



Introduction au git - Partie B



Contents

1 Objectifs	2
2 Opérations de base	3
3 Branch et Merge	11
4 Gitgraph	14
5 Gitflow	15
6 Extras	17
A GIT Commandes	19
B Commandes Git les plus utilisées	20
Bibliographie	21



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 référentiel créé est ensuite publié sur [GitHub](#). Les fonctions avancées `branch` et `merge` sont essayées dans un exemple dans le Chapitre 3. Dans le Chapitre 5, nous travaillons tous ensemble sur un repository. 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 référentiel 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.



2 | Opérations de base

2.1 Crée un repository git

Créez un fork du template repository à l'aide du lien suivant <https://classroom.github.com/a/O0eniBP2> (Fig. 1). Pour cela, vous devez vous connecter à GitHub.

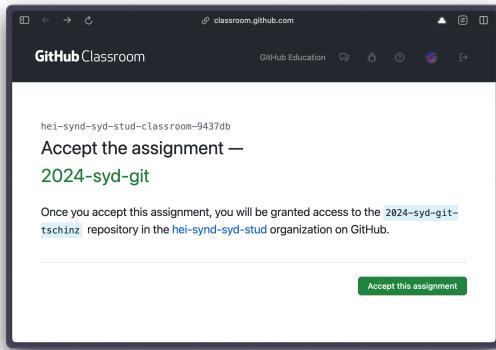


Fig. 1. – Lien d'invitation au forking

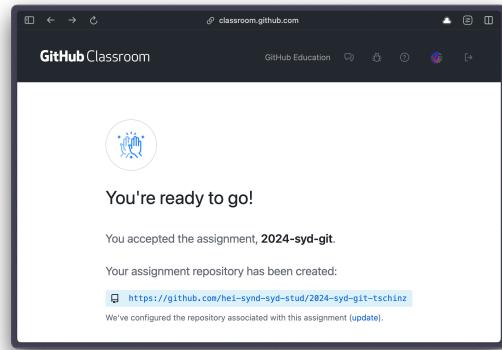


Fig. 2. – Lien pour le clonage de repository

2.2 Clone

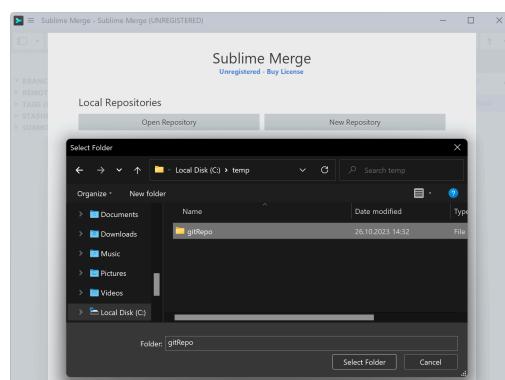
Après avoir créé le fork, vous recevrez un lien vers votre propre repository (Fig. 2). Clonez-le sur votre ordinateur local à l'emplacement de votre choix.

Commandline

```
git clone <linkurl>.git
e.g.
git clone https://github.com/hei-synd-synd-stud/2024-syd-git-tschinz.git
```

GUI

CTRL+T ⇒ Paste Source URL ⇒ Select Folder



Exercice 1



Que trouve-t-on dans le répertoire après le clonage du Git Repo ? A quoi servent les différents fichiers et dossiers ?

Notez les réponses dans le fichier `answers.md` !



2.3 Consulter l'état

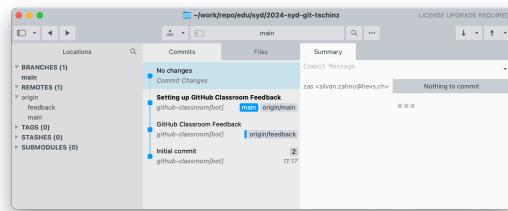
Obtenez des informations sur votre repo en utilisant les commandes suivantes :

Commandline

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

GUI

See the status in the main window.



2.4 Ajouter un fichier

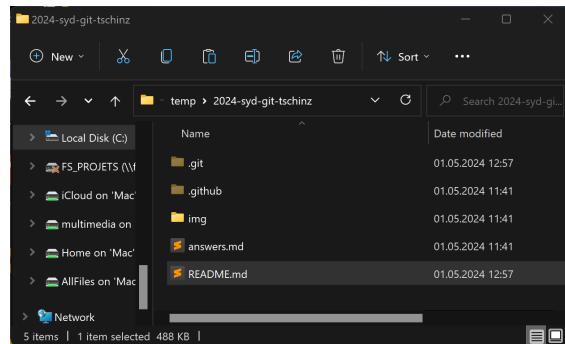
Créez maintenant un fichier vide nommé `README.md` dans le répertoire principal de votre repos.

Commandline

```
touch README.md
```

GUI

C:\temp ⇒ New ⇒ Any File ⇒ README.md



Utilisez les commandes précédentes pour récupérer à nouveau les informations sur votre repo.



Exercice 2

Qu'est-ce qui a changé dans `git status` et `git log -oneline`? Et pourquoi?
Écrivez la réponse!



Votre référentiel git local est composé de trois sections gérées par git :

- 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 Git Repo où le prochain commit doit être effectué.

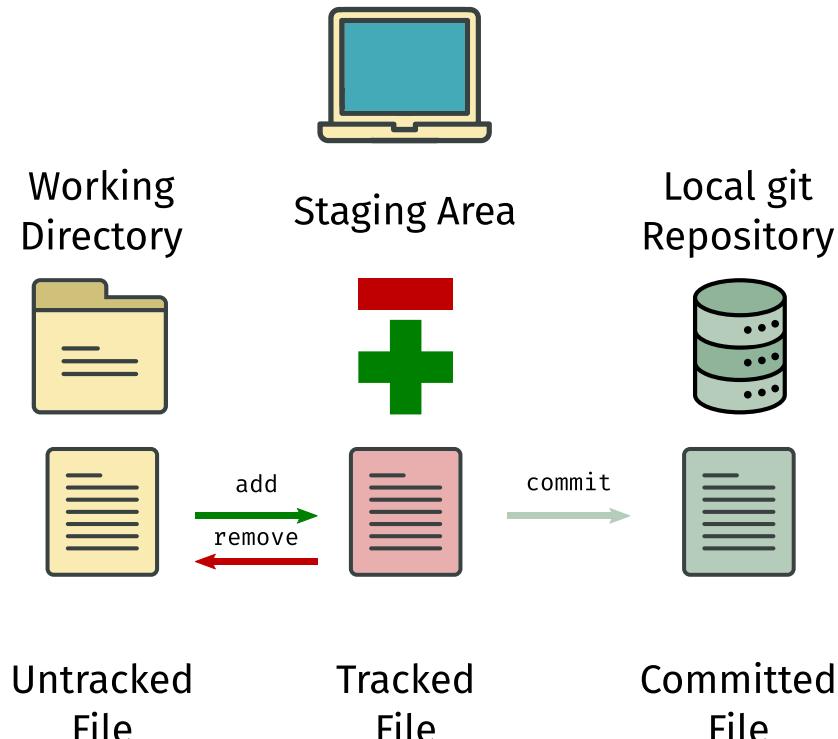


Fig. 3. – Types d'opérations locales de Git

Un simple repos 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.

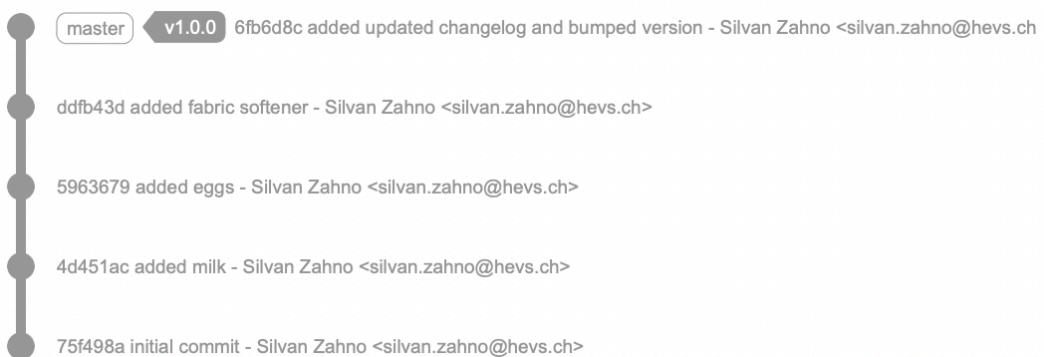


Fig. 4. – Cinq commits sur le repo local, chaque commit possède son propre identifiant



2.5 Ajouter le fichier au repo

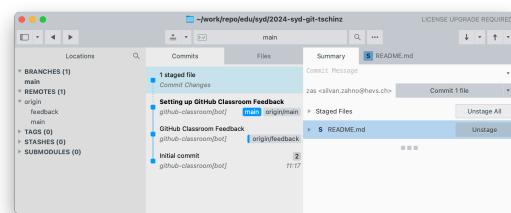
Ajoutez le fichier précédemment créé README.md au stage avec la commande :

Commandline

```
git add README.md
```

GUI

Untracked Files \Rightarrow README.md \Rightarrow Stage



Exercise 3

Consultez à nouveau les info de `git status` sur votre repo, que constatez-vous ?
Ecrivez la réponse!

Modifiez le fichier README.md avec un éditeur de texte et insérez le texte suivant (syntaxe Markdown) :

```
# Title of my readme
Text and more text, followed by a small list :
* Item 1
* Item 2
* Item 3

And finally a little code:
```sh
$ cd myDir
$ git init
$ git status
$ ls -al
```

```



Exercise 4

Consultez à nouveau les info de `git status` sur votre repo, que constatez-vous ?
Ecrivez la réponse!



2.6 Ajouter de nouvelles modifications

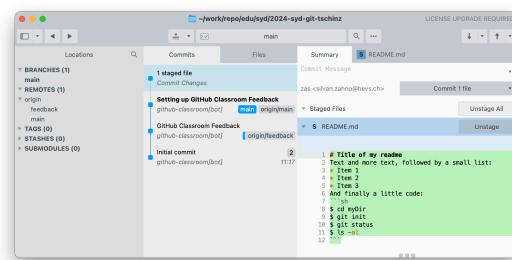
Ajoutez la dernière version du fichier README.md au stage.

Commandline

```
git add README.md
```

GUI

Untracked Files \Rightarrow README.md \Rightarrow Stage



2.7 Exécuter un commit

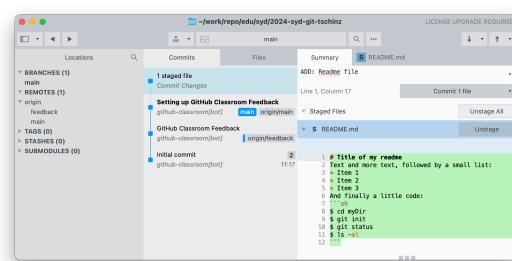
Exécutez maintenant un commit avec la commande suivante :

Commandline

```
git commit -m "ADD: README file."
```

GUI

Commit Message \Rightarrow ADD: README file. \Rightarrow Commit 1 file



L’option `-m` permet de spécifier directement le message du commit. Ce message doit être auto-explicatif. Il correspond à la description des modifications. Il est possible d’insérer un bloc de texte, par exemple via un éditeur de texte, sans utiliser l’option `-m`.

Vos modifications sont maintenant publiées dans votre repo Git local. Bravo !



2.8 Plus d'informations

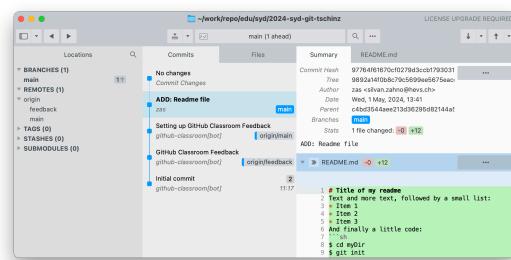
Quelles informations obtiendrez-vous maintenant avec la commande :

Commandline

```
git log --oneline
```

GUI

Select First Commit \Rightarrow See all informations



Expliquez clairement toutes les informations contenues dans cette ligne.

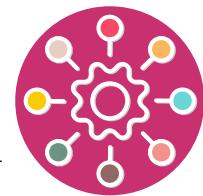
Exercice 5

Expliquez clairement toutes les informations contenues dans le premier ligne.



- Que signifie la chaîne de caractères au début ?
- Que signifient HEAD et main ?
- Qu'est-ce qui se trouve après les parenthèses ?

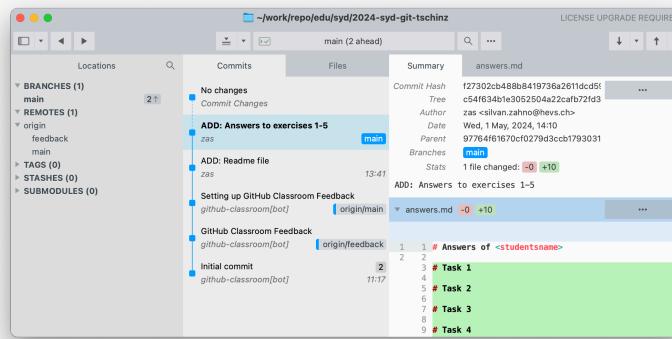
Écrivez la réponse!



2.9 Checkout commit

Pour ne pas perdre vos réponses, placez les modifications du fichier `answers.md` dans un commit.

```
git add answers.md
git commit -m "ADD : Answers to exercises 1-5" (ADD : réponses aux exercices 1-5)
```



Notez que chaque commit est accompagné d'un « hash » ou d'une « somme de contrôle » (de type sha1). Celle-ci, créée par la commande :

```
git log --oneline
```

ne sont que les premiers caractères de ce que l'on appelle les « short hashes ».

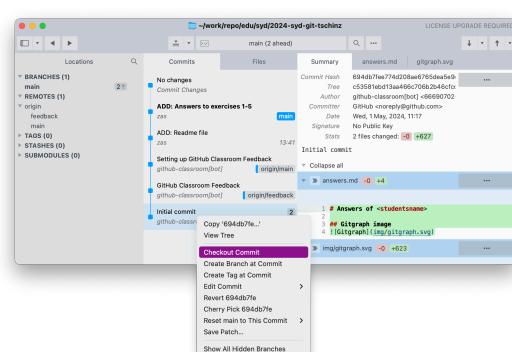
Effectuez maintenant un checkout avec la commande suivante, en utilisant le « short hash » qui correspond au premier commit.

Commandline

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

GUI

Select First Commit (Initial Commit) ⇒ Checkout commit



Examinez maintenant attentivement le contenu du dossier.



2.10 Checkout master

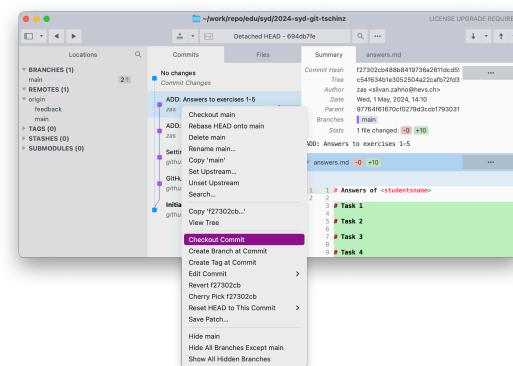
Effectuez maintenant un checkout avec la commande suivante :

Commandline

```
git checkout main
```

GUI

Branches (1) \Rightarrow master \Rightarrow checkout master



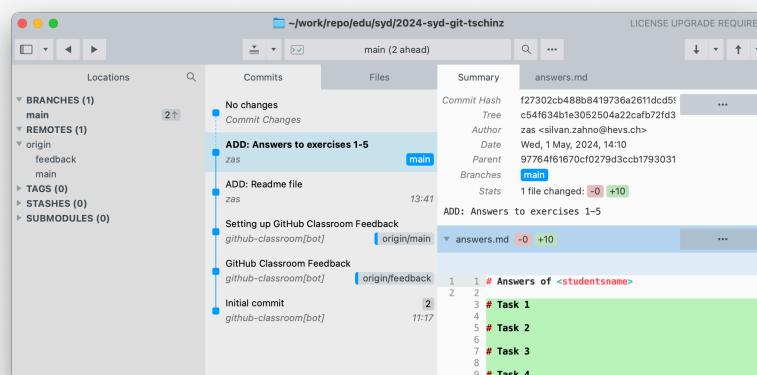
Exercise 6



Qu'avez-vous remarqué qui est arrivé aux fichiers et aux dossier dans le dossier de projet lorsque vous avez pris la « validation initiale » ? Que s'est-il passé lorsqu'ils sont revenus à la dernière validation ? Écrivez la réponse!

Pour ne pas perdre vos réponses, placez les modifications du fichier `answers.md` dans un commit.

```
git add answers.md
git commit -m "ADD : Answer to exercise 6" (ADD : réponses aux exercices 1-5)
```





3 | Branch et Merge

Jusqu'à présent, nous avons utilisé les fonctions de base de git. Il y a aussi les fonctions « branch » et « merge », que Git a grandement simplifiées par rapport aux outils existants auparavant.

Pour ce travail pratique, vous pouvez vous aider de la GUI Sublime Merge, qui vous fournit une représentation graphique et un historique visuel des commits dans votre repo.

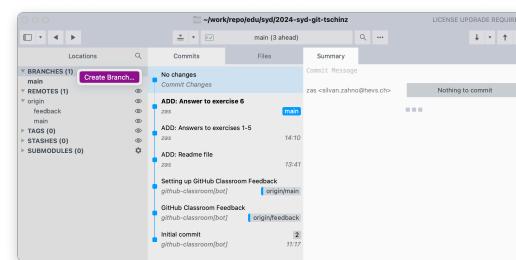
- Créez une branche de développement dev01 dans votre repo local.

Commandline

```
git checkout -b dev01
```

GUI

Branches ⇒ Create Branch ⇒ dev01



- Créez un commit sur cette branche :

- Le fichier `hello_world.py`.

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

```
git add hello_world.py
git commit -m "ADD: helloworld.py"
```

- Checkout le `main` branche

```
git checkout main
```

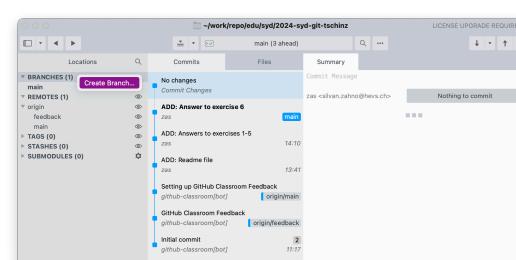
- A partir de la branche main, créer une nouvelle branche de développement dev02.

Commandline

```
git checkout -b dev02
```

GUI

Branches ⇒ Create Branch ⇒ dev02





5. Créez un commit sur cette branche :

- Un pour créer et remplir un fichier `hello_world.rs`.

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

```
git add hello_world.py
git commit -m "ADD: helloworld.py"
```

6. Checkout la branche `main`.

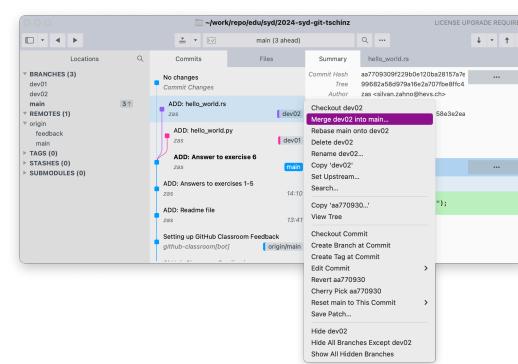
```
git checkout main
```

7. Merge la branche `dev02` dans `main`.**Commandline**

```
git merge dev02
```

GUI

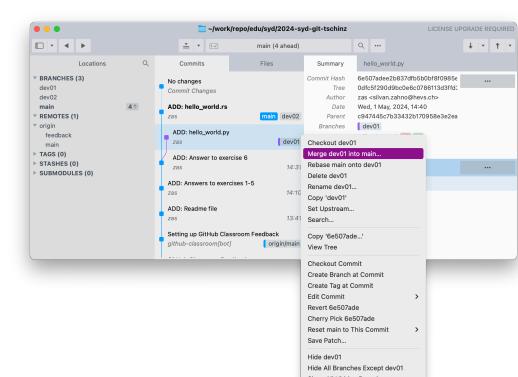
Select Commit ⇒ Merge dev02 into main ...

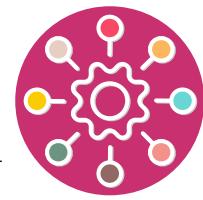
8. Merge la branche `dev01` dans `main`.**Commandline**

```
git merge dev01
```

GUI

Select Commit ⇒ Merge dev01 into main ...





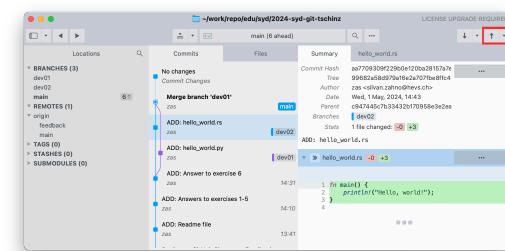
9. Poussez votre repository local vers votre repository distant GitHub.

Commandline

```
git push origin main
```

GUI

Top Right Arrow \Rightarrow Push changes



3.1.1 Résultat final

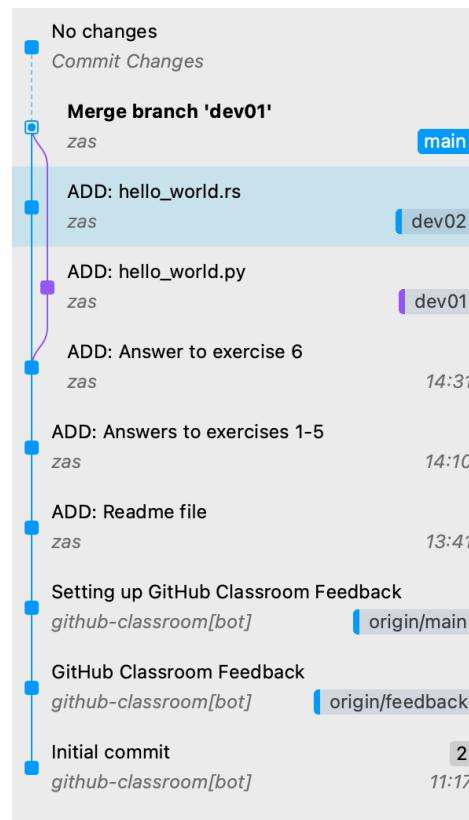


Fig. 10. – Statut après la merge des branches dev01 et dev02 du repo



Le statut de votre repo doit être similaire à Fig. 10. Cela fait partie de l'évaluation.



4 | Gitgraph

Dans la Fig. 11 est représenté un exemple d'une repo git. Nommez tous les éléments visibles sur l'image (points 1- 10).

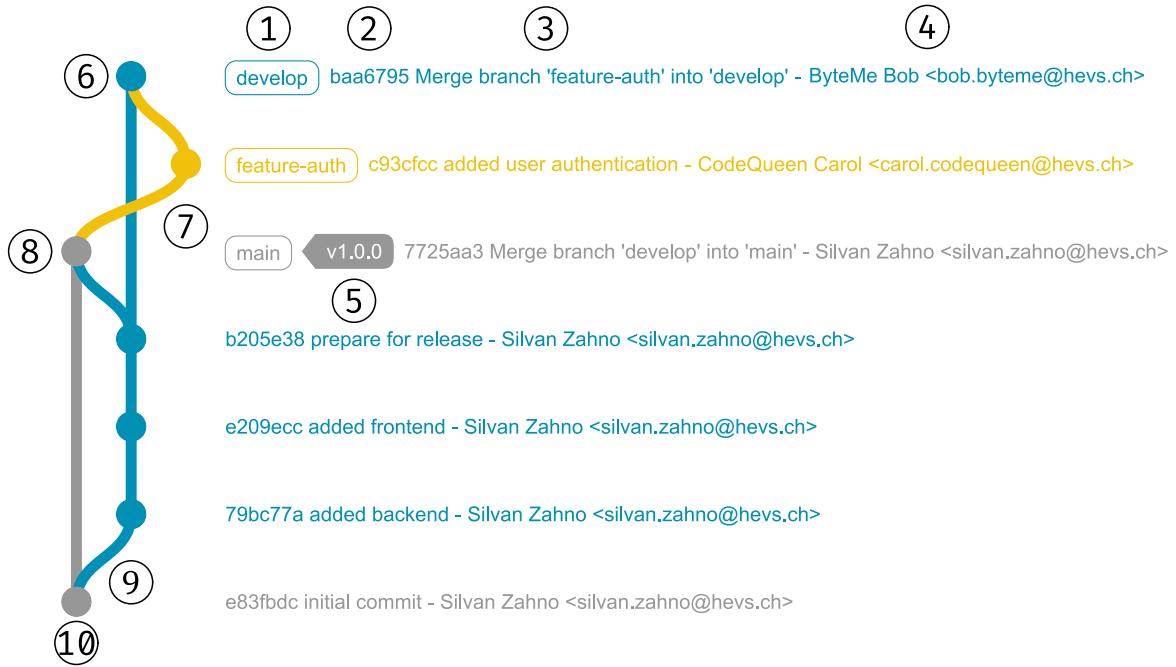


Fig. 11. – Exemple d'une gitgraph



Exercice 7

Nommez tous les éléments visibles dans l'image Fig. 11 (points 1 à 10).



5 | Gitflow

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

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

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

5.1 Fork

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

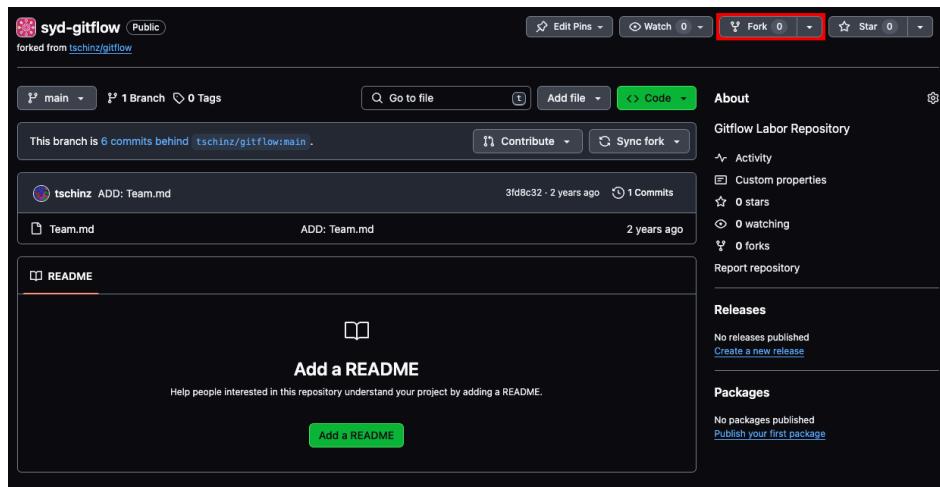


Fig. 12. – Bouton Fork pour un repo GitHub

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

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

5.2 Collaboration parallèle

Dans une branche de feature locale, modifiez le fichier `Team.md`. Remplacez votre numéro donné par votre prénom et votre nom.



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

« Commiter » et « pousser » votre branche dans le repo fork sur GitHub.



5.3 Pull Request

Créez une « Pull Request » (demande de fusion) sur GitHub. Utilisez pour cela l'interface du site web de GitHub.

Une fois que toutes les pull requests sont prêtes, les fusions sont effectuées en accord avec l'ensemble du groupe (et les enseignants).

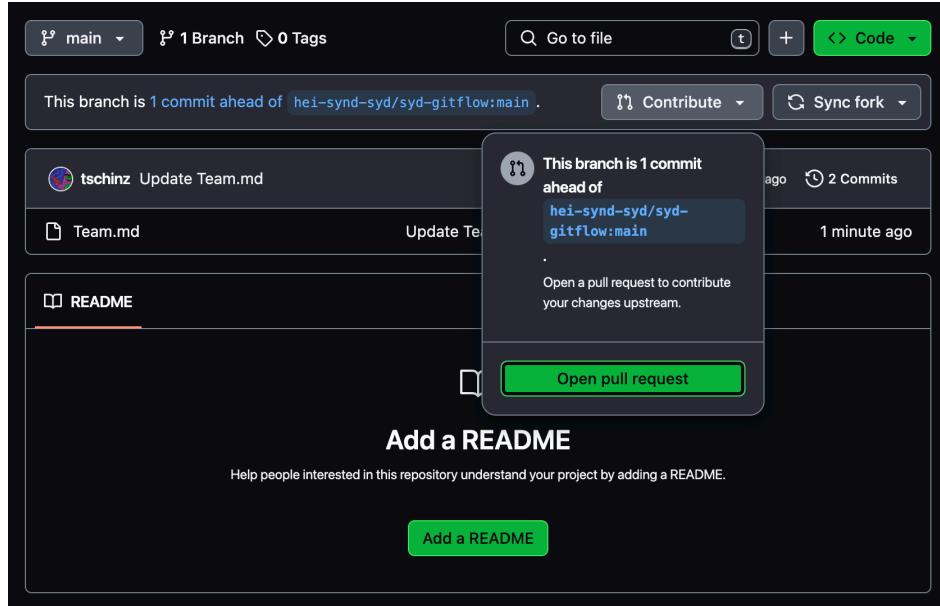


Fig. 13. – Créer une Pull Request sur Github



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 le [Markdown Syntax](#). 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 un [github action](#) approprié et ajoutez-le à votre projet.



Si vous avez besoin d'aide pour créer le github action. Consultez le ce [conseil](#).

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

Fig. 14. – Github-Actions Marktplatz



6.2 Apprendre le branchement Git

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

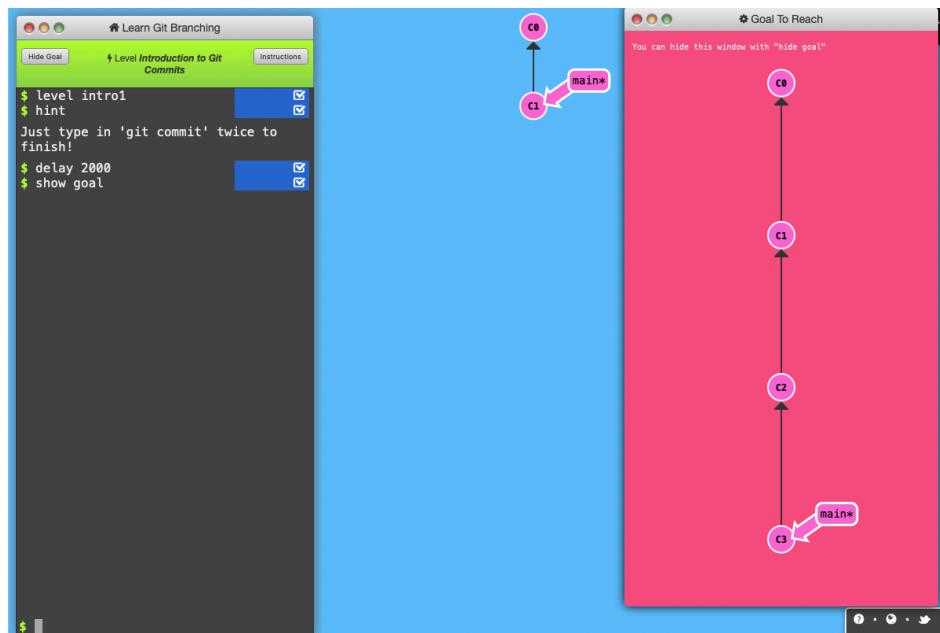


Fig. 15. – Lerne Git Branching Website



A | GIT Commandes

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

AA Examen des modification et création d'une opération de validation

```
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



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/hei-synd-syd/syd-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/>