Chapitre 1

Git and GitHub

Git et GitHub sont deux outils dont vous ne pourrez pas vous passer si vous voulez avoir une carrière qui inclut de la programmation. Normalement, vous devriez tous avoir un compte GitHub et avoir déjà installé Git sur votre ordinateur. Ces outils sont utilisés dans les cours d'introduction à la programmation puisqu'ils sont essentiels. Vous connaissez probablement plusieurs informations qui se trouvent dans ce tutoriel. Toutefois, il est possible que vous ne les compreniez pas suffisament pour vous en servir correctement au quotidien. Le but de ce tutoriel est de faire un rappel et de vous apprendre la base de Git et GitHub qui n'a pas été enseignée durant le cours d'introduction à la programmation.

Git est un outil de versionnage et d'hébergement de code. Git permet d'héberger son code sur un serveur, ce qui a l'avantage d'éviter d'avoir une seule version locale sur votre ordinateur et de faciliter la collaboration. En hébergeant votre code dans un dépôt central, vous pouvez l'utiliser et le modifier à partir de différents ordinateurs, tout en étant sûr que vous avez toujours une sauvegarde. Puisque le code est hébergé sur un serveur, plusieurs développeurs peuvent développer chacun de leur côté sur leur ordinateur, puis fusionner le code à la version globale, qui pourra ensuite être utilisée par tous les autres collaborateurs. Il est recommandé d'utiliser Git même lorsque vous travaillez seul, pour avoir accès aux différentes versions du projet si vous faites une erreur ou si votre ordinateur a des problèmes.

Chacun de vos projets sera placé dans un dépôt[1]. Dans votre ordinateur, le dépôt local sera simplement un dossier qui contient l'entièreté de votre code. Toutefois, il sera aussi synchronisé sur un dépôt central. Le dépôt central est la version hébergée sur les serveurs de GitHub. Dans les prochaines sections, je vous montrerai à utiliser les différentes lignes de commande Git.
GitHub est un serveur qui utilise Git comme interface avec l'utilisateur. C'est GitHub qui va héberger le code. Pour déposer votre code sur GitHub, vous

[1]Tout au long de ce tutoriel, j'utiliserai dépôt au lieu de *repository*.

devrez utiliser Git. GitHub est utile entre autres pour le partage de code. En effet, il est possible d'accéder au code d'un collègue simplement grâce à un url. Il existe de plus beaucoup de dépôts publics, que vous pourrez utiliser pour vos différents projets. Ils proposent de plus plusieurs outils qui permettent de faciliter le développement logiciel.
Avant de commencer à présenter les commandes git, au cas où vous n'auriez toujours pas installé Git sur votre ordinateur, je vous fournis les étapes.

1.1 Configuration de Git

Comme mentionné dans l'introduction, vous devriez déjà avoir un compte gitHub, avoir téléchargé Git et l'avoir configuré. Toutefois, si jamais ce n'est pas fait, voici comment le faire.
Commencez par ouvrir votre terminal et taper "git". La sortie de votre terminal[2] devrait ressembler à celle de la figure 1.1. Si c'est le cas, alors Git est déjà installé et vous pouvez passer à la prochaine section.

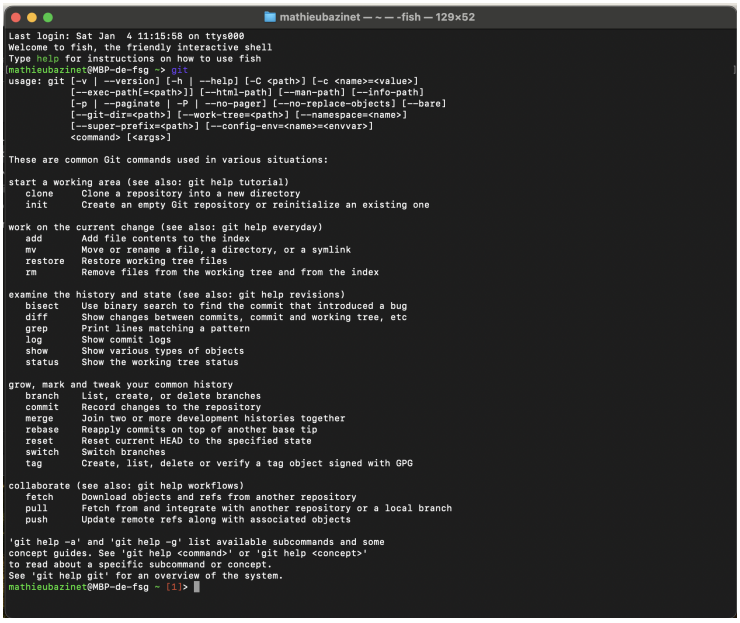

Figure 1.1: Sortie attendue dans le terminal lorsque la commande "git" est tapée dans le terminal.

Si Git n'est pas installé, vous devrez le télécharger. Normalement, sur Linux, il devrait déjà être installé. Sur Windows, téléchargez le ici. Pour les utilisateurs

[2]Quand je parle de Terminal, vous pouvez utiliser le terminal, git bash, etc. Tout ce qui compte, c'est que vous soyez en mesure d'utiliser des lignes de commande.

de Mac, nous recommandons de le télécharger en installant Xcode. Dans le terminal, tapez la commande suivante `xcode-select --install`.

Maintenant, vous devez configurer Git pour qu'il ait accès à votre profil GitHub. Dans le terminal, veuillez recopier les commandes suivantes puis remplacer votre email ainsi que votre nom d'utilisateur GitHub.

```
git config --global user.email "<votre-adresse-email>"
git config --global user.name "<votre-nom-d'utilisateur-github>"
```

Pour les utilisateurs d'un Mac, vous ne pourrez pas utiliser un mot de passe pour vous connecter à GitHub à travers Git. Vous devrez vous créer un *personal access token*, en suivant les instructions ici. À chaque fois que Git vous demandera un mot de passe, vous **devez** utiliser le token de GitHub.

1.2 Première étape avec Git et GitHub

Comme première étape, vous allez clôner le dépôt de ce tutoriel. Vous aurez ainsi accès à ce fichier PDF, qui pourra vous être utile plus tard. Lors du clônage d'un dépôt, Git va créer une copie locale du dépôt sur votre ordinateur, que vous pourrez modifier ensuite.

Pour clôner le dépôt, il vous est nécessaire d'avoir son url. Pour ce faire, suivez les étapes de l'exercice suivant.

Exercice : Trouver le dépôt du tutoriel

1. Sur GitHub, cherchez "Club d'intelligence artificielle de l'Université Laval".
2. Par défaut, GitHub cherche dans les noms de dépôt. Sélectionnez "Users" dans la barre de gauche.
3. Sélectionnez "Club d'intelligence artificielle de l'Université Laval"
4. Cherchez maintenant le dépôt nommé "tutoriels-h2024" et cliquez dessus.

Prenez le temps d'observer le contenu du dépôt. Vous y trouverez les dossiers associés aux trois tutoriels de la session, ainsi que quatre fichiers : `.gitignore`, `LICENSE`, `README.md` et `requirements.txt`. Un bon dépôt devrait toujours contenir ces quatres fichiers. Nous y reviendrons plus tard, après avoir clôné le dépôt.

Nous voulons maintenant clôner le dépôt. Pour ce faire, nous utiliserons notre première commande Git.

git clone

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

Exercice : Clôner le dépôt du tutoriel

1. Cliquez sur le bouton vert marqué “Code” et sélectionnez “HTTPS”.
2. Copiez le lien.
3. Dans le terminal, tapez `git clone <lien-du-repo>`.

Maintenant que vous avez téléchargez le dépôt, ouvrez un par un les fichiers. Je vous présenterai maintenant les quatre fichiers essentiels à un bon dépôt GitHub.

1.2.1 README

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 ainsi qu'une description du projet. Ainsi, on doit savoir quel est le but du projet ainsi que comment l'utiliser. Plus la description est claire, plus il est probable que d'autres utilisateurs veuillent utiliser votre code. Il est donc important de faire un beau README.

Par exemple, si c'est un dépôt qui fournit une fonction ou un groupe de fonction, donnez un exemple simple d'utilisation. Par exemple, si mon projet fourni la fonction `reverse_name` suivante[3]

```
def reverse_name(name):
    return name[::-1]
```

alors je pourrais fournir l'exemple suivant :

```
# Input your name
my_name = "John Doe"

# Reverse your name using the function
reversed_name = reverse_name(my_name)

# Print the reversed name
# This will output "Reversed Name: eoD nhoJ"
print("Reversed Name:", reversed_name)
```

Pour la majorité des projets de programmation, vous devriez spécifier la version de Python que vous avez utilisés lors du développement, ainsi que fournir un

[3]Exemple généré par ChatGPT sous ma demande.

fichier `requirements.txt`. De plus, si vous avez utilisés le code d'une autre personne, vous devez indiquer d'où provient ce code.
Pour plus d'information, je vous recommande ce blog.

1.2.2 LICENSE

Disclaimer : Je ne suis pas un expert légal, donc il ne faut pas croire tout ce que je dis.

La licence n'est pas un document que vous allez consulter souvent. Le but de la licence est de savoir ce que pouvez faire et ce que vous ne pouvez pas faire avec le code dans le dépôt. Les licences les plus classiques sont CC-BY 4.0, MIT, GNU et Apache 2.0. Généralement, vous devriez pouvoir utiliser et modifier le code sans problème, tant que vous citez le matériel emprunté. Il faut toutefois faire attention si jamais vous faites un projet de recherche ou un produit dans une compagnie, car certaines licences nécessitent que tout code qui utilise le projet soit sous la même licence et potentiellement rendu public.

S'il n'y a pas de licence, c'est équivalent à "ne pas toucher". Il y a beaucoup de chercheurs qui ne mettent pas de licence en se disant que ça devient "open-source" et peut-être utilisé par n'importe qui, mais c'est le contraire. Pas de licence 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 ne va vous taper sur les doigts. Si vous faites de la recherche ou vous voulez publier du code et laisser la communauté l'utiliser, il faut demander la permission aux auteurs ou suivre la licence[4].

Lorsque vous créez un dépôt, je vous encourage à suivre la procédure de GitHub pour la sélection de licence. Si vous avez d'autres questions, je vous recommande le site Choose A License.

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. Un autre exemple, 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 :

[4] Je vous encourage à aller lire cette page web : `https://choosealicense.com/no-permission/`

```
*.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.

Pour plus d'information, je recommande la documentation de Git.

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. Pour produire le fichier, vous pouvez utiliser la commande suivante :

`pip freeze > requirements.txt`

Quand vous ouvrez un dépôt qui contient un fichier `requirements.txt`, vous pouvez installer toutes les dépendances en utilisant la commande suivante :

`pip install -r requirements.txt`

1.3 Créer un dépôt Git

Vous allez maintenant 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 pour le dépôt
- Ajouter une description courte
- Choisir si le dépôt est privé ou public
- Ajouter un README
- Ajouter un .gitignore
- Choisir une licence

Lorsque vous choisissez un nom pour votre dépôt, il est important de savoir qu'il existe (au moins) deux noms de dépôts qui sont réservés pour des usages spécifiques. En effet, si vous choisissez votre nom d'utilisateur, GitHub va rajouter un README sur votre page d'accueil. Si vous choisissez un repo de style `<nom-d'utilisateur>.github.io`, GitHub va vous permettre d'héberger un

site web.

En effet, commencez par aller voir le profil GitHub de Nathaniel D'Amours. Vous aurez généralement accès aux dépôts publics de Nathaniel ainsi que ses contributions sur GitHub durant la dernière année.

Maintenant, allez voir mon profil sur GitHub. Dans un README, j'ai inclus mon CV, qui apparaît juste avant mes contributions sur GitHub. Ainsi, une personne qui s'intéresse à mon profil GitHub aura accès à mon CV et pourra en apprendre davantage sur ce que je fais.

Le deuxième nom de dépôt réservé vous permettra d'héberger un site web. Allez voir le dépôt nommé `MathieuBazinet.github.io`. Vous y trouverez tout le code et les fichiers qui se trouvent sur mon site web. Évidemment, ce n'est pas la seule façon de vous créer un site Web, mais pour les personnes qui n'ont jamais fait de programmation Web, c'est déjà très bien!

Vous allez maintenant créer un dépôt.

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.

Maintenant que vous avez créé un dépôt, je vais vous présenter les commandes Git dont vous aurez besoin.

1.4 Commandes Git

Dans cette section, je vais discuter des commandes les plus importantes pour l'utilisation de Git. Toutes les descriptions dans ce document sont extrêmement simplifiées, puisque les commande git sont extrêmement flexibles.

git add

La commande `git add <documents>` vous permet d'indiquer à Git quels documents doivent être ajoutés au projet ou les modifications qui doivent être synchronisées. La commande la plus utilisée est `git add .`, où le point représente le *current working directory*.

git rm

La commande `git rm <documents>` vous permet d'indiquer à Git quels documents doivent être supprimés du projet. Il est parfois nécessaire d'utiliser `git rm -r <dossier>` si vous voulez supprimer un dossier.

git commit

La commande `git commit` enregistre la version actuelle du code et la prépare à être soumise au dépôt global. La commande prend plusieurs arguments. L'usage le plus commun de cette commande est `git commit -m <votre-message-de-commit>`. Vous pouvez spécifier le message plus tard, mais ça ouvre généralement Vim ou un outil similaire et je le déconseille.

git push

La commande `git push` est généralement utilisée sans argument. Elle permet de soumettre tous les *commits* au dépôt global.

git pull

La commande `git pull` permet d'aller chercher les modifications faites sur le dépôt global puis de les fusionner au dépôt local. C'est équivalent à utiliser `git fetch` puis `git merge`, dont je ne discuterai pas ici.

1.4.1 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 s’est bien passé, vous devriez avoir un message d’erreur. En effet, vous devriez voir quelque chose de similaire à la figure 1.2.
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

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

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 1.3, suivez les étapes suivantes :
   (a) Tapez `git config pull.rebase false`.
   (b) Tapez `git pull`.
3. Tapez `git status`.

Git status

La commande `git status` permet d'afficher l'état du dépôt.

```
hint: You have divergent branches and need to specify how to reconcile them.
hint: You can do so by running one of the following commands sometime before
hint: your next pull:
hint:
hint:   git config pull.rebase false  # merge
hint:   git config pull.rebase true   # rebase
hint:   git config pull.ff only       # fast-forward only
hint:
hint: You can replace "git config" with "git config --global" to set a default
hint: preference for all repositories. You can also pass --rebase, --no-rebase,
hint: or --ff-only on the command line to override the configured default per
hint: invocation.
```

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.

Après toutes ces étapes, vous devriez avoir une sortie similaire à celle de la figure 1.4
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 1.5.

```
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 1.4: Sortie dans la console après la commande `git status`.

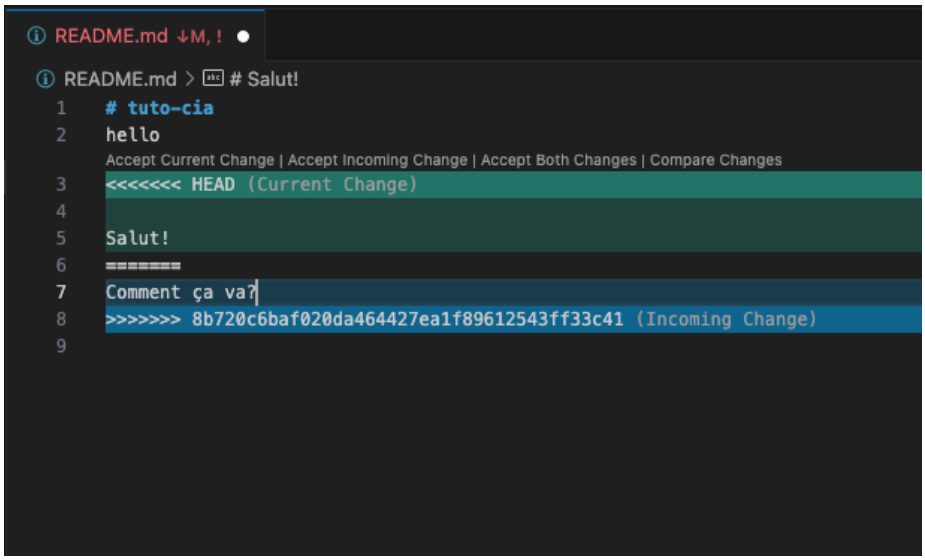

Figure 1.5: Apparence du fichier README lorsque la fusion a échoué.

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 1.5, 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 `<<<<<<<`, `=======`, `>>>>>>>`.

Suivez maintenant les étapes suivantes pour finaliser la fusion.

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.
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 court.
4. Sélectionnez `Checkout locally`.
5. Cliquez sur le bouton vert `Create branch`.

Au lieu de créer la branche sur GitHub, il est aussi possible de l'utiliser avec la commande `git branch`. Toutefois, c'est toujours plus facile de passer par GitHub.

git branch

La commande `git branch` peut être utilisée pour faire trois choses.

1. lister les branches : `git branch --list`
2. créer une nouvelle branche nommée `<nom-de-la-branche>` : `git branch <nom-de-la-branche>`
3. supprimer une branche nommée `<nom-de-la-branche>`: `git branch -d <nom-de-la-branche>`

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>`.

git checkout

La commande `git checkout` est généralement utilisée de deux façons différentes.

1. Changer de branche : `git checkout <nom-de-la-branche>`
2. Investiguer un commit : `git checkout <numéro-du-commit>`

Modifiez maintenant le README sur la nouvelle branche, puis poussez la modification sur la branche. Nous allons maintenant effectuer une *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 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 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.

1.5.4 Commandes Git pour quand vous avez fait une erreur

Assurément, vous allez faire des erreurs en utilisant Git. Ce qui est important, c'est d'être capable de s'en sortir après avoir fait une erreur. Je présente les trois commandes qui me semblent le plus pertinentes.

git reset

La commande `git reset <numéro-du-commit>` permet de supprimer les modifications qui ont été faites depuis le *commit* sélectionné. Si vous utilisez le drapeau `--soft`, les *commits* sont supprimés mais pas les modifications. Si vous utilisez le drapeau `--hard`, les *commits* et les modifications sont supprimées.

git revert

La commande `git revert <numéro-du-commit>` permet d'inverser les modifications faites dans un *commit*. Cette commande crée un nouveau *commit* avec exactement l'inverse des modifications du *commit* choisi. Pour plus d'information, voir ce post de StackOverflow.

git commit –amend

La commande `git commit` peut être utilisée avec le drapeau `--amend` pour faire des modifications aux commits qui n'ont pas encore été poussés. Cette commande permet de faire différents types de modifications, je vous propose donc deux sources différentes pour apprendre à la connaître : site de Git et site d'Atlassian.

Il existe plein d'autres commandes git qui peuvent être utilisées, mais qui sont trop spécialisées et ne sont pas assez utilisées pour qu'elles soient présentées dans ce tutoriel. Je vous recommande la documentation de Git pour toutes vos questions, ainsi que StackOverflow. Je vous recommande de plus la cheat sheet de GitHub!