Version Control

Notes by José A. Espiño P. [1]

Spring Semester 2023–2024

[1]The content in these notes is sourced from what was covered in the course the document is named after. I claim no autorship over any of the contents herein.

Contents

1 Software Collaboration

Version Control, also known as source control or source code management, is a system that records all modifications to the files in a project for tracking purposes and it allows for the recovery of previous versions of the files. This is achieved by providing developers with access to all previous versions of the code. It is also a tool for collaboration, as it allows for the coordination of the work of multiple developers. For tracking purposes, all changes are recorded with the identity of the person who made them, the time they were made, and a description of the changes. Another powerful aspect of version control is peer review, a process where other developers can review the changes made by a developer before they are merged into the main codebase, thus catching bugs and ensuring the code is of good quality. Version control is a key element of development operations (DevOps), a set of practices that aim to automate and integrate the processes between software development and IT teams, so they can build, test, and release software faster and more reliably. Often, DevOps philosophy implies the use of processes from the agile methodology, in which a team normally plans an executes work in spans of two weeks, known as iterations. Each iteration has a set of goals and a set of tasks that need to be completed.
There are two key types of control systems. Centralised control systems (CVCS) are systems where all the files are stored in a central server, and developers (clients) download the files to their local machines to work on them. They then upload the files back to the server when they are done. Every operation needs a connection to the server itself. Furthermore, viewing the history of changes requires that you are connected to the server to retrieve and view them. This kind of system is more accessible and gives more access controls to users, but it might be slower. The other kind of control system is the distributed control system (DVCS), where every developer has a copy of the entire repository on their local machine—essentially, every user is a server. This allows for offline work and faster operations, but it might be harder to manage access controls.
Some common tools usually used in conjunction with version control systems are:

- **Workflow**: A set of rules that dictate how the code is managed and how changes are made.
- **Continuous Integration**: A practice where developers integrate their code into a shared repository frequently, usually several times a day. Each integration is verified by an automated build, allowing teams to detect problems early. This also reduces the number of merge conflicts.

- **Continuous Delivery**: A practice where code changes are automatically built, tested, and prepared for a release to production. This allows for faster and more reliable releases.
- **Continuous Deployment**: A practice where every change that passes previous stages is released onto a test (staging) environment to validate the deployment package and software changes. Once validated, it is deployed automatically to the live (production) environment for customers.

2 Command Line

The **command line** is just one of many ways to interact with your computer; as opposed to GUIs, however, it allows developers to perform tasks more directly and efficiently. It is a text–based interface that lets you communicate with your computer through **commands**. Commands can be used alongside **flags**, which modify the command by either changing or extending their functionality. Let us start with some of the most basic commands:

- cd
  Stands for change directory. It is used to point the command line to a specific directory. For example, `cd /` will take you to the root directory, and `cd ..` will take you to the parent directory.
- touch
  This command is used to create a new file. For example, `touch file.txt` will create a new file called `file.txt`.
- mkdir
  This command is used to create a new directory. For example, `mkdir new_directory` will create a new directory called `new_directory`.
- history
  This command is used to display a list of all the commands that have been entered in the command line.
- code
  This command is used to open Visual Studio Code from the command line.
- man
  This command is used to display the manual for a specific command. For example, `man ls` tells you detailed instructions on the use of the command `ls`.
- ls
  This command is used to list the contents of a directory. For example, `ls` will list all the files and directories in the current directory. Some common flags used with this command are `-a` to list all files, including hidden ones, and `-l` to list files in long format, which includes permissions, owner, group, size, and date.

- pwd
  This command is used to display the current directory. For example, `pwd` will display the current directory.
- mv
  This command is used to move files or directories. For example, `mv file.txt new_directory` will move the file `file.txt` to the directory `new_directory`.
- rm
  This command is used to remove files or directories. For example, `rm file.txt` will remove the file `file.txt`.
- cp
  This command is used to copy files or directories. For example, `cp file.txt new_directory` will copy the file `file.txt` to the directory `new_directory`.
- cat
  This command is used to display the contents of a file. For example, `cat file.txt` will display the contents of the file `file.txt`.
- less
  This command is used to display the contents of a file one page at a time. For example, `less file.txt` will display the contents of the file `file.txt` one page at a time.
- grep
  This command is used to search for a specific string in a file. For example, `grep "string" file.txt` will search for the string `"string"` in the file `file.txt`.
- wc
  This command is used to count the number of lines, words, and characters in a file. For example, `wc file.txt` will display the number of lines, words, and characters in the file `file.txt`.

You can create your own commands by creating a **shell script**. A shell script is a file that contains a sequence of commands for a shell to execute. To create a shell script, you need to create a file with the extension `.sh` and write the commands you want to execute in it. For example, the following is a simple shell script that prints "Hello, World!" to the command line:

```
#!/bin/bash
echo "Hello, World!"
```

Note that shell script must always start with `#!/bin/bash` to indicate that it is a shell script. You can then execute the shell script by running the command `./script.sh` in the command line. Before you can execute the script, you need to give it execute permissions by running the command `chmod +x script.sh`.
Pipes are another strong element of pipes: they allow you to use the output of a command as the input of another. For example, the command `ls | grep "file"` will list all the files in

the current directory and then search for the string `"file"` in the output.
Linux commands take an input and give an output. **Redirection** allows us to change what exactly will be the input and output of a given command. There are three main types of IO redirections: standard input, which is represented by a zero, standard output, which is represented by a one, and standard error, which is represented by a two. If you want to redirect the standard input from the keyboard to, say, the output of a command, you can use the < operator. For example, the command `cat < file.txt` will display the contents of the file `file.txt`. If you want to redirect the standard output of a command to a file, you can use the > operator. For example, the command `ls > files.txt` will list all the files in the current directory and write the output to the file `files.txt`, as opposed to the output being sent to your computer screen.. If you want to redirect the standard error of a command to a file, you can use the 2> operator. For example, the command `ls 2> error.txt` will list all the files in the current directory and write any errors to the file `error.txt`. If you want to append the standard output of a command to a file, you can use the » operator. For example, the command `ls » files.txt` will list all the files in the current directory and append the output to the file `files.txt`.
Grep stands for global regular expression print and it is a tool for searching for a specific string in a file using regular expressions. Some common flags used with this command are `-i` to ignore case, `-v` to invert the match, and `-c` to count the number of matches. Exact matches can be found using the `-w` flag.

3 Working with Git

Git is a version control system that helps users keep track of the changes made to any of the files in a project. It was originally designed by Linus Torvalds to keep track of all the changes made to the Linux kernel. A closely related service to git is **GitHub**, which is a cloud–based hosting service that lets you manage Git repositories from a user interface. It uses and extends on Git version control features, such as access control, pull requests, and automation.
Git repositories contain a file named `.git` that contains all the information about the repository, including the history of changes, the current state of the repository, and the configuration of the repository. The `.git` file is hidden, so you will not see it when you list the contents of a directory. This folder is created automatically by GitHub or manually when initialising a repository using the command `git init`. Git has a three stage workflow. Firstly, the modified stage is when you have made changes to a file but it is still not tracked by Git. The second stage, staged, is files and modifications start being tracked by Git. Lastly, the committed stage is when the changes are saved to the repository and added to the remote repository.
Before making any changes to a repository, it is good practice to check if there are any changes or commits that have not yet been pushed to the remote repository. You can do this by running the command `git status`. Once you have made modifications, you can add new/-modified files to the staging area by running the command `git add`. You can then commit the changes to the repository by running the command `git commit`. You can also use the `-m` flag to add a message to the commit. For example, the command `git commit -m "Add new file"` will commit the changes to the repository with the message `"Add new file"`. If

at any point you would like to unstage a file, you just need to run the command `git restore -staged`. It is important to note that the commited changes will not appear on the remote repository until you push them. You can do this by running the command `git push -u origin main`. The `-u` flag is used to set the upstream branch, which is the default branch that the changes will be pushed to.

Branching is another important feature of Git. A branch is a separate line of development that allows you to work on a feature without affecting the main codebase. You can create a new branch by running the command `git checkout -b`. For example, the command `git checkout -b new_branch` will create a new branch called `new_branch`. Another way to create a branch is by using the command `git branch`. The difference is that the former command not only creates the branch but also moves you from the main branch onto the newly created one. The main branch exists in isolation; it will only be updated with any changes upon being merged with the main branch. Usually, this is achieved via **pull requests**, a way to ask for a peer to review the changes made in the branch before merging them onto the main branch.

The opposite of the push command is the `git pull` command, which is used to fetch and merge changes from the remote repository to the local repository. The `git clone` command is used to create a copy of an entire remote repository onto your computer; the difference is that `git pull` just updates an already existing repository.

In Git, a **head** refers to the current commit that you are working on. The `git log` command is used to display the history of commits in the repository. Some common flags used with this command are `-oneline` to display each commit on a single line, `-graph` to display the commit history as a graph, and `-decorate` to display the names of branches and tags next to the commits.

Another useful command in Git is **diff**, which is used to display the differences between two files. For example, the command `git diff file1.txt file2.txt` will display the differences between the files `file1.txt` and `file2.txt`. diff can also be sued to compare different commits and different branches. This is achieved by using the `git diff` command followed by the commit or branch you want to compare. For example, given branch1 and branch2, the command `git diff branch1 branch2` will display the differences between the two branches.

Blame is a useful command that is used to display the author and the last commit that modified each line of a file. For example, the command `git blame file.txt` will display the author and the last commit that modified each line of the file `file.txt`.

Forking is a feature of GitHub that allows you to create a copy of a repository in your own account. This is useful when you want to make changes to a repository without affecting the original repository. Once you have made the changes, you can create a pull request to ask the owner of the original repository to merge your changes.