

# Introduction au versionnage des fichiers avec git



## Contenu

1 Objectifs .....	2
2 Installation .....	3
2.1 <b>git</b> .....	3
2.2 Sublime Merge .....	4
2.3 Comptes en ligne .....	4
2.4 Configuration de Windows .....	5
3 Opérations de base .....	7
3.1 Créer un référentiel git .....	7
3.2 Initialiser .....	7
3.3 Consulter l'état .....	8
3.4 Ajouter un fichier .....	8
3.5 Ajouter le fichier au repo .....	9
3.6 Ajouter de nouvelles modifications .....	10
3.7 Exécuter un commit .....	11
3.8 Plus d'informations .....	11
3.9 Plus de commits .....	12
3.10 Checkout commit .....	13
3.11 Checkout master .....	14
4 Branch et Merge .....	15
5 Gitflow .....	17
5.1 Fork .....	17
5.2 Collaboration parallèle .....	17
5.3 Pull Request .....	18
6 Extras .....	19
6.1 Votre propre projet sur git .....	19
6.2 Apprendre le branchement Git .....	20



# 1 | Objectifs

Dans ce laboratoire, nous apprenons les principes de base du contrôle de version **git** [1].

Pour commencer, nous allons installer et configurer Git Chapitre 2 sur votre ordinateur. Nous créerons également des comptes sur [Github](#) [2] et [Hevs Gitlab](#) [3].

Dans Chapitre 3, nous apprenons les opérations de base pour pouvoir travailler avec Git.

Le référentiel créé est ensuite publié sur <https://github.com>.

Les fonctions avancées **branch** et **git** sont essayées dans un exemple dans Chapitre 4.

Au Chapitre 5, nous travaillons tous ensemble selon le principe Gitflow.

Enfin, il y a quelques travaux optionnels dans Chapitre 6.



Les réponses aux questions doivent être écrites dans un fichier Markdown. Le fichier doit être nommé **answers.md**.



## 2 | Installation

La première étape est l'installation de Git et de Sublimemerge.

### 2.1 git

Tu peux télécharger la dernière version 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

Démarre « Git Bash ». Il s'agit d'un éditeur de commandes de type Unix/Linux qui permet d'exécuter des commandes Git en mode console. C'est cette interface que nous utiliserons dans ce laboratoire.

```

Last login: Tue Mar  8 09:26:26 on ttys004
(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)
$

```

Figure 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 votre adresse électronique. Ces informations sont visibles publiquement afin d'identifier votre travail (vos commits). }

```
git config --global user.name "Firstname Lastname"
git config --global user.email first.last@email.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 [4].

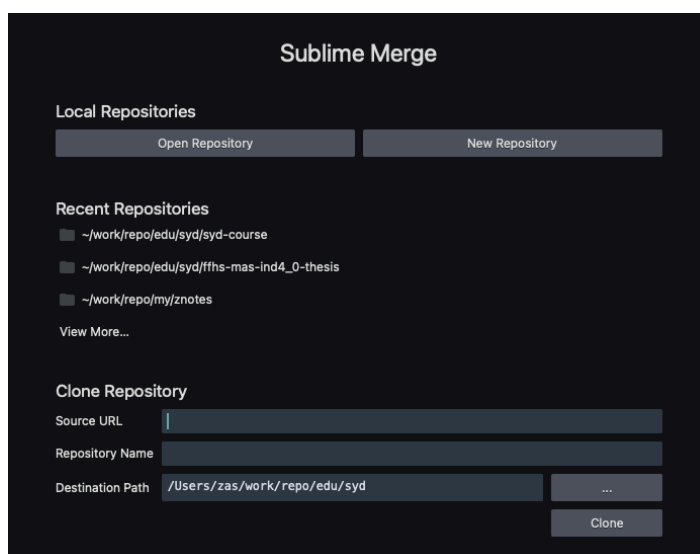


Figure 2 - Sublime Merge GUI

## 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).

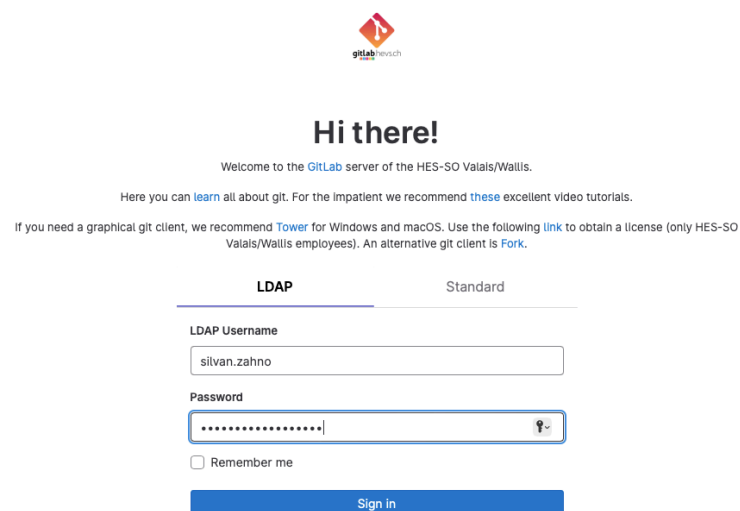

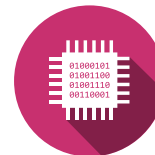


Table 1 - Connexion à Gitlab

### 2.3.2 Github

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

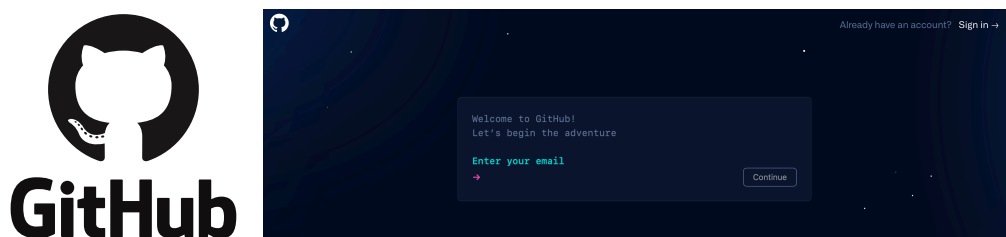


Table 2 - Connexion à GitHub

## 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 ⇒ View ⇒ Show ⇒ Activez « **Extensions de noms de fichiers** » et « **Éléments cachés** »

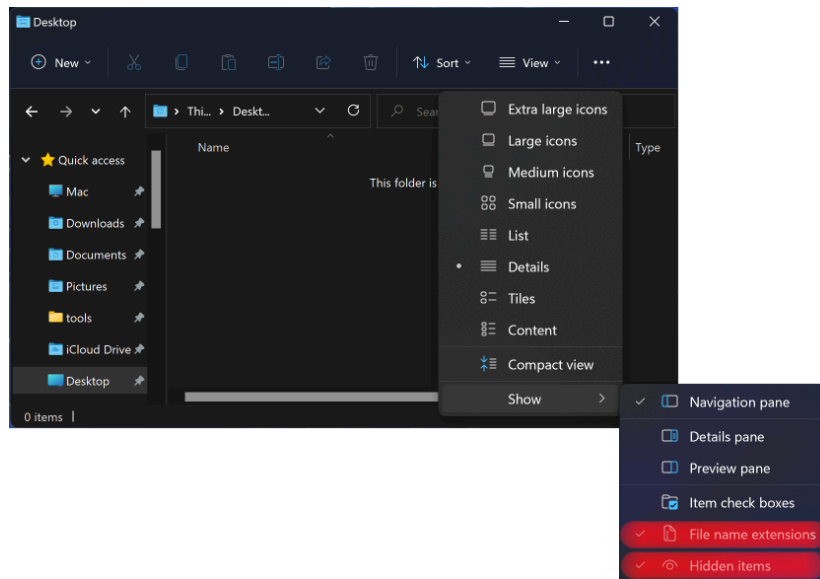


Figure 3 - Configuration de l'explorateur de fichiers Windows



## 3 | Opérations de base

### 3.1 Créer un référentiel git

À l'aide de la console Git Bash, créez un répertoire vide sur votre ordinateur, par ex. **C:\temp\gitRepo** ou **~/tmp/gitRepo**. Ce répertoire sera votre repo Git.

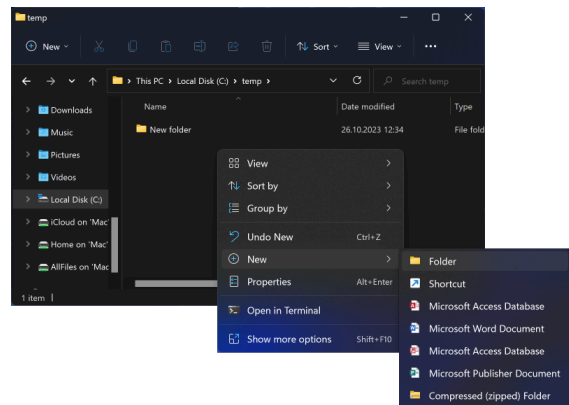
Vous pouvez utiliser les commandes Unix/Linux **ls** (list files and directories), **cd** (change directory), **pwd** (print current working directory) et **mkdir** (make directory) pour créer ce répertoire.

#### Commandline

```
mkdir -p c:/temp/gitRepo
cd c:/temp/gitRepo
```

#### GUI

**C:\temp** ⇒ **New** ⇒ **Folder** ⇒ **gitRepo**



### 3.2 Initialiser

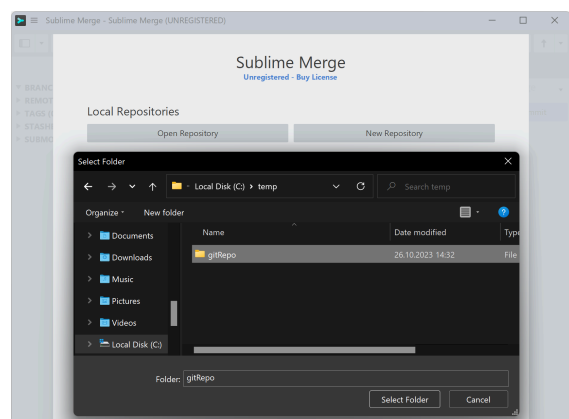
Une fois que vous êtes dans ce répertoire, initialisez-le en tant que Git Repo avec la commande :

#### Commandline

```
git init
```

#### GUI

**CTRL+T** ⇒ **New Repository** ⇒ **Select Folder**



Qu'est-ce qui a changé dans le répertoire après que tu l'aies initialisé en tant que git-repo ?



Écrivez la réponse!

### 3.3 Consulter l'état

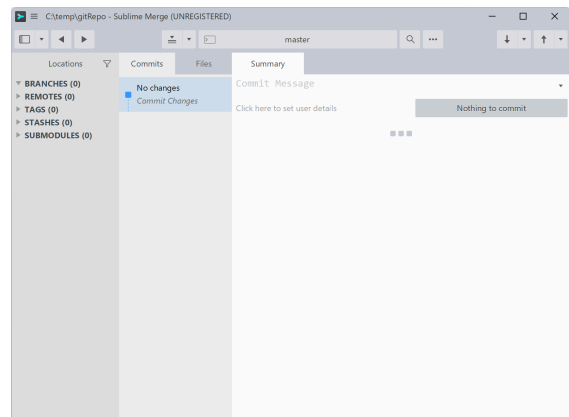
Obtenez des informations sur votre repo en utilisant les commandes suivantes :

#### Commandline

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

#### GUI

See the status in the main window.



### 3.4 Ajouter un fichier

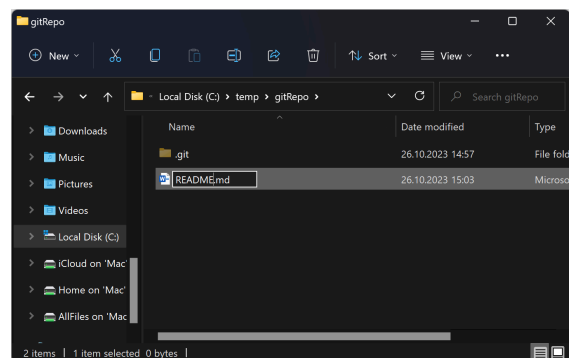
Créez maintenant un fichier vide nommé **README.md** dans le répertoire principal de votre repos.

#### Commandline

```
touch README.md
```

#### GUI

C:\temp ⇒ New ⇒ Any File ⇒ README.md



Utilisez la commande précédente pour récupérer à nouveau les informations sur votre repo. Qu'est-ce qui a changé ?



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.

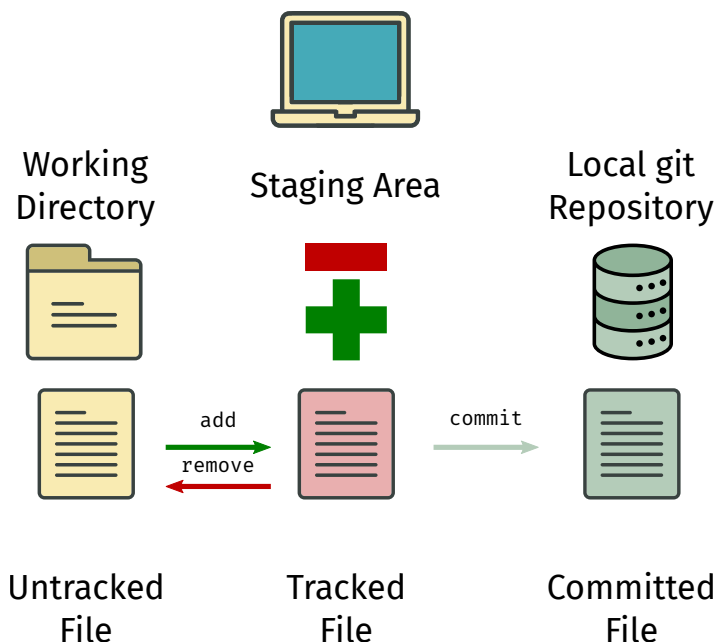


Figure 4 - Types d'opérations locales de Git

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.

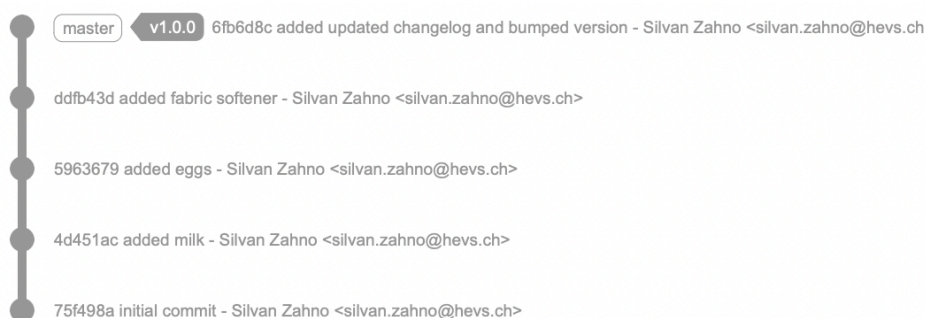


Figure 5 - Cinq commits sur le repo local, chaque commit possède son propre identifiant

### 3.5 Ajouter le fichier au repo

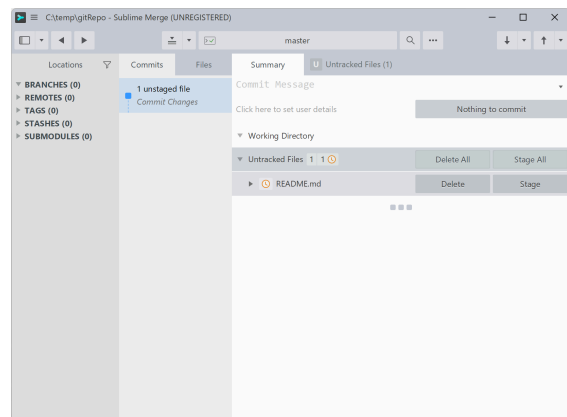
Ajoutez le fichier précédemment créé **README.md** au stage avec la commande :

**Commandline**

```
git add README.md
```

**GUI**

Untracked Files ⇒ README.md ⇒ Stage



Consultez à nouveau les informations 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
```
```

Reçois à nouveau les informations sur ton repo, que constates-tu ?



Écrivez la réponse!

### 3.6 Ajouter de nouvelles modifications

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

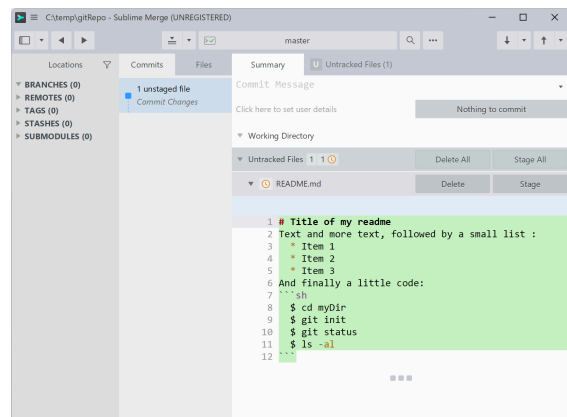
**Commandline**

**GUI**

**Untracked Files** ⇒ **README.md** ⇒ **Stage**



```
git add README.md
```



### 3.7 Exécuter un commit

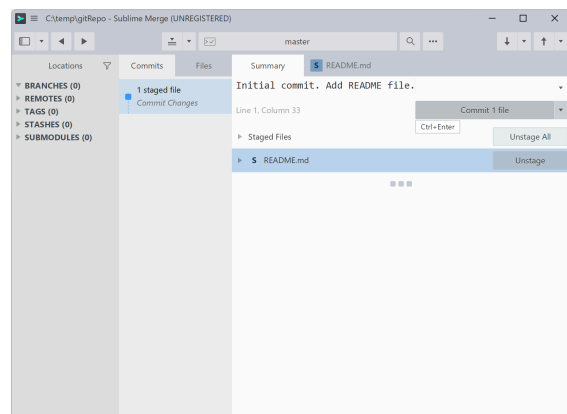
Exécutez maintenant un commit avec la commande suivante :

#### Commandline

```
git commit -m "Initial commit. Add README file."
```

#### GUI

**Commit Message** ⇒ **Initial commit. Add README file.** ⇒ **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 !

### 3.8 Plus d'informations

Quelles informations obtiendrez-vous maintenant avec la commande :

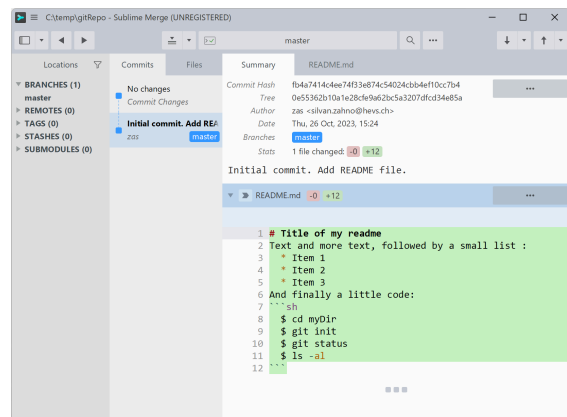
#### Commandline

#### GUI

**Select First Commit** ⇒ **See all informations**



```
git log --oneline
```



Expliquez clairement toutes les informations contenues dans cette ligne.



Écrivez la réponse!

### 3.9 Plus de commits

Effectuez un autre commit pour ajouter un nouveau fichier (vide) **hello\_world.py** dans votre repo.

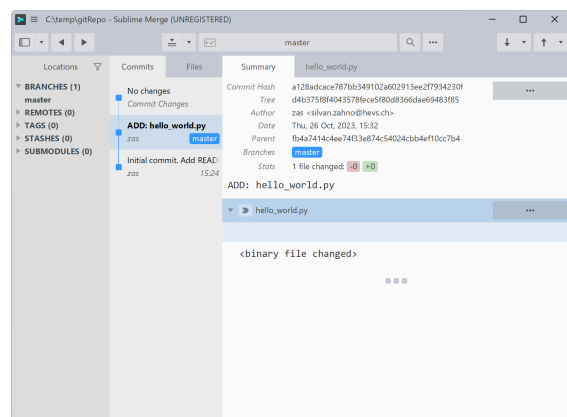
Quelles nouvelles informations vous fournit maintenant la commande :

#### Commandline

```
git log
```

#### GUI

Select Second Commit ⇒ See all informations



Ecrivez la réponse!

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 ».

### 3.10 Checkout commit

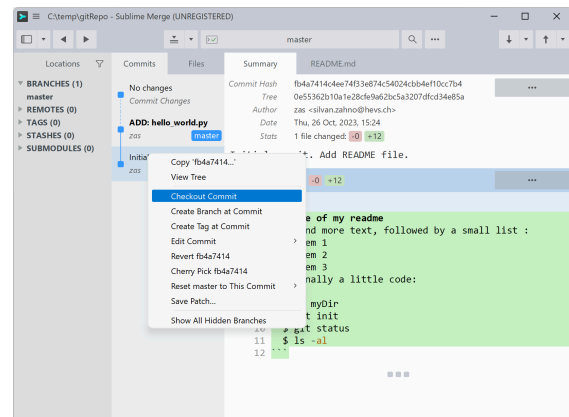
Effectuez maintenant un checkout avec la commande suivante, en utilisant le « short hash » qui correspond à votre premier commit.

#### Commandline

```
git checkout <SHA1> -b inspectingPrev  
# for example  
git checkout fb4a741 -b inspectingPrev
```

#### GUI

Select First Commit ⇒ Checkout commit



Que constatez-vous maintenant lorsque vous consultez le contenu du Working Directory ?



Ecrivez la réponse!



### 3.11 Checkout master

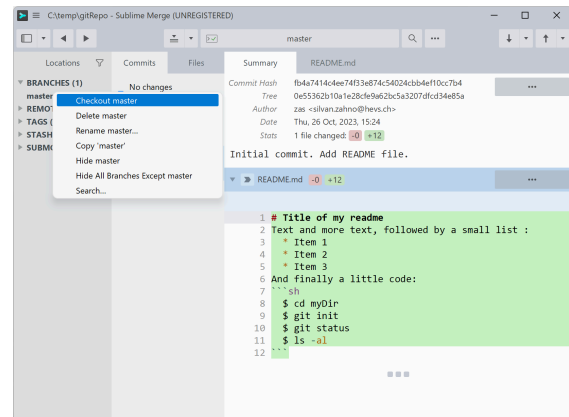
Effectuez maintenant un checkout avec la commande suivante :

#### Commandline

```
git checkout master
```

#### GUI

Branches (1) ⇒ master ⇒ checkout master



Que constatez-vous maintenant en examinant le contenu du working directory ?



Ecrivez la réponse!



## 4 | Branch et Merge

Jusqu'à présent, nous avons utilisé les fonctions de base de git. Il y a aussi les fonctions branch (branche) et merge (fusion), 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.

### 1. Créez un repo Github et ajoutez-le en tant que remote

**Create a new repository**

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)


Required fields are marked with an asterisk (\*).

**Repository template**

No template ▾

Start your repository with a template repository's contents.

**Owner \*** **Repository name \***

 tschinz / gitRepo

gitRepo is available.

Great repository names are short and memorable. Need inspiration? How about [stunning-octo-bassoon](#) ?

**Description (optional)**

test repo for course SyD

☒ **Public**  
Anyone on the internet can see this repository. You choose who can commit.

☐ **Private**  
You choose who can see and commit to this repository.

**Initialize this repository with:**

☐ **Add a README file**  
This is where you can write a long description for your project. [Learn more about READMEs.](#)

**Add .gitignore**

.gitignore template: None ▾

Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

**Choose a license**

License: None ▾

A license tells others what they can and can't do with your code. [Learn more about licenses.](#)

📘 You are creating a public repository in your personal account.

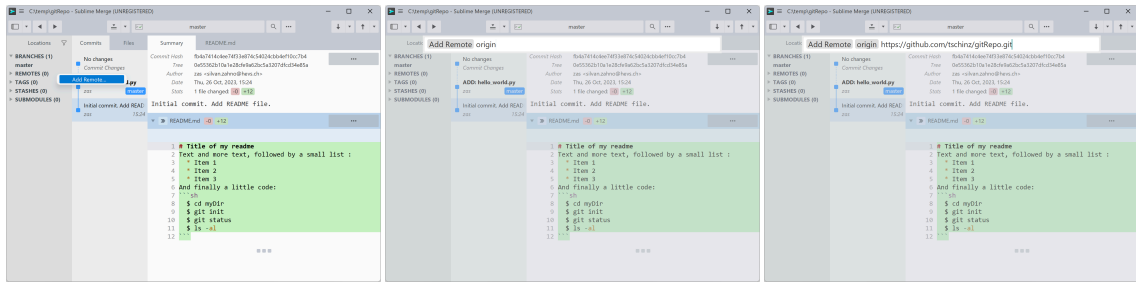
**Create repository**

### Commandline

```
git remote add origin https://github.com/<username>/<reponame>.git
```

### GUI

**Remotes (0) ⇒ Add Remote... ⇒ Remote name ⇒ origin ⇒ Remote URL ⇒ https://github.com/<username>/<reponame>.git**



2. Créez une branche de développement **dev01** dans votre repo local.
3. Créez deux commits sur cette branche :
  - Un pour créer un fichier **hello\_world.py** (ce commit existe déjà).
  - Un pour remplir ce fichier **hello\_world.py**.

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

4. Checkout le **master** branche
5. A partir de la branche master, créer une nouvelle branche de développement **dev02**.
6. Editez le fichier **README.md** et créez un commit sur la branche **dev02s**
7. Merge la branche **dev02** dans **master**.
  - Checkout la branche **master**.
  - Effectuer une fusion « merge **dev02** into **master** »
8. Merge la branche **dev01** dans **master**.
  - Checkout la branche **master**.
  - Effectuer une fusion « merge **dev01** into **master** »
9. Poussez votre repository local vers votre repository distant GitHub.  
**git push origin master.**

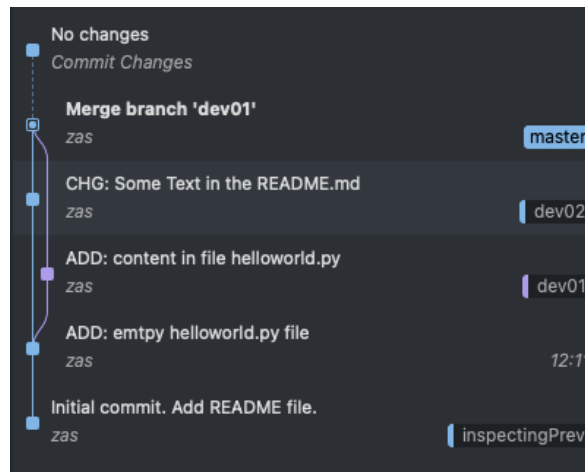


Figure 7 - Statut après la merge des branches **dev01** et **dev02** du repo





## 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/tschinz/gitflow> [5]

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.

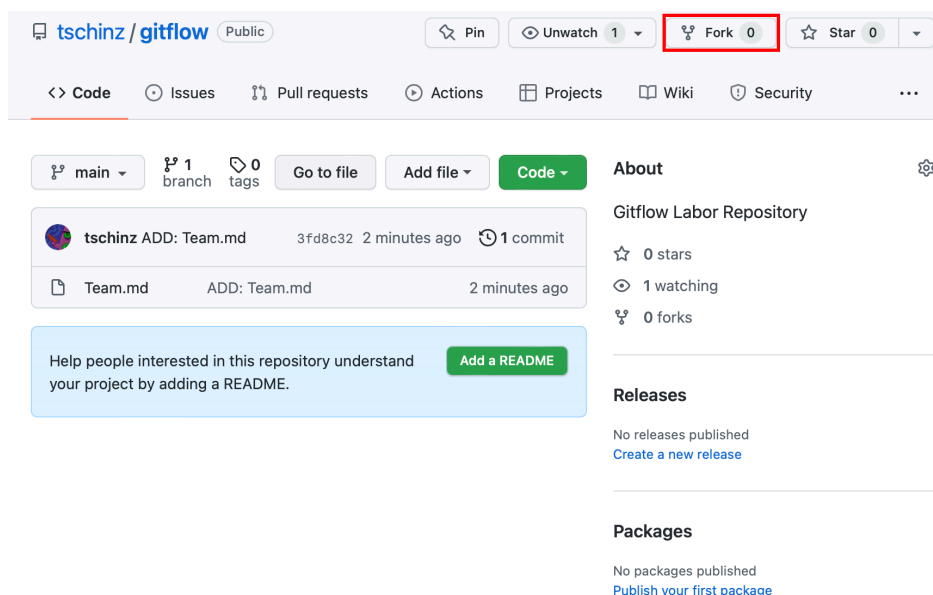


Figure 8 - Bouton Fork pour un repo GitHub

Clonez ensuite notre référentiel commun. L'URL de votre référentiel ressemblera à ceci:

```
git clone https://github.com/<username>/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).



## 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 [conseil](#).

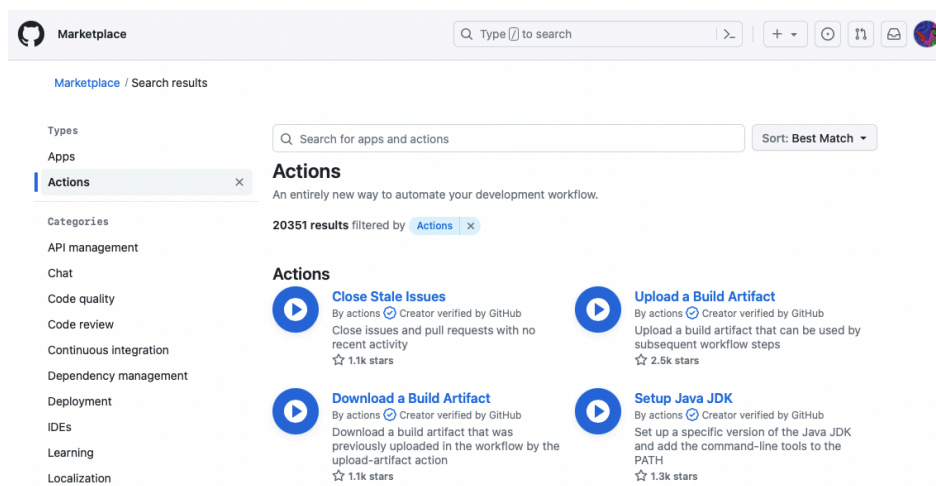
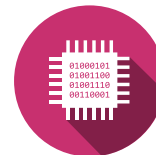


Figure 9 - Github-Actions Marktplatz



## 6.2 Apprendre le branchement Git

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

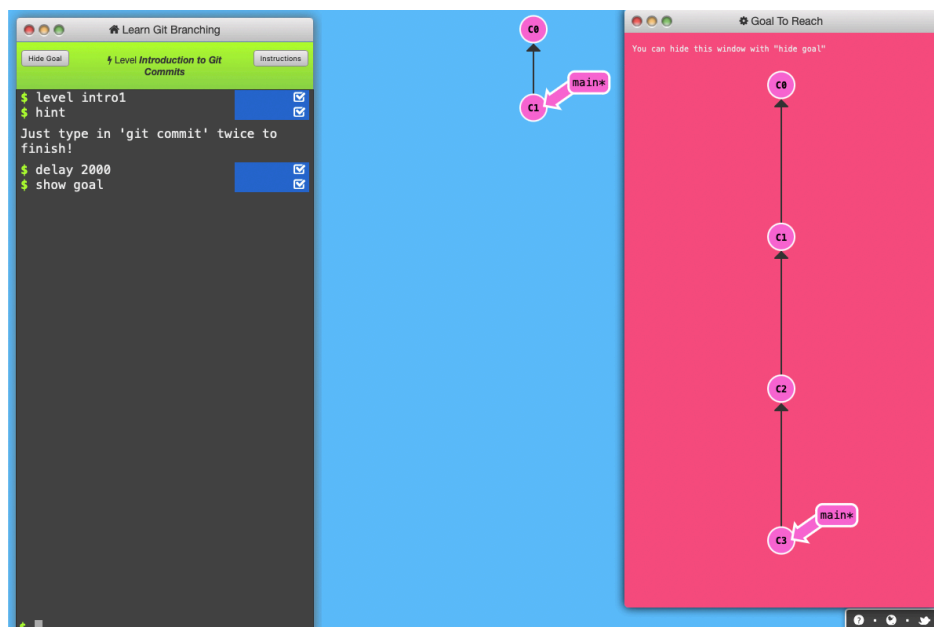


Figure 10 - Lerne Git Branching Website



# A | GIT Commandes

[Github git cheatsheet](#) [6], [7]

## 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