HOW GIT ?

Plusieurs ressources méritent d'être mentionnées pour leur complétude et efficacité dans l'explication du fonctionnement de GIT et ses commandes de base. Notamment,

- https://grafikart.fr/tutoriels/remote-push-pull-589 : un ensemble de vidéos explciatives du fonctionnement interne de GIT et des commandes qui vous accompagnerons chaque jour dans votre travail ;
- https://git-scm.com/book/fr/v2/Les-branches-avec-Git-Les-branches-en-bref ;
- https://www.atlassian.com/fr/git/tutorials/what-is-git ;
- https://openclassrooms.com/fr/courses/7162856-gerez-du-code-avec-git-et-github.

Configuration

Avant toutes choses, assurez vous d'avoir installé git sur votre machine.

```
sudo apt install git -y
```

Une fois git installé, la première chose est de nous présenter à GIT via un nom d'utilisateur et un mail public.

```
git config --global user.name "***"
git config --global user.email monmail@gmail.com
```

Également, il est possible de modifier le nom de la branche principal de chaque dépôt ainsi que notre éditeur de texte favori. Par défaut, la branche principale se nomme "Master".

```
git config --global init.defaultBranch main
git config --global core.editor vim
```

Puis créons un repo local dans notre répertoire courant 'Statisserie'.

```
cd ./Bureau/Statisserie
git init
```

Staging area.

Capturons l'image du projet actuellement : ce snapshot est stocké dans la zone d'attente - i.e. l'index. Cette zone d'attente est communément appelée "staging area". Pour ce faire, on doit utiliser la commande "add".

```
git add .
```

Notez la présence du point à la suite de la commande 'git add', cela permet d'envoyer en zone d'attente tous les fichiers et dossiers contenus dans votre répertoire (ceux modifiés et les nouveaux). Une autre façon de procéder aurait été de préciser "-all" :

```
git add --all # ou, -A
```

Si vous souhaitez plus d'informaiton concernant la commade "git add", vous pouvez faire appel à la documentation comme suit :

```
git add --help
```

Commit. Stockons de façon permantente cette image dans le repo git local.

```
git commit -m "Ajout test homogénéité variances"
```

Lors d'un **commit** il est important d'ajouter un message décrivant la nature du commit. Ce message est spécifié grâce à l'option -m suivi du message entre guillement. Maintenant que notre image est stockée nous pouvons l'envoyer sur le repo distant. Toutefois, entre temps, nous avons modifié plusieurs documents de travail. Nous ne savons plus lesquels, pour le savoir on peut utiliser la commage "git status".

```
git status
```

Voici le résultat de la commande :

```
On branch main:
    Changes to be committed:
        modified: analyse_bivariee.py
```

Cette commande renvoie l'ensemble des fichiers situés dans l'arborescence de notre répertoire et ayant subit une modification (ou une suppression).

On sait à présent qu'un seul document de travail à été modifié. Il faut donc l'ajouter à notre index avant de pouvoir stocker cette image sur notre repo local. On ajoute à l'index notre fichier 'analyse_bivariee.py' :

```
git add analyse_bivariee.py
```

Si vous êtes certains des modifications apportées à votre répertoire courant et que vous voulez les envoyer en staging area et les commit en même temps, exécutez la commande suivante :

```
git commit -a
```

Nous pouvons désormais commit notre nouvel index :

```
git commit -m "Ajout de l'analyse bivariée."
```

Maintenant nous pouvons envoyer sur le dépôt distant notre travail. Pour ce faire, on va utiliser la commande "git push".

```
git remote add origin https://github.com/JordanNSZ/statisserie (origin
existe déjà)
git push -u origin main
```

La commande git remote vous permet de créer, d'afficher et de supprimer des connexions avec d'autres dépôts. Ainsi, la première ligne de code nous permet de créer une connexion entre notre dépôt local et notre dépôt distant. Ici, le dépôt distant est mentionné via son URL. Notez la présence du nom "origin", il indique simplement le nom de notre dépôt distant - vous pouvez le renommer comme bon vous semble ; 'origin' n'est qu'une simple convention d'après moi. Ensuite, la commande "git push" précise qu'on envoi sur la branche "main" de ce dépôt distant (nommé "origin") l'ensemble des changements effectués localement et qui ont été capturés lors des précédents commit.

En ce qui concerne l'adresse du dépôt, vous pouvez soit utiliser une adresse https (connexion à un dépôt distant), soit utiliser le chemin absolu de votre dépôt local. Pour vous assurez que la connexion à bien été établie : git remote -v. Vous appercevez que votre dépôt est prêt pour envoie (réception) sur le dépôt distant (local) : git push (pull) origin main. En ce qui concerne l'option "-u" : Une fois que vous avez exécuté "git push -u origin main", Git enregistre cette relation de suivi entre votre branche locale main et la branche distante main du dépôt origin. Cela signifie que vous n'avez plus besoin de spécifier explicitement la branche et le dépôt distant chaque fois que vous souhaitez pousser ou tirer des modifications. Vous pourrez donc simplement utiliser "git push" ou "git pull" et git "fetch" pour envoyer ou revoir les mise à jour de votre projet.

Suite à ces commandes, vous devrez renseigner votre nom d'utilisateur ainsi que votre mot de passe. Attention, vous devez impérativement demander la génération d'un token sur le site GitHub. En effet, git n'accepte plus l'authentification par mot de passe.

> Pour créer un token, rendez-vous sur le site github, les explications sont simples et efficaces.

Revenons en arrière. Nous avons modifié le document "analyse_bivariee.py". Toutefois, nous ne sommes pas certain de sa pertinence dans notre projet et des suites qui lui seront réservées. Aussi, il semble plus prudent de créer une nouvelle branche et d'enregistrer cette partie du projet sur cette dernière. Une branche, c'est comme une bifurcation temporaire de votre projet : vous voulez essayer une nouvelle

fonctionnalité ou un nouveau modèle sans que ces modifications n'interagissent avec l'état de votre projet actuel.

Créons cette nouvelle branche :

```
git branch anova_non_prametrique
```

Puis, envoyons notre nouvelle idée sur cette branche :

```
get switch anova_non_parametrique
get commit -a
```

La commande "git switch" permet de changer de branche. On utilise la commande "git commit -a" car on sait que les seules modifications apportées à notre répertoire courant sont l'ajout du document "analyse_bivariee.py".

Super nouvelle, vous et votre équipe avez approuvé les changements apportés à votre projet. La branche "anova_non_parametrique" peut être ajoutée à la branche principale - i.e. "main". Pour ce faire, rien de plus simple : il vous suffit d'utiliser la commande "git merge" ou "git rebase", pas si simple finalement... La différence entre "merge" et rebase" réside dans la façon dont git va ajouter la branche de travail à la branche principale : rebase va ajouter les commits de la branche de travail au dessus de la branche principale alors que "merge" va ajouter tous les commits de la branche de travail à la branche principale. La fonction "merge" va donc conserver l'intégralité des commits de la branche de travail mais rendre la compréhension de l'historique plus complexe. À l'inverse, la fonction "merge" va ajouter l'état final de la branche de travail à la branche principal sans rapporter l'historique des commits.

Pour résumer, "git merge" conserve l'historique complet des deux branches - rendant sa lecture parfois complexe - alors que "git rebase" ajoute simplement le dernier commit à la suite de la branche principale - rendant intraçable les modifications apportées à la branche secondaire.

Selon vos besoin, vous choisirez l'une ou l'autre des commande. Puisque notre branche secondaire ne demande pas un suivi particulier des modifications apportées, on utilisera la commande "rebase".

```
git rebase anova_non_parametrique
```

> Rebase va ajouter les changements effectués en branche secondaire à la suite de la dernière modification en branche principale.

Si vous souhaitez utiliser la commande "merge" :

```
git merge anova_non_parametrique
```

> Merge va créer une jointure des modifications sur la branche principale.

Désormais, notre branche "anova_non_parametrique" n'est plus nécessaire, on décide donc de la supprimer.

```
git branch -d anova_non_parametrique
```

Mettre à jour notre dépôt local.

Pour cela il faut utiliser la commande **git pull**. Cette commande va d'abord récupérer les modifications effectuées par vos collègues **git fetch** puis fusionner ces modifications avec votre dépôt local. Lors de cette dernière étape git effectura soit un **git merge**, soit un **git rebase**.

```
git pull origin main
```

Ici *'origin'* représente le nom du dépôt distant et *'main'* le nom de la branche en local.

Mettre en file d'attente un fichier à supprimer.

Vous vous appercevez que, par erreur, vous avez envoyé un fichier en distant. Ce fichier contient des données personnelles, la RGPD vous interdit de les diffuser. Il vous faut donc retirer cette base de données du dépôt distant. Pour ce faire, on va utiliser la commande "git rm --cached" suivi du nom du fichier à supprimer. Notez que ce fichier sera simplement marqué comme "non suivi" puis sera supprimé lors du prochain add-commit-push.

```
git rm --cached .~donnees_clients.csv
```

Dans cette commande "git rmé est utilisée pour supprimer des fichiers de l'index (la zone de staging) et de votre répertoire de travail. Cependant, l'option "--cached" indique à Git de ne supprimer le fichier que de l'index, mais de le laisser intact dans votre répertoire de travail.

Ensuite, pour vous assurer que Git n'ajoute plus ce fichier à l'avenir, ajoutez son nom au fichier .gitignore.

```
echo ".~donnees_clients.csv" >> .gitignore
```

> Le fichier .gitignore est un fichier de configuration utilisé par Git pour déterminer quels fichiers ou répertoires doivent être ignorés par le système de contrôle de version. Cela signifie que les fichiers ou répertoires listés dans le .gitignore ne seront pas suivis, ajoutés ou commités dans le dépôt Git.

Puis commiter ce changement :

```
git commit -m "Arrêt du suivi des donnees clients."
```

Récapitulatif de la procédure d'envoie de modifications locales en distant.

Afin d'envoyer des modifications du dépôt local vers le dépôt distant, 3 étapes sont nécessaire : - add ; - commit ; - push. **La commande "add" permet d'ajouter à la staging area - i.e. file d'attente a.k.a index - les modifications apportées à votre dépôt -tout ou partie des modifications. La commmande "commit" permet de capturer les changements disponibles dans l'index. La commande "push" envoie les changements capturés vers la branche sur laquelle on se trouve -par défaut, main. Il est possible de changer la branche sur laquelle on souhaite envoyer nos changement via la commande git switch mon_autre_branche.**

Par exemple,

```
git add --all (git add README.md test_f.md)
   git commit -m "Ajout test homogénéité variances"
   git remote add origin https://github.com/JordanNSZ/statisserie (origin
existe déjà)
   git push -u origin main
   Username : ****
   Password : ****
```

Annuler les modifications locales apportées au projet.

Après avoir modifié mon projet, je me suis aperçu que les modifications n'étaient pas bonnes. Toutefois, la nouvelle version de mon projet a déjà été envoyé sur le dépôt distant. Je dois donc annuler ce "commit" tout en supprimant les modifications apportées en local. Pour cela il faut utiliser la commande **git revert**. Cette commande prend comme argument le hash du commit en question. Pour connaitre le hash du commit en question, il vous suffit d'employer la commande **git log**.

```
git revert 6815e2789328a3fec4005c8974201537322c0b86
```

Et voilà, en local j'ai récupéré la version de mon projet avant modification. Je n'ai plus qu'à effectuer un nouvelle envoie pour rétablir les choses sur le dépôt distant.

Récupérer un dépôt distant en local. Si vous souhaitez récupérer un dépôt distant en local vous devez utiliser la commande **git clone url_du_projet**. Celle-ci va créer un répertoir au nom du projet puis initialiser un dépôt git (**git init**) avant d'ajouter un serveur distant (**git remote add**). Enfin, le dépôt distant est récupéré via la commande **git fetch** et le dernier commit est extrait du répertoire de travail avec **git checkout**.

Voici comment récupérer la dernière version disponible de votre projet "mon_projet". Placer vous à l'endroit où vous souhaitez stocker le dossier de votre projet, puis lancer la commande *git clone*.

```
git clone https://github.com/JordanNSZ/portofolio
```

Si vous souhaitez que le nom du répertoire courant de votre projet diffère de celui du projet, vous pouvez en spécifier un.

```
git clone https://github.com/mon_nom_utilisateur/mon_projet mon_projet2024
```

Pour aller plus loin.

Si il y a des **conflits entre nos modifications** git crée un rapport 'problematic files' auquel on peut accéder via **git diff**.

Voyons ce qui est en attente d'envoie sur le repo git local.

```
git diff --cached
```

> Note. **git diff vs git status**. "The main difference between the commands is that git diff is specially aimed at comparisons, and it's very powerful at that: It can compare commits, branches, a single file across revisions or branches, etc. On the other hand, git status is specifically for the status of the working tree." (Internet)