Introduction to Programming in Python

Lucerne University of Applied Sciences and Arts

HOCHSCHULE LUZERN

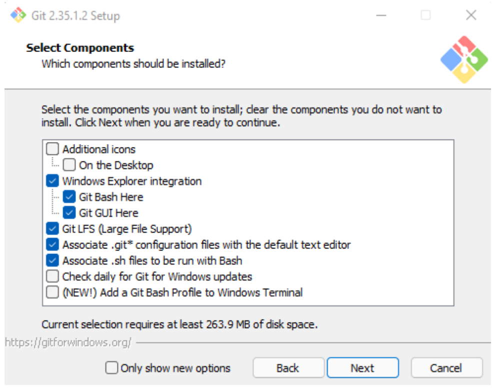
Plan for today

- Installing and using git
- Midterm poll
- Working in PyCharm

Installing git

Windows

- Download git from here: https://git-scm.com/download/win
- Run the installer.


Git 2.35.1.2 Setup

Select Components

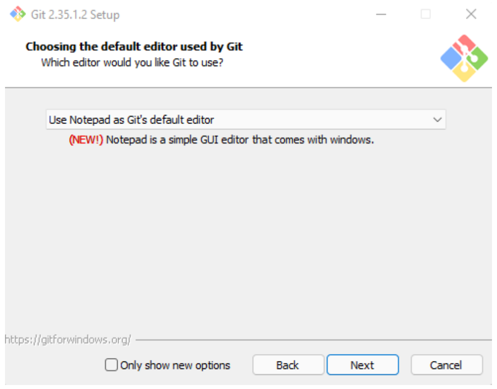
Which components should be installed?

Select the components you want to install; clear the components you do not want to install. Click Next when you are ready to continue.

- [ ] Additional icons
  - [ ] On the Desktop
- [✓] Windows Explorer integration
  - [✓] Git Bash Here
  - [✓] Git GUI Here
- [✓] Git LFS (Large File Support)
- [✓] Associate .git* configuration files with the default text editor
- [✓] Associate .sh files to be run with Bash
- [ ] Check daily for Git for Windows updates
- [ ] (NEW!) Add a Git Bash Profile to Windows Terminal

Current selection requires at least 263.9 MB of disk space.

https://gitforwindows.org/



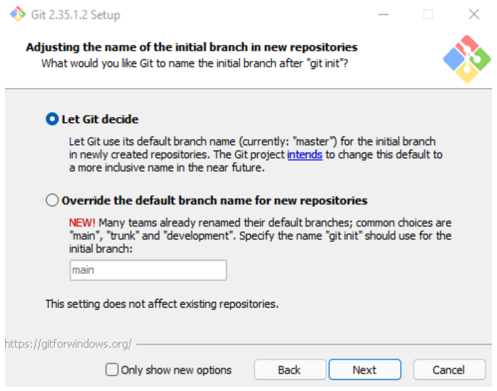
Git 2.35.1.2 Setup

Choosing the default editor used by Git
Which editor would you like Git to use?

Use Notepad as Git's default editor

(NEW!) Notepad is a simple GUI editor that comes with windows.

https://gitforwindows.org/


Git 2.35.1.2 Setup

Adjusting the name of the initial branch in new repositories

What would you like Git to name the initial branch after "git init"?

[✓ ● **Let Git decide**]

Let Git use its default branch name (currently: "master") for the initial branch in newly created repositories. The Git project intends to change this default to a more inclusive name in the near future.

○ **Override the default branch name for new repositories**

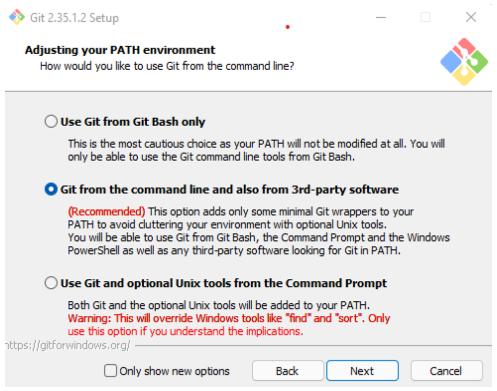
NEW! Many teams already renamed their default branches; common choices are "main", "trunk" and "development". Specify the name "git init" should use for the initial branch:

main

This setting does not affect existing repositories.

https://gitforwindows.org/

☐ Only show new options



Git 2.35.1.2 Setup

Adjusting your PATH environment

How would you like to use Git from the command line?

○ **Use Git from Git Bash only**

This is the most cautious choice as your PATH will not be modified at all. You will only be able to use the Git command line tools from Git Bash.

● **Git from the command line and also from 3rd-party software**

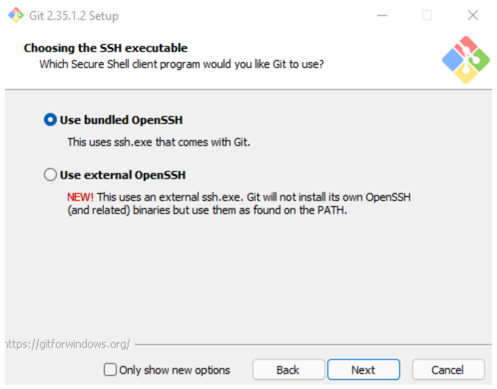
(Recommended) This option adds only some minimal Git wrappers to your PATH to avoid cluttering your environment with optional Unix tools. You will be able to use Git from Git Bash, the Command Prompt and the Windows PowerShell as well as any third-party software looking for Git in PATH.

○ **Use Git and optional Unix tools from the Command Prompt**

Both Git and the optional Unix tools will be added to your PATH. Warning: This will override Windows tools like "find" and "sort". Only use this option if you understand the implications.

https://gitforwindows.org/

☐ Only show new options


Git 2.35.1.2 Setup

Choosing the SSH executable

Which Secure Shell client program would you like Git to use?

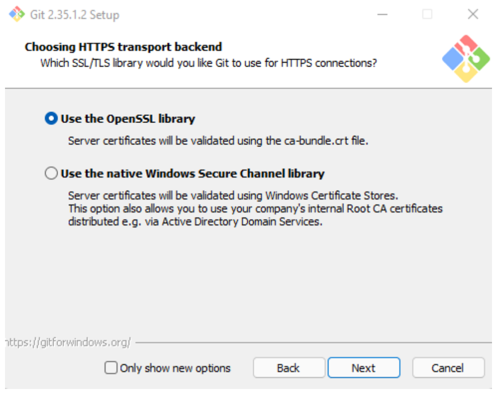
[✓ **Use bundled OpenSSH**]

This uses ssh.exe that comes with Git.

Use external OpenSSH

NEW! This uses an external ssh.exe. Git will not install its own OpenSSH (and related) binaries but use them as found on the PATH.

ttps://gitforwindows.org/


Git 2.35.1.2 Setup

Choosing HTTPS transport backend

Which SSL/TLS library would you like Git to use for HTTPS connections?

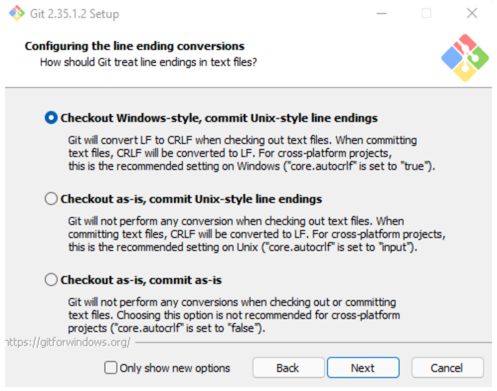
[✓ **Use the OpenSSL library**]

Server certificates will be validated using the ca-bundle.crt file.

Use the native Windows Secure Channel library

Server certificates will be validated using Windows Certificate Stores.
This option also allows you to use your company's internal Root CA certificates
distributed e.g. via Active Directory Domain Services.

https://gitforwindows.org/



Configuring the line ending conversions

How should Git treat line endings in text files?

[✓ **Checkout Windows-style, commit Unix-style line endings**]

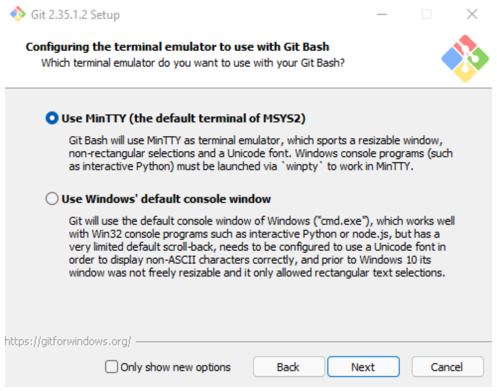
Git will convert LF to CRLF when checking out text files. When committing text files, CRLF will be converted to LF. For cross-platform projects, this is the recommended setting on Windows ("core.autocrlf" is set to "true").

Checkout as-is, commit Unix-style line endings

Git will not perform any conversion when checking out text files. When committing text files, CRLF will be converted to LF. For cross-platform projects, this is the recommended setting on Unix ("core.autocrlf" is set to "input").

Checkout as-is, commit as-is

Git will not perform any conversions when checking out or committing text files. Choosing this option is not recommended for cross-platform projects ("core.autocrlf" is set to "false").



Configuring the terminal emulator to use with Git Bash

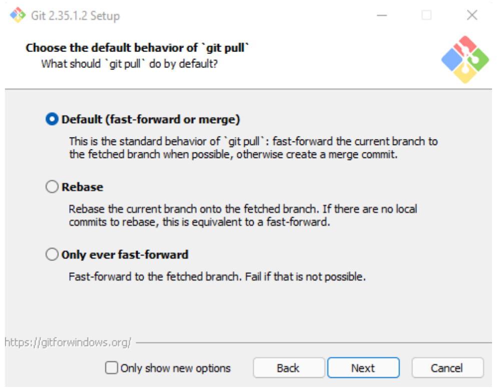
Which terminal emulator do you want to use with your Git Bash?

● **Use MinTTY (the default terminal of MSYS2)**

Git Bash will use MinTTY as terminal emulator, which sports a resizable window, non-rectangular selections and a Unicode font. Windows console programs (such as interactive Python) must be launched via `winpty` to work in MinTTY.

○ **Use Windows' default console window**

Git will use the default console window of Windows ("cmd.exe"), which works well with Win32 console programs such as interactive Python or node.js, but has a very limited default scroll-back, needs to be configured to use a Unicode font in order to display non-ASCII characters correctly, and prior to Windows 10 its window was not freely resizable and it only allowed rectangular text selections.



Choose the default behavior of `git pull`

What should `git pull` do by default?

[✓ **Default (fast-forward or merge)**]

This is the standard behavior of `git pull`: fast-forward the current branch to the fetched branch when possible, otherwise create a merge commit.

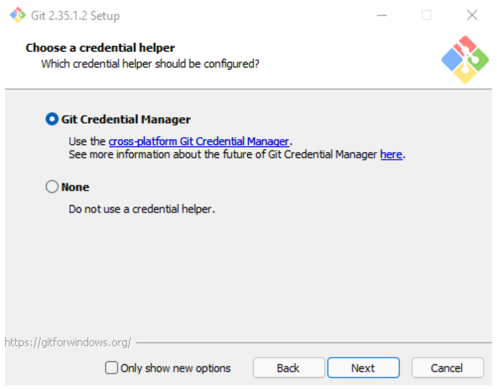
Rebase

Rebase the current branch onto the fetched branch. If there are no local commits to rebase, this is equivalent to a fast-forward.

Only ever fast-forward

Fast-forward to the fetched branch. Fail if that is not possible.

https://gitforwindows.org/



Choose a credential helper

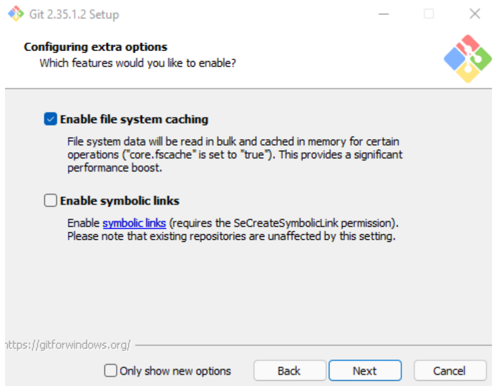
Which credential helper should be configured?

● **Git Credential Manager**

Use the cross-platform Git Credential Manager.
See more information about the future of Git Credential Manager here.

○ **None**

Do not use a credential helper.


Git 2.35.1.2 Setup

Configuring extra options

Which features would you like to enable?

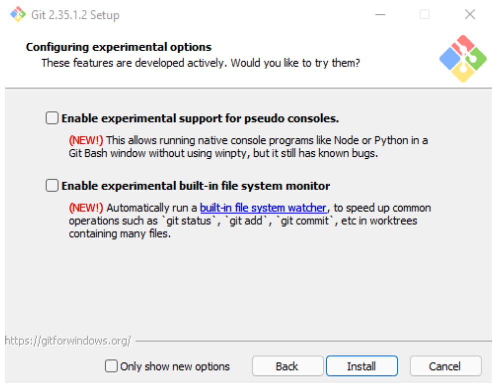
[✓] **Enable file system caching**

File system data will be read in bulk and cached in memory for certain operations ("core.fscache" is set to "true"). This provides a significant performance boost.

[ ] **Enable symbolic links**

Enable symbolic links (requires the SeCreateSymbolicLink permission). Please note that existing repositories are unaffected by this setting.

https://gitforwindows.org/


Git 2.35.1.2 Setup

Configuring experimental options

These features are developed actively. Would you like to try them?

☐ **Enable experimental support for pseudo consoles.**

(NEW!) This allows running native console programs like Node or Python in a Git Bash window without using winpty, but it still has known bugs.

☐ **Enable experimental built-in file system monitor**

(NEW!) Automatically run a built-in file system watcher, to speed up common operations such as `git status`, `git add`, `git commit`, etc in worktrees containing many files.

https://gitforwindows.org/

MacOS

Open a terminal and copy and paste the following two lines one by one, hitting enter after each.

```
/bin/bash -c "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"

brew install git
```

Using git/Github

- `git` is a tool for "version control". It allows several developers to collaborate on the same piece of code, and merge their work later.
- Github is a website that allows you to host your code in the cloud, using `git` to upload/download changes.
- I'll show you some first steps.

Creating a new repo

- Make an account on Github: https://docs.github.com/en/get-started/signing-up-for-github/signing-up-for-a-new-github-account
- Create a new repo: https://docs.github.com/en/get-started/quickstart/create-a-repo

Cloning the repo

- Open cmd (Windows) or a Terminal (on MacOS). Use the `cd` command to navigate to your Documents folder :
    - `cd c:\Users\<username>\Documents` (Windows)
    - `cd /Users/<<username>/Documents` (MacOS)
        - Check out the contents of the folder:
    - `dir` (Windows)
    - `ls` (MacOS)
        - If you don't have one yet, create a `Python` in folder inside `Documents`:
    - `mkdir` Python
- Clone your new repository:
    - `git clone <URL of repo>`
- This creates a new folder with the contents of the repo. You can now modify them, and then push them back to Github.

Making changes

- Open the README.md file in the repo with an editor. Make some changes, and save the file.
- Back in the terminal window, do
    - `git status`
- This shows that the file has changed. We now want to upload the changes back to Github. There are several steps.

Uploading changes

- You first have to tell `git` which changes you want to submit. This is called "staging". It works like this:
    - `git add <filename>`
- You do this for all the files whose changes you want to commit. Next, these changes are "commited" to `git`:
    - `git commit -m "<a nice commit message explaining the changes>"`
- Lastly, upload the changes to github:
    - `git push`
- The last command likely results in an error, because you need to specify your email and password.
- The password is *not* your Github password, but rather an access token that you have to create first. Follow the instructions here: https://docs.github.com/en/authentication/keeping-your-account-and-data-secure/creating-a-personal-access-token\
- You need separate tokens for each machine you want to work on.

Downloading changes

- If you are collaborating with someone (e.g., your teammate for the project), they can now download the changes using (in the project folder in CMD/the terminal)
    - `git pull`
- Problem: this fails if any local changes would be overwritten. One way out is to discard the local changes, using `git stash`, and then doing `git pull` again. This may be useful for the course materials, if you don't need your own changes anymore.

Branches

- For collaboration, one works in `branches`. A branch is basically a separate copy/version of the repo where you can independently make changes.
- Doing `git branch` shows you the branch you are currently on.
- Do `git checkout -b <branch name>` to create a new branch, and then `git push origin <branch name>` to publish the branch to Github (this last part won't work with the course repo, because you don't have commit access there).
- Now, when you `add`, `commit`, and `push` changes, they will be commited to the new branch.
- Later, you can merge the changes in your branch into the main branch by creating a [pull request] on Github.
- You can switch back to the `main` branch by doing `git checkout main`. Note: historically, the main branch was called `master`. That's also true for the course repo

Possible workflow for the course materials

- Clone the repo once:
  - `git clone https://github.com/s-broda/IntroPython`
- At the beginning of the lecture, do `git pull` to download the latest changes.
- If you want to work along, do that in your own branch, so that the `master` branch doesn't have any changes, which would create conflicts next week. Thus, at the beginning of a lecture, do:
  - `git checkout -b my_work`
- If you forget the above step, then you can also do this at the end of the lecture (before commiting).
- At the end of the lecture, commit your changes to your branch (the `-a` option adds any changed files automatically, skipping the `add` step)
  - `git commit -am "me following along"`
- Then, switch back to the master branch:
  - `git checkout master`

Poll

Please take some time to complete the anonymous poll on Ilias. Your feedback is greatly appreciated.

Working with PyCharm

There are no slides for this bit. Some nice tutorials are available on the web, like this one.

PyCharm also has extensive documentation:

- https://www.jetbrains.com/help/pycharm/creating-and-running-your-first-python-project.html#summary
- https://www.jetbrains.com/help/pycharm/debugging-your-first-python-application.html#summary

Required reading

- Read the "Beginner", "Getting Started", and "Collaborating" sections of https://www.atlassian.com/git/tutorials/what-is-version-control (note that these tutorials use BitBucket instead of Github)

Exercise / Homework

- Together with your team mate, set up a repo on Github (under one of your accounts; give your teammate access.
- Clone the repo to both your machines.
- Each of you, create a new branch (different names). Push the branch to Github.
- Inside your branch, create a new file each (different filenames). Push the branch with the new file to Github.
- On Github, create and merge two pull requests, thus merging the changes in the two branches into the "main" branch.

```
In [ ]:
```