



Introduction à git - Partie A & B



Contents

1 Objectif	1
2 Installation	3
3 Markdown	6
4 Outro	8
5 Objectifs	9
6 Outils	9
7 Opérations de base	10
8 Branch et Merge	19
9 Gitgraph	23
10 Gitflow	24
11 Extras	26
A GIT Commandes	28
B Commandes Git les plus utilisées	29
Bibliographie	30

1 | Objectif

Ce laboratoire est divisé en deux parties. La partie A doit être réalisée à la maison en guise de préparation, tandis que la partie B est traitée en commun dans le laboratoire. Le laboratoire sera réalisé de manière autonome sur votre ordinateur portable et sera évalué à la fin.

Dans ce laboratoire, nous allons apprendre les principes de base du contrôle de version [git](#) [1], en particulier les outils [ligne de commande Git](#) et [Sublime Merge](#), qui doivent être installés et configurés sur votre ordinateur (voir Chapitre 2). De plus, des comptes sont créés sur les plateformes [Github](#) et [Hevs Gitlab](#) [2].

Enfin, nous apprendrons les bases de Markdown dans le Chapitre 3 pour écrire facilement des fichiers texte.



Il est crucial que l'installation et la configuration soient effectuées avec soin afin d'éviter toute perte de temps pendant le partie B de laboratoire.



2 | Installation

La première étape est l'installation de Git et de Sublimemerge. Vous pouvez choisir vous-même si vous souhaitez utiliser la ligne de commande ou l'interface graphique pendant le laboratoire. Les deux outils doivent cependant être installés et configurés.

2.1 git

La dernière version de git peut être téléchargée sur le site officiel <https://git-scm.com/> [1]. Git est disponible pour Linux, Mac et Windows. Pour ce laboratoire, git ≥ 2.27 est nécessaire.

2.1.1 Ligne de commande

Lancez « Git Bash » sous Windows ou « Terminal » sous MacOS. Il s'agit d'un éditeur de commandes de type Unix/Linux qui permet d'exécuter des commandes Git en mode console.

```

zas - zas@zac ~ - zsh - 99x52
Last login: Tue Mar  8 09:26:26 on ttys004
[zas@zac] ~ (base)
[zas@zac] ~ (base) $ git --help
usage: git [--version] [--help] [-C <path>] [-c <name>=<value>]
           [--exec-path=<path>] [--html-path] [--man-path] [--info-path]
           [-p | --paginate | -P | --no-pager] [-no-replace-objects] [--bare]
           [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
           [--super-prefix=<path>] [--config-env=<name>=<envvar>]
           <command> [<args>]

These are common Git commands used in various situations:

start a working area (see also: git help tutorial)
clone      Clone a repository into a new directory
init       Create an empty Git repository or reinitialize an existing one

work on the current change (see also: git help everyday)
add        Add file contents to the index
mv        Move or rename a file, a directory, or a symlink
restore   Restore working tree files
rm        Remove files from the working tree and from the index

examine the history and state (see also: git help revisions)
bisect    Use binary search to find the commit that introduced a bug
diff      Show changes between commits, commit and working tree, etc
grep      Print lines matching a pattern
log       Show commit logs
show     Show various types of objects
status   Show the working tree status

grow, mark and tweak your common history
branch   List, create, or delete branches
commit   Record changes to the repository
merge    Join two or more development histories together
rebase   Reapply commits on top of another base tip
reset   Reset current HEAD to the specified state
switch  Switch branches
tag     Create, list, delete or verify a tag object signed with GPG

collaborate (see also: git help workflows)
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

'git help -a' and 'git help -g' list available subcommands and some
concept guides. See 'git help <command>' or 'git help <concept>'
to read about a specific subcommand or concept.
See 'git help git' for an overview of the system.

[zas@zac] ~ (base)
[zas@zac] ~ (base) $
```

Fig. 1. – git Terminal

Notez que pour toutes les commandes de Git Bash, vous pouvez obtenir de l'aide en insérant `help` après la commande.



`git --help`



2.1.2 Configuration globale

Un grand nombre de paramètres peuvent être configurés dans Git. Il est possible de modifier les paramètres globalement sur votre ordinateur (indicateur `global`) ou seulement pour un repo particulier.

Nous allons maintenant procéder à la configuration minimale. Utilisez les commandes suivantes pour configurer votre identité dans Git `global` sur le système. Utilisez votre nom et adresse électronique. Ces informations sont visibles publiquement afin d'identifier votre travail (commits).

```
git config --global user.name "Firstname Lastname"
git config --global user.email first.last@email.ch
```

Par exemple:

```
git config --global user.name "Silvan Zahno"
git config --global user.email silvan.zahno@hevs.ch
```

Vous pouvez vérifier la configuration à l'aide de la commande suivante :

```
git config --list
```

Vous pouvez également vérifier un paramètre spécifique :

```
git config user.name
```

2.2 Sublime Merge

Visitez le site <https://www.sublimemerge.com> et téléchargez et installez l'outil Sublime Merge [3].

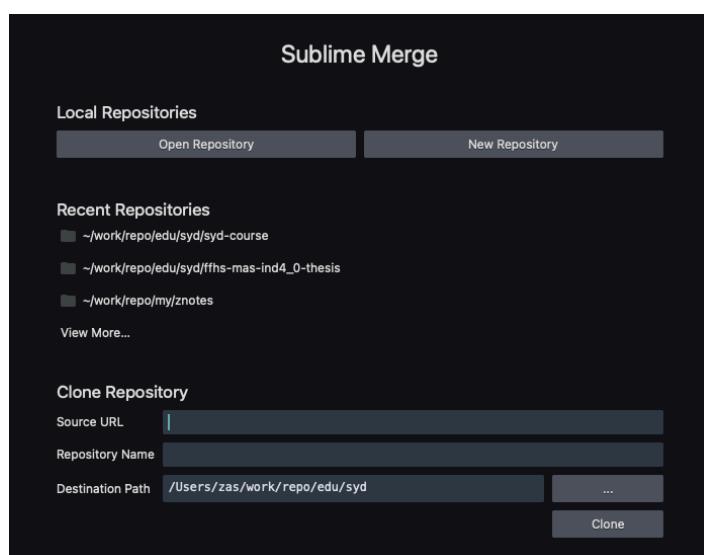


Fig. 2. – Sublime Merge GUI



2.3 Comptes en ligne

2.3.1 Gitlab

Visitez le site <https://gitlab.hevs.ch> et connectez-vous avec votre compte d'école (SwitchEDU-ID).

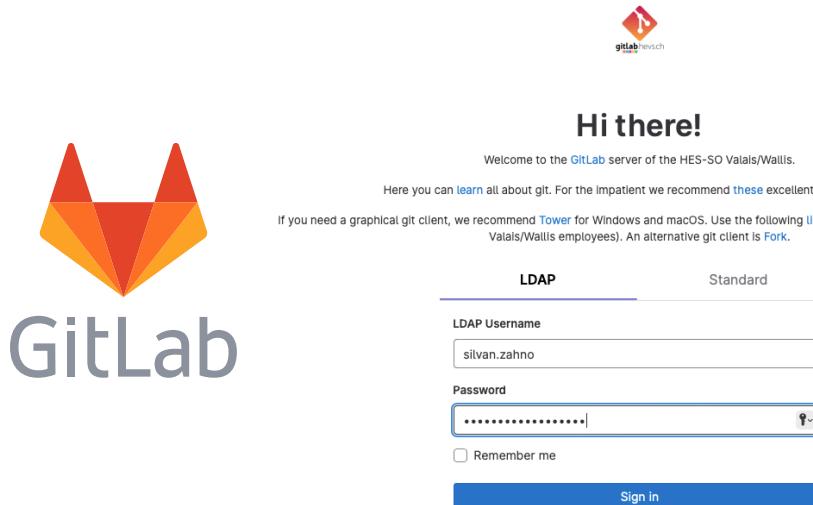


Fig. 3. – Connexion à Gitlab

2.3.2 Github

Visitez le site <https://github.com>, créez un compte et connectez-vous.

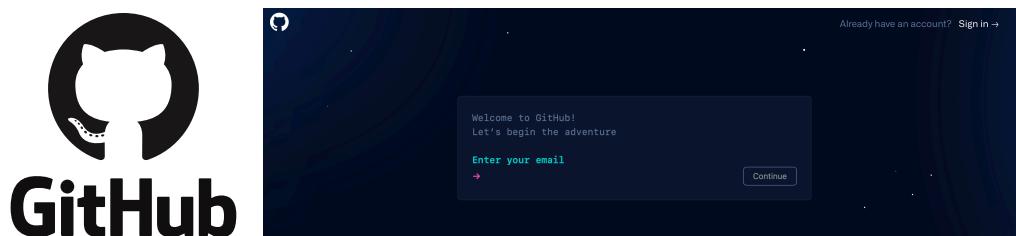


Fig. 4. – Connexion à GitHub

2.4 Configuration de Windows

Afin de voir également le dossier caché .git/ ainsi que les extensions de fichiers, configurez votre explorateur de fichiers Windows comme suit :

Explorateur de fichiers ⇒ Afficher ⇒ Afficher ⇒ Activez « **Extensions de noms de fichiers** » et « **Eléments cachés** »

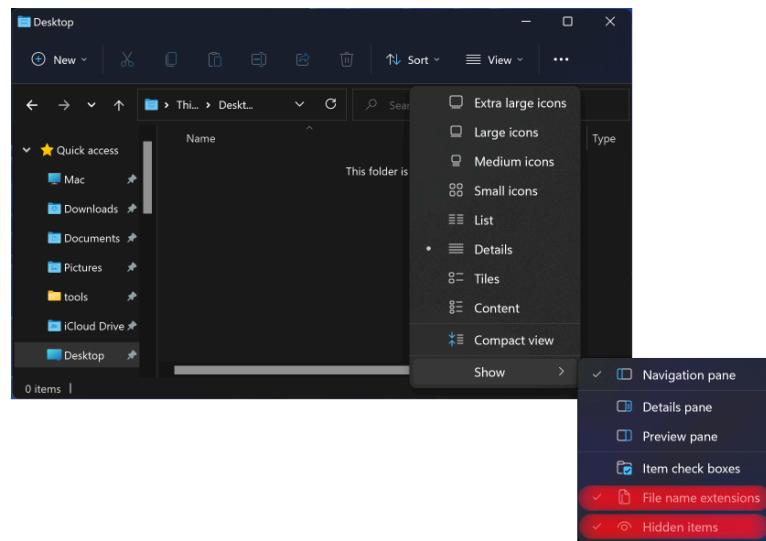


Fig. 5. – Configuration de l'explorateur de fichiers Windows

3 | Markdown

Markdown est un langage de codage léger avec une syntaxe de formatage en texte brut. Il est conçu pour être facile à lire et à écrire, tout en étant facilement converti en PDF, HTML ou autres formats. Markdown est couramment utilisé pour formater du texte sur le web, par exemple dans les fichiers README.md, la documentation, les messages sur les forums et la messagerie.

Pour écrire du Markdown, vous avez besoin de votre éditeur de texte préféré ou vous pouvez installer un éditeur Markdown spécialisé tel que [Marktext](#).

Par exemple, la [page d'introduction](#) de ce cours est écrit en Markdown. Vous pouvez voir le code source de la page en cliquant sur le bouton « Editer cette page » dans le coin supérieur droit de la page ou via ce [lien](#).

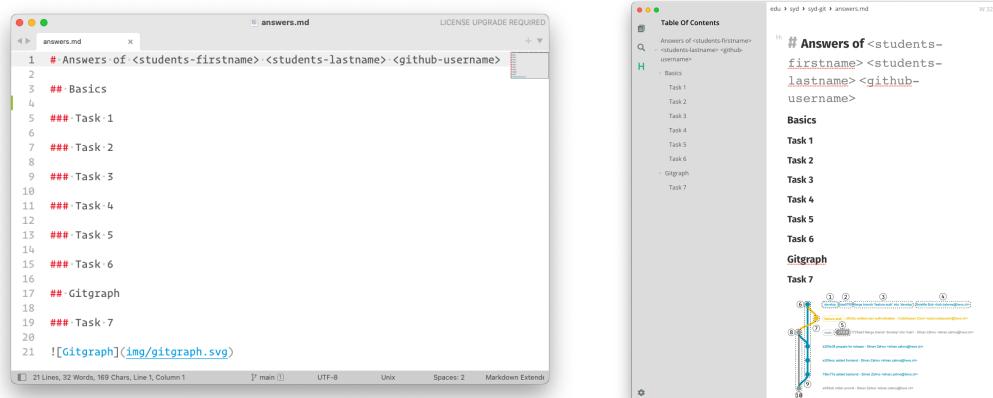


Fig. 6. – A gauche : Editeur de text simple (Sublime Text), à droite : Marktext



Pour le laboratoire, vous devrez écrire un document au format Markdown. Préparez votre éditeur.

3.1 Syntaxe Markdown

Voici un bref aperçu de la structure d'un fichier Markdown. La syntaxe est simple et facile à apprendre. Le fichier doit être sauvegardé avec l'extension .md. Une liste syntaxique plus complète est disponible à l'adresse [ici](#).

```
# Title 1
## Title 2
### Title 3

Some simple Text italic **bold**
~~Strikethough~~ `monospaced`

Formulas $S = \sum_{i=1}^n x_i^2

- List Item 1
- List Item 2
1. Numbered List Item 1
2. Numbered List Item 2

[Link name](https://hevs.ch/synd)
![logo](logo.svg)

```rust
// A rust code bloc
fn main(){
 println!("Hello World");
}
```

Tables	Are	Cool
col 1	left-align	$f_{clk}$
col 2	centered	$12
col 3	right-align	1024

---
```

urces > git > markdown-cheatsheet.md W 90

Title 1

Title 2

Title 3

Some simple Text *italic* **bold** ~~Strikethough~~ `monospaced`

Formulas $S = \sum_{i=1}^n x_i^2$

- List Item 1
- List Item 2

1. Numbered List Item 1
2. Numbered List Item 2

Link name



```
// A rust code bloc
fn main(){
    println!("Hello World");
}
```

| Tables | Are | Cool |
|--------|-------------|-----------|
| col 1 | left-align | f_{clk} |
| col 2 | centered | \$12 |
| col 3 | right-align | 1024 |



4 | Outro

Félicitations, vous avez maintenant installé et configuré tout ce dont vous avez besoin pour travailler avec Git. Préparez-vous pour le prochain laboratoire :

- Étudiez la théorie
- Familiarisez-vous avec les commandes Git
- Familiarisez-vous avec l'outil graphique SublimeMerge
- Entraînez-vous à écrire des documents avec Markdown



Dans les annexes Chapitre A et Chapitre B, vous trouverez un résumé des principales commandes Git.



5 | 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 7, 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 8. Dans le Chapitre 10, nous travaillons tous ensemble sur un même dépôt. Enfin, il y a quelques travaux optionnels dans le Chapitre 11.



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.

6 | Outils

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



Git CMD
App



Sublime Merge
App

Fig. 7. – Git CMD Ligne de commande Fig. 8. – Sublime Merge GUI



7 | Opérations de base

7.1 Créer un dépôt git

Créez un fork du template dépôt à l'aide du lien suivant <https://classroom.github.com/a/O0eniBP2> (Fig. 9). Pour cela, vous devez vous connecter à GitHub.

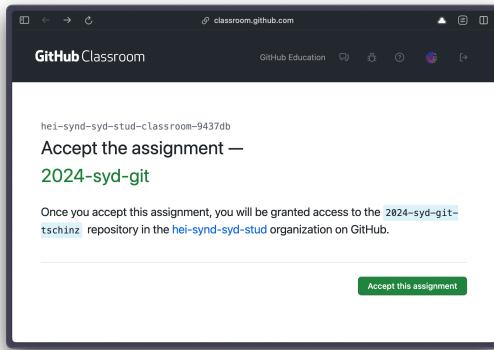


Fig. 9. – Lien d'invitation au forking

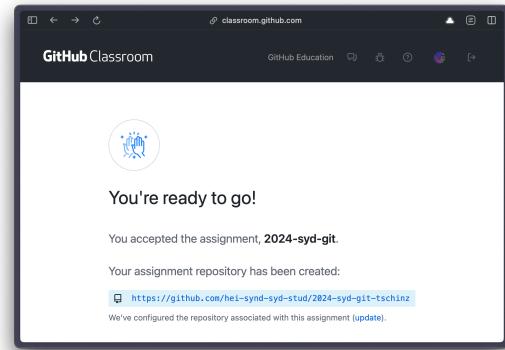


Fig. 10. – Lien pour le clonage du dépôt

7.2 Clone

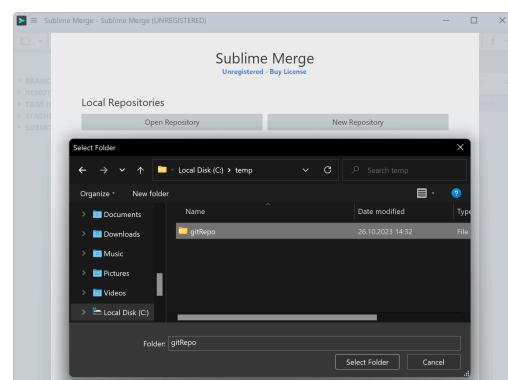
Après avoir créé le fork, vous recevrez un lien vers votre propre dépôt (Fig. 10). Clonez-le sur votre ordinateur local à l'emplacement de votre choix.

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 0



Modifiez le titre du 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

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` !

7.3 Consulter l'état

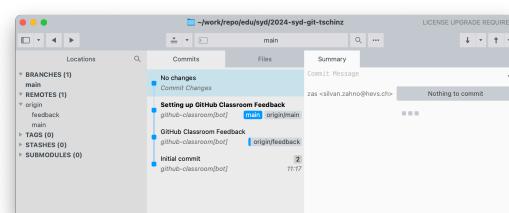
Obtenez des informations sur votre repo en utilisant les commandes suivantes :

Commandline

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

GUI

See the status in the main window.



7.4 Ajouter un fichier

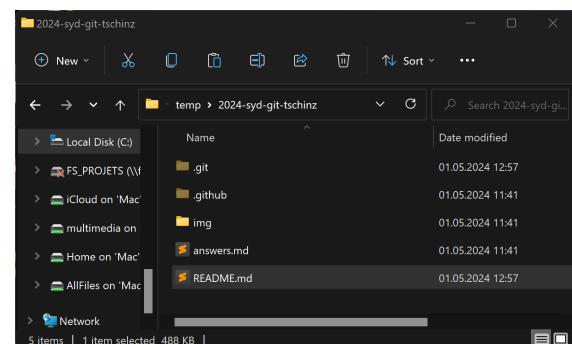
Créez maintenant un fichier vide nommé `README.md` dans le répertoire principal de votre repos.

Commandline

```
touch README.md
```

GUI

C:\\[myRepo] ⇒ 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 dépôt 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.

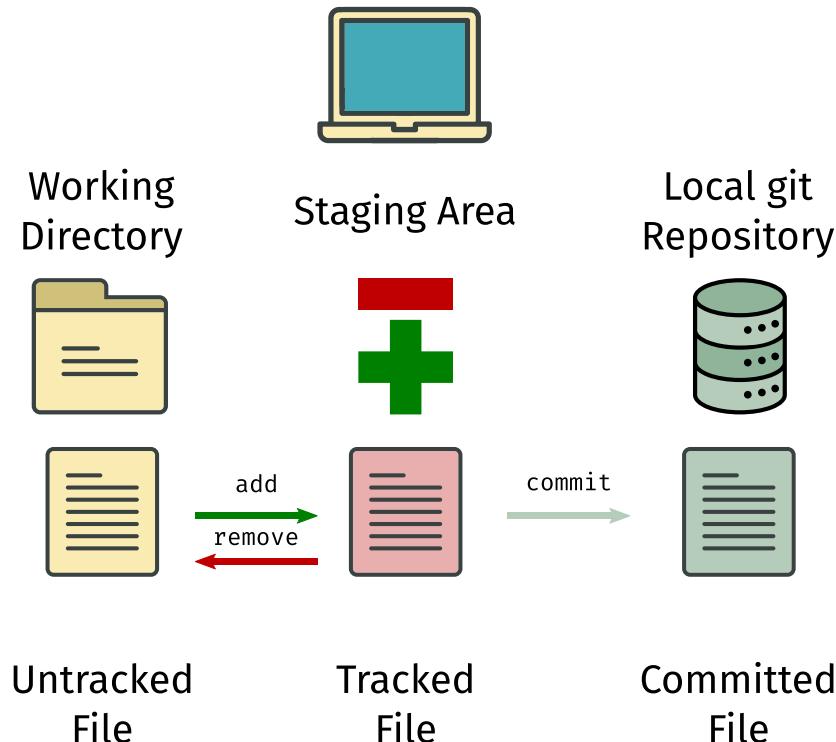


Fig. 11. – Types d'opérations locales de Git

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.

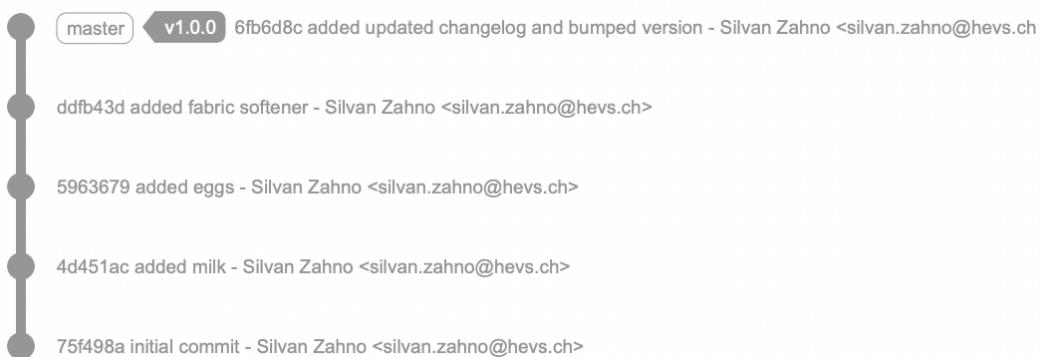


Fig. 12. – Cinq commits sur le repo local, chaque commit possède son propre identifiant



7.5 Ajouter le fichier au repo

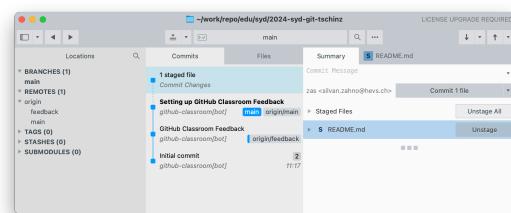
Ajoutez le fichier précédemment créé README.md au stage avec la commande :

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!



7.6 Ajouter de nouvelles modifications

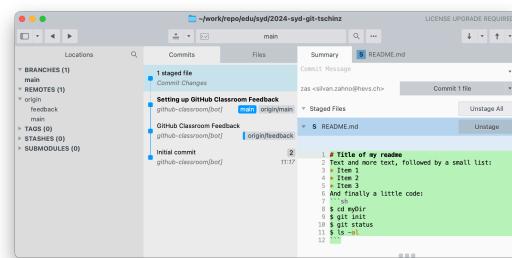
Ajoutez la dernière version du fichier README.md au stage.

Commandline

```
git add README.md
```

GUI

README.md \Rightarrow Stage



7.7 Exécuter un commit

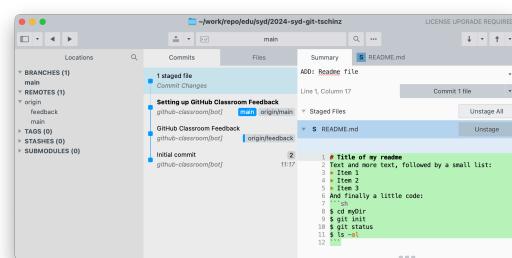
Exécutez maintenant un commit avec la commande suivante :

Commandline

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

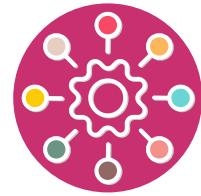
GUI

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 !



7.8 Plus d'informations

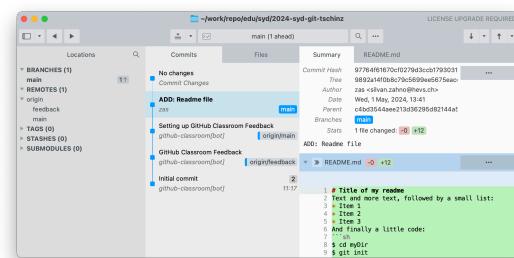
Quelles informations obtiendrez-vous maintenant avec la commande :

Commandline

```
git log --oneline
```

GUI

Select First Commit ⇒ 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 ?

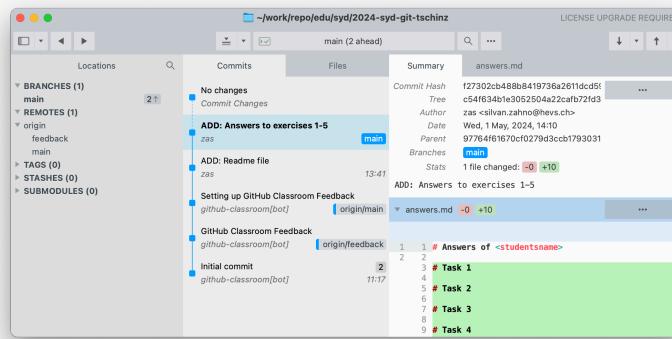
Écrivez la réponse!



7.9 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 ».

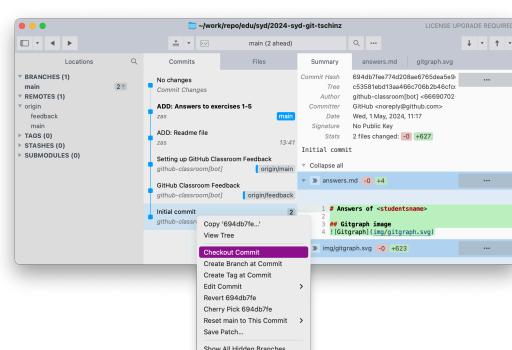
Effectuez maintenant un checkout avec la commande suivante, en utilisant le « short hash » qui correspond au premier commit.

Commandline

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

GUI

Select First Commit (Initial Commit) ⇒ Checkout commit



Examinez maintenant attentivement le contenu du dossier.



7.10 Checkout master

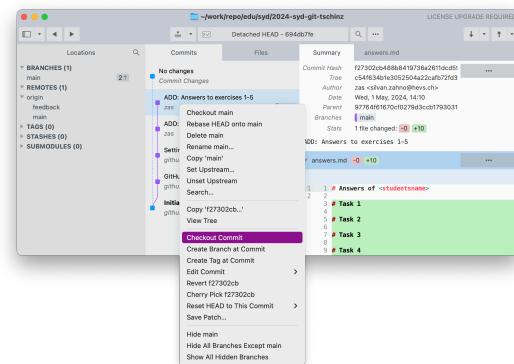
Effectuez maintenant un checkout avec la commande suivante :

Commandline

```
git checkout main
```

GUI

Branches (1) ⇒ master ⇒ checkout master



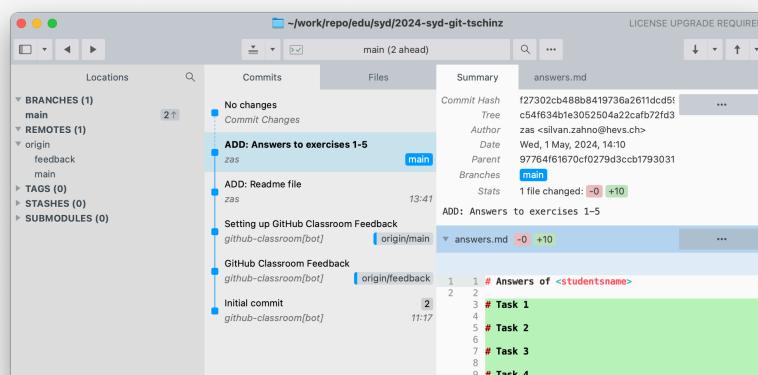
Exercise 6



Qu'est-il arrivé aux fichiers et aux dossier dans le dossier de projet lorsque vous êtes revenus sur le commit « Initial commit » ? Que s'est-il passé lorsque vous êtes revenus au dernier commit ? 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)
```





8 | 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.

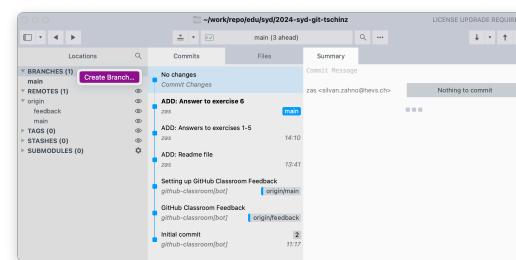
- Créez une branche de développement dev01 dans votre repo local.

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: hello_world.py"
```

- Revenez sur la branche `main`

```
git checkout main
```

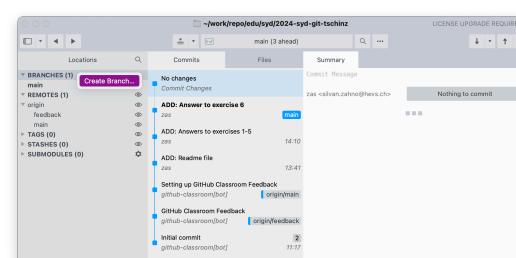
- A partir de la branche `main`, créer une nouvelle branche de développement `dev02`.

Commandline

```
git checkout -b dev02
```

GUI

Branches ⇒ Create Branch ⇒ dev02





5. Créez un commit sur cette branche :

- Le fichier `hello_world.rs`.

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

```
git add hello_world.rs
git commit -m "ADD: hello_world.rs"
```

6. Checkout la branche `main`.

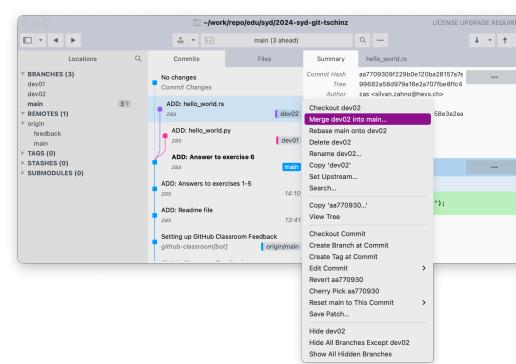
```
git checkout main
```

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

```
git merge dev02
```

GUI

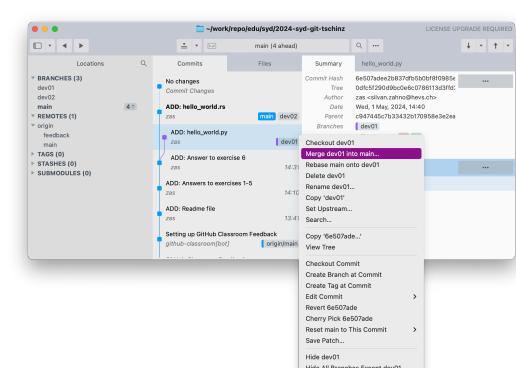
Select Commit ⇒ Merge dev02 into main ...

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

```
git merge dev01
```

GUI

Select Commit ⇒ Merge dev01 into main ...





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

Commandline

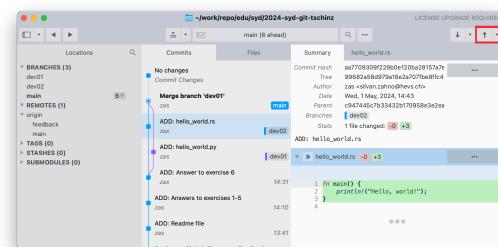
```
git push origin main
git push origin dev01
git push origin dev02
```

GUI

Checkout dev01 ⇒ Top Right Arrow ⇒ Push changes

Checkout dev02 ⇒ Top Right Arrow ⇒ Push changes

Checkout main ⇒ Top Right Arrow ⇒ Push changes



8.1.1 Résultat final

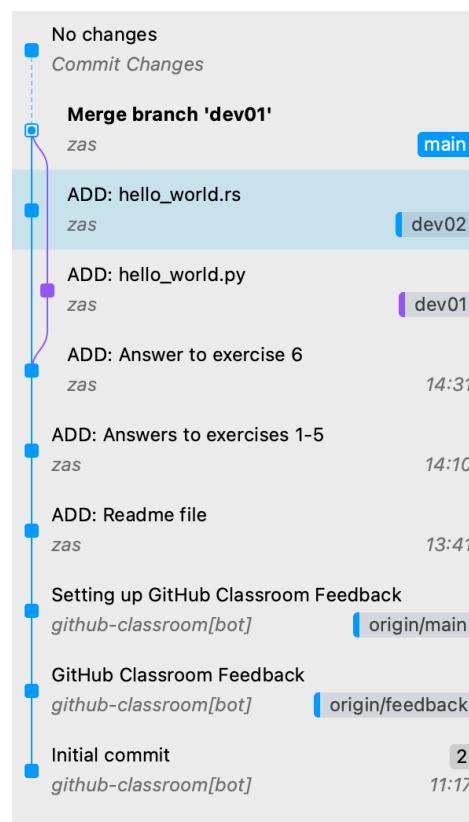


Fig. 18. – Statut après la fusion des branches dev01 et dev02 du repo



Le statut de votre repo doit être similaire à Fig. 18. Cela fait partie de l'évaluation.



9 | Gitgraph

Dans la Fig. 19 est représenté un exemple d'un dépôt git. Nommez tous les éléments visibles sur l'image (points 1- 10).

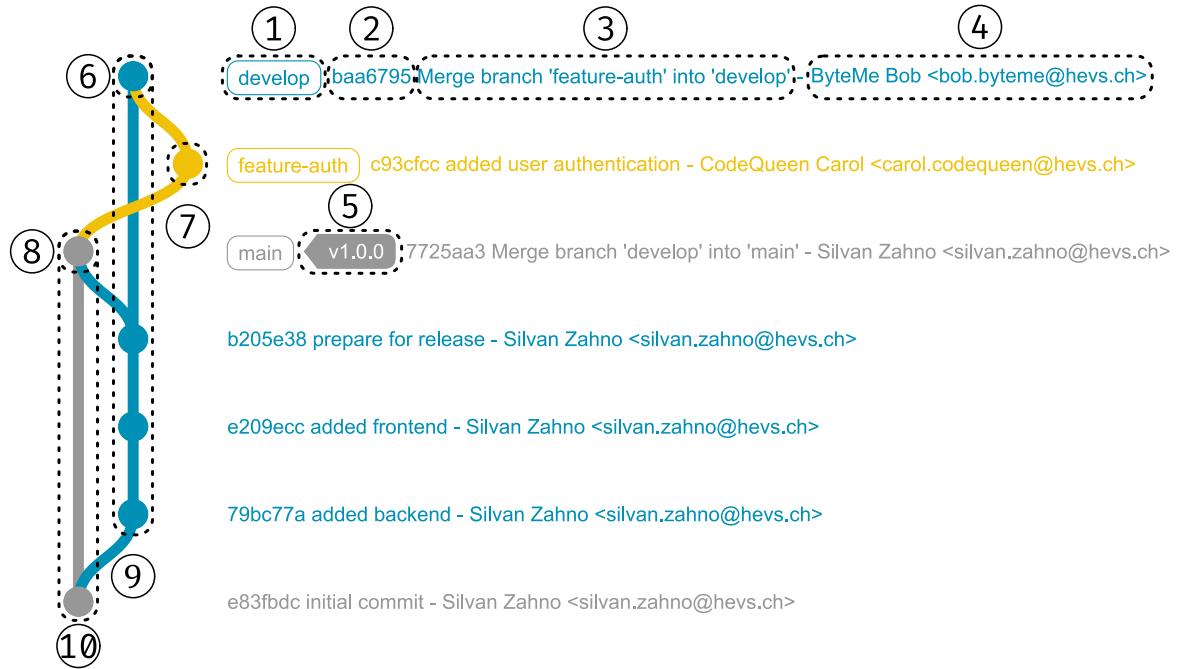


Fig. 19. – Exemple d'une gitgraphe



Exercice 7

1. Nommez tous les éléments visibles dans l'image Fig. 19 (points 1 à 10).
2. Poussez votre dépôt local vers votre dépôt distant GitHub.



10 | 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> [4]

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

10.1 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.

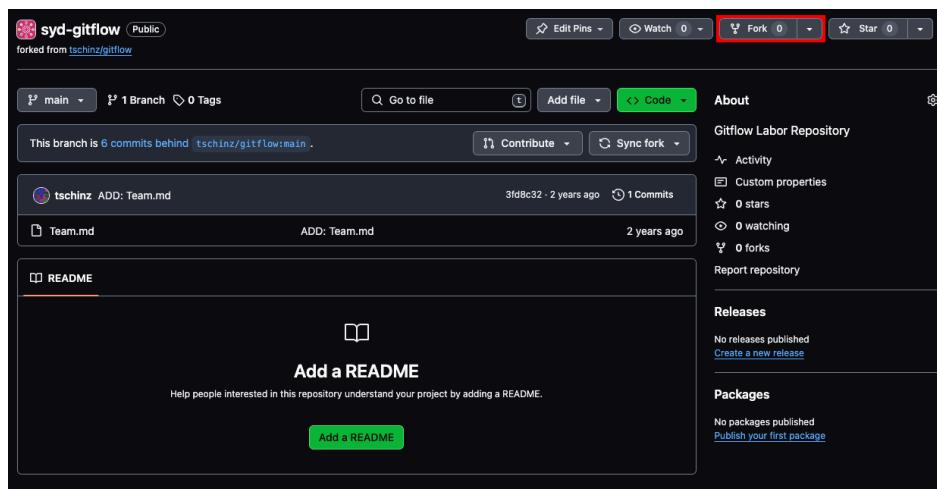


Fig. 20. – Bouton Fork pour un dépôt GitHub

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

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

10.2 Collaboration parallèle

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.

« Commitez » et « poussez » votre branche dans le dépôt fork sur GitHub.



10.3 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).

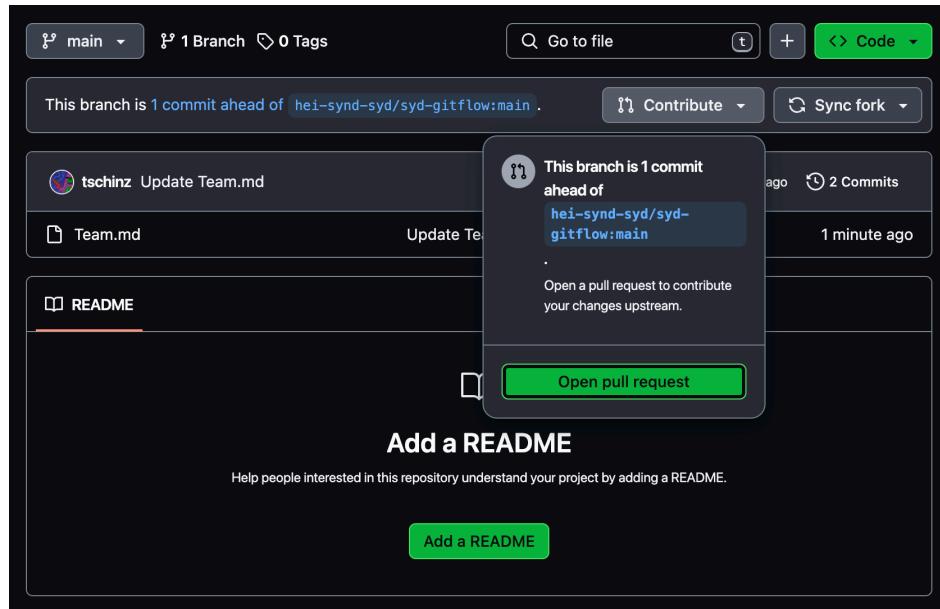


Fig. 21. – Créer une Pull Request sur Github



11 | 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 »

11.1 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](#).

Marketplace

Type to search | + | ⌂ | ⌂ | ⌂ | ⌂ | ⌂

Marketplace / Search results

Sort: Best Match

Types

Actions

Categories

API management

Chat

Code quality

Code review

Continuous integration

Dependency management

Deployment

IDEs

Learning

Localization

Actions

An entirely new way to automate your development workflow.

20351 results filtered by Actions

Close Stale Issues
By actions Creator verified by GitHub
Close issues and pull requests with no recent activity
1.1k stars

Upload a Build Artifact
By actions Creator verified by GitHub
Upload a build artifact that can be used by subsequent workflow steps
2.5k stars

Download a Build Artifact
By actions Creator verified by GitHub
Download a build artifact that was previously uploaded in the workflow by the upload-artifact action
1.1k stars

Setup Java JDK
By actions Creator verified by GitHub
Set up a specific version of the Java JDK and add the command-line tools to the PATH
1.3k stars

Fig. 22. – Github-Actions Marktplatz



11.2 Apprendre le branchement Git

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

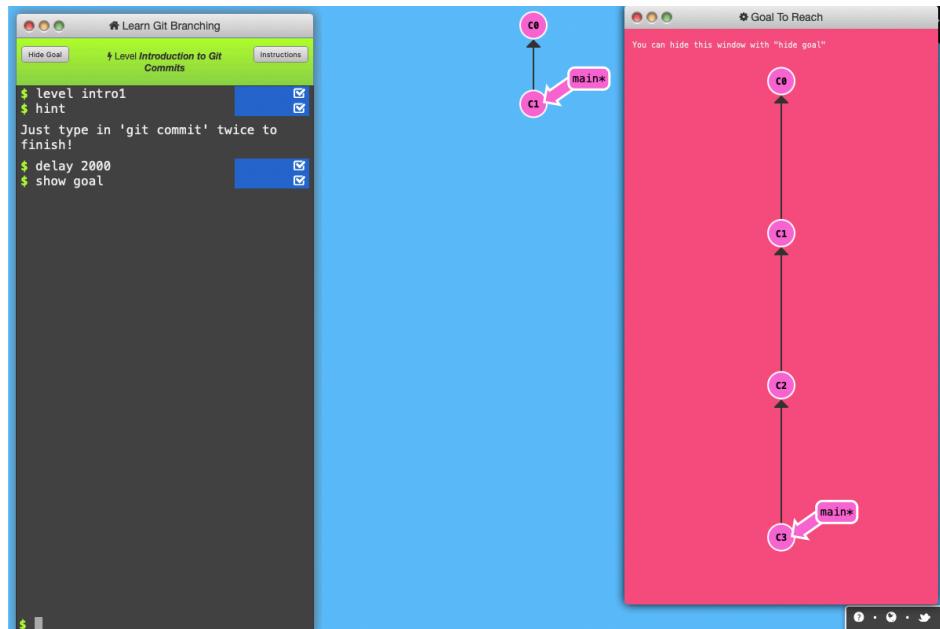


Fig. 23. – Lerne Git Branching Website



A | GIT Commandes

[Github git cheatsheet](#) [5], [6]

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



Bibliographie

- [1] T. Linus, « Git ». Consulté le: 25 avril 2023. [En ligne]. Disponible sur: <https://git-scm.com/>
- [2] « GitLab Hevs ». Consulté le: 25 avril 2023. [En ligne]. Disponible sur: <https://gitlab.hevs.ch/>
- [3] « Sublime Merge - Git Client from the Makers of Sublime Text ». Consulté le: 25 avril 2023. [En ligne]. Disponible sur: <https://www.sublimemerge.com/>
- [4] tschinz, « Tschinz/Gitflow ». Consulté le: 25 avril 2023. [En ligne]. Disponible sur: <https://github.com/tschinz/gitflow>
- [5] gitlab, « Git Cheatsheet ». 2023.
- [6] « GitHub Git Spickzettel ». Consulté le: 25 avril 2023. [En ligne]. Disponible sur: <https://training.github.com/downloads/de/github-git-cheat-sheet/>