Introduction to version control

In this module, you'll be introduced to the concept of version control, which will make managing and rolling back your code look super easy. You'll learn how to differentiate between files and the tools at your disposal to make this happen. Next, you'll be introduced to Git and you can leverage that platform to improve your coding abilities. Once you've got a grasp on what Git is, you'll install it and start using it to create and clone code repositories. Last up, you'll deep dive into Git in order to get more familiar with the different tools and commands it has to offer.

Objectives:
- Understand the concept of version control and the benefits it brings
- Utilize the diff and patch commands to automate differentiating and editing files.
- Understand what Git is and the benefits that it brings.
- Install Git on their machine.
- Utilize Git by creating and cloning repositories, adding code, checking the status of code, and commiting code.

Before version control

Intro to Module 1: Version Control

Keeping historical copies

Work on the most primitive form of version control, keeping historical copies. These copies let you see what the project was like before, and go back to that version if you end up deciding that the latest changes were wrong. They also let you see the progress of the changes over time, and maybe even help you understand why a change was made. We say that this is primitive because it's very manual and not very detailed. First, you need to remember to make the copy. Second, you usually make a copy of the whole thing, even if you're changing one small part. And third, even if you're emailing your changes to your colleagues, it may be hard to figure out at the end who did what, and more important, why they did it.

The principle of version control is keep track of the changes in our files.

Diffing Files

diff command - used to take two files or even two directories, and show the differences between them in a few formats.

```
diff old-file.py new-file.py
```

"<" tell us that the first line was removed from the first file.
">" tell us that the second line was added to the second file.
In other words, the old line got replaced by the new one.

"5c5,6" shows a line in the first file "5" that was replaced by two different lines in the second file "5,6". The "c" in between the numbers means that a line was changed.

"11 a 13,15" shows three lines that are new in the second file "13,15". The "a" stands for added.

```
diff -u old-file.py new-file.py
```

the unified format "-u" shows the change lines together with some context.
"-" to mark lines that were removed.
"+" to mark lines that were added.

There are other tools that can be used to compare files
wdiff - highlights the words that have changed in a file instead of working line by line like diff does.
meld, KDiff3, vimdiff - graphical tools that display files side by side and highlights differences by using color.

Applying changes

```
diff -u old-file.py new-file.py > change.diff
```

">" redirects the output of the diff command to a file.
The generated file is usually refered to as a diff file or sometimes a patch file. It includes all the changes between the old file and the new one, plus the additional context needed to understand the changes and to apply those changes back to the original file.

patch command - takes a file generated by diff and applies the changes to the original file.

```
patch original_file.py < generated_file.diff
```

"<" to redirect the contents of the file to standard input.

Practical Application of diff and patch

sys.exit - to make the return number of the exit code of our script, which is the code that causes a program to exit with the corresponding exit value.

Version control systems

What is version control?

Version Control System (VCS) - Keeps track of the changes we made to our files.

Commit - make edits to multiple files and treat that collection of edits as a single change

A VCS even provides a mechanism to allow the author of a commit to record why the change was made, including what bugs, tickets or issues were fixed by the change. This information can be a lifesaver when trying to understand a complex series of changes, or to debug some obscure issue. So, be sure to record this extra info in your commits to be truly commited to better code.

Files are usually organized in repositories which contains separate software projects or just group all related code.

Version Control and Automation

A VCS stores your code and configuration. It also stores the history of that code and configuration. A VCS can function like a time machine, giving you insights into the decisions of the past.

What is Git?

Git - free open source VCS available for installation on different platforms. Unlike some VCS that are centralized around a single server, Git has a distributed architecture.

Git can work as a standalone program, as a server and as a client. This means that you can use Git on a single machine without even having a network connection, or you can use it as a server on a machine where you want to host your repository, and you can use Git as a client to access the repository from another machine or even the same one.

Installing Git

Using Git

First Steps with Git

```
git config --global user.email "me@example.com"   ← set email
git config --global user.name "My name"            ← set name
```

↳ to set this value for all git repositories that we'd use

"git init" create directories from scratch
"git clone" to make a copy of a repository that already exists somewhere else.

Working tree - current version of your project.

Staging area (index) - file maintained by Git that contains all of the information about what files and changes are going to go into your next commit

"git commit" to set commited into the .git directory

Tracking Files

The Git project will consist of three sections:
Git directory - contains the history of all the files and changes
Working tree - contains the current state of the project, including any changes that we've made.
Staging area - contains the changes that have been marked to be included in the next commit.

The files can be:
Tracked - the files are part of the snapshots.
Untracked - the files aren't part of the snapshots yet.

A tracked file can be in three main states:
Modified state - it means that we've made changes to it that we haven't commited yet. The changes could be adding, modifying or deleting the contents of the file.
Staged state - the changes to those files are ready to be commited to the project.
Commited state - the changes made to it are safely stored in a snapshot in the Git directory.

```
git add file.py
```

"git add" - tell to Git we want to add the current changes in that file in the list of changes to be commited

```
git commit -m 'name the changes made'
```

"git commit" - to commit our stage changes

```
git status
```

"git status" - to see the current status of our files