Professional Toolkit
BA (Hons) Digital Media Arts

Group Workshop

Week 3 [b], Semester 1: Code Storage and Versioning with Git

Why store and version your code

Storing and versioning your code keeps your progress safe and flexible. You can return to a working state when an experiment goes wrong, capture meaningful milestones as a project evolves, collaborate without overwriting each other and move between computers without juggling loose or zipped files.

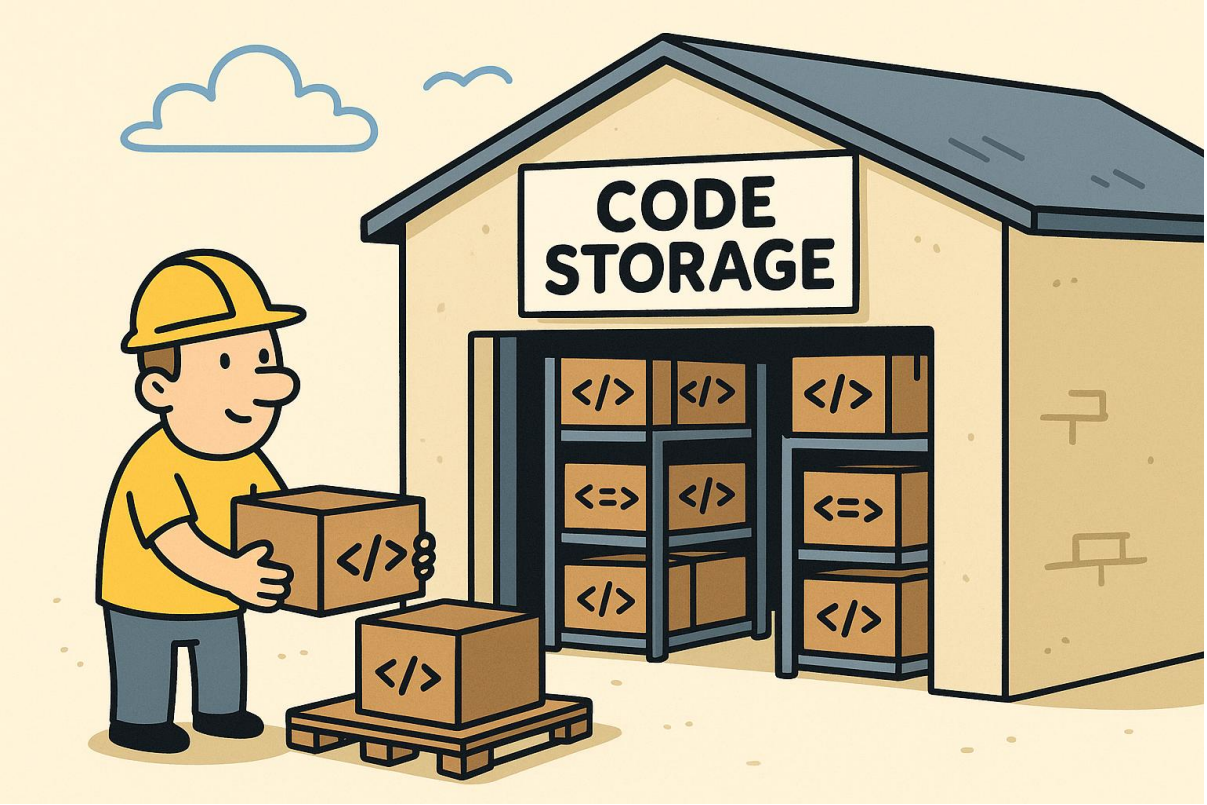

Figure 1 An Illustration of Code Storage and Versioning

In this exercise you will use Git to store and manage versions of your projects so you can work with confidence and keep a reliable record of changes.

Git

Git is a distributed version control software system that enables you to manage different versions of your code or data. It is often used to control source code by programmers who are developing software. However, it is also used by artists and

others to store and version their code and data, including to work collaboratively in a team.

Technologies for storing and versioning code

There are several version control systems in common use. They are defined as follows:

- Git - a distributed version control system that records snapshots (commits), supports branching and merging and works offline.
- Mercurial (hg) - a distributed version control system with a streamlined command set and similar concepts to Git.
- Subversion (SVN) - a centralized version control system where history lives on a central server and clients commit to it.

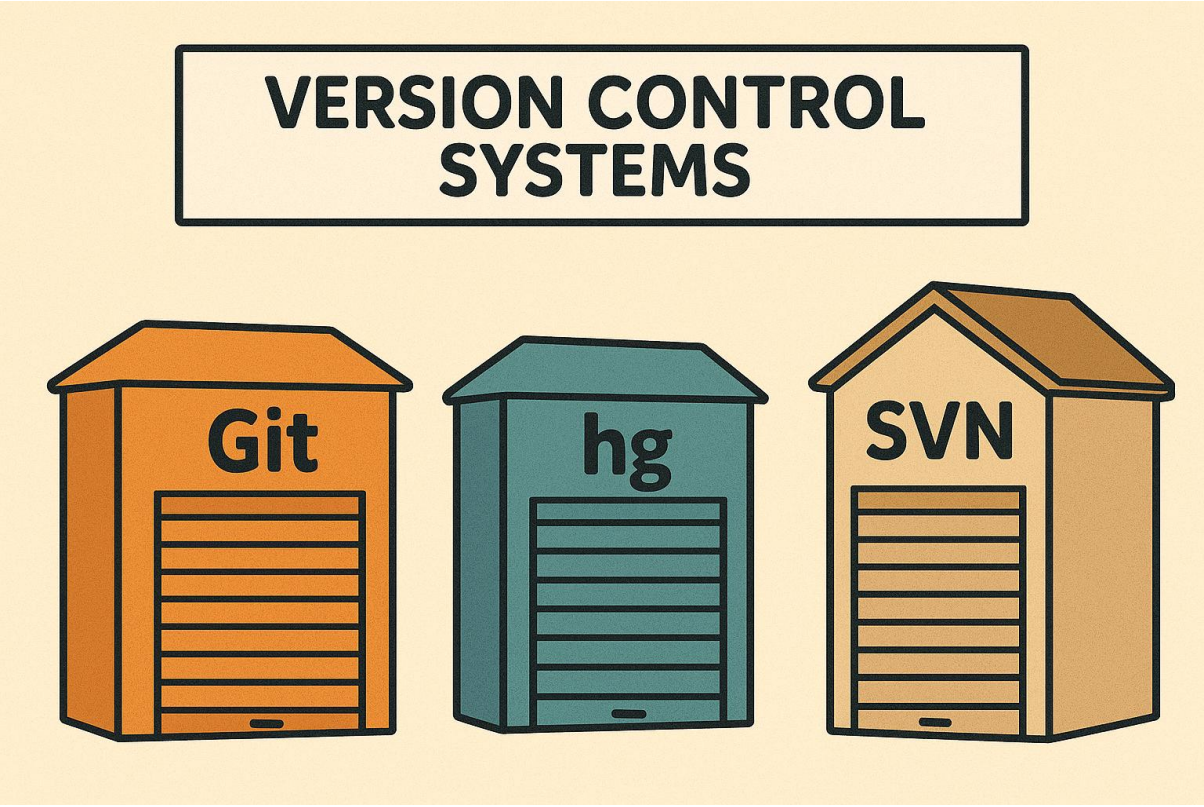

Figure 2 Git, Mercurial (hg) and Subversion (SVN) are three types of versional control systems

In this tutorial we will use Git. It is widely available, integrates with GitHub and Gitee, and makes local commits fast and reliable. The ideas generalize to other systems, but the commands differ.

Professional Toolkit
BA (Hons) Digital Media Arts

How it works

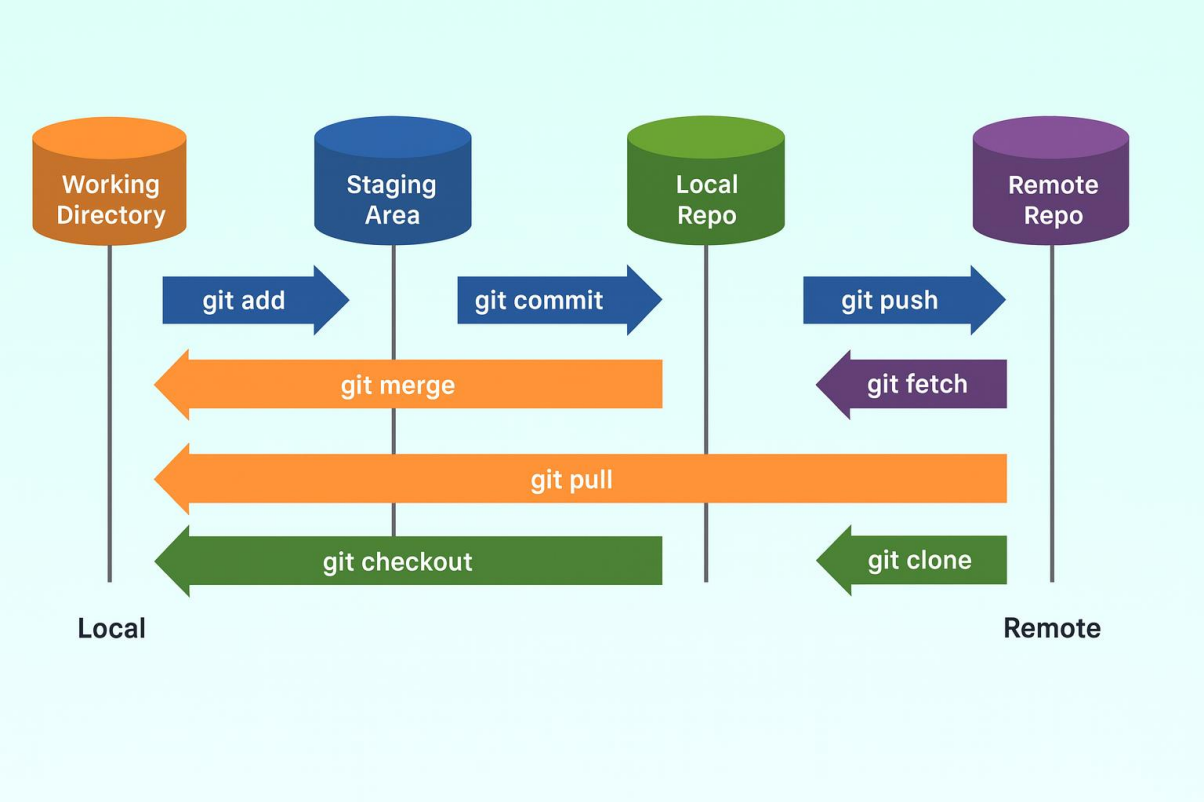

Figure 3 An overview of how Git works

A repository (repo) is a project folder tracked by Git. A commit is a snapshot with a short message describing what changed. A branch is a safe lane to try ideas without affecting the main line of work, and a merge brings work from one branch into another. A remote is the online copy on a service like GitHub or Gitee. In practice you edit files, stage them with add, make a commit, push to the remote and others can pull those changes.

Choose your platform

Host your Git repository at https://github.com. If you cannot use GitHub, then use Gitee at https://gitee.com as an alternative. Follow the tutorial below for the platform that you choose.

Professional Toolkit
BA (Hons) Digital Media Arts

Tutorial A - GitHub

1. Create an account and repository

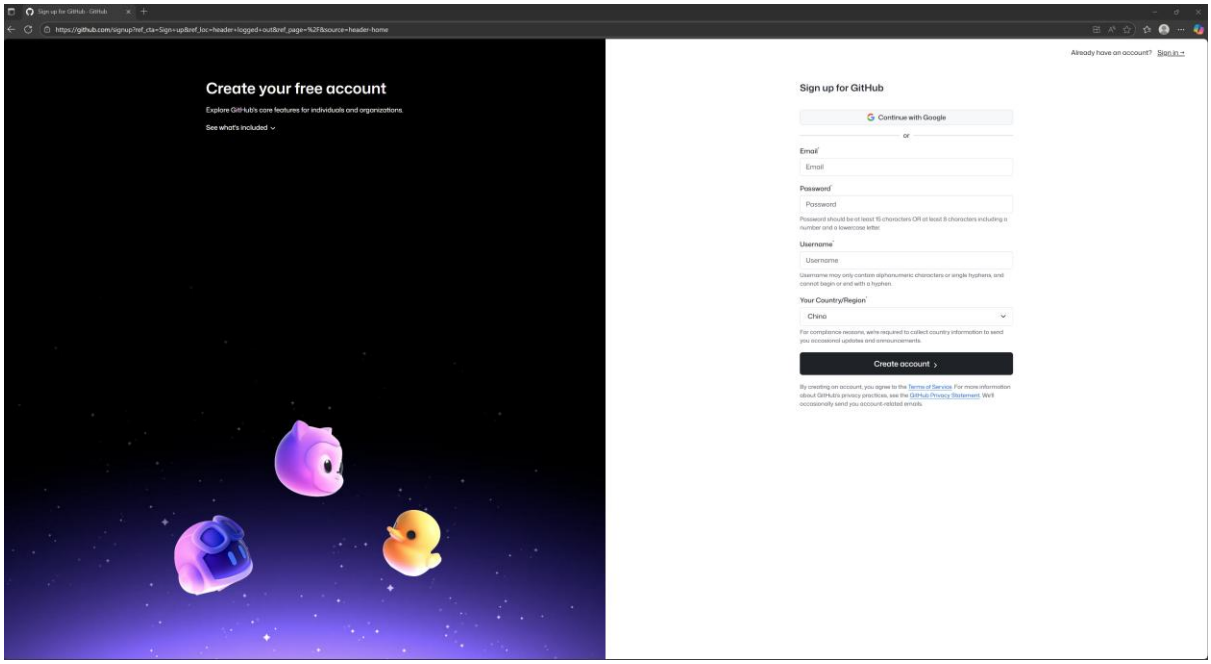

Figure 4 GitHub sign-up page

Open https://github.com and sign up if you do not already have an account (otherwise, sign in if you already do). Create a new repository by selecting New, entering a name for your project and confirming to create the repository. The repository page provides a remote URL, which you will use to connect your local project to GitHub in the next steps.

2. Install and configure Git

Download Git for Windows from https://github.com/git-for-windows/git/releases/download/v2.51.0.windows.1/Git-2.51.0-64-bit.exe and install it (Mac should already have it installed by default).

3. Open your project folder

Open your project folder in the Windows Terminal

Professional Toolkit
BA (Hons) Digital Media Arts

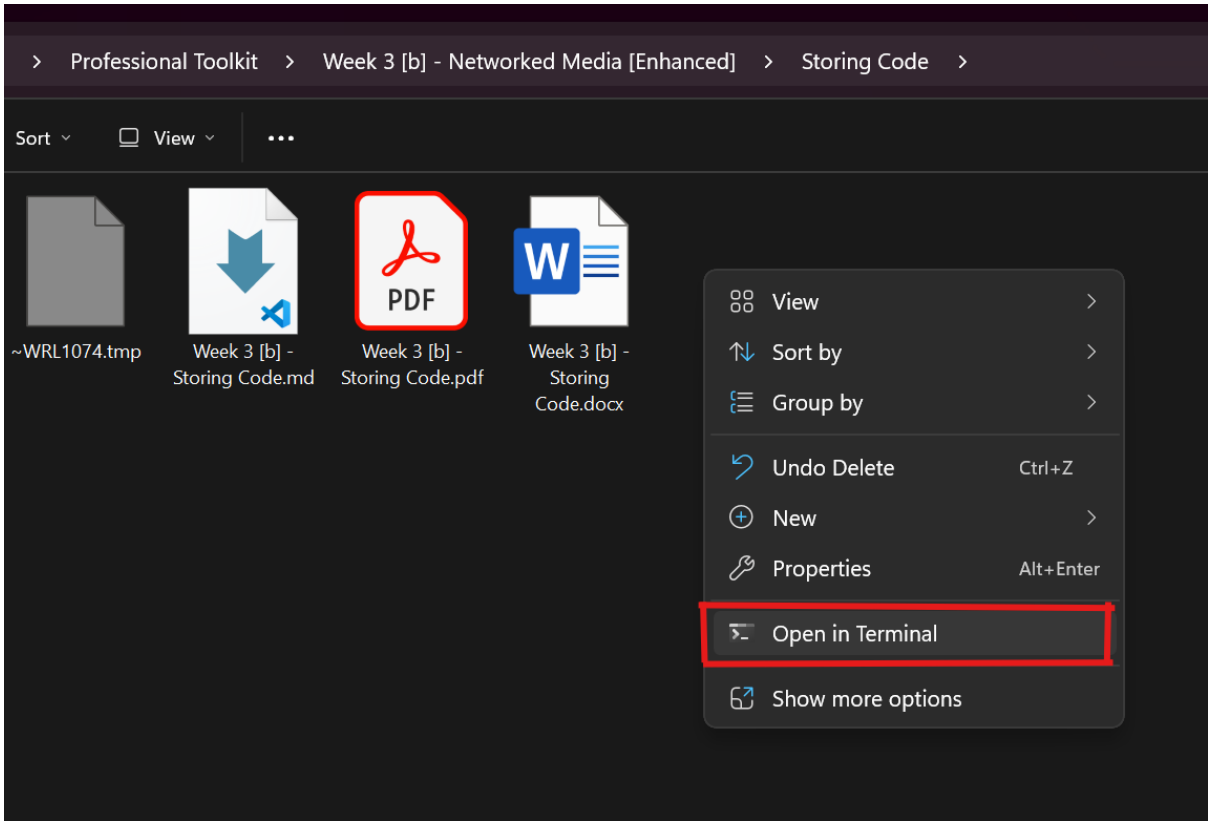

Figure 5 How to open your project in the Windows Terminal (Right-Click in your project folder and choose Open in Terminal)

Continue by running the following commands with your own information replacing what is in the quotes.

```
git config --global user.name "Your Name"
```

```
git config --global user.email "you@example.com"
```

4. Initialize the repo locally and push the first version

You are about to turn the current folder into a Git repository, save a first snapshot, connect it to your GitHub repository and upload that snapshot. This establishes the link so future commits can be pushed easily.

Initialize Git in this folder

```
git init
```

Optionally add a .gitignore file first (See section 6)

Save your current project as the first commit

Professional Toolkit
BA (Hons) Digital Media Arts

```
git add .
```

```
git commit -m "Initial commit"
```

Connect to your GitHub repo (replace USERNAME/REPO)

```
git branch -M main
```

```
git remote add origin https://github.com/USERNAME/REPO.git
```

Push your versioned code to GitHub

```
git push -u origin main
```

5. Daily versioning routine

Each time you finish a meaningful change, create a commit and push it. Commits are your named versions; pushing stores them on GitHub.

Stage and commit meaningful changes

```
git add .
```

```
git commit -m "Describe what changed"
```

Publish your new version

```
git push
```

Optional - SSH (avoid password prompts after setup)

ssh-keygen -t ed25519 -C "you@example.com"

Copy `%USERPROFILE%\.ssh\id_ed25519.pub` to GitHub under Settings, SSH and GPG keys, then add a new SSH key. Switch the remote with: `git remote set-url origin git@github.com:USERNAME/REPO.git`.

6. Create a minimal .gitignore file

Some files don't need to be committed to your project's history. Typical examples are generated build output, caches, logs, editor/OS settings, machine-local configuration and private files such as passwords or API keys.

We create and use a `.gitignore` file to tell Git to skip these files, so that your history stays clean and the repository stays small.

Make a file named `.gitignore` in your project folder and add lines like these:

```
Build/temp
```

Professional Toolkit
BA (Hons) Digital Media Arts

```
node_modules/

build/

.dist/

.vscode/

.DS_Store

Thumbs.db

*.log
```

7. Large files (optional Git LFS)

If you must version large files such as .mp4, .mov or .psd, then use Git Large File Storage (Git LFS), which is an open-source Git extension designed to handle large files efficiently within Git repositories. To install and use it, do the following:

```
git lfs install
```

In your repo, track needed types

```
git lfs track "*.mp4"

git lfs track "*.mov"

git lfs track "*.psd"
```

Commit the LFS config

```
git add .gitattributes

git commit -m "chore: enable LFS for media"
```

Free plans often limit LFS storage and bandwidth.

8. What you should be able to do now

You should have a repository on GitHub or Gitee, a local repository initialized with Git, an initial commit pushed online, and the ability to make a change, commit it and push a new version. If needed, you can also set up SSH for easier authentication and Git LFS for large files.

Professional Toolkit
BA (Hons) Digital Media Arts

Tutorial B - Gitee

1. Create an account and repository

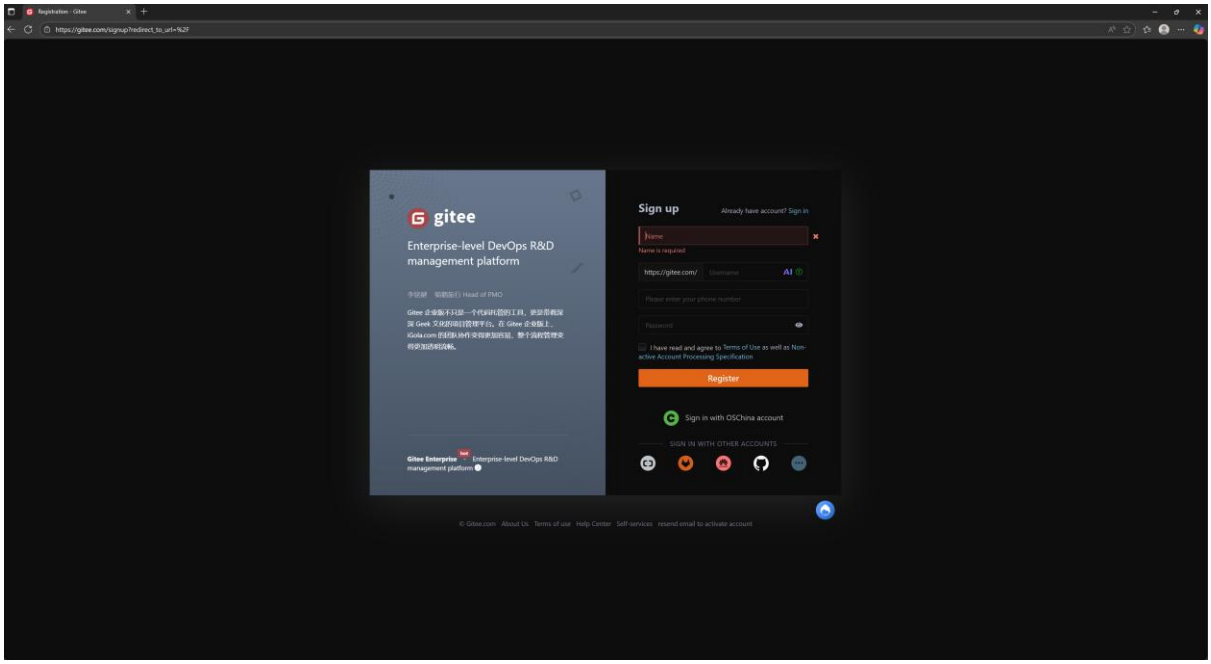

Figure 6 Gitee sign-up page

Open https://gitee.com/signup to create an account if needed. Use the plus menu to create a repository, give it a name and create it. The repository page shows a remote URL; which you will connect to from your local project shortly.

2. Install and configure Git

Download Git for Windows from https://github.com/git-for-windows/git/releases/download/v2.51.0.windows.1/Git-2.51.0-64-bit.exe and install it (Mac should already have it installed by default).

3. Open your project folder

Open your project folder in the Windows Terminal

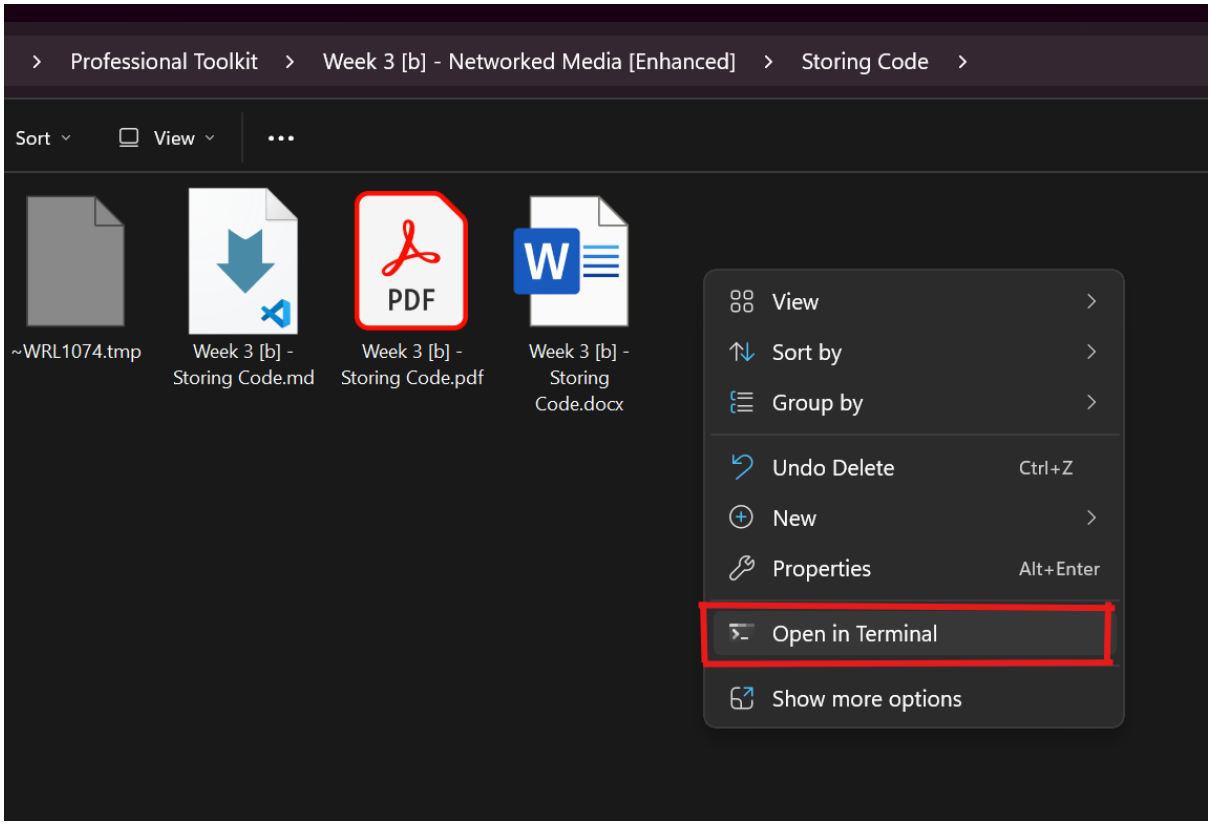

Figure 7 How to open your project in the Windows Terminal (Right-Click in your project folder and choose Open in Terminal)

Continue by running the following commands with your own information replacing what is in the quotes.

```
git config --global user.name "Your Name"
```

```
git config --global user.email "you@example.com"
```

4. Initialize the repo locally and push the first version

These commands initialize Git in your folder, save a first commit, connect to your Gitee repository, and upload that commit so it's stored online.

```
git init
```

```
git add .
```

```
git commit -m "Initial commit"
```

```
git branch -M main
```

```
git remote add origin https://gitee.com/USERNAME/REPO.git
```

Professional Toolkit
BA (Hons) Digital Media Arts

```
git push -u origin main
```

5. Daily versioning routine

After meaningful changes, commit and push to save the new version online.

```
git add .
```

```
git commit -m "Describe what changed"
```

```
git push
```

Optional - SSH on Gitee

ssh-keygen -t ed25519 -C "you@example.com"

Add `%USERPROFILE%\.ssh\id_ed25519.pub` in Gitee under Settings, SSH Public Keys, then add. Switch the remote with: `git remote set-url origin git@gitee.com:USERNAME/REPO.git`.

6. Create a minimal .gitignore file

Some files don't need to be committed to your project's history. Typical examples are generated build output, caches, logs, editor/OS settings, machine-local configuration and private files such as passwords or API keys.

We create and use a `.gitignore` file to tell Git to skip these files, so that your history stays clean and the repository stays small.

Make a file named `.gitignore` in your project folder and add lines like these:

```
Build/temp
```

```
node_modules/
```

```
build/
```

```
.dist/
```

```
.vscode/
```

```
.DS_Store
```

```
Thumbs.db
```

```
*.log
```

7. Large files (optional Git LFS)

Professional Toolkit
BA (Hons) Digital Media Arts

If you must version large files such as .mp4, .mov or .psd, then use Git Large File Storage (Git LFS), which is an open-source Git extension designed to handle large files efficiently within Git repositories. To install and use it, do the following:

```
git lfs install
```

In your repo, track needed types

```
git lfs track "*.mp4"
```

```
git lfs track "*.mov"
```

```
git lfs track "*.psd"
```

Commit the LFS config

```
git add .gitattributes
```

```
git commit -m "chore: enable LFS for media"
```

Free plans often limit LFS storage and bandwidth.

8. What you should be able to do now

You should have a repository on GitHub or Gitee, a local repository initialized with Git, an initial commit pushed online, and the ability to make a change, commit it and push a new version. If needed, you can also set up SSH for easier authentication and Git LFS for large files.

9. Troubleshooting

If something goes wrong while committing or pushing, try the following quick fixes.

Push rejected due to large files

Remove generated/build folders; use Git LFS for required big assets.

Repeated login prompts

Complete browser sign-in when asked or switch to SSH.

Merge conflicts

Open conflicted files, keep the intended lines, remove `<<<<<<<`, `=======`, `>>>>>>>`, then commit.

Windows line endings

git config --global core.autocrlf true` can help normalize line endings.