



Introduction au git - Partie A

Installation et configuration



Contents

1 Objectif	1
2 Installation	2
3 Markdown	5
4 Outro	7
A GIT Commandes	8
B Commandes Git les plus utilisées	9
Bibliographie	10

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 apprenons les bases de Markdown dans la 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 dans le 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

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

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.

```
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)
$
```

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 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
```

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].

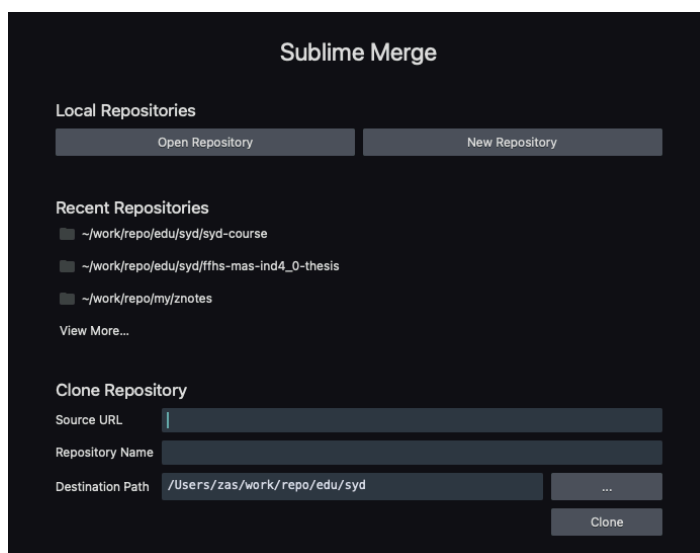


Fig. 2. – Sublime Merge GUI



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).

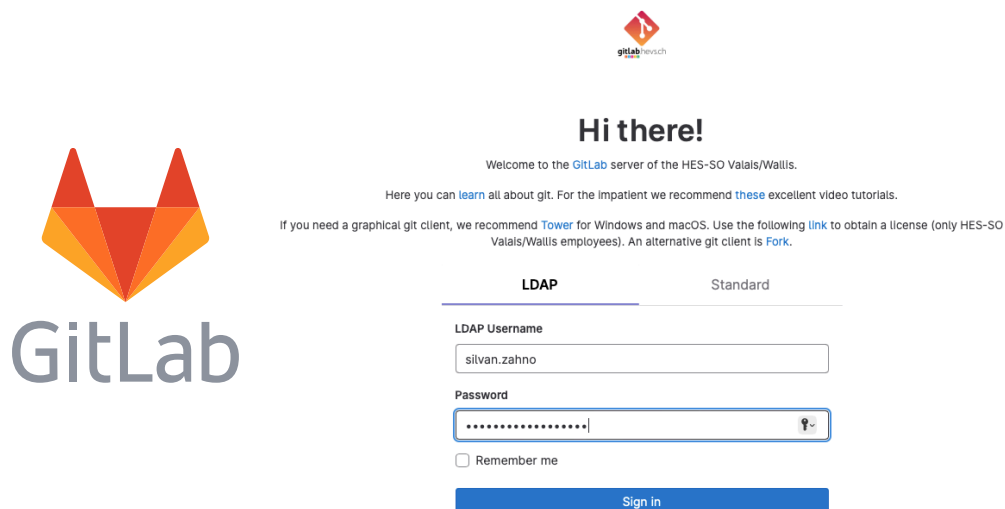


Fig. 3. – Connexion à Gitlab

2.3.2 Github

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

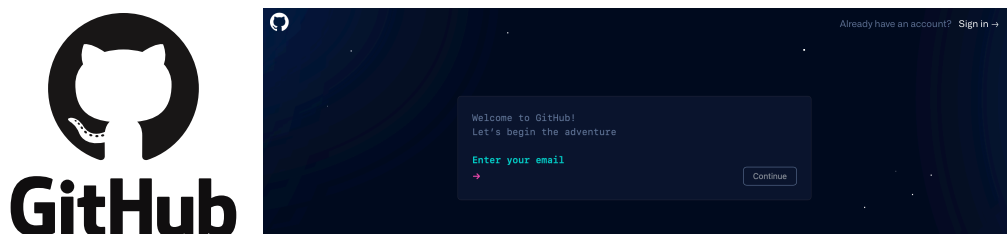


Fig. 4. – Connexion à GitHub

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

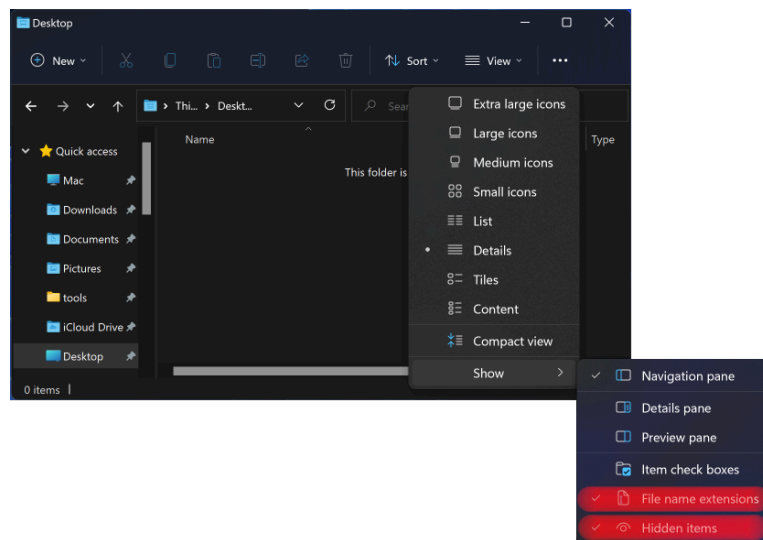


Fig. 5. – Configuration de l'explorateur de fichiers Windows

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, le [intropage](#) 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 le [link](#).

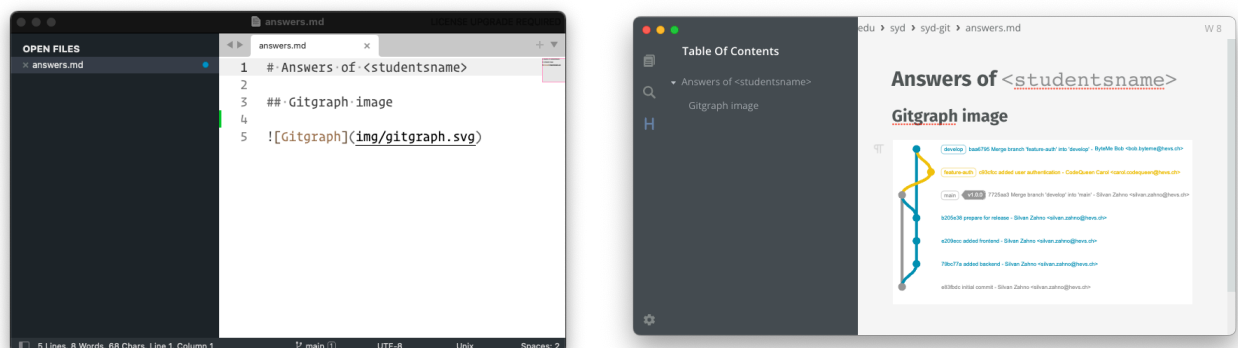


Fig. 6. – A gauche : Editeur de text simple (Sublime Text), à droite : Marktext



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**
~~Strikethrough~~ `monospaced`

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

- 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 python 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** ~~Strikethrough~~ monospaced

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

- List Item 1
- List Item 2

- Numbered List Item 1
- Numbered List Item 2

[Link name](#)

```
// A python 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.



A | GIT Commandes

[Github git cheatsheet](#) [4], [5]

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] « 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] gitlab, « Git Cheatsheet ». 2023.
- [5] « GitHub Git Spickzettel ». Consulté le: 25 avril 2023. [En ligne]. Disponible sur: <https://training.github.com/downloads/de/github-git-cheat-sheet/>