Chapter 1

Git and GitHub

intro
Explain what git and GitHub are and their purposes
dépôt au lieu de repository/repo dans ce tutoriel

1.1 Config

Cette section devrait déjà être faite, vous devriez déjà avoir un compte github, avoir téléchargé git et avoir configuré votre email. Si jamais ce n'est pas fait, voici comment le faire.
Commencez par ouvrir votre terminal et taper "git"
Download : `https://git-scm.com/` xcode-select –install
Configuring git :

```
git config --global user.email "votre_courriel@ulaval.ca"
git config --global user.name "votre_nom"
```

token d'identification (Mac) (`https://docs.github.com/fr/authentication/keeping-your-account-and-data-secure/managing-your-personal-access-tokens#cr%C3%A9ation-dun-personal-access-token-classic`)

1.2 Première étape avec Git

Cloner le repo de ce tutoriel aller chercher le lien du repo
cloner un repo

Exercice : Clôner le dépôt du tutoriel

Vous allez maintenant cloner le dépôt Git du tutoriel.
`git clone https://github.com/cia-ulaval/tutoriels-h2024.git`

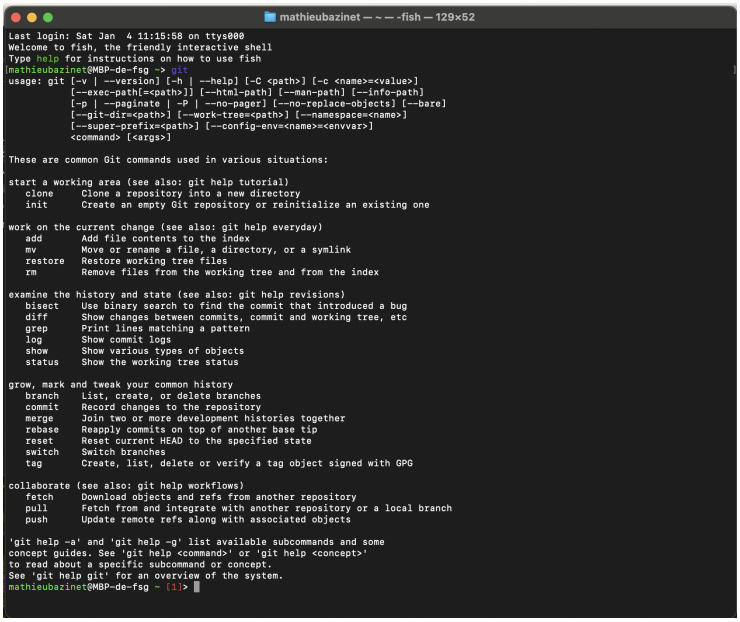

Figure 1.1: Sortie attendue dans le terminal lorsque la commande “git” est tapée dans le terminal.

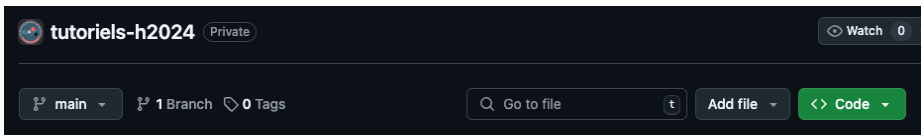

Figure 1.2: À remplir

git clone

La commande `git clone <url-du-repo>` permet de créer une copie locale du projet.

Normalement, un bon repo va avoir les quatre fichiers suivants :

1. README.md
2. LICENSE
3. .gitignore
4. requirements.txt

1.2.1 README

https://www.freecodecamp.org/news/how-to-write-a-good-readme-file/
Le README est le premier contact entre l'utilisateur et le dépôt. C'est un fichier markdown qui doit donner les informations nécessaires à utiliser le code du dépôt. Le README va contenir un titre,
Project's title Project description How to install the project/ en lien avec le requirements.txt + la version de Python si applicable
How to use the project (quels fichiers font quoi, si c'est des expériences dans un papier, comment rouler les expériences, etc.)
Crédits, si du code provient d'un autre endroit, le nom des collaborateurs, sources, etc. contact information

1.2.2 LICENSE

S'il n'y a pas de license, c'est équivalent à "ne pas toucher". Il y a beaucoup de chercheurs qui ne mettent pas de license en se disant que ça devient "open-source" et peut-être utilisé par n'importe qui, mais c'est le contraire. Pas de license veut dire que tous les droits sont à l'auteurs et que tu ne peux pas utiliser le code. Dans un petit projet, ce n'est pas très grave, personne va vous taper sur les doigts. Si vous faites de la recherche/ vous voulez publier du code et laisser la communauté l'utiliser, il faut demander la permission aux auteurs ou suivre la license.
Les licenses les plus classiques CC-BY 4.0 MIT Licence publique générale GNU v3.0 Licence Apache 2.0
https://choosealicense.com/no-permission/ https://choosealicense.com/
https://docs.github.com/fr/repositories/managing-your-repositorys-settings-and-features/customizing-your-repository/licensing-a-repository

1.2.3 .gitignore

Le fichier gitignore n'est pas le fichier le plus intéressant. C'est purement un fichier utilitaire. Il va être utile uniquement si l'utilisation de votre code crée des fichiers qui ne doivent pas se retrouver sur github. Par exemple, si votre code télécharge MNIST, on ne veut pas qu'il se retrouve sur GitHub. De plus, directement dans le dossier Tuto1-GitHub, vous trouverez un fichier ".tex". Si vous compilez le fichier, vous aurez un pdf ainsi que des fichiers avec des extensions tels que ".aux", ".fls", ".log", ".out", etc. Ces fichiers ne sont pas pertinents, et je ne veux pas oublier de les enlever et mettre des fichiers inutiles sur github. Si vous regardez le fichier .gitignore, vous verrez en bas les lignes suivantes :

```
*.aux
*.fdb_latexmk
*.fls
*.out
*.synctex.gz
```

Vous ne les trouverez pas sur GitHub, puisqu'ils sont automatiquement ignorés. L'astérisque au début veut dire : tous les fichiers de style nom_du_fichier.aux va être matché par *.aux.
`https://git-scm.com/docs/gitignore`

1.2.4 requirements.txt

En plus de la version de Python donnée dans le README, il est de bonne usage de donner la version des librairies utilisées par le projet. Vous garantissez que le code va fonctionner et sera reproductible si l'utilisateur utilise la version des librairies que vous avez utilisés.
pip freeze > requirements.txt pip install -r requirements.txt

1.3 Créer un dépôt Git

Techniquement, il est possible de créer un dépôt avec `git init`. Je trouve qu'il est beaucoup plus facile de le faire directement sur GitHub. Vous devrez choisir :
un nom Description courte Si le repo est privé ou public Ajouter un README Ajouter un gitignore Choisir une licence
Dans la section sur comment créer un repo, Parler des repo ¡Nom-d'utilisateur¿ et ¡nom-d'utilisateur¿.github.io
`https://github.com/MathieuBazinet` contient uniuqmenet un readme qui va être display
`https://github.com/MathieuBazinet/MathieuBazinet.github.io` contient le contenu qui va se retrouver dans le site web. `https://mathieubazinet.github.io/fr/`
`https://github.com/NathanielDamours` un repo sans readme

Exercice : Créer un dépôt Git

Dans cet exercice, vous allez créer un dépôt Git.

1. Sur GitHub, créez un nouveau dépôt en cliquant sur le bouton vert "New".
2. Choisissez le nom du dépôt. Nous recommandons "tuto-cia" ou d'utiliser le nom aléatoire généré par GitHub.
3. Ajoutez une petite description du dépôt.
4. Cliquez sur "Private" pour créer un dépôt privé.
5. Ajoutez un README
6. Choisissez le .gitignore de votre choix en tapant le langage de programmation que vous utilisez.

7. Choisissez une licence de votre choix. Je recommande “MIT License”.
8. Créez votre dépôt.

1.4 Commandes Git

git add

hello

git commit

hello

git push

hello

git branch

hello

git checkout

hello

Git log

hello

Git status

hello

1.4.1 Commandes Git pour quand vous avez fait une erreur

git reset

hello

git revert

hello (https://stackoverflow.com/questions/19032296/how-to-use-git-revert)

git commit —amend

hello

1.4.2 Commandes Git que je n'utilise jamais

1. git init
2. git fetch
3. git stash
4. git switch

1.4.3 Exemple concret

Nous allons maintenant pratiquer l'utilisation de ces commandes. De plus, nous allons générer un problème de fusion (*merge*) entre deux *commits*.
Premièrement, nous allons commencer par cloner le dépôt créé précédemment dans la section 1.3.

Exercice : Cloner le nouveau dépôt

1. Ouvrez votre dépôt sur GitHub
2. Obtenez le lien du dépôt, en cliquant sur le bouton vert "code" et en choisissant "https".
3. Utilisez `git clone <url-du-dépôt>` dans votre terminal.

Avant de modifier le dépôt et d'utiliser les commandes apprises dans la section précédente, nous allons ajouter un README.md directement sur GitHub.

Exercice : Modifier le README sur GitHub

1. Sur GitHub, cliquez sur le bouton de modification représenté par un crayon.
2. Ajoutez un titre pertinent au README. Je recommande "Mon premier dépôt git".
3. Ajoutez une petite description du contenu du laboratoire.
4. Cliquez sur le bouton "Commit changes...".

5. Ajoutez une courte description du *commit*.
6. Choisissez “commit directly to the `main` branch”.
7. Cliquez de nouveau sur “Commit changes”.

Nous allons maintenant modifier le contenu du README directement sur l'ordinateur. Le but ici est de vous montrer un exemple de l'utilisation des commandes de base de git, ainsi que la gestion de fusion.

Exercice : Modifier le README dans le dépôt local

1. Ouvrez le README sur votre ordinateur, puis ajoutez une courte phrase dans le document.
2. Dans le terminal, tapez `git add README.md`.
3. Tapez `git commit -m "mon premier commit"`
4. Tapez `git push`

Si tout c'est bien passer, vous devriez avoir un message d'erreur. En effet, vous devriez voir quelque chose de similaire à la figure 1.3.

```
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

Figure 1.3: Message d'erreur quand des modifications ont été faites sur le dépôt git mais pas dans la version locale sur votre ordinateur.

En effet, après avoir clôner le dépôt, nous avons fait des modifications directement sur GitHub. Puis, nous avons modifié la version locale. Les deux versions n'étant pas identique, il est nécessaire de se mettre-à-jour avant de soumettre nos modifications.

Exercice : Gérer manuellement la fusion

1. Tapez `git pull`
2. Si vous obtenez une erreur similaire à la figure 1.4, suivez les étapes suivantes :
   (a) Tapez `git config pull.rebase false`
   (b) Tapez `git pull`
3. Tapez `git status`.

```
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

Figure 1.4: Message d'erreur quand des modifications ont été faites sur le dépôt git mais pas dans la version locale sur votre ordinateur.

Après toutes ces étapes, vous devriez avoir une sortie similaire à celle de la figure 1.5

```
On branch main
Your branch and 'origin/main' have diverged,
and have 1 and 1 different commits each, respectively.
  (use "git pull" to merge the remote branch into yours)

You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)
        both modified:   README.md

no changes added to commit (use "git add" and/or "git commit -a")
```

Figure 1.5: Sortie dans la console après la commande `git status`.

On voit bien en rouge que les deux branches ont modifiées le README. Nous allons donc ouvrir le README dans votre éditeur de code préféré. Dans le README, vous devriez voir les symboles suivant : `<<<<<<<`, `=======`, `>>>>>>>`, similairement à la figure 1.6.
La section entre `<<<<<<<` et `=======` correspond au code local en conflit. La section entre `=======` et `>>>>>>>` correspond au code déjà soumis sur GitHub qui rentre en conflit avec votre code local. Dans la figure 1.6, VSCode propose quatre options pour gérer le conflit. Vous pouvez accepter les modifications locales, accepter les modifications entrantes ou faire un mélange des deux. Si vous savez quelle version choisir, laissez l'éditeur faire son travail et cliquer sur une des deux options. Sinon, vous pouvez manuellement modifier le fichier jusqu'à ce qu'il soit à votre goût. Assurez vous d'enlever les symboles `<<<<<<<`, `=======`, `>>>>>>>`.

Exercice : Finir la fusion

1. Ouvrez le fichier README dans un éditeur de code.
2. Faites les modifications nécessaires. Retirez les symboles associés au conflit.

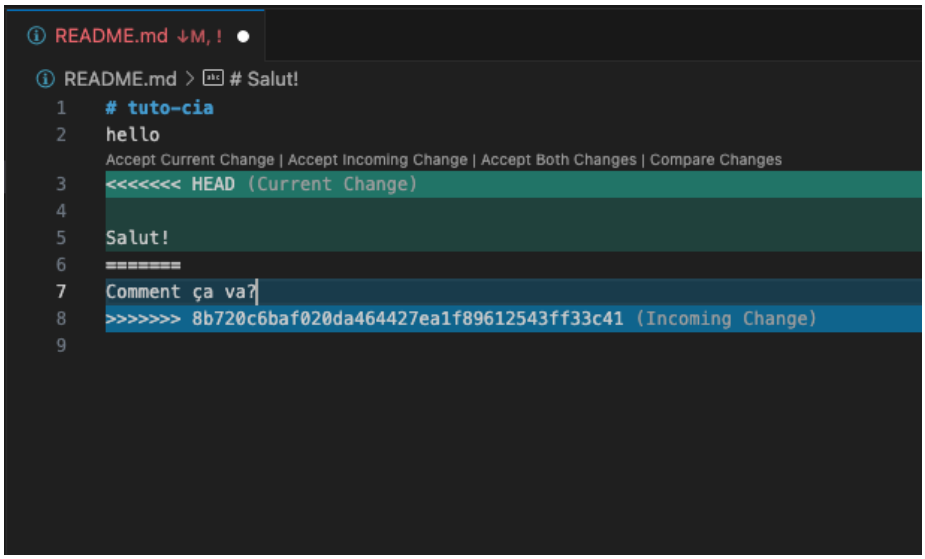

Figure 1.6: Apparence du fichier README lorsque la fusion a échoué.

3. Dans le terminal, tapez `git add README.md`
4. Tapez `git commit -m "Mon premier conflit!"`
5. Tapez `git push`

Normalement, si tout a fonctionné, les modifications devraient apparaître sur GitHub!
Pour plus d'information sur les conflits, allez voir ici : Lien du tutoriel GitHub.

1.5 Fonctionnalités GitHub

Normalement, avec tout ce qui a été discuté jusqu'à maintenant, vous devriez être en mesure d'utiliser Git et GitHub. Mais, il existe des fonctionnalités de GitHub dont nous n'avons toujours pas parlé. Par exemple, peut-être vous souviendrez-vous avoir vu, lors de la modification du README sur GitHub, l'option `Create a new branch for this commit and start a pull request`.
Les branches permettent de travailler sans avoir peur de faire une erreur dans le code. De plus, lors d'un travail d'équipe, cela permet à chacun de développer tranquillement sans gérer de fusion, avant de pousser le tout dans la branche principale.
Un autre outil utile est la création d'*issues* sur GitHub. Les *issues* GitHub permettent de décrire un problème ou une tâche.

1.5.1 Créer des issues

Tel que mentionné plus haut, les *issues* sont des tickets de tâches qui devront être accomplies plus tard. Ces tickets contiennent un titre et une description, et peuvent être assigner à un membre du dépôt.

Exercice : Créer un *issue*

1. Ouvrez votre dépôt GitHub et ouvrez la section *issues*
2. Cliquez sur le bouton vert `New issue`
3. Ajoutez un titre. Je recommande “Mon premier issue”
4. Ajoutez une courte description.
5. Dans le coin à droite, assignez vous-même à l’issue.
6. Créez l’issue

Maintenant que l’*issue* est créé, il sera disponible jusqu’à ce qu’il soit fermé. Dans le cadre de certains gros projets, par exemple Scikit-Learn, il est possible de reporter un bug ou poser une question. Un *issue* peut être fermé de deux façons différentes : `Close as completed` et `Close as not planned`. Lorsque vous avez réglé le problème, l’*issue* va être fermée en tant que complétée. Toutefois, si vous ne souhaitez plus réglez le problème, ou c’était une amélioration que vous n’avez pas le temps ou l’envie de faire, vous pouvez fermer l’*issue* en tant que “non planifié”.
Après l’ouverture d’une *issue*, GitHub propose automatiquement de créer une branche sur laquelle régler le problème.

1.5.2 Créer une branche

Nous allons maintenant créer la branche. Sur la branche, vous pouvez faire toutes les modifications en rapport avec le problème que vous résolvez. Lorsque l’*issue* sera réglé, vous pourrez fusionner votre branch avec la branche principale. Créons maintenant la branche.

Exercice : Créer une branche

1. Ouvrez l’issue qui vous intéresse
2. Cliquez sur `Create a branch` en bas à droite
3. Modifiez le nom de l’issue pour un nom un peu plus courte
4. Sélectionnez `Checkout locally`
5. Cliquez sur le bouton vert `Create branch`

Pour accéder à la branche, vous pouvez suivre les instructions de GitHub.

Exercice : Accéder à la branche

1. Dans le terminal, tapez `git fetch origin`
2. Entrez `git checkout <nom-de-la-branche>`

Modifiez maintenant le README sur la nouvelle branche, puis poussez la modification sur la branche. Nous allons maintenant effectuer un *pull-request*.

1.5.3 Pull request

Les *pull requests* permettent de vérifier les changements qui ont été effectués sur la branche. Elles sont aussi utilisées pour faire vérifier le code par des collègues avant de fusionner les modifications à la branche principale. Vos collègues pourront mettre des commentaires et demander des modifications avant d'accepter la fusion.
Si vous ne l'avez pas encore fait, faites une modification dans le README sur la nouvelle branche, puis poussez la modification sur la branche. En allant sur GitHub dans la section `Pull requests`, vous verrez maintenant un nouveau message disant `<nom-de-la-branche> had recent pushes` x `minutes ago`. Cliquez maintenant sur `Compare & pull request`.

Exercice : Créer une *pull request*

1. Cliquez sur `Compare & pull request`
2. Ajoutez un titre parlant
3. Ajoutez une courte description de la modification
4. Cliquez sur le bouton vert `Create pull request`

Normalement, si c'était un projet avec plusieurs coéquipiers, vous seriez amener à ajouter des reviewers. Sans l'approbation des reviewers, vous ne serez pas en mesure de fusionner votre code avec la branche principale. De plus, GitHub vous indique s'il y a un conflit avec la branche principale. Dans ce cas, j'utilise généralement ce tutoriel.
Après la création d'une *pull request*, vous avez accès à la conversation avec les reviewers, les différents *commit* que vous avez poussé sur la branche ainsi qu'une comparaison entre le code sur la branche principale et sur les modifications entrantes.
Quand le code est approuvé, cliquez sur `Merge pull request`. Le code sera maintenant fusionné à la branche principale. GitHub vous proposera de plus de supprimer la branche. Puisqu'elle ne sera plus utilisée, généralement, vous pouvez la supprimer immédiatement. En fermant la *pull request*, généralement, l'*issue* va être fermée en tant que complétée.