**Try it! Working with GIT in terminal**

**Terminology**

A **repository** is like a folder, and you work with it as if it were a simple folder. However, it also contains information about all changes made to all files in this folder.

A **commit** is a record of a change in the repository. It allows you to track the evolution of a file. So, a Git repository, storing initial files and all changes, allows you to view the state of the repository at any moment in time.

**Steps to repeat**

1. Create a folder for your project and open it in terminal/command line.
2. Initialize repository in the folder with git init.
3. Add a file to the folder (todolist.txt for example).
4. Stage added file with git add todolist.txt.
5. Commit changes with git commit -m “name of commit”.

**Try it! Working with GIT in graphical interface**

**Steps to repeat**

1. Open your repository in VSCode or Sourcetree, modify an existing file and add a new folder with a file inside.
2. Stage the modified and newly created files using GUI.
3. Commit your changes with a clear commit message.
4. Repeat staging and committing steps using a different GUI application.
5. Add or modify another file, stage all changes (git add .), and commit via command line.

**Try it! Viewing history of changes**

**Steps to repeat**

1. View commit history in terminal with git log and use git log --oneline for shorter output.
2. Filter commit history by author or date using options like --author, --since, and --until.
3. View commit history in VSCode using Git History extension.
4. View commit history and file changes in Sourcetree.
5. View previous file versions in terminal with git show <commit\_hash> <file\_path>.
6. Compare commits using git diff <hash1> <hash2> <file\_path>.
7. Navigate to previous commits with git checkout <commit\_hash> and return to latest commit with git checkout main.

**Try it! Branching**

**Terminology**

**Branch** – an independent version of your repository that allows you to safely develop features or make changes without affecting the main (or master) branch.

**Merge** – combining changes from one branch into another.

**Steps to repeat**

1. Create a new branch:
   * git branch new\_branch
   * or directly switch: git checkout -b new\_branch
2. Switch between branches:
   * git checkout branch\_name or git switch branch\_name.
3. Create a branch from a previous commit:
   * Switch to specific commit (git checkout <hash>), then create branch from it:git switch -c branch\_from\_old\_commit.
4. Modify files and commit changes to the newly created branches.
5. Merge branch into master:
   * Switch to master branch:git checkout master
   * Merge branch:git merge branch\_name
6. Perform squash merge to combine multiple commits into one:
   * git merge --squash source\_branch
7. Manage branches with GUI:
   * Create, switch, merge, and delete branches in VSCode or Sourcetree GUI.
8. Delete branches:
   * Terminal:git branch --delete branch\_name
   * GUI: right-click branch and select delete.

**Try it! Resolving conflicts**

**Terminology**

**Merge Conflict** – occurs when Git cannot automatically merge changes from two branches.

**Steps to repeat**

1. Modify the same line in a file in two separate branches.
2. Attempt to merge these branches (git merge branch\_name) and identify conflicts.
3. Open the conflicted file to manually resolve conflicts, editing conflicting sections marked by Git (<<<<<<<, =======, >>>>>>>).
4. Stage the resolved files (git add file\_name), then complete merge (git merge --continue).
5. Verify the merge result in a GUI tool (e.g., Sourcetree or VSCode).
6. Repeat conflict resolution using VSCode's Merge Editor.
7. Complete or abort the merge (git merge --continue or git merge --abort).

**Try it! Collaborative development**

**Terminology**

**Remote Repository** – A shared repository hosted remotely (GitHub, GitLab) for collaboration.

**Fetch** – Retrieves updates from the remote repository without merging.

**Pull** – Retrieves updates and merges them into your current branch.

**Push** – Uploads local commits to the remote repository.

**Steps to repeat**

1. Create a remote repository on GitHub or GitLab.
2. Connect your local repository to the remote by running:
   1. git remote add origin <repository\_url>
   2. git push -u origin master
3. Push local changes to remote using:
   1. git push
4. Edit files directly in the remote repository interface and commit changes.
5. Update your local repository by running:
   1. git fetch
   2. git pull
6. Create and view remote branches:
   1. View remote branches: git branch -r
   2. Fetch branches: git fetch
   3. Merge remote branches:
      1. git merge origin/branch\_name
7. Update files in remote repository interface, fetch and pull those changes locally:
   1. git fetch
   2. git pull
8. Explore remote branches in GUI tools (Sourcetree, VSCode).

**Try it! Undoing changes**

**Steps to repeat**

1. Modify a file without committing, then discard changes using the GUI (VSCode Source Control Panel).
2. Make unwanted changes, commit them, and revert the commit:
   * Using GUI (Sourcetree) with the "revert" option.
   * Using command line: git revert <commit\_hash>.
3. Confirm the reverted changes appear correctly.
4. Practice viewing commit hashes using git log and revert a commit from the command line.

**Try it! Useful GIT features**

**Steps to repeat**

1. Create a .gitignore file and specify files or patterns to ignore.
2. Clone an existing remote repository using git clone <repository\_url>.
3. Merge latest changes from the main branch into your branch or use git rebase.
4. Use git cherry-pick to apply specific commits from other branches.  
   (Resolve conflicts with git cherry-pick --continue.)