Data Science Technology and Systems (11523)

Semester 2 2024

UNIVERSITY OF CANBERRA

DATA SCIENCE TECHNOLOGY AND SYSTEMS

Lecture 4

Dr. Ibrahim Radwan

DISTINCTIVE BY DESIGN

OUTLINE

- Use Case:
  - Predictive modelling on premises – a complete case study
- Version Control Systems
  - Git

CASE STUDY

Import the data

Explore the data

Build a predictive model

Evaluate the model on the testing data

Interpret the results

DATA

- In this example, we will be working on the following dataset:
  - https://rdrr.io/cran/rattle.data/man/weatherAUS.html

- First, we need to read the description of the data and understand the column names and formats

FULL CODE

- The full code of this use case can be downloaded from:
  - Link

CAN WE USE OTHER TECHNIQUES?

- Random forest trees
- Support vector machines
- Neural networks

- How about you check this possibility using the techniques from SKLearn package?
  - https://scikit-learn.org/stable/supervised_learning.html

RECOMMENDED ACTIVITIES

- Please re-do the steps of the use case in your own time and ensure that you have understood all steps

VERSION CONTROL

What?

Why?

How?

VERSION CONTROL

- Version Control is a software tool that can be used to manage the different versions of files (*e.g.,* source code files)
- These tools help in tracking the changes that are made on files
- With the version, you assign a timestamp with each change being done

VERSION CONTROL SCENARIOS

- For any data scientist, version Control can be used for several scenarios, such as:
  - ❖ When you are developing a piece of software, and there are many changes in your code, at some point, you may need a previous change, or if there are accidental changes to your file and you want to roll back these changes, version control achieves this easily
  - ❖ When you work in a team, and they make different changes in different places in code or on the same source code file after the team finishes the work, you will need to take the changes made by the whole team
  - ❖ When you have a stable version of your software that needs to go for production, you need two copies of your source code: one for the stable version that goes for production that may need fixes to apply on it and another version that your team is working on to develop the rest of requirements

VERSION CONTROL TYPES

UNIVERSITY OF CANBERRA

Centralised

- A centralised version control system keeps a single copy of the repository (source code and change history) on a *central server*
- You may pull or push on the server
- The drawback of this type is the possible failure of the server so that you may lose the tracking history of your work, so it is a single point of failure
- Example: Microsoft Team Foundation Server (TFS)

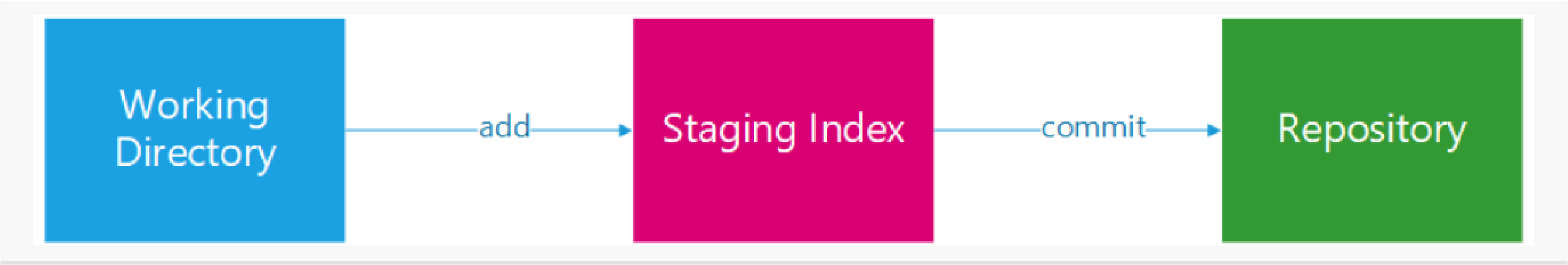
Decentralised

- A distributed version control system keeps a full version of the repository for each client (machine),
- Thus, if you lose connection with other clients, you still have your own repository. You can work on it until you are connected back to pull the changes of other contributors or push your own changes.
- Example: Git

In this lecture, we will focus on understanding the Git VCS

GIT

- Git is an open-source distributed version control system
- It's fast and easy to use.
- You can use it through the command line or through any IDE that integrates with Git, such as Microsoft Visual Studio or Visual Studio code



GIT – HANDS ON

INITIALIZING GIT

UNIVERSITY OF CANBERRA

Description

- To work with Git, you would first need to tell it who you are and which editor you use for commits' comments
- You will then need to create a repository to work in. Creating a repository is simple. Use the `git init` command in a created (new/existing) directory

Script

```
git config --global user.name "your name goes here"

git config --global core.editor "'C:/Program Files/Notepad++/notepad++.exe'"

mkdir python_project

cd python_project
git init
git status
```

What is the output of this step?

ADDING FILES

UNIVERSITY OF CANBERRA

Description

- We will start adding files to the repository.
- Create a file named "*prediction.py*“ in the project directory, and then run the following commands

Script

```
git status


git add .
git status


git restore --staged prediction.py
git status
```

Check the output of each step. Can you add files by their names?

COMMITTING CHANGES

UNIVERSITY OF CANBERRA

Description

- This is the step of shipping the files (or adding changes to the files) to be under the control of the git server
- Create two more files called “*stg.txt*”, and “*working.txt*”

Script

```
git commit
git status
git add stg.txt
git status
```

Check the output of each step and note down your reflection

IGNORING FILE TRACKING / WARNING

Description

- However, if we want to keep the files in the working directory and get the git's commands to ignore it, you can add the “*.gitignore*” file into the working directory.
- Add all files or folders you don't need to commit or get the git system to handle.
- Let’s add the "*working.txt*“ file into it and run the following

Script

```
• git add .
• git status
```

What is the output?

IGNORING FILE TRACKING / WARNING

UNIVERSITY OF CANBERRA

Description

- Also, if you want to ignore a whole folder or files with *specific* extensions, you may use expressions to do that.
- For example, if you have added two folders to your directory, "dataset“ and images, where the "dataset" folder contains "ds1.txt", "ds2.txt", "ds3.json" files and the “images” folder includes "img1.png" and "img2.png", and you need to ignore the whole “images” folder, and only the JSON files for the dataset folder, you may do the following

Script

```
git status
# add the files to the .gitignore file
git status
git add .
git status
git commit –m “ignore the non-necessary files”
git status
```

What is the output?

WORKING WITH CODE

Description

- Now let's add some simple code to our “*prediction.py*” file and check the power of using Git

Script

```
# add this code
import numpy as np
Import pandas as pd
# then in the git shell:
git add .
git commit –m “import the numpy and pandas libs”
```

Check the output of each step and note down your reflection

INSPECTING LOG CHANGES

UNIVERSITY OF CANBERRA

Description

- At any point of time, you can inspect the code changes via checking the log

Script

```
git log
git log --reverse
git log --p
 # or
git log --patch
```

Check the output of each step and note down your reflection

STAGED AND UPSTAGED FILES - DIFF

UNIVERSITY OF CANBERRA

Description

- Suppose you have a file, and you have made changes to it but have not committed yet
- So, it's still in the working directory, and you want to know the difference between the committed one (staged) and the current (un-staged) file
- To achieve this, let's add some code to the prediction.py file and then use the following command

Script

```
git diff prediction.py
```

What have you observed?

RECOMMENDED ACTIVITIES

- Please check the complete notes of the git document that we have shared under week 4 and ensure that you get yourself familiar with the git commands