



Introduction à git - Partie B



Contenu

1	Objectifs	3
1.1	Outils	3
1.2	Rappel Sublime Merge	4
2	Opérations de base	5
2.1	Création d'un dépôt git	5
2.2	Clone	6
2.3	Consulter l'état	7
2.4	Stage & commit	8
2.5	Ajout de fichiers	11
2.6	Markdown	12
2.7	Mise en ligne	13
2.8	Retour vers le passé (pas le futur)	14
2.9	Questions générales	15
2.10	Tag	15
3	Branch et Merge	16
3.1	Première branche	16
3.2	Seconde branche	17
3.3	Merge dev02	19
3.4	Merge dev01	19
3.5	Résultat final	20
4	Gitgraph	22
5	Gitflow	23
5.1	Fork	23
5.2	Collaboration parallèle	24
5.3	Pull Request	24
6	Extras	26
6.1	Votre propre projet sur git	26
6.2	Apprendre le branchement Git	27
	Bibliographie	28
7	Annexes	29
	A GIT Commandes	29



AA	Commandes GIT	29
AB	Synchronisation des changements	29
B	Commandes Git les plus utilisées	31
BA	Start a working area	31
BB	Work on the current change	31
BC	Examine the history and state	31
BD	Grow, mark and tweak your common history	31
BE	Collaborate	31



1 | Objectifs

Dans ce laboratoire, nous apprenons les principes de base du contrôle de version [git \[1\]](#). Vous devez déjà avoir réalisé la partie A du laboratoire chez vous pour pouvoir commencer la partie B.

Dans [Chapitre 2](#), nous apprenons les opérations de base pour pouvoir travailler avec Git. Le dépôt créé est ensuite publié sur [GitHub](#). Les fonctions avancées `branch` et `merge` sont appliquées dans un exemple au [Chapitre 3](#). Dans le [Chapitre 5](#), nous travaillons tous ensemble sur un même dépôt. Enfin, il y a quelques travaux optionnels dans le [Chapitre 6](#).



Les réponses aux questions **doivent** être écrites dans le fichier Markdown `answer.md`. Ce fichier se trouve dans le dépôt que vous allez créer à l'étape suivante.



À la fin du laboratoire, assurez-vous que vous avez publié toutes les modifications sur GitHub. Seules les modifications publiées sur GitHub seront évaluées.

1.1 Outils

Dans ce laboratoire, vous utiliserez Sublime Merge comme outil graphique et/ou Git CMD comme ligne de commande.



Git CMD
App



Sublime Merge
App

Figure 1 - Git CMD Ligne de commande Figure 2 - Sublime Merge GUI



N'utilisez le procédé en lignes de commande uniquement si vous êtes à l'aise avec un terminal. Il vous sera nécessaire de connaître les commandes pour naviguer, afficher des informations etc. `cd`, `ls`, `cat`, `touch`, `nano` ...



1.2 Rappel Sublime Merge

L'interface de Sublime Merge est présentée dans la [Figure 3](#):

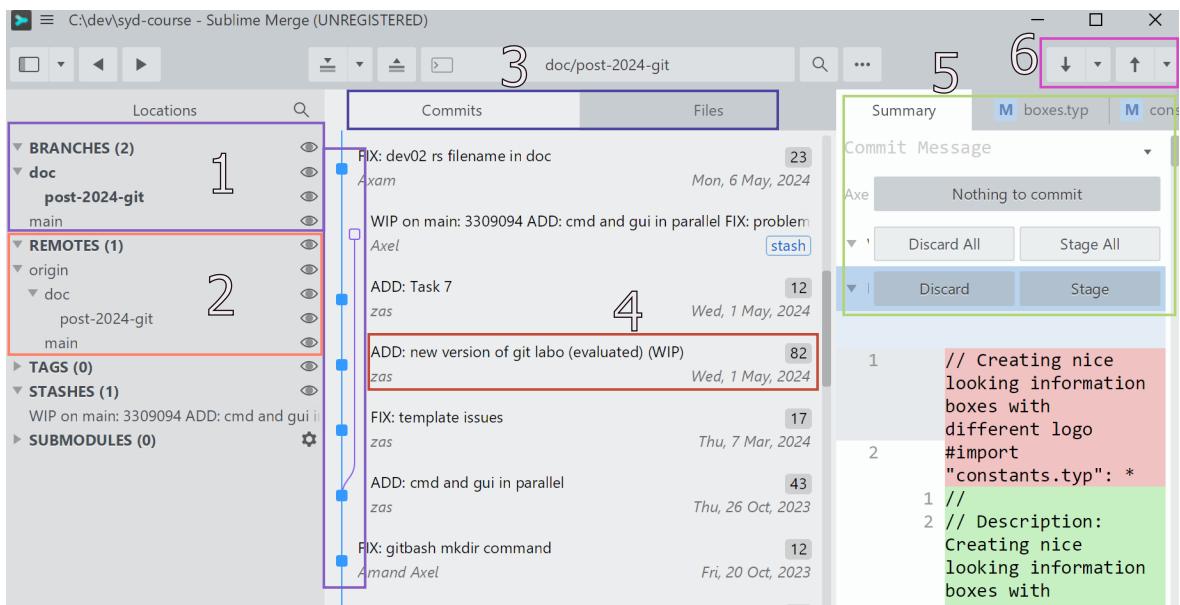


Figure 3 - Sublime Merge GUI

- Branches:** liste des branches du dépôt. Vous pouvez créer, supprimer et renommer des branches. Elles sont aussi affichées dans leur chronologie sous l'onglet **Commits** (point 3).
- Remotes:** liste des dépôts distants, c.à.d. les serveurs sur qui la copie du code est **pushée** et donc stockée.
- Commits/Files:** onglets permettant de passer de la vue chronologique des commits à celle présentant les modifications effectuées pour le commit sélectionné.
- Description du commit:** un commit est composé d'un message, d'un auteur et d'une date.
- Modifications actuelles:** les fichiers modifiés par rapport au dernier commit sont listés ici. Il est possible de sélectionner les fichiers à **commit** en les sélectionnant et en cliquant sur le bouton **Stage**. Tant que les fichiers n'ont pas été **commit**, il est bien sûr possible de retirer un fichier du staging en cliquant sur le bouton **Unstage**, voir de complètement effacer les modifications d'un fichier en appuyant sur **Discard** (⚠ effacement définitif ⚠).
- Pull / Push:** les deux boutons permettent de **pull** - prendre les dernières modifications du dépôt distant - et de **push** - envoyer les modifications locales vers le dépôt distant. Il est aussi possible de **fetch** en cliquant sur la petite flèche à côté du bouton **Pull**, ce qui permet de voir les nouveaux commits sans modifier le dépôt local.



2 | Opérations de base

2.1 Crédit d'un dépôt git

Pour travailler sur une base commune, nous allons créer une copie du dépôt modèle. Cette opération s'appelle *fork* - [Figure 4](#). Les modifications apportées sont reflétées dans votre propre dépôt, pas dans celui qui a été copié. Vous pouvez donc travailler sans craindre de créer des problèmes de synchronisation.

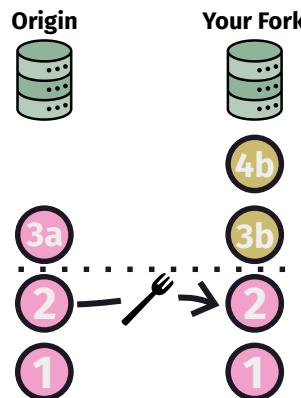
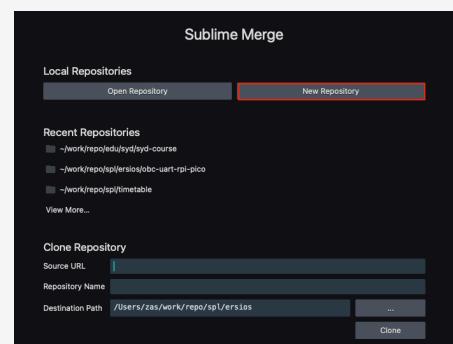


Figure 4 - Fork du dépôt modèle

Si vous souhaitez créer un nouveau dépôt à partir de zéro (non nécessaire dans ce labo), vous pouvez le faire avec la commande suivante ou via l'interface graphique de Sublime Merge :

```
cd /path/to/my/repo
git init my-new-repo
```

Fichier ⇒ Nouveau dépôt



Le dépôt nouvellement créé sera vide, sans `remote`, c'est-à-dire sans connexion à un dépôt distant (par exemple GitHub). Vous pouvez ajouter un dépôt distant plus tard avec la commande `git remote add origin <url>` ou via le GUI.



Créez un fork du dépôt modèle en utilisant le lien <https://classroom.github.com/a/wfORJke1>.



Pour cela, vous devrez vous connecter à GitHub. Acceptez l'invitation Classroom - *Figure 6*, qui vous donnera le lien vers VOTRE propre dépôt - *Figure 7*.

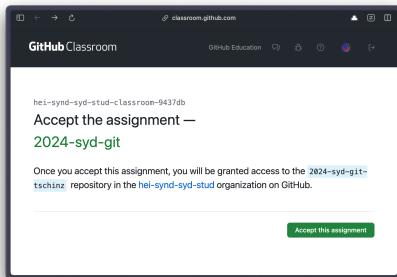


Figure 6 - Lien d'invitation du fork

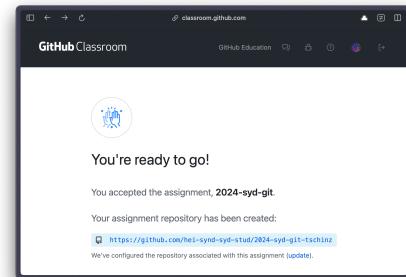


Figure 7 - Lien pour le clonage du dépôt



Notez le lien de votre dépôt pour la prochaine étape.
Profitez-en pour consulter votre dépôt en ligne sur le lien donné en retirant le .git à la fin de l'URL.

2.2 Clone

Après avoir créé le fork, vous recevrez un lien vers votre propre dépôt - *Figure 7*. Clonez-le sur votre ordinateur local à l'emplacement de votre choix.

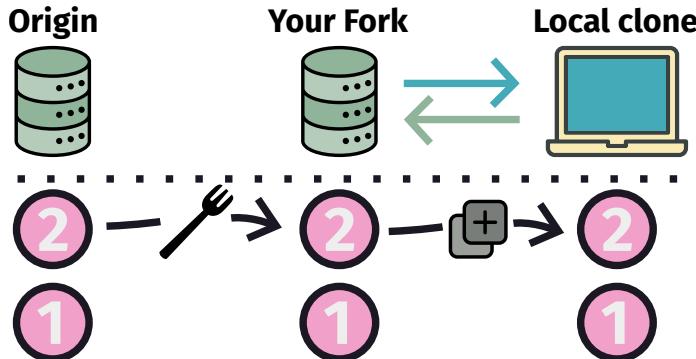


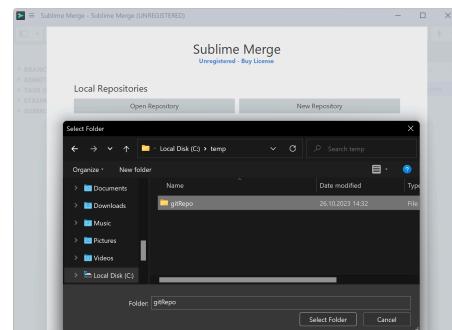
Figure 8 - Clonage

Commandline

```
cd /path/to/my/repo
git clone <linkurl>.git

# Example
git clone https://github.com/hei-synd-syd-stud/
2025-syd-git-tschinz.git
```

GUI - CTRL+T ⇒ Paste Source URL ⇒ Select Folder





2.3 Consulter l'état

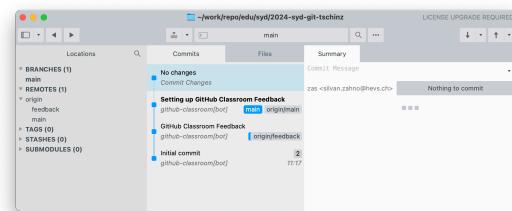
Obtenez des informations sur votre dépôt en utilisant les commandes suivantes :

Commandline

```
git status  
git log --oneline
```

GUI

See the status in the main window.



Actuellement, votre dépôt n'a aucun changement vu que rien n'a été modifié. Sur Sublime Merge, en haut de la fenêtre, il est noté **No changes**.



2.4 Stage & commit

Nous allons maintenant modifier des données dans le dépôt, en l'occurrence le fichier `answers.md`.

Exercice 0

Ouvrez le document `answers.md` et remplacez :



- `<students-firstname>` par votre prénom
- `<students-lastname>` par votre nom de famille
- `<github-username>` par votre nom d'utilisateur GitHub

Par la suite, les sections **Exercices** devront être répondues dans ce même fichier `answers.md`.

Toujours **SAUVEGARDER** votre fichier après avoir répondu à une tâche.

Comme présenté sous le [Chapitre 2.3](#), contrôlez le statut de votre dépôt.



Exercice 1

Quel est son statut ? Que signifie-t'il ?

Nous allons enregistrer les changements localement. Pour ce faire, il est nécessaire de **Committer** les modifications.



i

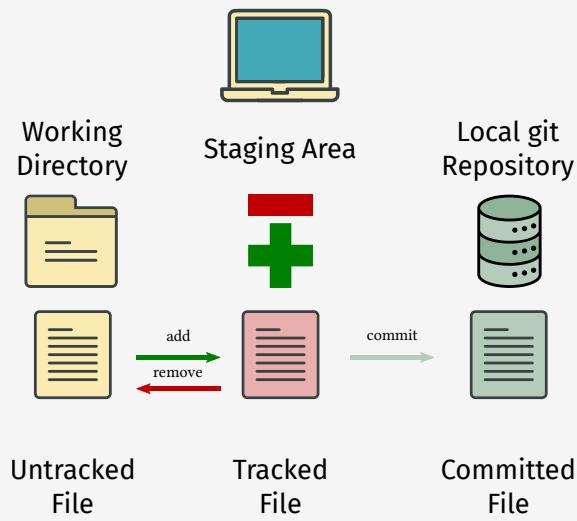


Figure 9 - Types d'opérations locales de Git

Votre dépôt git local est composé de trois sections gérées par git : 5

- Le **Working Directory** est un répertoire qui contient la version actuelle de vos fichiers (aux yeux de votre système d'exploitation, c'est un répertoire de fichiers normal).
- **Stage** contient les modifications à inclure dans le prochain commit.
- Le **Head** pointe vers l'endroit de l'arborescence du dépôt où le prochain commit doit être effectué.



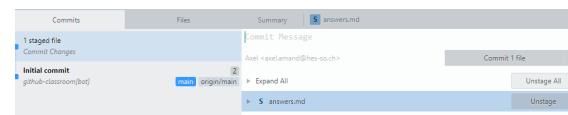
Pour enregistrer les modifications *localement*, vous devez d'abord sélectionner les modifications qui seront enregistrées dans le dépôt. Ce processus s'appelle **Stage**.

Commandline

```
git add answers.md
```

GUI

Select the file in the upper-right corner, and click on **Stage**.



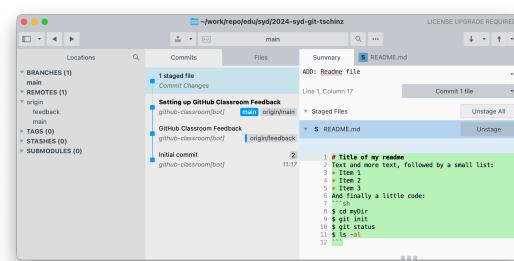
Une fois les modifications sélectionnées, le commit est effectué. Un commit est un instantané de votre dépôt à un moment donné et une fois effectué, l'état du dossier est enregistré dans le repo. Le **Commit** est identifié par un identifiant unique (hash) et contient des informations sur l'auteur, la date et le message de commit, qui doit refléter les modifications apportées :

Commandline

```
git commit -m "CHG: personal data in answers.md"
```

GUI

Commit Message \Rightarrow CHG: personal data in answers.md \Rightarrow **Commit 1 file**



Commitez vos changements

Exercice 2

Après avoir réalisé un commit comme expliqué précédemment, répondez aux questions suivantes:



- Que signifie le message de commit proposé en exemple ?
- Peut-on créer un commit sans message ?
- Qu'est-ce qui change désormais dans le dépôt ?
- Le dépôt distant (ici Github) est-il à jour avec nos modifications ?



2.5 Ajout de fichiers

Créez ensuite un fichier vide nommé `README.md` dans le répertoire principal de votre dépôt:

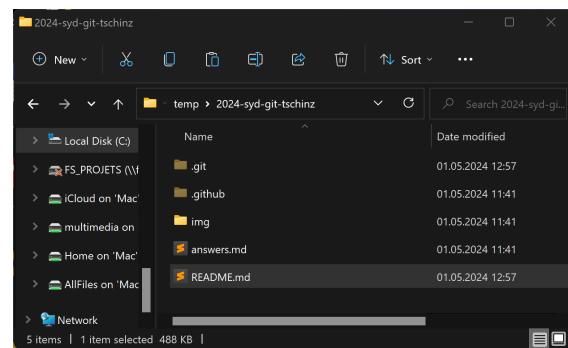
Commandline

```
touch README.md
```

GUI

(Windows) `C:\path\to\repo\` ⇒ New ⇒ Any File
⇒ `README.md`

(Linux&MacOS) `/path/to/repo` ⇒ New ⇒ Any File
⇒ `README.md`



Exercice 3

Consultez à nouveau le statut de votre dépôt, que constatez-vous ?



Réalisez un commit avec le nouveau fichier ainsi que les réponses aux exercices 2 et 3.

Exemple de dépôt

Un simple dépôt Git, composé de cinq commits, peut être représenté de la manière suivante. La position `Head` est une référence à un commit qui représente l'état actuel/la vue actuelle du repos, ici la dernière modification:

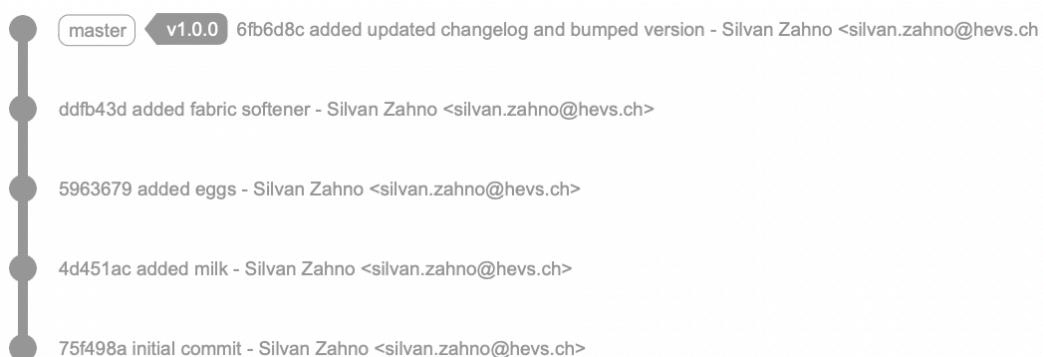


Figure 10 - Cinq commits sur le repo local, chaque commit possède son propre identifiant



2.6 Markdown

Modifiez le fichier `README.md` avec un éditeur de texte pour qu'il ressemble à peu près à ce qui suit:

My README.md

The goal of this document is :

- Approach the Markdown syntax
- Show the result on a Github repository

How-To

1. Identify constructs from given result
2. Write a corresponding Markdown syntax

Git - a command example

Git can be ran through command lines and the current status shown with:

```
# Going inside directory
cd /my/git/dir
# Asking for current status
git status
git log --oneline
```

It is also important to note that:

"It is easy to shoot your foot off with git, but also easy to revert to a previous foot and merge it with your current leg."

— Jack William Bell

Figure 11 - Structure Markdown à reproduire

Pour vous aider à écrire et prévisualiser votre fichier markdown, utilisez un outil comme présenté dans le chapitre Labo Part A - Markdown.

Voici un aide-mémoire Markdown pour vous aider à créer votre fichier `README.md`:

Title
Subtitle
Sub-subtitle

Some text

Some code:
```bash  
cd myDir  
```

A list:
* Item 1
* Item 2

> Quoting something

A numbered list:
1. Item 1
1. Item 2



Complétez votre fichier `README.md`.



Une fois satisfait, commitez vos modifications.



2.7 Mise en ligne

Actuellement, tout le travail effectué n'est sauvegardé que *localement*. Pour le mettre en ligne, il est nécessaire de pusher les commits sur le dépôt distant (Github) afin d'en avoir une copie et, dans le cadre d'un travail de groupe, permettre aux autres de le voir:

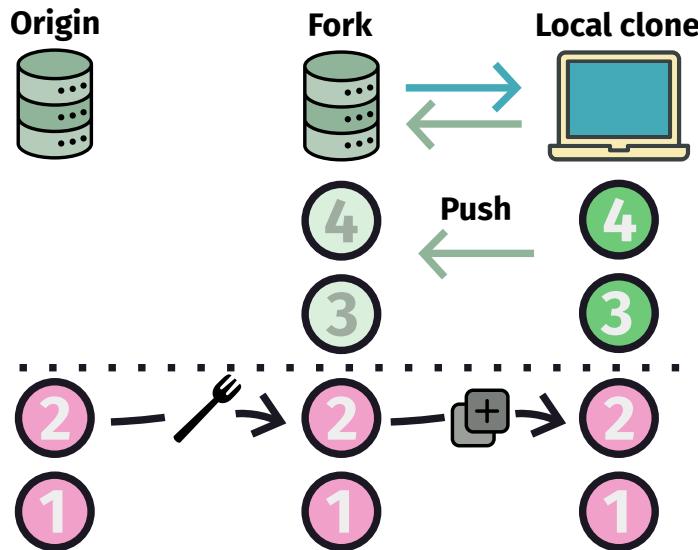


Figure 12 - Push sur le dépôt distant

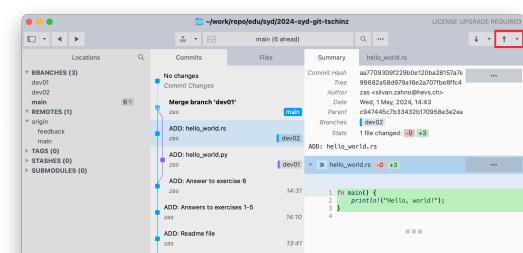
Pour ce faire, une fois toutes les modifications commitées:

Commandline

```
git push origin main
```

GUI

Top Right Arrow \Rightarrow Push changes



Rendez-vous en ligne sur Github pour voir le résultat de votre travail.
Vous devriez voir le fichier `README.md` avec le contenu que vous avez créé sous forme lisible.



2.8 Retour vers le passé (pas le future)

Git est un système de gestion de versions. Il est donc possible de revenir à une version antérieure du dépôt. Pour cela, il faut d'abord identifier le commit sur lequel vous souhaitez revenir. Ici, nous retournons au tout premier commit.

Notez que chaque commit est accompagné d'un « hash » ou d'une « somme de contrôle » (de type sha1). Ce hash est un identifiant unique permettant d'isoler un commit précis. Ce dernier est visible en cliquant sur un commit pour SublimeMerge, ou en utilisant la commande suivante:

```
git log --oneline
```

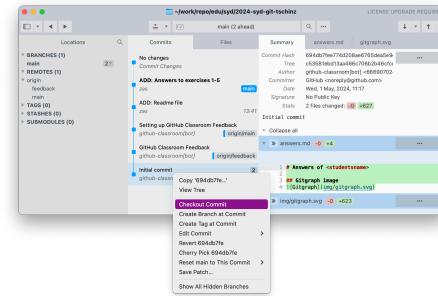
Pour changer de commit, on effectue l'opération de checkout:

Commandline

```
git checkout <SHA1>
# for example
git checkout 694db7f
```

GUI

Select First Commit (Initial Commit) ⇒ Right click ⇒ Checkout commit



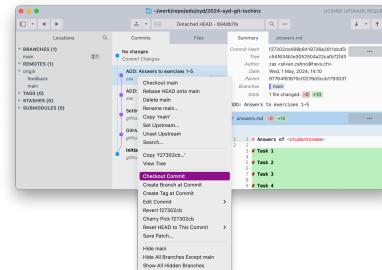
Ensuite, revenez au dernier commit de la même manière. Hormis le hash, il est possible de revenir au dernier commit en utilisant le nom de la branche, ici `main`.

Commandline

```
git checkout main
```

GUI

Branches (1) ⇒ main ⇒ checkout main



Exercice 4

Que se passe-t-il lorsque vous revenez au commit « Initial commit » ? Que se passe-t-il lorsque vous revenez au dernier commit ?



2.9 Questions générales



Exercice 5

Quelle est la différence entre le dépôt local et le dépôt distant ?

Que se passerait-il si vous supprimiez le dépôt local ?



Exercice 6

Avec toutes ces manipulations, qu'en est-il du [dépôt originel](#), celui ayant été forké ?

A-t'il été modifié ?



Réalisez un commit contenant les réponses précédentes.

2.10 Tag

Afin d'identifier des commits clés, il est possible de créer des tags. Un tag est un pointeur vers un commit particulier. Il est souvent utilisé pour marquer des versions de code, par exemple v1.0, v2.0, etc.

Pour marquer la fin de ce chapitre, créez un tag chapter2 sur le dernier commit. Pour cela:

Commandline

```
git tag chapter2
git push origin chapter2
```

GUI

- Right click on commit ⇒ Create Tag at Commit
⇒ chapter2 ⇒ Enter
- Right click on tag ⇒ Tag chapter2 ⇒ Push Tag chapter2



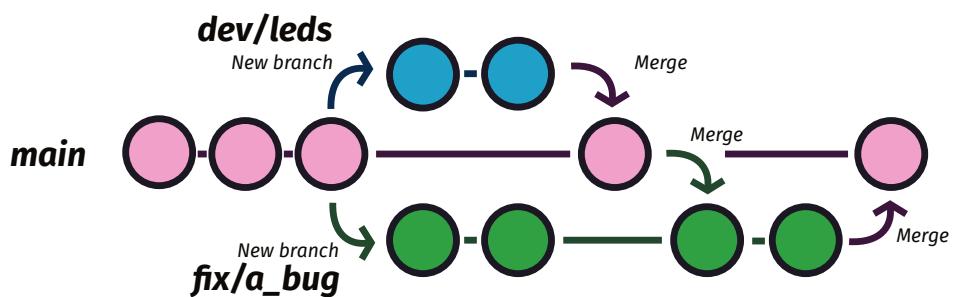
Ce chapitre est à compléter sur le même dépôt que précédemment.



Soyez sûr d'avoir commis toutes vos modifications avant de commencer ce chapitre, et que ces dernières aient été taguées par chapter2.

3 | Branch et Merge

Jusqu'à présent, nous avons utilisé les fonctions de base de git - [ligne de commits rose](#). Il existe aussi les fonctions `branches` et `merge`, que Git a grandement simplifiées par rapport aux outils existants auparavant:



Les branches permettent de travailler en parallèle sur plusieurs versions d'un même projet. Par exemple, vous pouvez créer une [branche pour développer une nouvelle fonctionnalité](#), pendant que votre collègue travaille sur un [patch pour fixer un bug](#). L'action de fusionner (`merge`) permet de rassembler les modifications apportées sur une branche dans une autre.



Git n'est pas omniscient. Si deux branches ont modifié les mêmes fichiers, Git ne saura pas comment les fusionner et vous devrez manuellement résoudre les conflits.

3.1 Première branche

La première branche simule le travail d'une première personne sur un nouveau fichier, `hello_world.py`. Cela pourrait être n'importe quel fichier: code, image, dessin Inventor, simulation Matlab ...

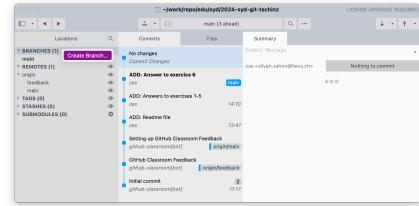
- Créez une branche de développement `dev01` dans votre repo local:

Commandline

```
git checkout -b dev01
```

GUI

Branches ⇒ Create Branch ⇒ dev01



2. Créez un commit sur cette branche :

- Créez un fichier `hello_world.py`
- Remplissez-le avec le code suivant :

```
print("Hello, world!")
```



Réalisez un commit comme présenté au chapitre [Chapitre 2.4](#), avec un message clair.

3.2 Seconde branche

La seconde branche simule le travail d'une deuxième personne sur un autre fichier, `hello_world.rs`.

En supposant que la branche ait été créée en même temps que l'autre, il est donc nécessaire de commencer par revenir sur le même commit que précédemment.

1. Revenez sur la branche `main`:

Commandline

```
git checkout main
```

GUI

`main` ⇒ Right click ⇒ Checkout `main`



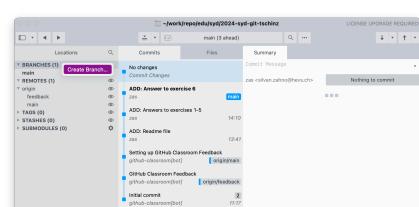
2. A partir de la branche `main`, créer une nouvelle branche de développement `dev02`:

Commandline

```
git checkout -b dev02
```

GUI

Branches ⇒ Create Branch ⇒ `dev02`



3. Créez un commit sur cette branche:

- Créez un fichier `hello_world.rs`
- Remplissez-le avec le code suivant:



```
fn main() {  
    println!("Hello, world!");  
}
```



Réalisez un commit comme présenté au chapitre [Chapitre 2.4](#), avec un message clair.



3.3 Merge dev02

Maintenant que les deux branches ont reçues des commits, il est temps de les fusionner dans la branche principale `main`.

1. Revenez sur la branche `main`:

Commandline

```
git checkout main
```

GUI

`main` ⇒ Right click ⇒ Checkout `main`



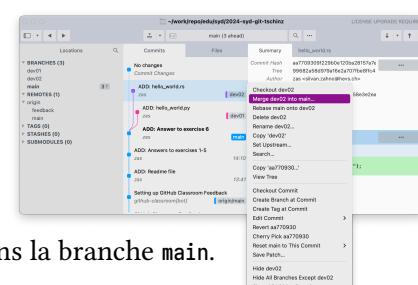
2. Merge de la branche `dev02` dans `main`:

Commandline

```
git merge dev02
```

GUI

Select Commit ⇒ Merge `dev02` into `main` ...



Le travail de la branche `dev02` est maintenant intégré dans la branche `main`.

3.4 Merge dev01

De la même manière, il est maintenant temps de fusionner la branche `dev01` dans `main`.

1. Assurez-vous d'être sur la branche `main`:

Commandline

```
git checkout main
```

GUI

`main` ⇒ Right click ⇒ Checkout `main`



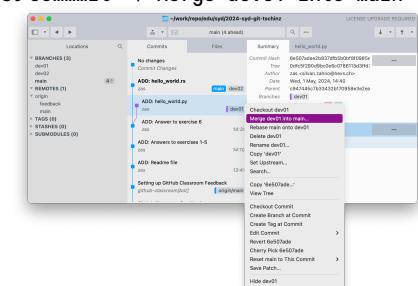
2. Merge de la branche `dev01` dans `main`.

Commandline

```
git merge dev01
```

GUI

Select Commit ⇒ Merge `dev01` into `main` ...





3.5 Résultat final

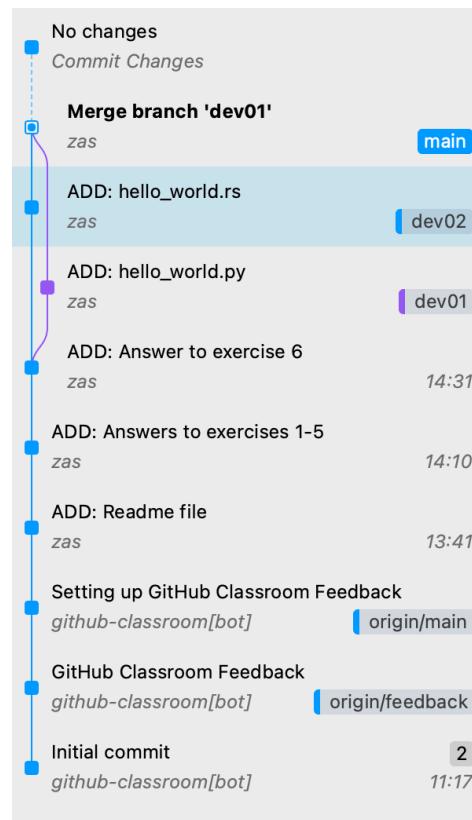


Figure 22 - Statut après la fusion des branches `dev01` et `dev02` du repo

Il se peut que, par défaut, le dépôt n'affiche qu'une ligne bleue. Comme le travail est très linéaire, Sublime Merge cache les commits. Vous pouvez forcer



l'affichage de tous les commits en cliquant sur le bouton sur la ligne de vie des commits.

1. Poussez votre dépôt sur le serveur distant GitHub. **Il est nécessaire de réaliser trois push pour pousser chaque branche séparément:**

Commandline

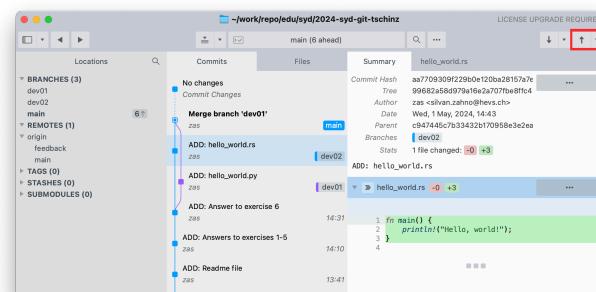
```
git push origin main
git push origin dev01
git push origin dev02
```

GUI

Checkout `dev01` \Rightarrow Top Right Arrow \Rightarrow Push changes

Checkout `dev02` \Rightarrow Top Right Arrow \Rightarrow Push changes

Checkout `main` \Rightarrow Top Right Arrow \Rightarrow Push changes





2. Taggez la branche `main` avec le tag `chapter3` comme présenté sous [Chapitre 2.10](#). N'oubliez pas de pousser le tag !



Le statut de votre repo doit être similaire à [Figure 22](#) plus incluse les tags `chapter2` et `chapter3`. Cela fait partie de l'évaluation.



Ce chapitre est à réaliser dans le même dépôt que précédemment.



Assurez-vous d'avoir commisé toutes vos modifications avant de commencer ce chapitre, et que ces dernières aient été taguées par chapter3.

4 | Gitgraph

Dans la [Figure 24](#) est représenté un exemple d'un dépôt git. Nommez tous les éléments visibles sur l'image (points 1- 10).

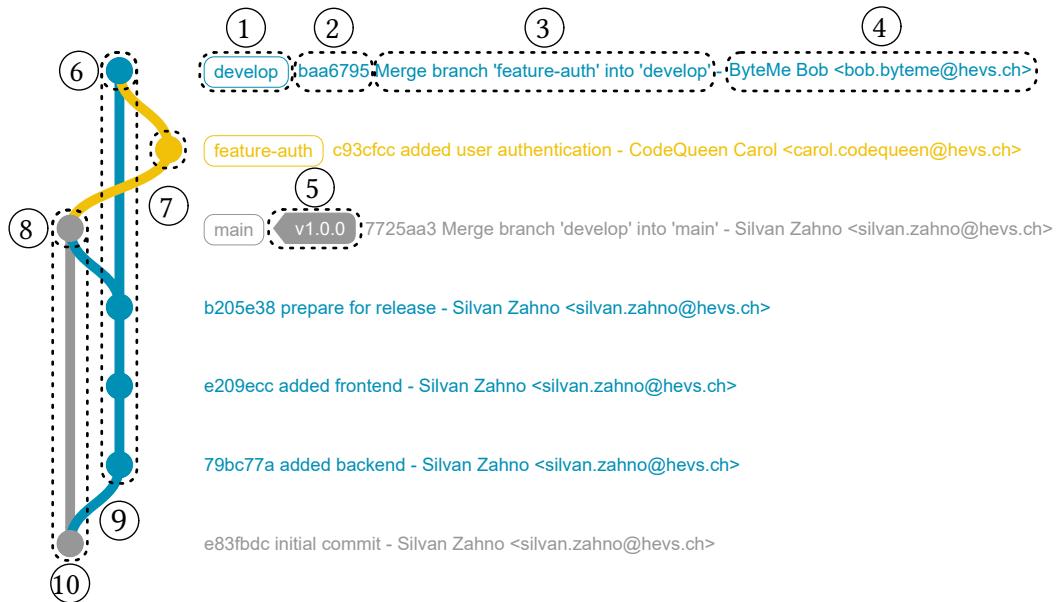


Figure 24 - Exemple d'une gitgraphhe

Exercice 7

1. Nommez tous les éléments visibles dans l'image [Figure 24](#) (points 1 à 10)



Commitez vos changements

1. Poussez votre dépôt local vers votre dépôt distant GitHub
2. Taggez et poussez le tag chapter4 selon [Chapitre 2.10](#)



Félicitation, le travail sur votre premier dépôt Github est terminé.



Ce chapitre concerne **un autre dépôt** que celui utilisé précédemment.



Assurez-vous d'avoir commité toutes vos modifications avant de commencer ce chapitre, que ces dernières aient été taguées par chapter4 et pushées en ligne. Les modifications, commits, tags ... qui ne sont pas disponibles en ligne compteront comme travail non effectué pour la notation du laboratoire.
Vous pouvez contrôler en ligne sous [Github](#) si votre dépôt est bien disponible et à jour.

5 | Gitflow

Pour cette tâche, utilisez la philosophie Gitflow présentée dans le cours. Vous allez tous collaborer sur le dépôt Git suivant, comme si vous formiez une équipe de développement :

<https://github.com/hei-synd-syd/syd-gitflow> [2]

Il s'agit d'un dépôt Git public hébergé sur Github.

5.1 Fork

Pour des raisons de sécurité, vous n'êtes pas autorisé à travailler directement sur ce dépôt. Vous devez créer votre propre copie (**fork**) pour pouvoir effectuer des modifications. Veuillez donc créer un **fork** de ce dépôt dans votre compte GitHub. Pour ce faire, utilisez le bouton **Fork** dans l'interface web de Github.

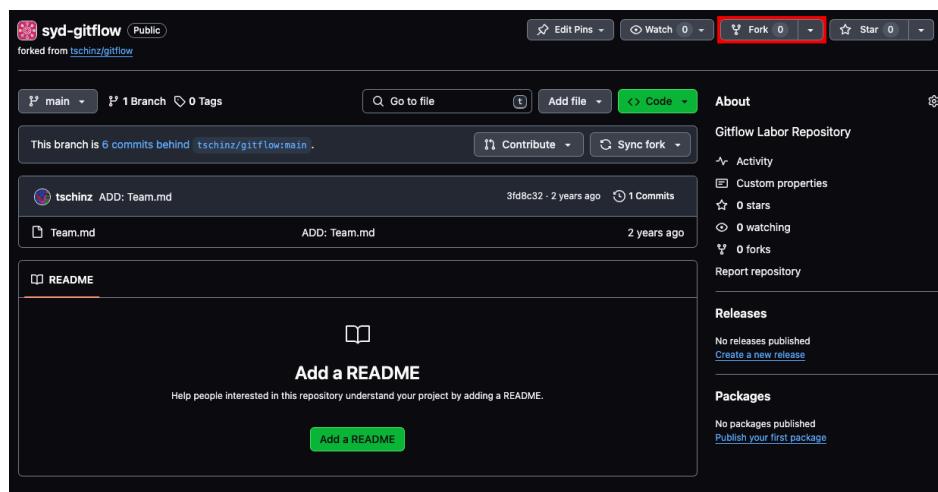


Figure 25 - Bouton Fork pour un dépôt GitHub

Clonez ensuite ce dépôt forké. L'URL de votre nouveau dépôt ressemblera à ceci:

```
git clone https://github.com/<username>/syd-gitflow.git
```



5.2 Collaboration parallèle

Modifiez le fichier `Team.md`. Remplacez votre numéro donné par votre prénom et votre nom.



Committez et Pushez votre branche vers votre fork du dépôt sur GitHub.



Vous allez tous modifier le même fichier. Pour éviter tout conflit, ne modifiez que la ligne qui vous concerne.

Demandez au professeur ou à l'assistant votre numéro de ligne !

5.3 Pull Request

Maintenant que vos modifications sont prêtes, il est encore nécessaire de réaliser une fusion sur le dépôt originel.

Comme vous ne possédez pas les droits, il est possible de réaliser une demande de fusion - pull request - sur le dépôt d'origine.

Ainsi, l'administrateur pourra accepter les modifications, vous demander de corriger des éléments avant d'accepter la fusion, discuter de problèmes etc.

Pour ça, utilisez l'interface du site web de GitHub. **Contribute** ⇒ **Open pull request**.

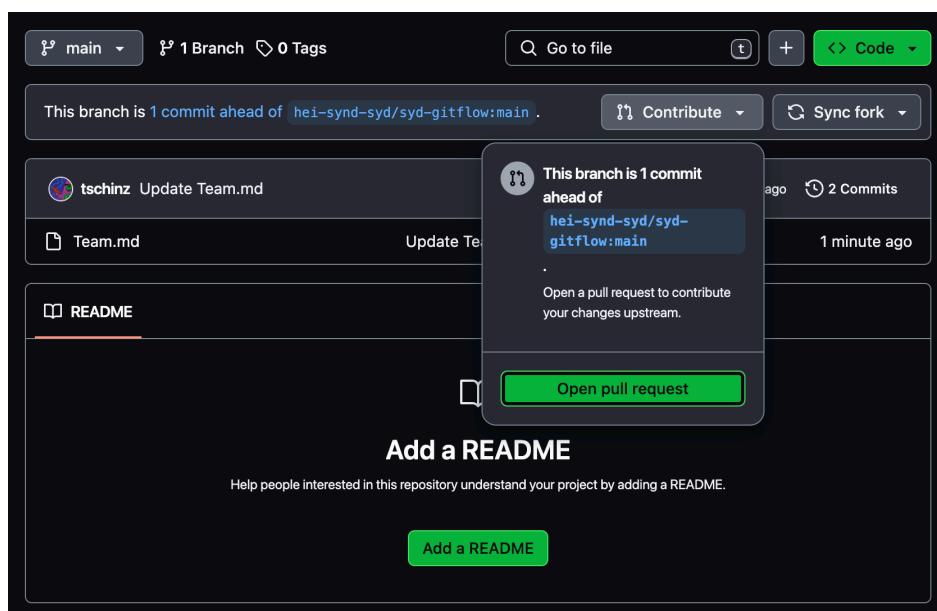


Figure 26 - Créer une Pull Request sur Github

Vous devrez donner un titre parlant ainsi qu'une courte description à votre pull request. Il est possible d'écrire la description en Markdown - [Chapitre 2.6](#).

Les pull-requests incomplètes, mal décris, mal intitulées ... seront refusées.



Une fois les pull requests prêtes, les fusions sont effectuées par le professeur et les assistants.



Contrôlez sur [Github](#) que votre pull request a correctement été **ouverte**, qu'elle contient un **titre** et une **description** clairs, et que vous avez modifié uniquement votre **numéro de ligne**. Les pull requests manquantes compteront comme travail non effectué pour la notation du laboratoire.



6 | Extras

Ce chapitre optionnel peut être lancé à condition que les tâches précédentes aient été effectuées. Il y a 2 tâches à faire :

1. Mettre votre propre projet sur Github et lui fournir un `README.md` et un CI/CD.
2. Suivre le tutoriel sur le site web « Learn Git Branching »

6.1 Votre propre projet sur git

- Mettez un projet sur lequel vous travaillez actuellement sur Github.
- Créez un fichier `README.md` pour le projet en utilisant la [syntaxe Markdown](#). Le fichier `README.md` doit contenir les éléments suivants:
 - Titre
 - Image
 - Description du projet
 - Explication de l'exécution et de l'utilisation du projet
 - Liste des auteurs
- Maintenant, créez une action github pour transformer le `README.md` en PDF à chaque « push ». Pour cela, trouvez une [action github](#) appropriée et ajoutez-la à votre projet.



Si vous avez besoin d'aide pour créer l'action, consultez [ce conseil](#).

The screenshot shows the GitHub Marketplace interface with the 'Actions' category selected. The main search bar is empty. On the left, there's a sidebar with categories like Types, Apps, and Actions. The Actions section has a sub-header 'Actions' with the subtext 'An entirely new way to automate your development workflow.' Below this, it says '20351 results filtered by Actions'. There are four main action cards displayed:

- Close Stale Issues**: By actions (Creator verified by GitHub). Description: Close issues and pull requests with no recent activity. Stars: 1.1k.
- Upload a Build Artifact**: By actions (Creator verified by GitHub). Description: Upload a build artifact that can be used by subsequent workflow steps. Stars: 2.5k.
- Download a Build Artifact**: By actions (Creator verified by GitHub). Description: Download a build artifact that was previously uploaded in the workflow by the upload-artifact action. Stars: 1.1k.
- Setup Java JDK**: By actions (Creator verified by GitHub). Description: Set up a specific version of the Java JDK and add the command-line tools to the PATH. Stars: 1.3k.

Figure 27 - Github-Actions Marktplatz



6.2 Apprendre le branchement Git

Suivez le tutoriel sur <https://learngitbranching.js.org>.

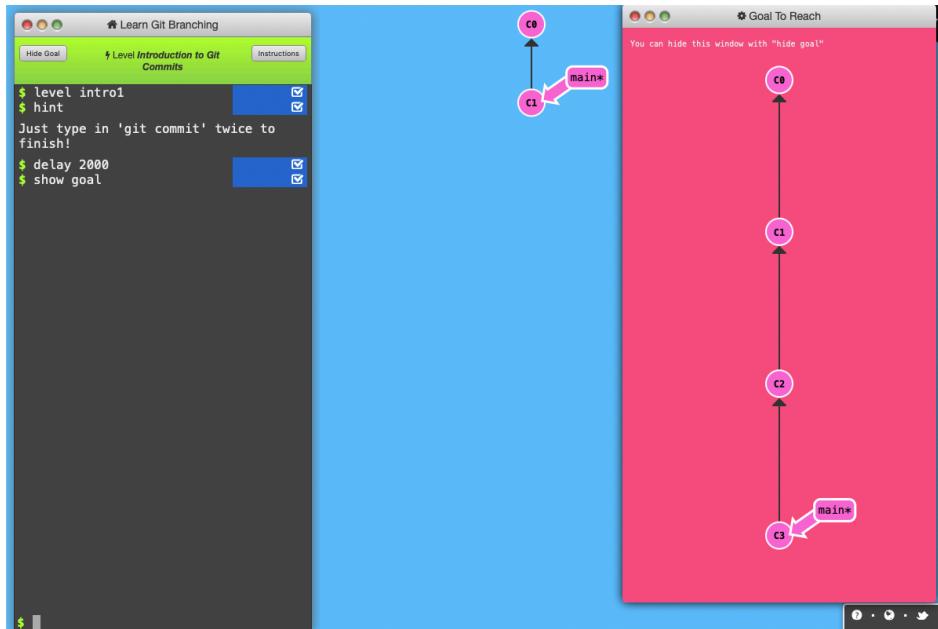


Figure 28 - Lerne Git Branching Website



Bibliographie

- [1] T. Linus, « Git ». Consulté le: 25 avril 2023. [En ligne]. Disponible sur: <https://git-scm.com/>
- [2] tschinz, « Tschinz/Gitflow ». Consulté le: 25 avril 2023. [En ligne]. Disponible sur: <https://github.com/tschinz/gitflow>
- [3] gitlab, « Git Cheatsheet ». 2023.
- [4] « GitHub Git Spickzettel ». Consulté le: 25 avril 2023. [En ligne]. Disponible sur: <https://training.github.com/downloads/de/github-git-cheat-sheet/>



7 | Annexes

A | GIT Commandes

[Github git cheatsheet \[3\], \[4\]](#)

AA Commandes GIT

```
git status
```

Liste tous les fichiers nouveaux ou modifiés qui sont prêts à être commit.

```
git diff
```

Affiche les modifications de fichiers qui n'ont pas encore été indexées.

```
git add [file]
```

Ajoute le fichier au versionning.

```
git diff --staged
```

Affiche les différences entre l'index (« staging area ») et la version actuelle du fichier.

```
git reset [file]
```

Retire le fichier de l'index (« staging area ») mais ne le supprime pas du disque.

```
git commit -m "[descriptive message]"
```

Inclut tous les fichiers actuellement indexés de façon permanente dans l'historique des versions.

AB Synchronisation des changements

Enregistrement d'un référentiel externe (URL) et échange de l'historique du repository.

```
git fetch [remote]
```

Télécharge l'historique complet d'un repository externe.

```
git merge [remote]/[branch]
```

Intègre la branche externe dans la branche locale.



```
git push [remote] [branch]
```

Pousse la branch locale (donc tous les commits de celle-ci) sur GitHub.

```
git pull
```

Récupération de l'historique du repository externe et intégration des modifications sur le repository local.



B | Commandes Git les plus utilisées

BA Start a working area

- `clone` - Clone a repository into a new directory
- `init` - Create an empty Git repository or reinitialize an existing one

BB Work on the current change

- `add` - Add file contents to the index
- `mv` - Move or rename a file, a directory, or a symlink
- `reset` - Reset current HEAD to the specified state
- `rm` - Remove files from the working tree and from the index

BC Examine the history and state

- `log` - Show commit logs
- `show` - Show various types of objects
- `status` - Show the working tree status

BD Grow, mark and tweak your common history

- `branch` - List, create, or delete branches
- `checkout` - Switch branches or restore working tree files
- `commit` - Record changes to the repository
- `diff` - Show changes between commits, commit and working tree, etc
- `merge` - Join two or more development histories together
- `rebase` - Reapply commits on top of another base tip
- `tag` - Create, list, delete or verify a tag object signed with GPG

BE Collaborate

- `fetch` - Download objects and refs from another repository
- `pull` - Fetch from and integrate with another repository or a local branch
- `push` - Update remote refs along with associated objects