.txt** avec le contenu suivant sans l'ajouter dans l'index :

```
Pizza : Super pizz
Coulis de tomates cerises & tomates russes
jambon de Parme
champignons de Paris
oeuf
supplément : gruyère + 4.5 €
```

- 2/ Re-modifiez le fichier calzone.txt en mettant l'option gruyère à 300 euros.

Le dépôt se retrouve clairement dans l'état suivant :

- Un fichier modifier & un fichier non suivi.

Ajoutez ces 2 fichiers dans l'index. Vous vous apercevez alors que vous avez fait une erreur : les 300 euros ne correspondent pas à la réalité (...) du prix d'un supplément pour une pizza !

- Annulez les modifications dans le fichier calzone.txt
- Commitez pour la création du fichier super_pizz.txt

Modifier un message de commit

- On peut modifier un message de commit dans l'historique à deux conditions :

1/ Qu'il n'y a pas eu d'autre commit entre temps de fait.

2/ Qu'on a pas "publié" le commit en question sur une branche distante (serveur distant).

05 Exercice ajouter un category

Ajoutez un fichier category.txt dans notre dossier Pizza, il listera l'ensemble de nos pizzas par catégorie, mais nous allons oublié volontairement les noms des pizzas vegans :

```
Pizzas : liste
- Itialia
    - calzone
    - bellachao
- Specials
    - grill
    - merguez
- Vegans
```

Faites un nouveau commit avec le message suivant "Ajout du fichier categorie: liste des pizzas/catégorie."

- Corrigez maintenant le petit oubli ?

Puis modifiez le fichier **category.txt** pour ajouter la liste complète des pizzas avec les pizzas vegans. Nous aimerions modifier maintenant le dernier message de commit sans recréer un nouveau message de commit, voici les commandes à réaliser :

```
# Avant toute chose
git status

# vérifiez les deux derniers commits avant
git log -2

git commit -m "liste complète de toutes les pizzas" --amend

# vérifiez les deux derniers commits après
git log -2
```

Notez bien qu'il n'y a pas eu de nouveau commit de créé !

Pour terminez considérer la commande suivante toute seule :

```
$ git commit "un message" --amend --no-edit
```

Elle permet de changer uniquement le message du dernier commit.

git reset HEAD~1

Nous parlerons ici que du cas où nous reculons d'un commit dans l'historique.

Parfois vous souhaiterez annuler des actions ou même les supprimer après avoir fait un commit.

Voici quelques commandes qui vous permettrons d'annuler des commits sans perdre votre travail :

```
# Annule le dernier commit et met tout dans le WD sans perte (revient au commit précédent -
$ git reset HEAD~1

# Annule le dernier commit et met tout dans la staging area sans perte
$ git reset --soft HEAD~1
```

06 Exercice key_private & reset

Créez un nouveau dépôt **example-reset_soft**, les fichiers créés ci-dessous ne nécessitent pas de contenu, ils seront utilisés uniquement pour exemple :

- Initialisez un nouveau dépôt.
- Dans ce dépôt créez un fichier file1.txt (commit).
- Créez maintenant un fichier .gitignore ne mettez rien pour l'instant dans ce fichier (commit).
- Créez un autre fichier file2.txt (commit).
- Créez un fichier key_secret.sh et mettez des fausses clés secrètes dans ce fichier (commit).

Vous vous apercevez maintenant que vous avez commité des clés secrètes.

- 1/ Corrigez le problème.
- 2/ Excluez le fichier key_secret.sh de l'historique de Git.

git reset [commit]

Vous pouvez annuler plusieurs commits en revenant en arrière dans l'historique, **mais dans ce cas attention à la cohérence de votre historique. . . .**

Supposons que nous ayons l'historique suivant et que la copie de travail est propre :

```
124f561 (HEAD -> master) file33
6695308 file22
0637561 file11
8ef0ef2 ignore
0a70edb file2
bf3f73c first commit
```

Alors si vous tapez la commande suivante :

```
git reset 8ef0ef2
```

Le HEAD -> master se déplacera sur le commit 8ef0ef2 en annulant les autres commits (messages) **sans perte** (sans perdre votre travail):

```
8ef0ef2 (HEAD -> master) ignore
0a70edb file2
bf3f73c first commit
```

Les fichiers et leurs modifications ne sont pas perdu tout est replacé dans le WD (espace de travail).

git reste –hard

Parfois vous voudriez supprimer un commit avec le(s) fichier(s). Bien sûr cette commande **est très dangeureuse** car, vous allez supprimer des fichiers ou des modifications. Donc elle est à faire **avec beaucoup beaucoup de prudence!**.

```
# Annule le dernier commit et supprime les modifications...(danger)
$ git reset --hard HEAD~1
```

git checkout

Nous allons utiliser cette commande pour remettre un fichier dans son état initial.

07 Exercice d'application checkout

- Créez un dépôt **example-checkout**
- Initialisez le dépôt

Chaque étape ci-dessous doit donner lieu à un commit :

- Créez un fichier index.html dans lequel vous mettez uniquement la chaîne de caractères suivante : Hello world.
- Créez le fichier category.html.

- Modifiez le fichier index.html dans lequel vous mettez maintenant du code HTML et dans le body et un h1 la chaîne de caractères “Hello World”. Le message du commit s’appelera “refonte de la page d’accueil”.

Vous devriez avoir quelque chose qui ressemble à l’historique suivant :

```
5050a3a (HEAD -> master) refonte de la page d'accueil
9b85968 add category
8a56381 add index
```

Nous aimerions annuler la refonte de la page d’accueil et revenir dans l’état initial (Hello World) :

```
# Permet de revenir avant la refonte
$ git checkout 9b85968 index.html

$ git status
# modifié :         index.html
```

- Pour terminer

En interprétant le message du git status que proposez-vous comme solution pour valider les modifications et donc revenir dans l’état initial du fichier index.html (avant la refonte) ?

git revert

Nous allons voir maintenant une autre commande pour annuler un commit. La méthode revert annule un commit en créant un commit d’annulation.

Exercice d’application

Mettez en place cet exemple dans votre d’exercices :

```
$ mkdir example_revert
$ cd example_revert
$ git init
$ echo "Title : file readme" > README.md
$ git add README.md
$ git commit -m "first commit"
$ echo "Bad update" > README.md
$ git commit -am "bad update commit..."
```

Nous allons maintenant afficher en console les hash des commits (court) :

```
git --no-pager log --oneline
```

```
(HEAD -> master) 419e298 bad update
f9cbeaa first commit
```

Tapez maintenant la commande suivante elle va créer un commit revert d'annulation du dernier commit réalisé :

```
git revert 419e298
```

Ecrivez un message de commit, vous venez d'annuler le "bad commit" pour vous remettre dans l'état initial, vérifiez bien que le titre est celui que l'on a mis au début dans le fichier README.md à savoir "Title : file readme".