# Command Line Fundamentals

Presented by the **SCIS ACE Programme.**

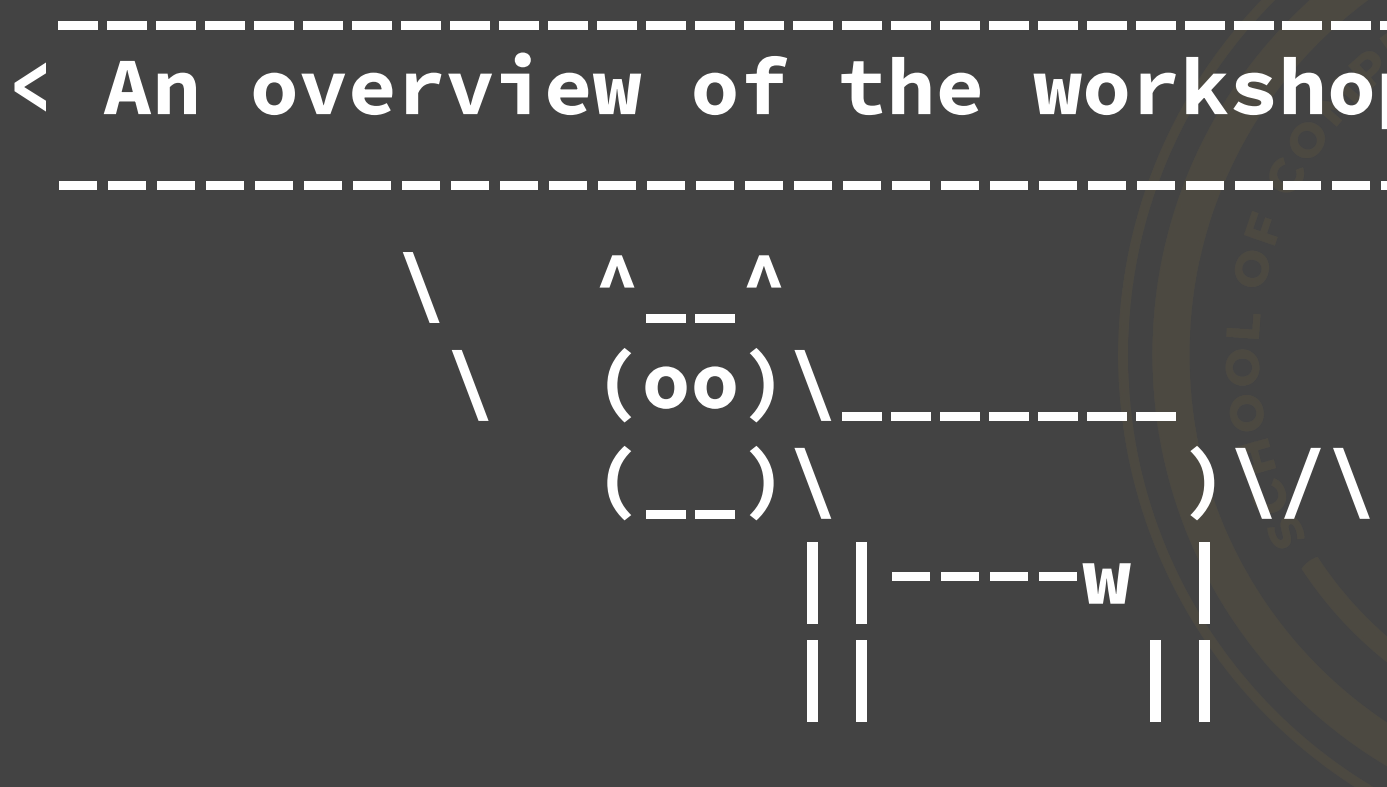

**Instructors**
Tan Jing Zhi
Ex-President, SMU Whitehat Society (Year 3, CS)

Matthew Ho
Vice President, SCIS ACE Programme (Year 3, CS)


# For the optimal learning experience…

1. Please **have your nametent out** so that we can address you.

2. Do not hesitate to **stop us** and **ask questions** at **any time** during the workshop.


An overview of the workshop

# Topics

1. An overview of the shell
2. Connecting to a remote server over SSH
3. Navigating and Managing Files
4. Installing Packages
5. Environment Variables
6. Permissions and Privileges
7. Manipulating Text Files
8. Connecting to the internet
9. System information
10. Basic user management

# Why use a CLI?

Preferred

- Docker, Network troubleshooting, scripts, Package Managers

Rather use a UI

- Browsing web and folders, visual media, moving files (Drag and drop)

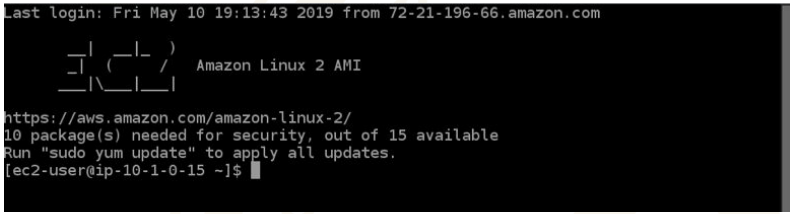
# Isn't clicking easier?

**ABSOLUTELY, but…**

The environment may not have a UI

The UI might change for the same tasks

Some tools may not have buttons





GitLab


# Why Click?

Knowing some commands exist is useful for automation

- Healthchecks
- Setting up environments
- Building programs

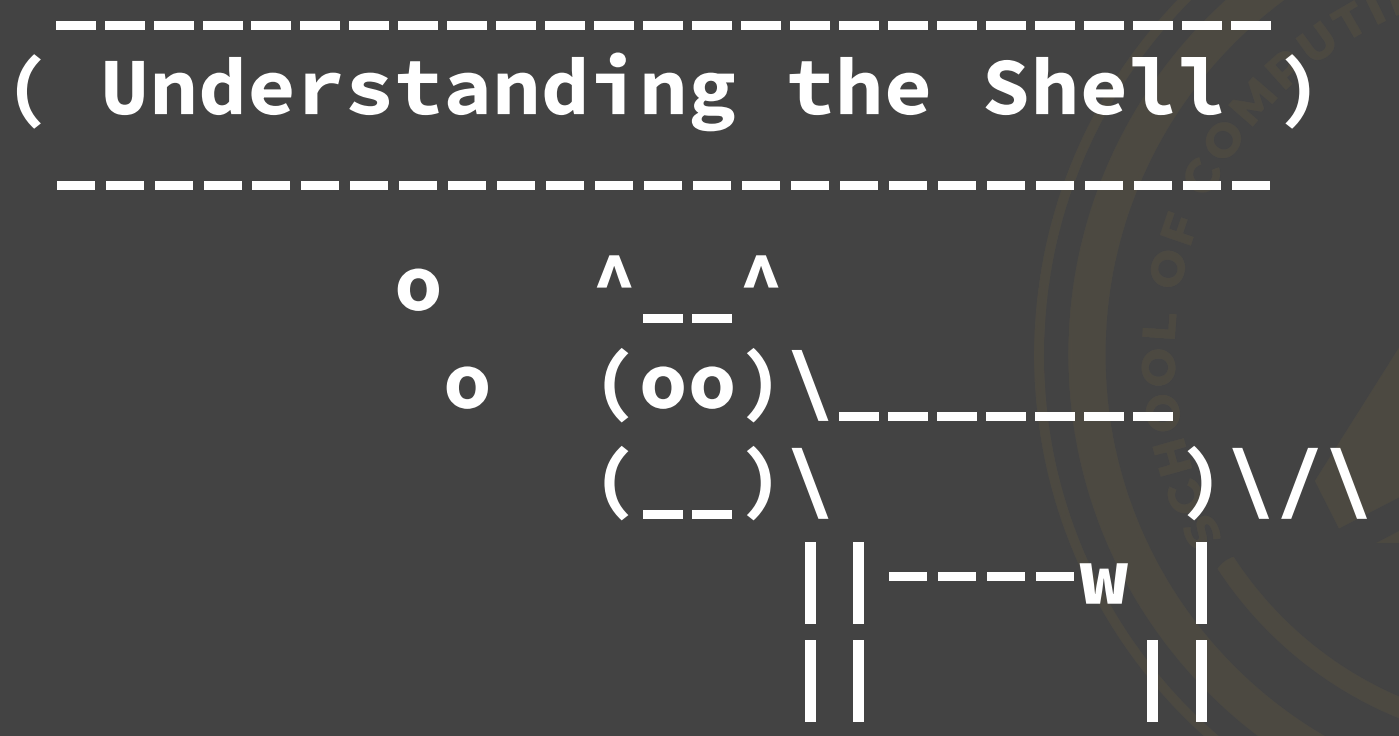
```
run-linting-container:
  stage: Validation
  image: docker
  variables:
    DOCKER_HOST: "tcp://docker:2375"
    DOCKER_TLS_CERTDIR: ''
  services:
  - name: docker:dind
  before_script:
  - docker info
  script:
    - docker login -u $CI_REGISTRY_USER -p $CI_REGISTRY_PASSWORD $CI_REGISTRY
    - docker compose -f compose.test.yml up --exit-code-from workflow-tester
```

# Learning Objectives

1. Learn about the CLI, its use cases, and its transferable skills
2. Learn command line syntax and conventions
3. Learn how to perform basic tasks with CLI and commands

You may not use all these skills daily, but will need these skills eventually

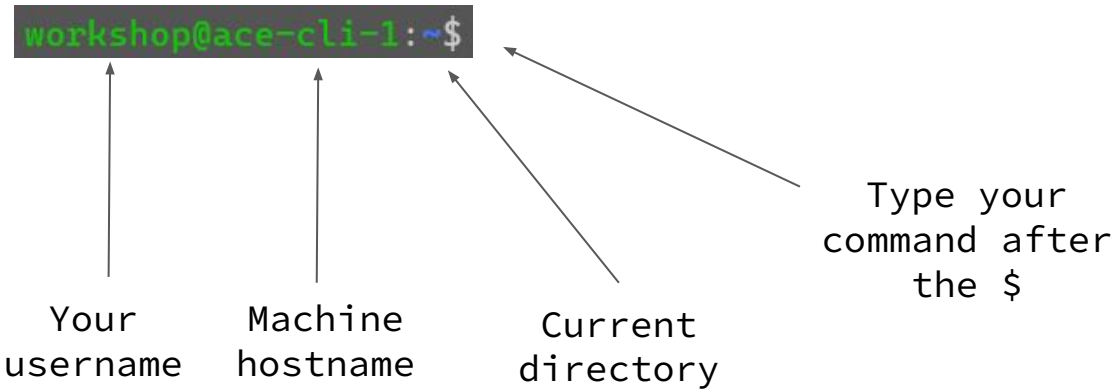
- "Skip the basics" later


Understanding the Shell

# Basic Shell Prompt

What's in a shell prompt?

(Varies slightly all the time)



# Bash

**Bash** (also known as Bourne Again Shell) is a **shell**. It is one of the more commonly used **shells** out there.

Some other shells include: **sh, zsh, csh, ksh**.

A shell allows you to:

1. Interact with an environment without a UI
2. Run scripts and perform **automation** tasks.

# Connecting to the Lab virtual machines

1. Choose a **portnumber** between 2200 and 2250 (Based on your seat number)
2. Run the command below, replacing **portnumber** with your chosen port number
3. When prompted for the password, enter "**aceCLI!**"

Example:

```
$ ssh workshop@acecli.jeijii.com -p portnumber
```

# Connecting to a remote shell & your lab environment

You can use **SSH (secure shell)** to connect to another computer and remotely control it.

**Basic usage:**

```
$ ssh username@hostname -p portnumber
```

**Example:**

```
$ ssh workshop@1.2.3.4 -p 2200
```

# Getting help with commands

Getting help with commands

**$ [command] -h**

**$ [command] --help**

Reading the manual for a command

**$ man [command]**

Navigate up and down with **arrow keys**, next page with **space** and **q** to quit.

# What's in a command

```
(base) matthewho@Matt-Book Test % ls -al --color=always
total 88
drwxr-xr-x   5 matthewho  staff    160 Dec 29 14:12 .
drwx------@ 13 matthewho  staff    416 Dec 29 14:11 ..
drwxr-xr-x   2 matthewho  staff     64 Dec 29 14:12 Work Folder
-rw-------@  1 matthewho  staff  19624 Dec 27 23:51 download-1.png
-rw-------@  1 matthewho  staff  22200 Dec 27 23:51 download.png
```

**command** -<shorthand arguments> --<longer arguments>

- Arguments are separated with spaces
- Some arguments can be followed with an input parameter
  - e.g. docker build -t <Tag> .

# Simple Keyboard Shortcuts

**Go to previous or next command**

↑ or ↓

**Autocomplete**

Tab

**Exiting most applications (SIGINT)**

Ctrl-C, or else try q


Navigating and Managing
the Filesystem

# Browsing a Filesystem

Get your current directory:

**$ pwd** (a.k.a. Print Working Directory)

Changing your current directory

**$ cd foldername** (relative path to folder)

**$ cd /foldername** (absolute path to folder)

**$ cd ../foldername**

**$ cd ~**

# Directory Shorthands

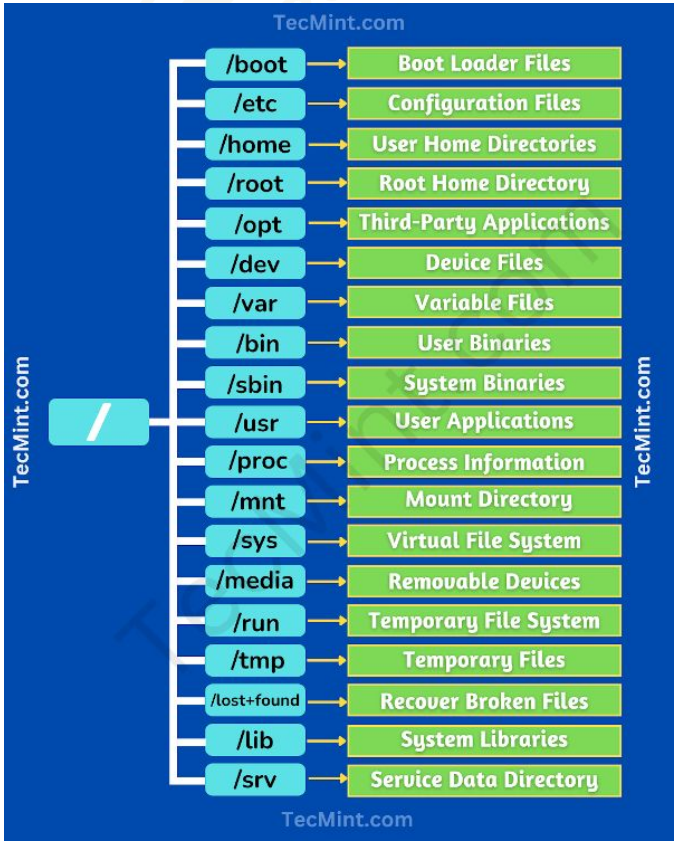
**Common directories:**

`~` Your home directory (/home/yourusername)

`/` System root directory

`..` Parent directory

`.` Current directory



# Listing the Directory

Showing files in a directory

**$ ls**

**$ ls -l**

**$ ls -al**

**$ ls -al /**

What's the difference between these commands?

- Use **ls --help** or **man ls** to find out

# Examining a File

Determining the type of a file

```
$ file [filename]
```

Full file information

```
$ stat [filename]
```

To print the contents of the file

```
$ cat [filename]
```

# Directory Tree

Showing files in all subdirectories

```
$ tree
```

# Creating Empty Files and Folders

Creating an empty directory

**$ mkdir foldername**

**$ mkdir -p foldername/foldername/foldername**

What does -p do?

Creating an empty file

**$ touch filename**

# Deleting Files and Folders

Deleting a file

**$ rm**

Recursively deleting everything in a directory

**$ rm -r**

Deleting an empty directory

**$ rmdir**

**PLEASE BE CAREFUL**

DO NOT **rm -rf /**

# Copying, Renaming and Moving Files

Copying a file (src) to another directory (dst)

```
$ cp src/file dst/file
```

Moving a file (src) to another directory (dst)

```
$ mv src/file dst/file
```

Renaming a file (oldname) to a new name (newname)

```
$ mv oldname newname
```

# What if I want to select multiple files?

You may use “globbing” in Bash to:

*** to match multiple characters**

**? to match a single character**

If no files are matched, the raw string is used.

Although this works for all commands, it is **most appropriate** for **commands that involve file names**.

# Finding Files

Finding all files ending with .txt, starting from the current directory (.)

```
$ find . -name "*.txt"
```

Finding all files modified in the last day (-1)

```
$ find . -mtime -1
```

Searching large directories like **/** or **~** take a while

# Exercise 1 (15 mins)

1. Navigate to **exercises/ex1**. Try using **cd**, **ls** and **pwd**

2. Remove **removeme.txt** with **rm**

3. Move **welcome.txt** from **source** to **destination**

4. Find the hidden **.ex1.txt** somewhere in **mess**

# Further Reading

Do something to the found files

```
~$ find mess -name ".ex1.txt" -exec cat {} \;
```

**Explanation**

**-exec** - Run a command on the found paths

**cat** - Output file contents

**{}** - Placeholder for the resulting file path

**\;** - End of command


< Installing Packages >

# apt (Advanced Packaging Tool)

apt is a utility for managing packages on Linux distributions.

Package: All the files needed for a function or application on a machine. Contain 1 or more applications.

**Easily Install/Uninstall/Update programs**

Some environments may use different package managers like yum, brew, apk

# Managing Packages

Update current package index (Think of **sudo** as "run as administrator")

```
$ sudo apt update
```

Search for packages

```
$ sudo apt search packagename
```

Get information on a package

```
$ sudo apt show packagename
```

# Managing Packages

Install a package

```
$ sudo apt install packagename
```

Remove a package

```
$ sudo apt remove packagename
```

List all installed packages

```
$ sudo apt list --installed
```

# Managing Packages

Upgrading installed packages

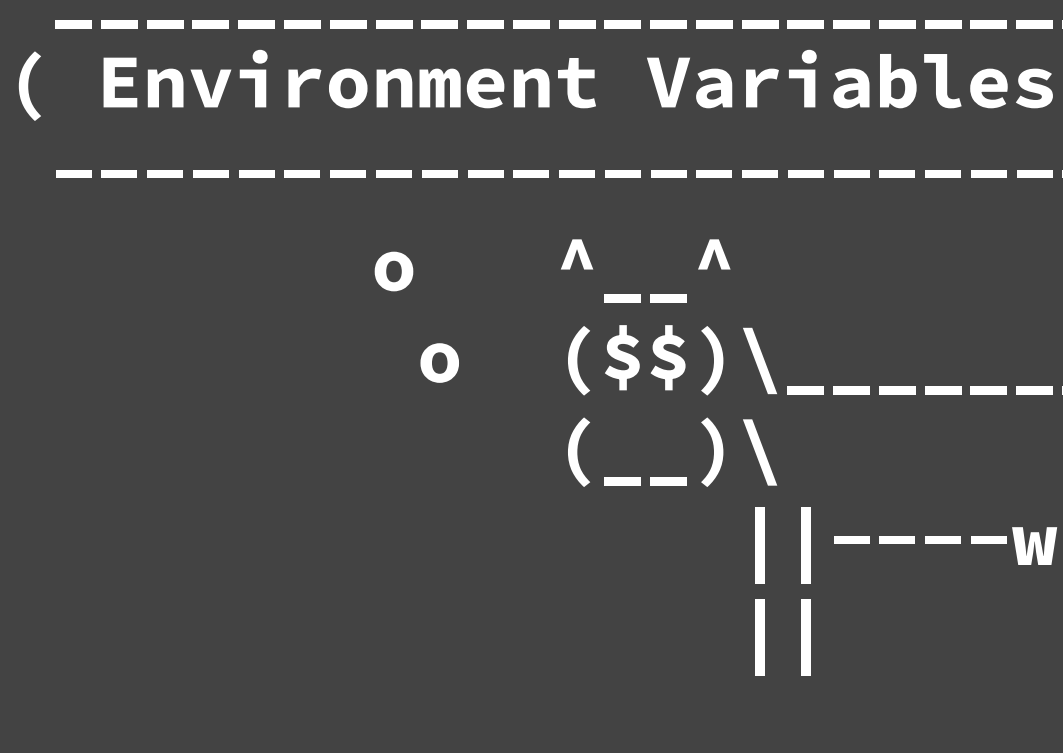
```
$ sudo apt upgrade
```

# Exercise 2 (10 mins)

Try to perform the following:

1. Install the following package: **cowsay**

2. Find out what the **cowsay** command does and try to make the cow appear “happy” while saying something.


( Environment Variables )
o ^__^
o ($$)_______
(__)\ )\/\
||----w |
|| ||

# Why Environment Variables?

Storing passwords in files is not a good practice

Some applications retrieve variables from environment variables instead

Applications can run in their own containers or "environments"

# Environment Variables

Environment variables are **key-value variables** passed to programs started by the command line shell.

Usually **CAPITALISED_WITH_UNDERSCORES**

Viewing the currently set environment variables

```
$ printenv
```

Viewing a specific environment variable

```
$ echo $varname
$ echo $PATH
```

| Environment Variables | |
|---|---|
| **NAME** | Matt |
| **PATH** | /Library/… |
| **SHELL** | /bin/zsh |

# Understanding the PATH environment variable

The PATH environment variable is a special variable which shows **which directories are searched** to find the executable for a command.

You can find the executable location a command by

```
$ which <command>
$ which python
```

# Modifying Environment Variables

Setting an environment variable

```
$ export varname=value
```

Unsetting an environment variable

```
$ unset varname
```

# Solution to Exercise 3

Setting the environment variable **MY_NAME**

```
$ export MY_NAME="Matt"
```

Or set the variable in a one-off fashion:

```
$ MY_NAME="JingZhi" python3 ex3.py
```


Permissions and Privileges
^__^
(OO)_______
(__)\ )\/\
||----w |
|| ||

# Super User Do

Running a command as **root**
(the highest privileged account)

**$ sudo [the original command]**

# Read, Write and Execute (Files)

Read, Write and Execute are permissions that are set for **files and directories**.

Files

**Read**: Allows you to view the contents.

**Write**: Allows you to modify the contents.

**Execute**: Allows you to run the code in the file or run as if its a self-contained binary.

# Read, Write and Execute (Directories)

Read, Write and Execute are permissions that are set for **files and directories**.

Directories

**Read**: Allows you to view the contents of the directory.

**Write**: Allows you to create and remove files in that directory.

**Execute**: Allows you to “cd” into the directory.

# User, Group and Other

**User**: The user who owns the file

**Group**: The group who owns the file

**Other**: Everybody else

You can view the file/dir ownership by performing

```
$ ls -l
```

# Read, Write and Execute (Directories)

| dir permissions | Octal | del rename create files | dir list | read file contents | write file contents | cd dir | cd subdir | subdir list | access subdir files |
|---|---|---|---|---|---|---|---|---|---|
| --- | 0 | | | | | | | | |
| -W- | 2 | | | | | | | | |
| R-- | 4 | | only file names (*) | | | | | | |
| RW- | 6 | | only file names (*) | | | | | | |
| --X | 1 | | | X | X | X | X | X | X |
| -WX | 3 | X | | X | X | X | X | X | X |
| R-X | 5 | | X | X | X | X | X | X | X |
| RWX | 7 | X | X | X | X | X | X | X | X |

# Changing File Permissions

Use **chmod**

**$ chmod [mode] [file]**

For example:

**$ chmod +x myfile** grants execute permissions to user, group and other.

**$ chmod g+w myfile** grants write permissions to group only.

Conversely:

**$ chmod u-r myfile** removes read permissions from the file.

How about **chmod ug+rw myfile**?

# Changing File Permissions (Octal System)

Read = 4
Write = 2
Execute = 1

An octal of 6 represents Read and Write only.

**$ chmod 755 myfile**
Read, Write and Execute (7) permissions for user,
Read and Execute (5) permissions group and other.

# Changing File Permissions (Octal System)

| | User | Group | Others |
|---|---|---|---|
| Read | 4 | 4 | 4 |
| Write | 2 | 2 | 2 |
| Execute | 1 | 1 | 1 |
| **Result** | 7 | 5 | 5 |

Read = 4
Write = 2
Execute = 1

**$ chmod 755 <file>**
The user (owner) can Read, Write and Execute [7].

Those in the same group and others can Write and Execute [5].

# Practical Scenario

Some files/directories should have a certain level of access control

.ssh directory: 700

SSH public keys: 644

SSH private keys: 600

Home directory: 755 (What does this mean?)

# Changing Ownership

You can use the chown command to change the user or group ownership of a file.

```
$ chown newuser:newgroup myfile
```


( Scripts )
o ^__^
o (@@)_______
(__)\ )\/\
:q ||----w |
|| ||

# Scripts

Put a series of commands in a file to make them easier to run

Use **.sh** for shell scripts

You may have to do **chmod +x <script>.sh** to make them runnable

# Shebang!

Specified at the top of the file.

**#!/bin/bash**

Run the file with /bin/bash

**#!/usr/bin/env python3**

Run the file with python3

# Exercise 4 (10 mins)

1. Note three files in the ex4 folder.
    a. all4me.sh
    b. pubkey.pub
    c. private.key

2. Check their permissions with **ls -l**

3. Change their permissions to those described in the table

| | |
|---|---|
| all4me.sh | rwx --- --- |
| pubkey.pub | rw- r-- r-- |
| private.key | rw- --- --- |


( Editing and Manipulating Files )
^__^
(@@)_______
(__)\ )\/\
:q ||----w |
|| ||

# Text Editors

Nano and Vim are the two most popular command line text editors.

- **Nano is installed by default on Ubuntu**
- Vim may require installation.

Nano is a simple and fairly easy to use editor, whereas Vim is much more complex, but offers a lot more functionality out of the box.

# Nano

Opening a file to edit (Does not have to already exist)

**$ nano file_name**

Straightforward, commands at the bottom to guide you

Crucial commands:

**Ctrl + O -** Writes to a file

**Ctrl + X -** To exit

# Vim

Opening a file to edit

**$ vim file_name**

Vim is a mode-based editor.

Toggle between different modes to access different functionalities of the editor.

# Vim - Basics

**Keybinds:**

i - insert mode

Esc - normal mode

Arrow keys, hjkl - Move around

In normal mode,

**:q!** - Exit Vim without saving changes

**:w** - Save Changes

**:wq** - save and exit

# Exercise 5 (10 mins)

1. Get familiar with either Nano or Vim
2. Using Nano or Vim, edit count.py to count the number of files with 'SCIS' in its file name.

3. Check for correctness

   **$ python3 count.py | ./ex5.sh**

4. (Optional) Try writing a bit of code in geometric.py

# Exercise 5

Some platforms may have only Vim/Nano

Get to know a bit about both

Try to identify your preferences!


Pipes & Redirection
o ^__^
o (oo)_______
(__)\ )\/\
|| ||----w |
|| ||

# Output Redirection

Writing to a file quickly

```
$ echo "hello world" > hi.txt
```

Appending to an existing file

```
$ echo "hello world" >> hi.txt
```

# Pipes

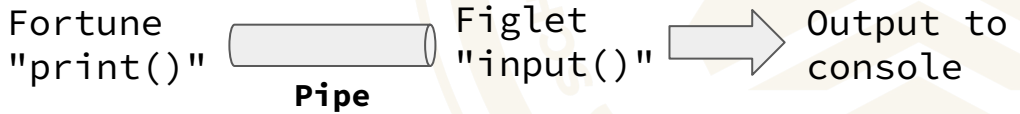
Passing output to another program

```
$ cat copypasta1.txt | grep [search term]
```

Now try this!

```
$ fortune | figlet
```

```
$ cat copypasta2 | cowsay
```



# Command Interpolation

You can use the **output of a command** as the **input to another command.**

```
$ echo The time now is $(date)
```

Now try

```
$ echo "The time now is $(date)"
```

```
$ echo 'The time now is $(date)'
```

# Exercise 6: Pipes (15 mins)

How many times does the word **bee** occur in the Bee Movie script?

```
$ [your command(s)]
7
```

Check your answer:

```
$ [your command(s)] | ex6
```

# Hints

Use **curl** to download the text. The link is in url.txt

Use **grep** to print only the matching part of the lines

Use **wc** to count the number of lines/matches

Pipe the output into ex6.py

**curl <url> | grep <arguments> | wc <arguments> | ex6.sh**

# Takeaways

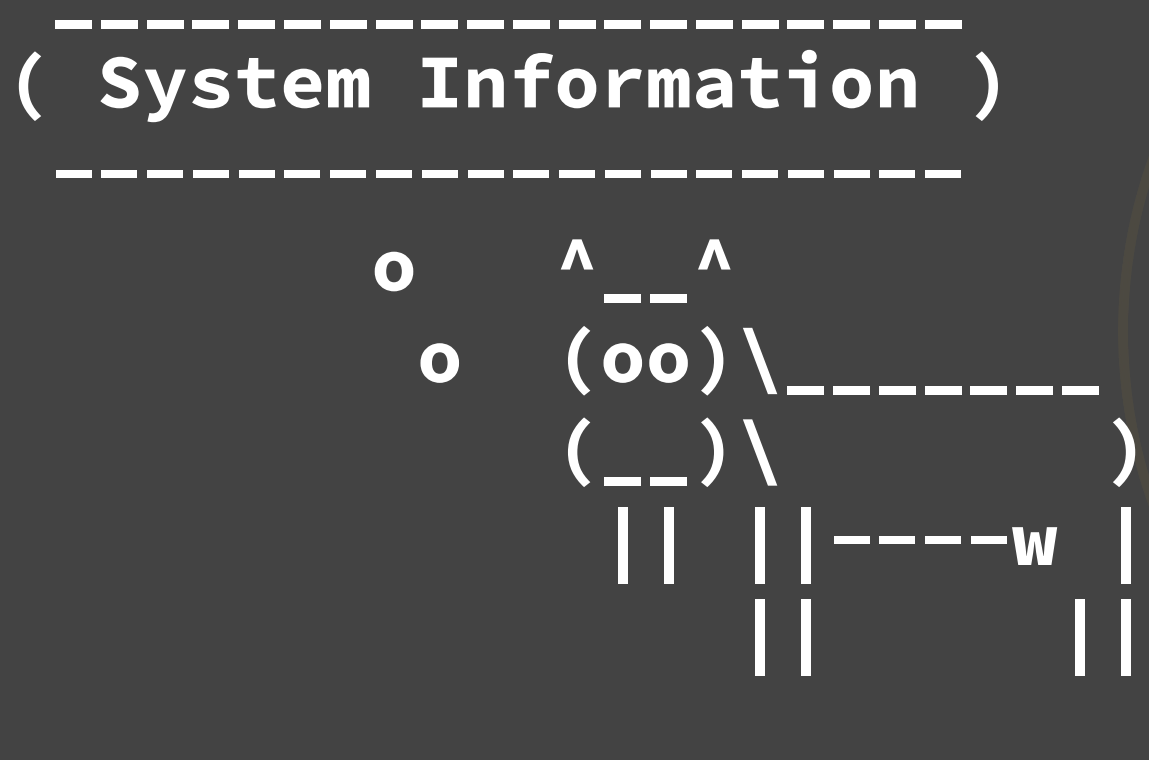
Pipes allow you to join commands together to perform simple operations

Best used in scripts, not manually typed every time

**When to use Python/Bash scripts?**
Bash scripts are run directly on the shell, which is faster

Bash can be harder to use, python may need fewer lines

The environment may not always have a python interpreter


( System Information )
o ^__^
o (oo)_______
(__)\ )\/\
|| ||----w |
|| ||

# System Information

Viewing how long a system has been running

**$ uptime**

Viewing running process information

**$ top** (Continuous)

**$ ps** (Instant)

# System Information

Viewing system information

```
$ uname -a(Your OS)
$ lscpu      (CPU info)
```

File system information

```
$ lsblk      (Block devices)
$ df         (Space on a file system)
$ df -h      (Same as above but formatted)
```


( Network Tasks )
o ^__^
o (oo)_______
(__)\ )\/\
||----w |
|| ||

# Making HTTP Requests

HTTP request using curl

**$ curl www.google.com** (Can upload & download)

HTTP request using wget

**$ wget www.google.com** (Downloading)

# Performing DNS Lookups

Resolving a domain

```
$ nslookup www.google.com
```

If you need more information

```
$ dig www.google.com
```

# Troubleshooting Network Connectivity

Checking if a host is up

**$ ping <hostname/IP address>**

Obtaining the network hops

**$ traceroute google.com**

# Exercise 7: Network Tasks

Try to **ping** a website or IP address

Use **dig** to get DNS information on a domain name

Use **curl** to download a webpage

**curl parrot.live**

**curl wttr.in**


< User Management >

# Managing Linux users and Groups

Some environments like docker containers run applications as root

Need to know some basics to not run these as root all the time

# Managing new users and groups

Creating a user

**$ adduser [username]**

Creating a group

**$ addgroup [group_name]**

Deleting the user

**$ deluser [username]**

Becoming the user

**$ su [username]**

Changing the password

**$ passwd [password]**

Add a user to a group

**$ sudo usermod -g [gid] [user]**

# Exiting

# Exiting

Getting out of your user

```
$ exit
```

Closing the SSH connection

```
$ exit
```

Closing your local terminal session

```
$ exit
```


Thank you for attending
the ACE Command Line
Workshop!

https://www.redhat.com/en/services/training/rh124-red-hat-system-administration-i