

# Fundamental tools for data science

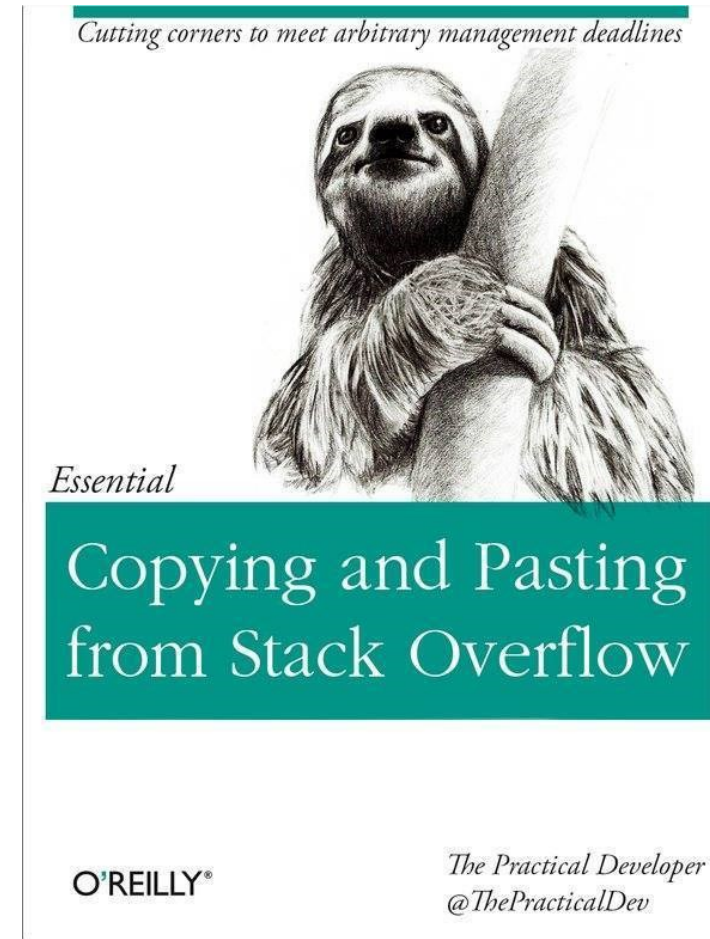
Szu-Chi Chung

Department of Applied Mathematics, National Sun Yat-sen University

# Motivation

---

- ▶ As data scientists, we know that computers are great at aiding in repetitive tasks
  - ▶ We have a vast range of tools available at our fingertips that enable us to be more productive and solve more complex problems when working on any computer-related problem
  - ▶ Yet many of us utilize only a tiny fraction of those tools; In this mini-course, I will try my best to help you become familiar with what kind of tools may be useful in your research



# Course Outline

---

- ▶ Specifically, what we will cover includes the following topics
  - ▶ Bash commands to help you be comfortable with the command line
    - ▶ <https://github.com/RehanSaeed/Bash-Cheat-Sheet>
    - ▶ [https://oit.ua.edu/wp-content/uploads/2020/12/Linux\\_bash\\_cheat\\_sheet-1.pdf](https://oit.ua.edu/wp-content/uploads/2020/12/Linux_bash_cheat_sheet-1.pdf)
  - ▶ Git for version control to help you track the data and code
    - ▶ <https://github.com/arslanbilal/git-cheat-sheet>
    - ▶ <https://training.github.com/downloads/github-git-cheat-sheet/>
    - ▶ [Search tips](#)
  - ▶ Colab to help you exploit the GPU power and interactive experiment
  - ▶ Kaggle to help you explore the datasets and learn useful concepts in the world's largest data science community

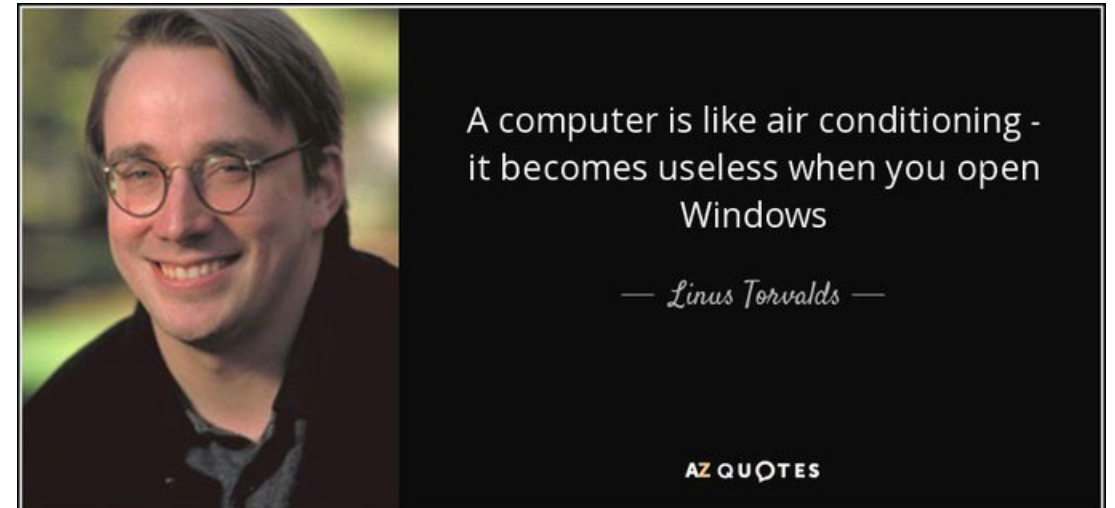


Shell

# What is the shell

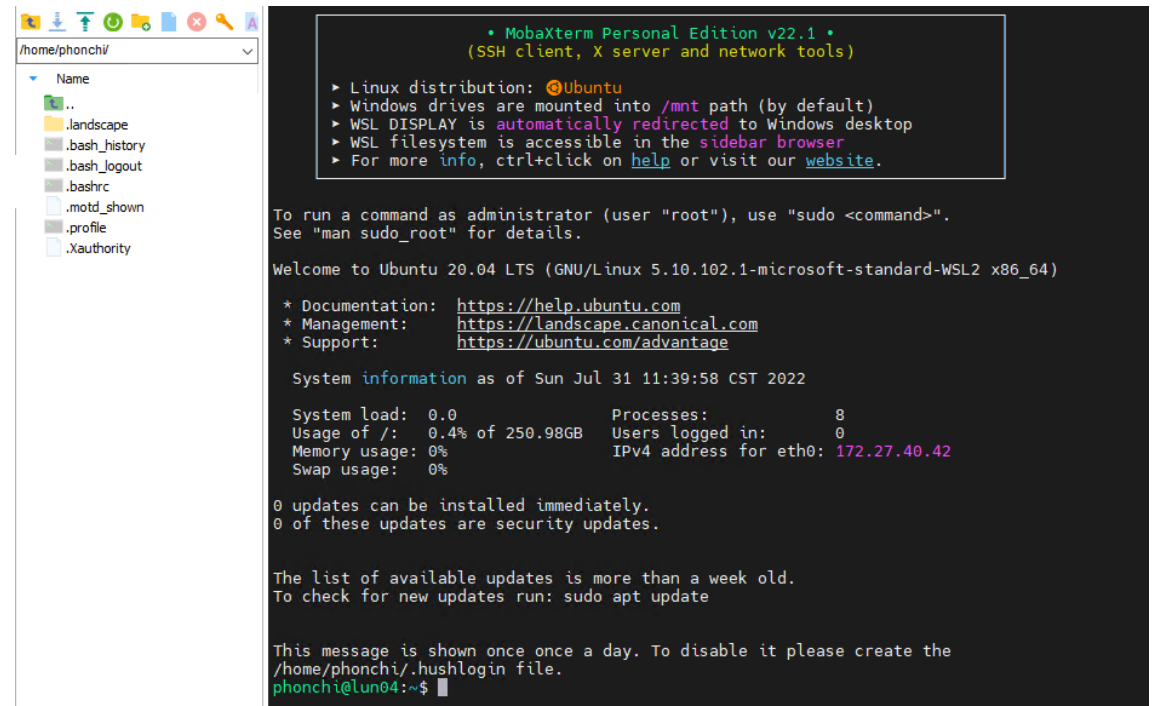
---

- ▶ Computers these days have a variety of interfaces for giving them commands
  - ▶ Fanciful graphical user interfaces. Voice interfaces, and even AR/VR are everywhere
  - ▶ These are great for 80% of use-cases, but they are often fundamentally restricted in what they allow you to do — you cannot press a button that isn't there or give a voice command that hasn't been programmed. To take full advantage of the tools your computer provides, we have to go old-school and drop down to a textual interface: The Shell
- ▶ In this lecture, we will focus on the Bourne Again SHell, or “bash”
  - ▶ This is one of the most widely used shells. To open a shell prompt (where you can type commands), you first need a terminal. Your device probably shipped with one installed, or you can install it



# Using the shell

- ▶ You will see a *prompt*. It tells you that you are on the machine `lun04` and that your “current working directory”, or where you currently are, is `~` (short for “home”). The `$` tells you that you are not the root user (more on that later)
  - ▶ At this prompt, you can type a command, which will then be interpreted by the shell
  - ▶ Tab can be used for auto-completing
  - ▶ The program will be searched under the `$PATH` variable
  - ▶ `env` will list all environment variables
  - ▶ `export` can be used to set environment variables
  - ▶ `explorer.exe .` (open `.` In mac) to open the home directory



```
MobaXterm Personal Edition v22.1
(SSH client, X server and network tools)

▶ Linux distribution: Ubuntu
▶ Windows drives are mounted into /mnt path (by default)
▶ WSL DISPLAY is automatically redirected to Windows desktop
▶ WSL filesystem is accessible in the sidebar browser
▶ For more info, ctrl+click on help or visit our website.

To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

Welcome to Ubuntu 20.04 LTS (GNU/Linux 5.10.102.1-microsoft-standard-WSL2 x86_64)

* Documentation:  https://help.ubuntu.com
* Management:    https://landscape.canonical.com
* Support:        https://ubuntu.com/advantage

System information as of Sun Jul 31 11:39:58 CST 2022

System load:  0.0          Processes:      8
Usage of /:   0.4% of 250.98GB   Users logged in: 0
Memory usage: 0%             IPv4 address for eth0: 172.27.40.42
Swap usage:   0%

0 updates can be installed immediately.
0 of these updates are security updates.

The list of available updates is more than a week old.
To check for new updates run: sudo apt update

This message is shown once once a day. To disable it please create the
/home/phunchi/.hushlogin file.
phunchi@lun04:~$
```

# 1. Navigating Directories

---

- ▶ A path on the shell is a delimited list of directories, separated by / on Linux and macOS and \ on Windows. On Linux and macOS, the path / is the “root” of the file system, under which all directories and files lie, whereas on Windows there is one root for each disk partition (e.g., C:\)
  - ▶ Absolute path starts with /
  - ▶ Relative path starts with .. or .
- ▶ To see what lives in a given directory, we use the *ls* command
- ▶ Usually, running a program with the *-h* or *--help* flag will print some help text that tells you what flags and options are available
  - ▶ *man* and *tldr* are also useful
  - ▶ If this is the first time you initialize the system try *sudo apt update* and *sudo apt install tldr*

# Navigating Directories

---

- ▶ *ls -la* will give you more information about each file or directory present
  - ▶ First, the *d* at the beginning of the line tells us that it is a directory
  - ▶ Then follow three groups of three characters (rwx). These indicate what permissions the owner of the file, the owning group (phonchi), and everyone else respectively has on the relevant item. A - indicates that the given principal does not have the given permission
  - ▶ You can check the groups using *groups*
  - ▶ You can change the permission using *chmod*

```
phonchi@lun04:~$ ls -la
total 32
drwxr-xr-x 3 phonchi phonchi 4096 Jul 31 11:39 .
drwxr-xr-x 3 root    root    4096 Jul 30 17:36 ..
-rw----- 1 phonchi phonchi   51 Jul 31 11:39 .Xauthority
-rw----- 1 phonchi phonchi    3 Jul 30 19:13 .bash_history
-rw-r--r-- 1 phonchi phonchi  220 Jul 30 17:36 .bash_logout
-rw-r--r-- 1 phonchi phonchi 3771 Jul 30 17:36 .bashrc
drwxr-xr-x 2 phonchi phonchi 4096 Jul 30 17:37 .landscape
-rw-r--r-- 1 phonchi phonchi    0 Jul 31 11:39 .motd_shown
-rw-r--r-- 1 phonchi phonchi  807 Jul 30 17:36 .profile
```



## 2. File operations

---

- ▶ *mkdir* can be used to make new directories while *mkdir -p* can be used to create nested directories
- ▶ *cp* can be used to copy files while *mv* can be used for moving files or renaming files
  - ▶ *cp -r* can be used to copy directories
- ▶ *rm* can be used to remove files and *rm -rf* can be used to remove directories recursively
- ▶ *touch* can be used to create files, while *cat/head/tail* can be used to print the content of a file
- ▶ *ln -s* can be used to create a symbolic link

### 3. Connecting programs

---

- ▶ In the shell, programs have two primary “streams” associated with them: their input stream and their output stream
  - ▶ Normally, a program’s input and output are both your terminal. That is, your keyboard as input and your screen as output. However, we can also rewire those streams!
  - ▶ The simplest form of redirection is `< file` and `> file` (Overwrite)
- ▶ You can also use `>>` to append to a file. Where this kind of input/output redirection really shines is in the use of pipes. The `|` operator lets you “chain” programs such that the output of one is the input of another
- ▶ The `xargs` command will execute a command using STDIN as arguments. For example, `ls | xargs rm` will delete the files in the current directory.

## 4. Finding files/code

---

- ▶ All UNIX-like systems come packaged with *find*, a great shell tool to find files. *find* will recursively search for files matching some criteria
  - ▶ *locate* is another useful tool, but you need to install it with `apt` on Ubuntu
- ▶ Finding files by name is useful, but quite often, you want to search based on file content
  - ▶ A common scenario is wanting to search for all files that contain some pattern, along with where in those files said pattern occurs. To achieve this, most UNIX-like systems provide *grep*

## Finding shell commands

---

- ▶ You may want to find specific commands you typed at some point. Typing the up arrow will give you back your last command, and if you keep pressing it, you will slowly go through your shell history
  - ▶ The `history` command will let you access your shell history
  - ▶ In most shells, you can make use of `Ctrl+R` to perform the backward search
- ▶ `Ctrl+A` can move the cursor to the front while `Ctrl+E` can move the cursor to the end

## 5. Job Control

---

- ▶ *top/htop* will list all process
- ▶ When typing *Ctrl – C* this prompts the shell to deliver a SIGINT signal to the process and many programs can be stopped
- ▶ & suffix in command will run the command in the background
- ▶ *jobs* will list all background jobs
- ▶ *ps – ef* will list all processes
- ▶ A more generic signal for asking a process to exit gracefully is to use the *kill* command, with the syntax *kill – 9 < PID >*
- ▶ *tmux* can be used as multiplexers

## Resources check

---

- ▶ `df -h` can be used to check the disk usage
- ▶ `du -sh *` can be used to check the disk usage of the current directories

## 6. Dotfiles

---

- ▶ Many programs are configured using plain-text files known as dotfiles (because the file names begin with a .)
  - ▶ Shells are one example of programs configured with such files. On startup, your shell will read many files to load its configuration
  - ▶ For bash, editing your `~/.bashrc`
  - ▶ For git, editing your `~/.gitconfig`
- ▶ Use *source* to activate the dotfiles

## Try the GUI Programs

---

- ▶ xclock
- ▶ xeyes
- ▶ xcalc
- ▶ gedit





Git

# Version Control (Git)

---

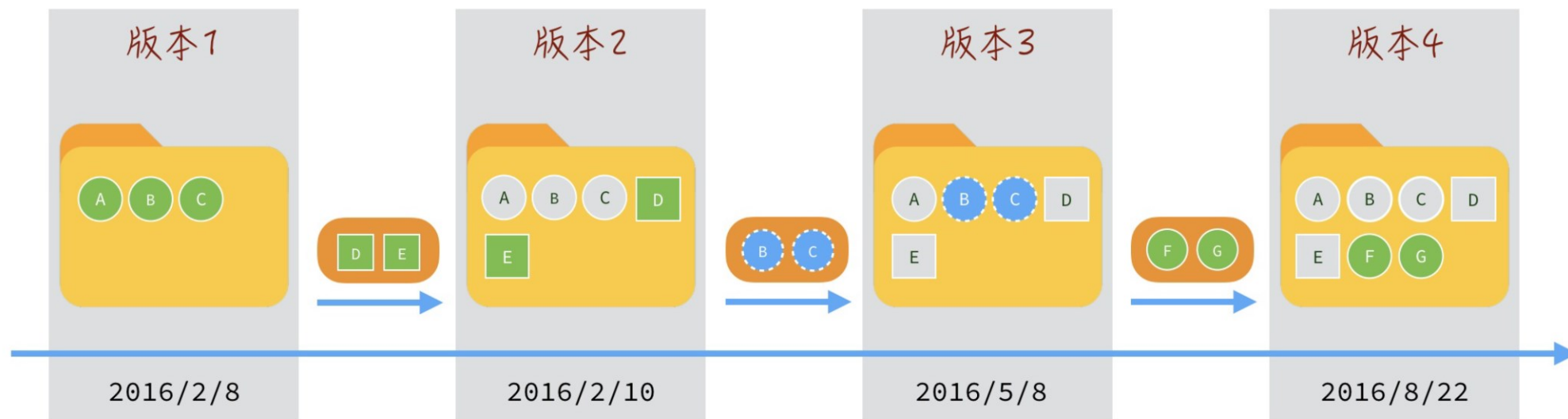
- ▶ Version control systems (VCSs) are tools used to track changes to source code (or other collections of files and folders). As the name implies, these tools help maintain a history of changes; furthermore, they facilitate collaboration
  - ▶ Even when you're working by yourself, it can let you look at old snapshots of a project, keep a log of why certain changes were made, work on parallel branches of development, and much more
  - ▶ Who wrote this module?
  - ▶ When was this particular line of this particular file edited? By whom? Why was it edited?



```
▼ 📁 resumes
  > 📁 resume-2016-02-08
  > 📁 resume-2016-02-10
  > 📁 resume-2016-05-08
  > 📁 resume-2016-08-22
  ▼ 📁 resume-2016-11-28
    📄 eddie.md
    📄 john.md
    📄 kao.md
    📄 mary.md
    📄 sherly.md
    📄 tracy.md
  > 📁 resume-bak
  > 📁 resume-for-5xruby
```

# Git

---



<https://gitbook.tw/>

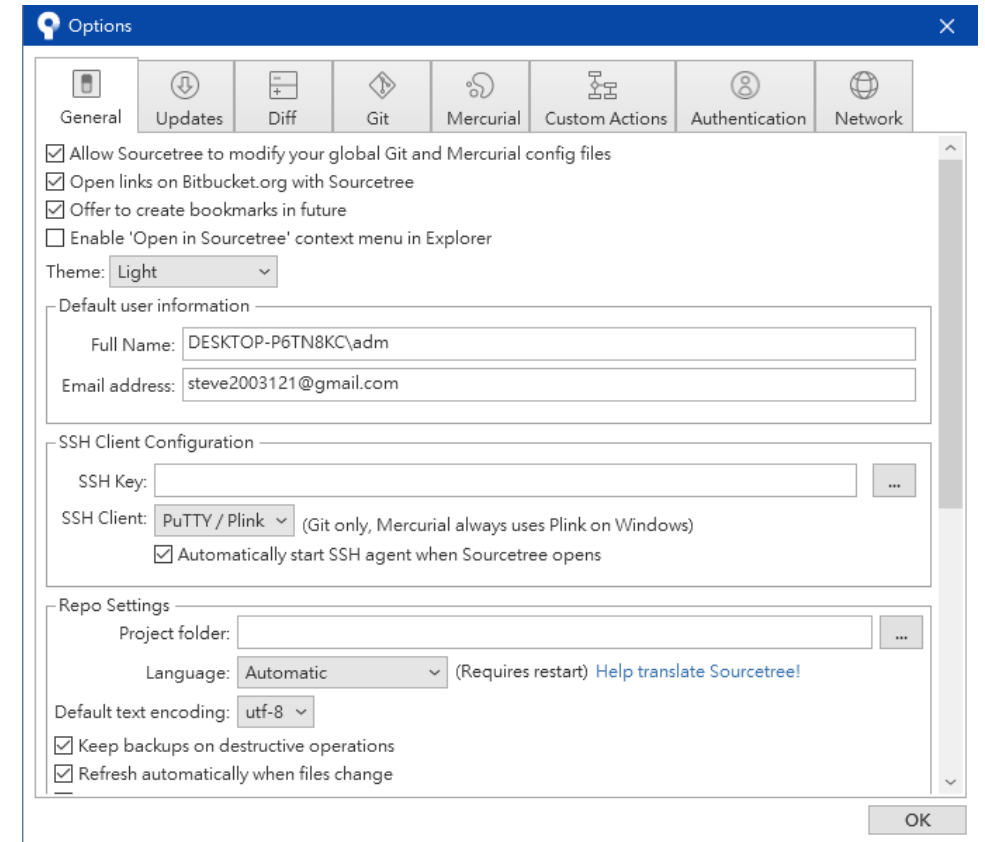
# Start using Git

## ► Tools->option

► `$ git config --global user.name "chung"`

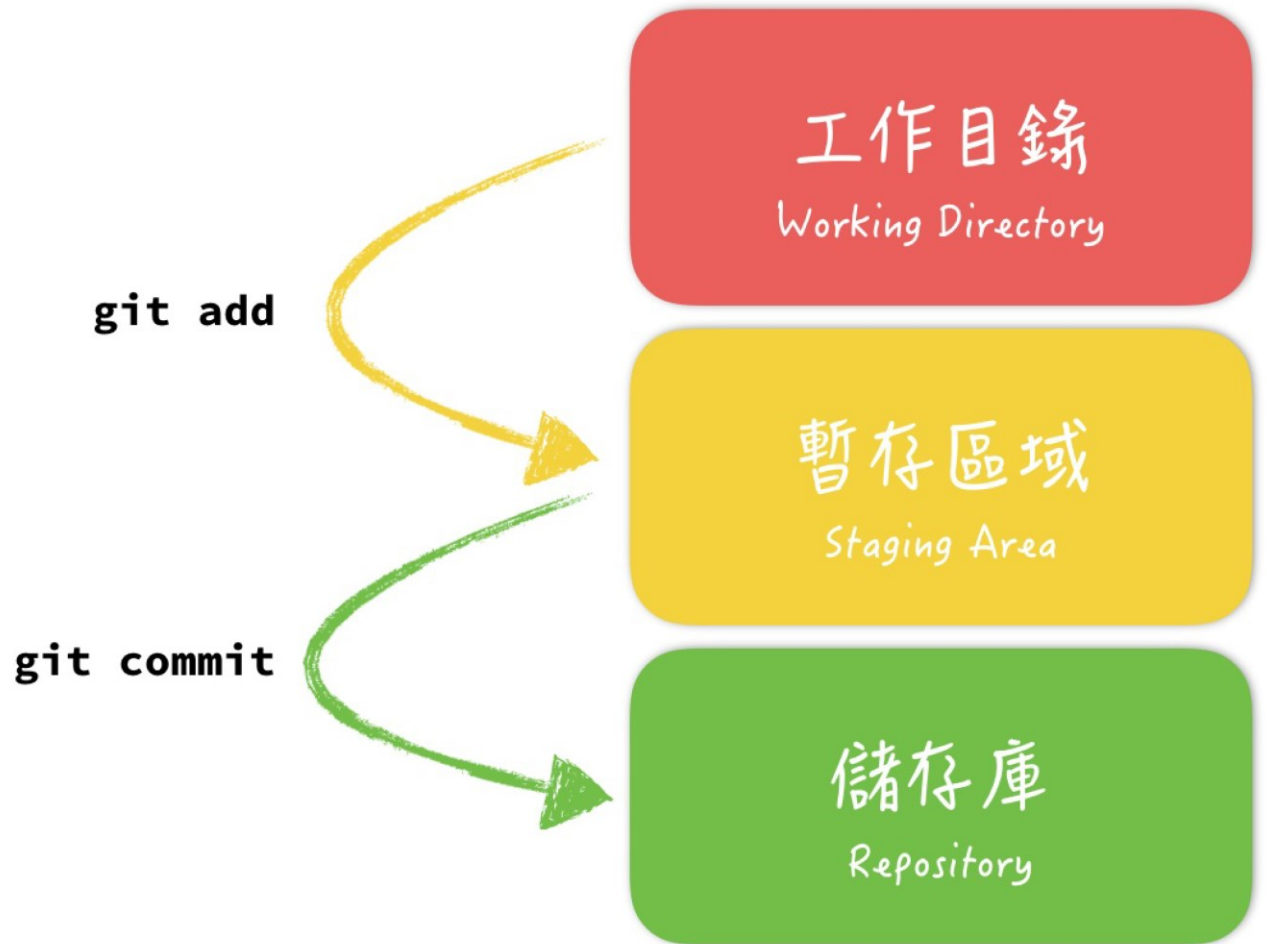
► `$ git config --global user.email "steve2003121@gmail.com"`

► `$ git init`: creates a new git repo, with data stored in the `.git` directory



# 1. Start tracking

- ▶ `$ git add .`
- ▶ `$ git commit -m "message"`
- ▶ `$ git status`
  - ▶ 1. 完成一個「任務」的時候
  - ▶ 2. 下班的時候：雖然可能還沒完全搞定任務，但至少先 Commit 今天的進度，除了備份之外，也讓公司知道你今天有在努力工作。（然後帶回家繼續苦命的做？）
  - ▶ 3. 你想要 Commit 的時候就可以 Commit。
- ▶ `$ git log --oneline -g`



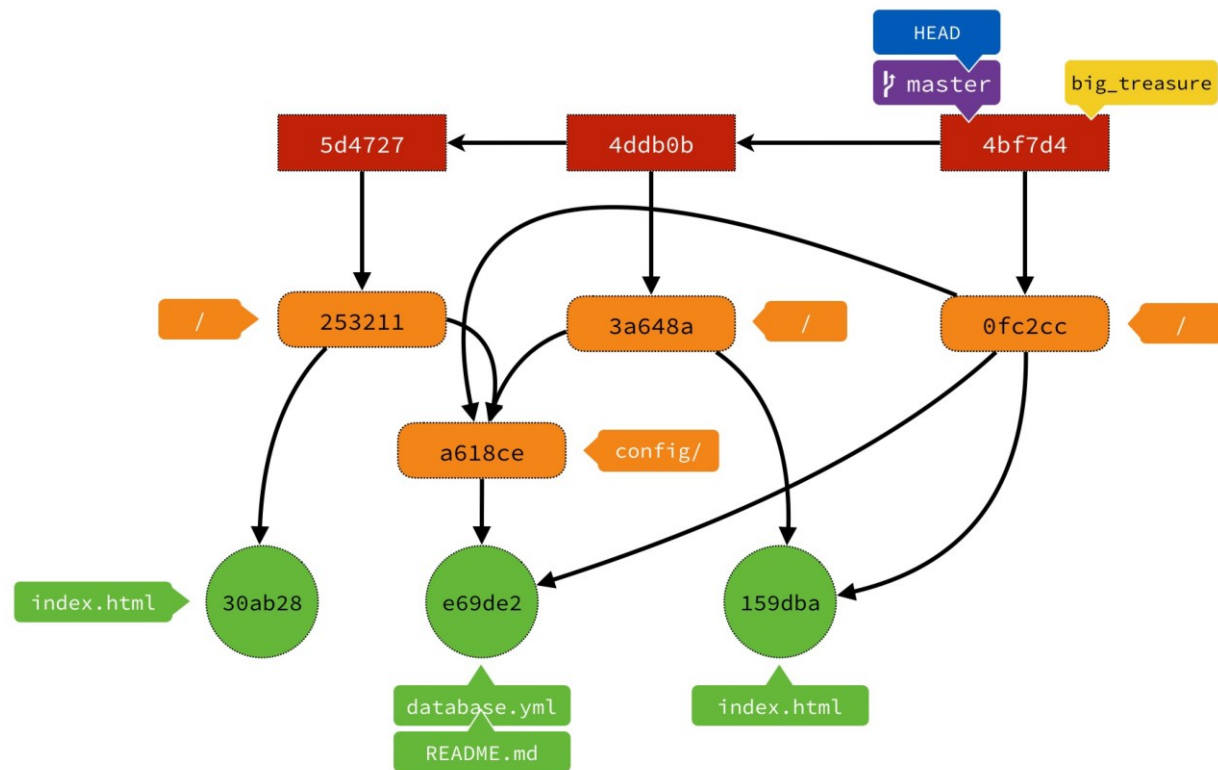
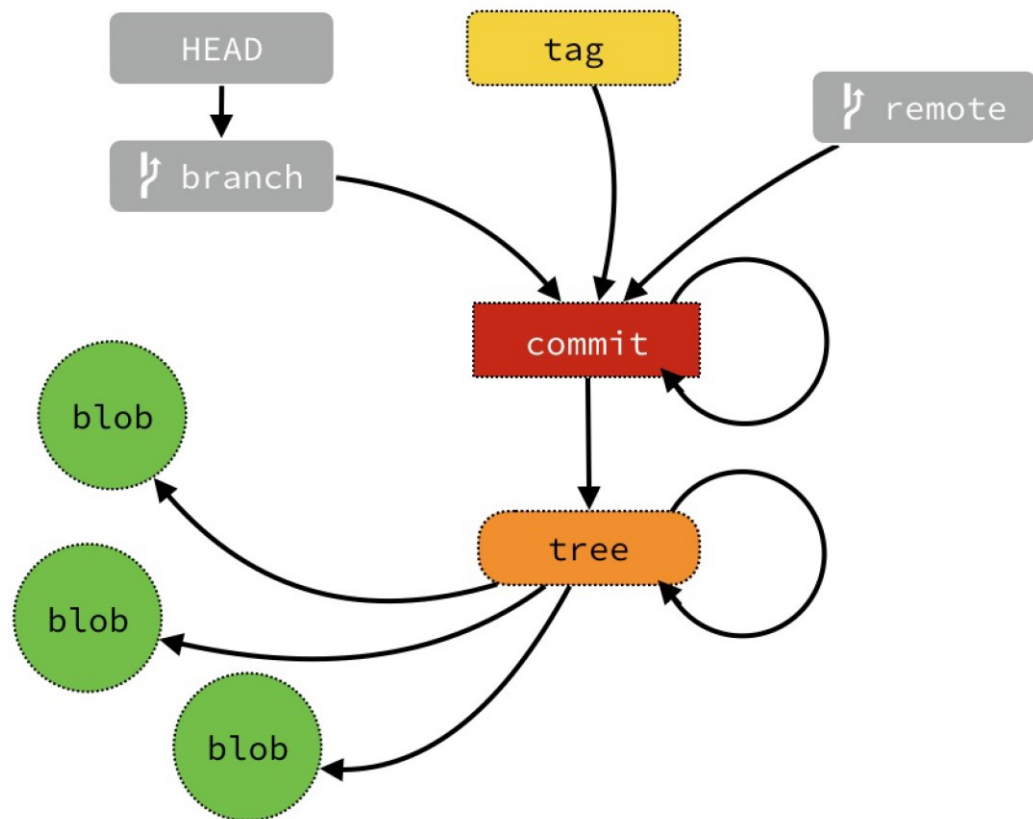
## 2. Undo and stop tracking

- ▶ .gitignore: specify intentionally untracked files to ignore
- ▶ `$ git rm <file> --cache`: Stop tracking certain file
- ▶ `$ git checkout .`: discard all changes
- ▶ `$ git checkout <revision>`: updates HEAD and current branch
- ▶ `$ git reset --hard <commit>`

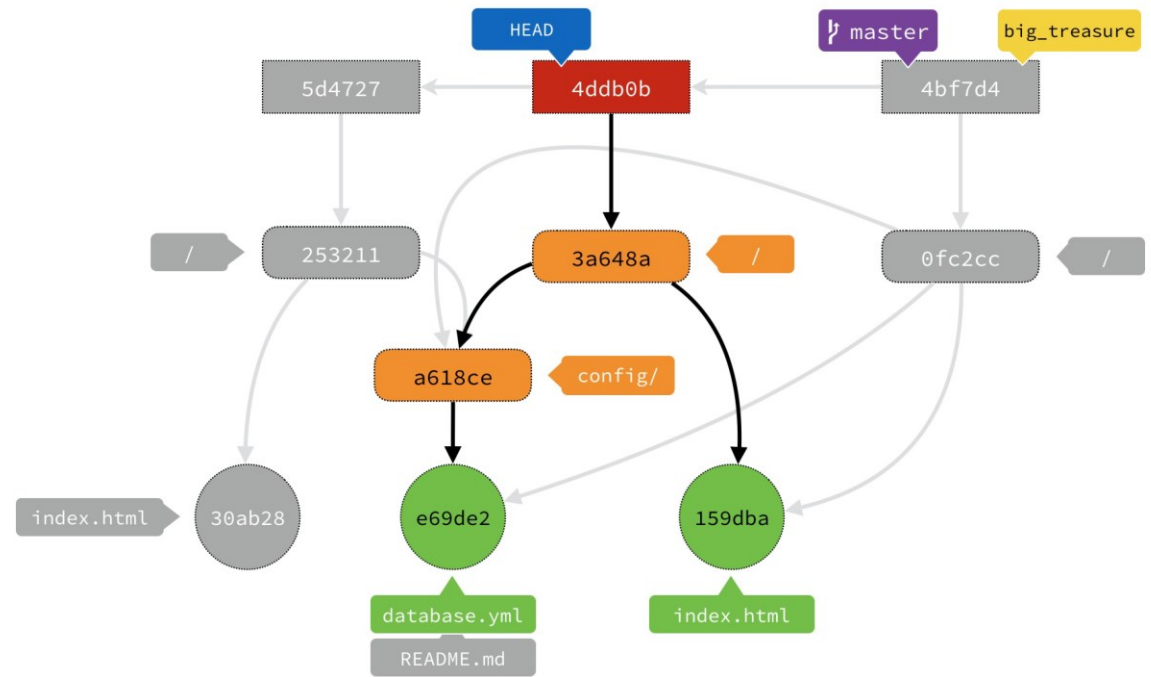
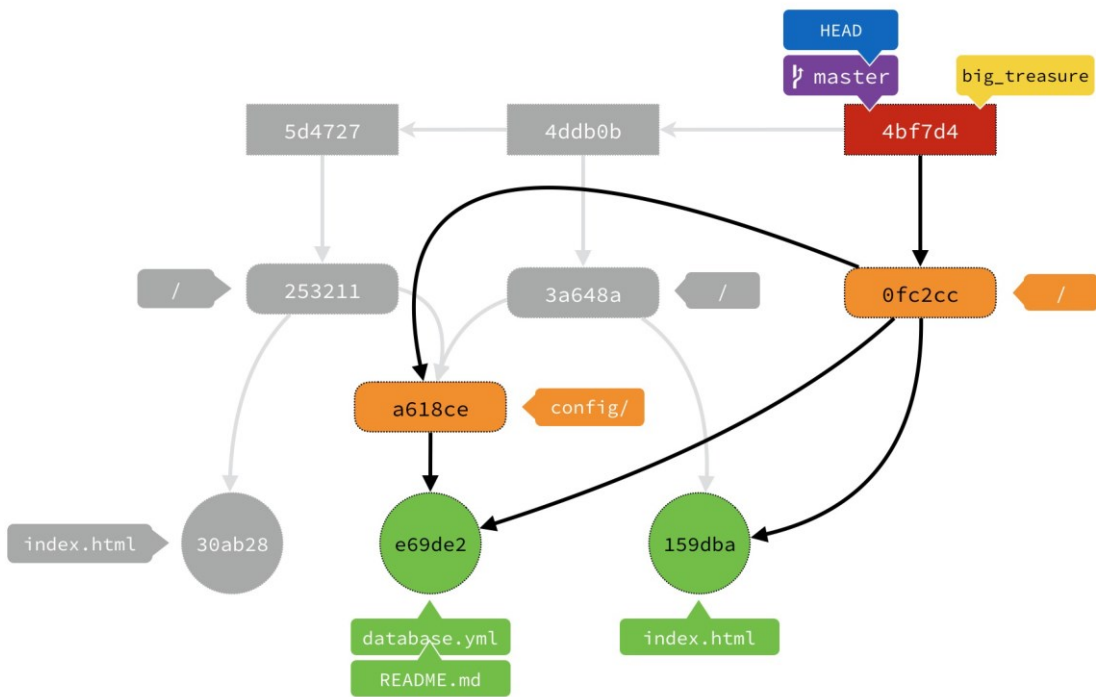
| 模式   | mixed 模式 | soft 模式 | hard 模式 |
|------|----------|---------|---------|
| 工作目錄 | 不變       | 不變      | 丟掉      |
| 暫存區  | 丟掉       | 不變      | 丟掉      |

<https://gitbook.tw/>

# Underlying



# Underlying





## Add a branch

---

- ▶ `$ git checkout -b <name>`: creates a branch and switches to it
  - ▶ same as `git branch <name>; git checkout <name>`
- ▶ `$ git merge <revision>`: merges into the current branch
- ▶ `$ git rebase`: rebase set of patches onto a new base

# Add a branch

---

## ► Conflict

### ► Modified the conflict files

- \$ git checkout --ours cute\_animal.jpg
- \$ git checkout --theirs cute\_animal.jpg

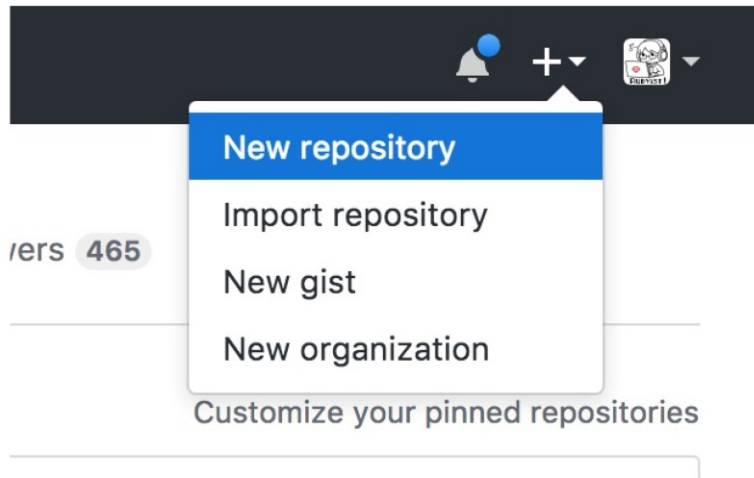
```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>首頁</title>
  </head>
  <body>
    <div class="container">
<<<<<< HEAD
      <div>我是 Cat</div>
=====
      <div>我是 Dog</div>
>>>>>> dog
    </div>
  </body>
</html>
```

<https://gitbook.tw/>

# Working with remote repositories

## ► GitHub

- Largest open source repositories control using git server



### Create a new repository

A repository contains all the files for your project, including the revision history.

Owner:  / Repository name:

Great repository names are short and memorable. Need inspiration? How about [supreme-broccoli](#).

Description (optional):

☒ **Public**  
Anyone can see this repository. You choose who can commit.

☐ **Private**  
You choose who can see and commit to this repository.

☐ **Initialize this repository with a README**  
This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore:  Add a license:  ⓘ

## Working with remote repositories

---

- ▶ `$ git remote add <name> <url>`: add a remote
- ▶ `$ git push -u <remote> <local branch>:<remote branch>`: send objects to remote, and update remote reference
- ▶ `$ git branch --set-upstream-to=<remote>/<remote branch>`: set up a correspondence between local and remote branch
- ▶ `$ git fetch`: retrieve objects/references from a remote
- ▶ `$ git pull`: same as git fetch; git merge
  
- ▶ `$ git clone`: download repository from remote (Only once)
- ▶ Submit your pull request and fork

# How to find an interesting fork?

---

- ▶ <https://stackoverflow.com/questions/54868988/how-to-determine-which-forks-on-github-are-ahead>

## Search tips

---

- ▶ <https://www.google.com/search?q=resume+site:cs.cmu.edu+filetype:pdf>
- ▶ <https://github.com/search?o=desc&q=dotfiles&s=stars&type=Repositories>
- ▶ <https://github.com/search?q=GAN>
- ▶ <https://github.com/search?q=statement+of+purpose>
- ▶ <https://github.com/search/>
- ▶ <https://paperswithcode.com/>



Colab

# Colab

---

- ▶ Jupyter Notebooks have rapidly grown in popularity among data scientists to become the standard for quick prototyping and exploratory analysis
  - ▶ For example, [Netflix](#) based all of their machine learning workflows on them, effectively building a whole notebook infrastructure to leverage them as a unifying layer for scheduling workflows



## Use different environment

---

- ▶ Remember DO NOT store input data in your drive and load from there. The input/output is very slow (store at ./ instead). Your output data should be stored in your google drive so that it can be accessed next time.
- ▶ Use a different environment
  - ▶ <https://stackoverflow.com/questions/60775160/install-python-3-8-kernel-in-google-colaboratory/71511943#71511943>

```
"collapsed_sections": [], "kernel_spec": {"name": "py39", "display_name": "py39", "language_info":
```

- ▶ Use [condacolab](#)
  - ▶ Only works for Python3.8

# Use different language

---

- ▶ Temporary notebook
  - ▶ <https://colab.to/>
- ▶ You can use R or Julia programming language in Google Colab
  - ▶ By going to <https://colab.to/r> or <https://colab.to/julia>. It will open a new notebook with R set as the kernel instead of Python



Kaggle

# Kaggle

---

- ▶ Kaggle is an online community of data scientists and machine learners owned by Google, Inc
  - ▶ Kaggle allows users to find and publish data sets, explore and build models in a web-based data-science environment, work with other data scientists and machine learning engineers, and enter competitions to solve data science challenges
  - ▶ Kaggle got its start by offering machine learning competitions and now also offers a public data platform, a cloud-based workbench for data science
- ▶ You can search the competitions, datasets and notebooks using the site search on the top bar of the website
  - ▶ But you can get fine-grained control using the search in each panel from the sidebar on the left-hand

# Kaggle

---

## ▶ The flexibility of selecting a script or notebook

### ▶ Scripts

- ▶ The first type is a script. Scripts are files that execute everything as code sequentially. To start a script, click on “Create Notebook” and select “Script”. This will open the Scripts editing interface
- ▶ From here you may select what type of script you would like to execute. You may write scripts in R or in Python.

### ▶ RMarkdown Scripts

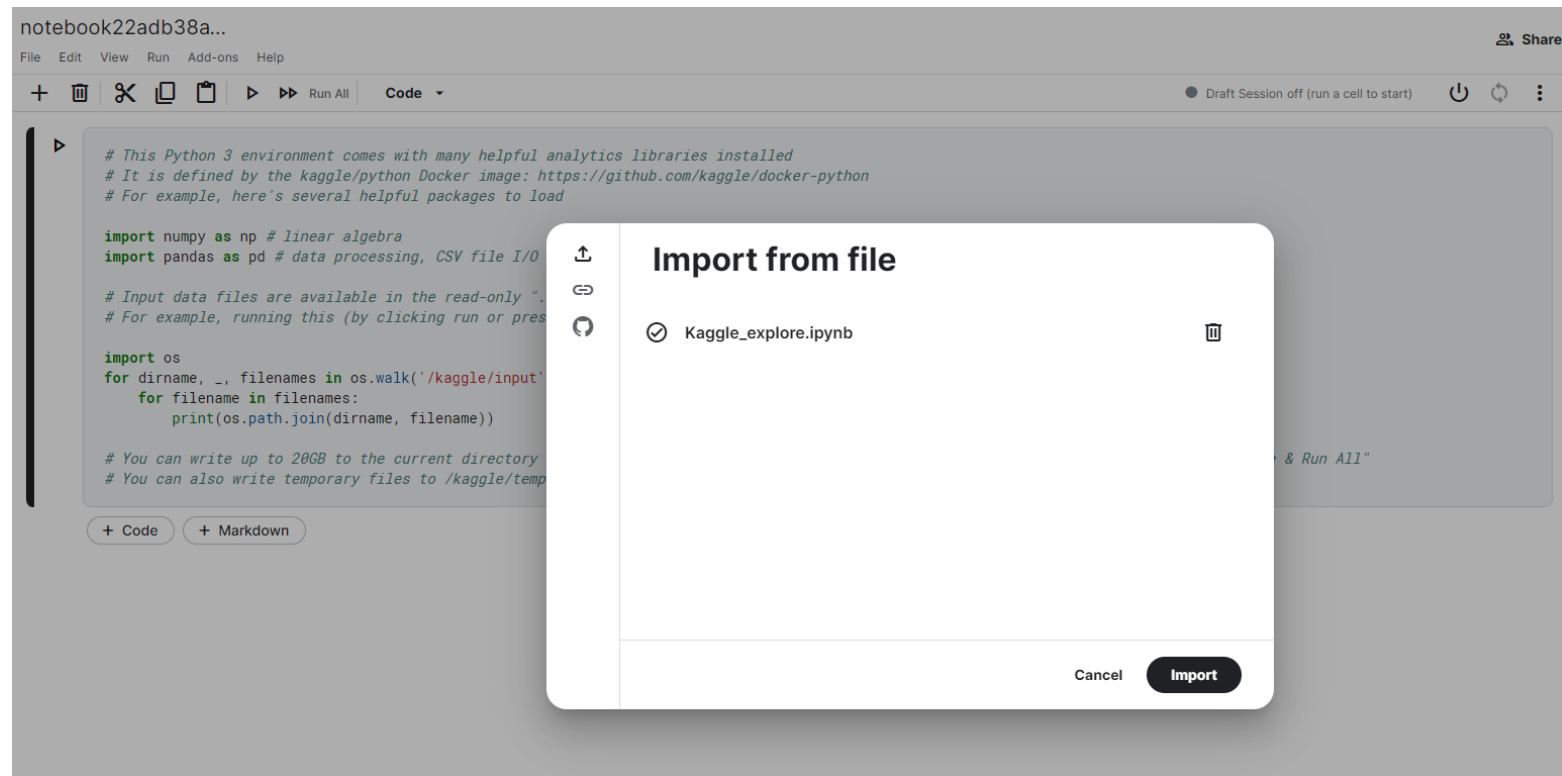
- ▶ RMarkdown scripts are a special type of script that executes not just R code, but RMarkdown code. To start editing an RMarkdown script, click on “Create Notebook”, navigate to the “Scripts” pane, and click on that. Then, in the language dropdown, click on “RMarkdown”.

## ▶ Notebooks

- ▶ The last type is Jupyter notebooks (usually just “notebooks”). To start a notebook, click on “Create Notebook”, and select “Notebook”. This will open the Notebooks editing interface. Notebooks may be written in either R or Python.

# Kaggle kernel (notebook)

## ► File-> Import Notebook



# The GPU accelerator

- ▶ The Resources are listed below
  - ▶ Kaggle GPU: 16G NVIDIA TESLA P100
    - ▶ <https://www.kaggle.com/docs/efficient-gpu-usage>
  - ▶ Limited to 30+ hrs/week depending on usage.
  - ▶ Limited to 12hrs/run

The screenshot shows a Kaggle notebook titled 'notebooke587b6bbad' with a 'Draft saved' status. The interface includes a top bar with 'File', 'Edit', 'View', 'Run', 'Add-ons', and 'Help' menus. Below this is a toolbar with icons for adding, deleting, and running code cells, along with a 'Run All' button. The main area contains a code cell with the following Python code:

```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserved as output when you create a version using "Save & Run All"
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current session
```

At the bottom of the code cell, the file paths are listed: `/kaggle/input/data-science-capstone-project-spring-2022/sample_submission.csv` and `/kaggle/input/data-science-capstone-project-spring-2022/mappings.csv`.

On the right side of the notebook, there is a sidebar with various controls:

- Data**: Includes 'Add data' and a list of input files, such as 'data-science-capstone-project-...'.
- Output**: Shows the current working directory as '/kaggle/working'.
- Competitions**: A dropdown menu.
- Settings**: A section with various configuration options:
  - Language**: Set to 'Python'.
  - Environment**: 'Loading environment...'.
  - Accelerator**: Set to 'GPU' (highlighted with a red box).
  - GPU Quota**: '09:20 / 43 hrs'.
  - Internet**: A toggle switch that is currently turned on.

# Save the notebook

## ► Quick Save

- Skips the top-to-bottom notebook execution and just takes a snapshot of your notebook exactly as it's displayed in the editor. This is a great option for taking a bunch of versions while you're still actively experimenting. You can choose to reserve the output

**Save Version**

Version Name (optional):

Version 1

9 / 50

✓ Quick Save  
Save a version of your notebook the way it currently looks

Advanced Settings Cancel Save

**← Version Settings**

Save outputs when creating a Quick Save

☐ Never save output

☒ Always save output

☐ Save output for this version

Save & Run All with an accelerator

ACCELERATOR  
Run with GPU for all sessions

Cancel Save

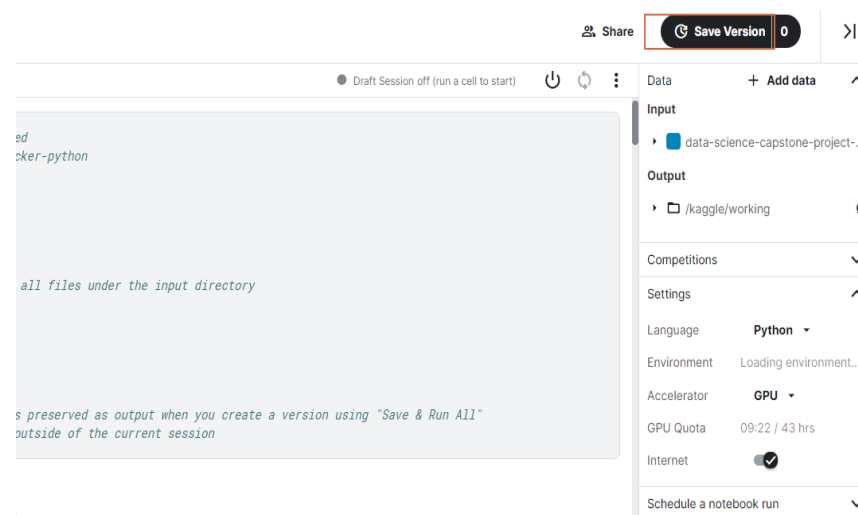
## ► Save & Run All

- Creates a new session with a completely clean state and runs your notebook from top to bottom. In order to save successfully, the entire notebook must execute within 12 hours (9 hours for TPU notebooks). Save & Run All is identical to the “Commit” behavior.



# Running the code in the background

- ▶ You can also run the code in the background with Kaggle. Firstly, make sure your code is bug-free, as any error in any code block would result in early stopping. Click the “Save Version” button as follows:
  - ▶ *The concept of “Versions” is a collection consisting of a Notebook version, the output it generates, and the associated metadata about the environment.*



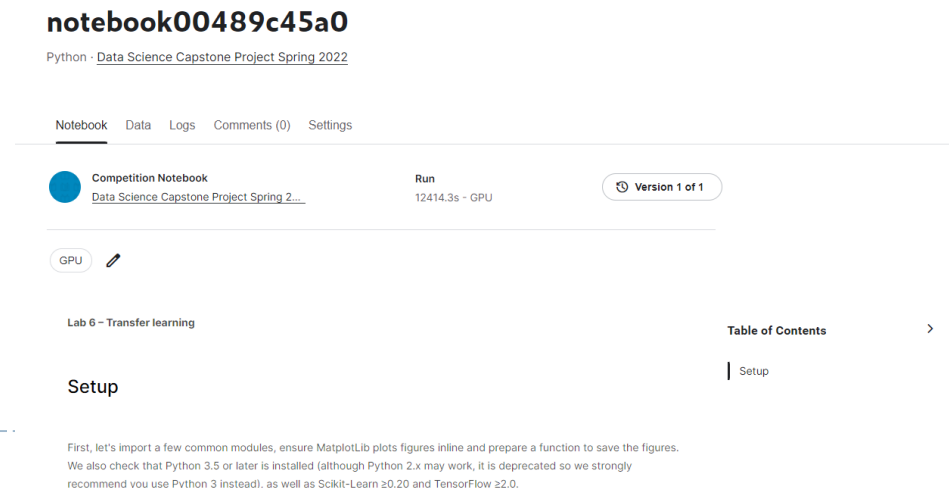
- ▶ Notice that the output is limited to 20G and the max run time is limited to 12hrs

# Running the code in the background

- ▶ You can then see the results by clicking the version number:



- ▶ The log will be shown in the Notebook panel



# Running the code in the background

- ▶ Your output can be accessed through the “Data” panel, where you can download your data. You can also save your model as a new dataset and import your new dataset into the notebook via “Add data” so that you can modify your code to load your checkpoint:

**notebook00489c45a0**  
Python · [Data Science Capstone Project Spring 2022](#)


Notebook **Data** Logs Comments (0) Settings

**Data**

ev2\_fine\_tuning\_0319.keras (56.21 MB)

Submit ⬇ >

Unable to show preview



Previews for binary data are not supported

Input (12.87 MB)

- Data Sources
  - Data Science Capstone ...

Output

- ev2\_fine\_tuning\_0319.keras
- my\_keras\_model\_0319.keras

⬇ Download All

+ New Dataset

+ New Notebook



# Appendix

# Resource

---

- ▶ Grammar check

- ▶ <https://www.grammarly.com/>

- ▶ <https://www.wordtune.com/>

- ▶ <https://www.facebook.com/yenhuan.li/posts/10220880099577177>

# Reference

---

- ▶ <https://missing.csail.mit.edu/>
- ▶ GUI
  - ▶ <https://docs.microsoft.com/en-us/windows/wsl/setup/environment>
    - ▶ <https://www.thewindowsclub.com/error-0x80370102-the-virtual-machine-could-not-be-started>
    - ▶ HyperV
- ▶ GPU
  - ▶ <https://docs.nvidia.com/cuda/wsl-user-guide/index.html#getting-started-with-cuda-on-wsl>
  - ▶ 需安裝21H2
    - ▶ <https://stackoverflow.com/questions/70011494/why-does-nvidia-smi-return-gpu-access-blocked-by-the-operating-system-in-wsl2>