Hacker Tools:
Shell & Scripting

Jing Yen UWU

7 February 2023
Slides at `https://hckr.cc/ht-shell-slides`
(Materials developed by Julius)

Where are we?

Introduction

Shell

Scripting

Conclusion

NUS Hackers

http://nushackers.org

hackerschool

Friday **Hacks**

Hack & Roll

Hacker Tools

About Me

Hi! I'm Jing Yen. My GitHub is
https://github.com/JingYenLoh

I'm a Year 4 Computer Science Undergraduate who loves hacking and building systems.

I don't watch anime.

What you will learn today

How to hack on a Unix-like environment:

- How to use the shell
- How to create scripts for automation

## Required Software

Unix-like environment, either one of these:

- Linux (you're good if you attended and installed Linux during our Linux Install Fest last week)
- macOS[1]
- BSD
- Other Unix-like OS'es (Minix, Solaris, AIX, HP-UX, etc.)
- WSL (Windows Subsystem for Linux) should also be alright, but no guarantee

---

[1]Open Terminal, and run first

Unix? Can I eat that?

- A family of multitasking, multiuser OS'es.
- First developed in the 1970's.
- Popularised the use of interactive command line.
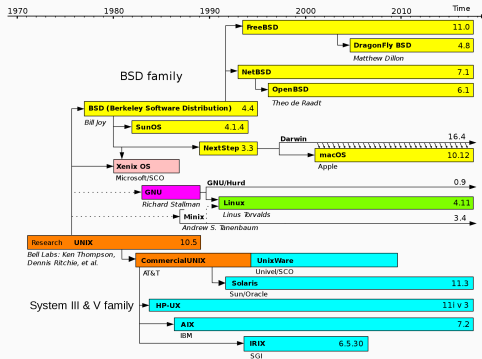
The Unix Philosophy

1. Write programs that do one thing and do it well.
2. Write programs to work together.
3. Write programs to handle text streams, because that is a universal interface.

Where are we?

Introduction to Shell

- An efficient, textual interface to your computer.

- Provides an interactive programming language
  ("scripting").

- Many shells to choose from:
  - Standard ones: sh or bash
  - Shells that match languages: csh
  - "Better" shells: fish, zsh

- For this workshop, the focus is on the ubiquitous sh and
  bash.[2]

---

[2]Feel free to explore other shells. On macOS, many people prefer fish
or zsh

The Shell Prompt

- What greets you when you open a terminal.

  ```
  0 16:21:57 julius@r-165-105-25-172:~/GitHub/hackerschool-hackertools
  501 (master) $
  ```

- Lets your run programmes and commands.

Common Commands

- to get the **man**ual pages of a command
- to **c**hange **d**irectory
- to **lis**t files and directories
- to **m**ake **dir**ectory
- to **rem**ove files and directories
- to **co**py file
- to **m**ove file
- to **p**rint **w**orking **d**irectory

Command Editing Shortcuts

bash has shortcuts based on emacs keybindings:

- ☐ Ctrl + a : beginning of line
- ☐ Ctrl + e : end of line
- ☐ Alt + b : move back one word
- ☐ Alt + f : move forward one word
- ☐ Ctrl + k : delete from cursor to the end of line
- ☐ Ctrl + _ : undo

And some special ones:

- ☐ Ctrl + u : delete from cursor to the start of line
- ☐ Ctrl + w : delete from cursor to start of word

You can find more in documentation for readline

Command Control Shortcuts

- ■ Ctrl + c : terminates the command
- ■ Ctrl + z : suspends the command (to continue)
- ■ Ctrl + l : clears the screen
- ■ Ctrl + s : stops the output to the screen
- ■ Ctrl + q : allows output to the screen

Fun (?) Commands

- - what is fork bomb
- - what is destroy everything

Where are we?

Script (1/2)

You can write programs directly at the prompt, or write into a file (writing scripts)

- Open an editor (for beginner, is recommended), save the script as example-script
- On your shell, run
- You can run your script as

Script (2/2)

Magic?

- is also known as the **shebang**, specifies the interpreter[3]
- is a command that prints its arguments to the standard output.

---

[3]You can use other interpreters too, e.g.   for a python script.

Flags (1/3)

- Most command line utilities take parameters using **flags**.
- They come in short form (-h) and long form (--help)
- Usually, running or will give you a list of the flags the program takes.
- Short flags can be combined: is equivalent to or

Flags (2/3)

- A double dash -- is used in to signify the end of command options, after which only positional parameters are accepted.
    - For example, to create a file called -v, Use instead of
    - For example, to grep a file called -v, will work while will not.

Flags (3/3)

Some common flags are a de facto standard:

- -a commonly refers to all files (i.e. also including those that start with a period[4])

- -f usually refers to forcing something, e.g.

- -h displays the help for most commands

- -v usually enables a verbose output

- -V usually prints the version of the command

---

[4]In Unix, by convention files whose names begin with a period is hidden. The origin is an accident, find out more here

Unix Directory Structure

Unix has a different directory structure from Windows.

There is no concept of drives.

Everything is files and directories. The root directory is /

We use forward slash / instead of backward slash \

Specifically for Linux, there is FHS[5]

---

[5]https:
//en.wikipedia.org/wiki/Filesystem_Hierarchy_Standard

Important Unix Directories

- /bin, /sbin, /usr/bin, /usr/local/bin, /opt = executables
- On Linux: /home = user home directories
- On macOS: /Users = user home directories
- /var/log = log files
- /tmp = temporary files
- /dev/urandom = random number generator

Where are we?

Running a command

Variables (1/3)

- Used to store text
- to set variable
- to access variable

Variables (2/3)

There are also a number of special variables:

- ■ : get exit code of the previous command
- ■ to : arguments to a script
- ■ : name of the script itself
- ■ : number of arguments
- ■ : process ID of current shell

Variables (3/3)

Create a script `variable-example` containing the code
below, then try running it with various arguments.

Loop (1/4)

Loop is used to run a command a bunch of times.

For example:

Loop (2/4)

Let's unpack this!

- terminates a command – equivalent to newline
- Split , assign each to , and run
- Split by "whitespace" – we will get into it later
- Compared to C, no curly braces, instead and

Loop (3/4)

Let's unpack this!

- Run the program with arguments 1 and 5
- Substitute the block with the output of the program
- Equivalent to

Loop (4/4)

Let's unpack this!

- Everything in a shell script is a command
- Here, it means run the command, with argument hello.
- All commands are searched in (colon-separated)
- Find out where a command is located by running , e.g.

Conditionals (1/2)

Let's unpack this!

- is a command.
- If its exit code is 0 (success), then   is run.
- Optionally, you can also hook in an   or

Conditionals (2/2)

Let's unpack this!

- is a program that provides various checks and comparison which exits with exit code 0 if the condition is true[6].
- Alternate syntax: , e.g.

---

[6]Remember, you can check exit code using

Everything Together

Let's create a command like that only prints directories:

Bug!

Hold on! What if the directory is called "My Documents"?

- expands to
- Will first perform the test on My, then on Documents
- Not what we wanted!

Argument Splitting

- Bash splits arguments by whitespace (tab, newline, space)
- Same problem somewhere else:
- If contains whitespace, will error!
- Need to use quote to handle spaces in arguments
- How do we fix our script?
- What do you think does?

Globbing (1/2)

- knows how to look for files using patterns:
    - : any string of characters
    - : any single character
    - : any of these characters
- Thus, means all files in this directory
- When globbing, each matching file becomes its own argument
- However, still need to make sure to quote, e.g.

Globbing (2/2)

You can make advanced patterns

  ■ :

Globbing (2/2)

You can make advanced patterns

- : all files starting with a in the current directory
- :

Globbing (2/2)

You can make advanced patterns

- ■ : all files starting with a in the current directory
- ■ : all .txt files in foo
- ■ :

Globbing (2/2)

You can make advanced patterns

- ■ : all files starting with a in the current directory
- ■ : all .txt files in foo
- ■ : all three-letter text files, starting with p, in subdirectories of foo

Other whitespace issues

- : What's the issue?

Other whitespace issues

- : What's the issue?
- What if is empty? arguments to are = and bar
- Possible workaround: , but very hacky

Other whitespace issues

- : What's the issue?
- What if is empty? arguments to are = and bar
- Possible workaround: , but very hacky
- Instead, use : bash built-in comparator that has special parsing
- Good news: it also allows instead of , instead of , etc.

shellcheck

- The mentioned problems are the most common bugs in shell scripts.
- A good tool to check for these kinds of possible bugs in your shell script: `https://www.shellcheck.net/`

Where are we?

Composability

- Shell is powerful, in part because of **Composability**

- You can chain multiple programs together, rather than one program that does everything

- Remember **The Unix Philosophy**:
    1. Write programs that do one thing and do it well.
    2. Write programs to work together.
    3. Write programs to handle text streams, because that is a universal interface.

Pipe (1/2)

Let's unpack this!

- Means run both a and b, but send all the output of a as input to b, and then print the output of b

Pipe (2/2)

You can chain this even longer!

- prints the system log
- This output is fed into , which looks for all entries from today.
- This output is then further fed into , which prints only the last 10 lines.

Streams

- All programs launched have 3 streams:
  - STDIN: the program reads input from here
  - STDOUT: the program prints to here
  - STDERR: a second output that the program can choose to use.
- By default, STDIN is your keyboard, STDOUT and STDERR are both your terminal

Stream Redirection (1/2)

- However, this can be changed!
- ■ : makes STDOUT of a the STDIN of b.
- ■ : STDOUT of a goes to the file foo
- ■ : STDERR of a goes to the file foo
- ■ : STDIN of a is read from the file foo
- ■ : STDIN of a is read from what comes after <<<
- You can also pipe to tee (look up in man what tee does)

Stream Redirection (2/2)

So why is this useful?

Stream Redirection (2/2)

So why is this useful?

It lets you manipulate output of a program!

Stream Redirection (2/2)

So why is this useful?

It lets you manipulate output of a program!

- ■ : all files that contain the word foo
- ■ : all processes that contain the word foo
- ■ On Linux: : last 5 system log messages with the word intel (case-insensitive)
- ■ Note that this forms the basis for **data-wrangling**, which will be covered later.

Grouping Commands

- Run a, then b, and send all their output to tac[7]
- For example:

---

[7]tac print in reverse

Process Substitution

- Run a, generate a temporary file name for its output stream, and pass that filename to b
- To demonstrate:
- On Linux:
- This shows the difference between the first 20 lines of the last boot log and the one before that.

Where are we?

Job (1/2)

Used to run longer-term things in the background.

- Use the suffix
    - It will give back your prompt immediately.
    - For example:
    - Note that the running program still has your terminal as
      STDOUT. Instead, can redirect STDOUT to file.
    - Handy especially to run 2 programs at the same time
      like a server and client:
    - For example:

Job (2/2)

- ■ : see all jobs
- ■ : bring the job corresponding to the id to the foreground (with no argument, bring the latest job to foreground)
- ■ You can also background the current program: ^Z[8], then run
  - ■ ^Z stops the current process and makes it a job.
  - ■ runs the last job in the background.
- ■ is the PID of the last background process.

---

[8] Ctrl is usually denoted as ^, thus Ctrl + z is denoted as ^Z

Process Control (1/2)

- ■ : lists running processes
  - ■ : lists processes from all users
  - ■ Check out the man page for other arguments.
- ■ : find processes by searching (like)
  - ■ : find processes with arguments
- ■ : send a *signal* to a process by ID (to search and run)
  - ■ Signal tells a process to do something
  - ■ SIGKILL (-9 or -KILL): tell it to exit *right now*
    (equivalent to ^\\)
  - ■ SIGTERM (-15 or -TERM): tell it to exit gracefully
    (equivalent to ^C)

Process Control (2/2)

- ■ : send a *signal* to a process by ID (to search and run)
    - Signal tells a process to do something
    - Most common[9]:
        - SIGKILL (-9 or -KILL): tell it to exit *right now* (equivalent to ^\)
        - SIGTERM (-15 or -TERM): tell it to exit gracefully (equivalent to ^C)

---

[9]Prefer SIGTERM over SIGKILL:

https://turnoff.us/geek/dont-sigkill/

More Resources

- If you are completely new to the shell, you might want to read a comprehensive guide, such as BashGuide[10].

- For a more in-depth introduction, The Linux Command Line[11] is a good resource.

---

[10]http://mywiki.wooledge.org/BashGuide
[11]http://linuxcommand.org/tlcl.php

Where are we?

xargs

- Sometimes piping doesn't quite work because the command being piped into does not expect the newline separated format.
- For example, `file` command tells you properties of the file.
- Try running and
- What is doing?

Other Exercises

- Try running , then . What appeared?
- Sometimes you want to keep STDIN and still output to a file. Try running
- Run , then . What are the contents of hello.txt? How is different from ?

Where are we?

Talk to us!

- **Feedback form**:
  https://hckr.cc/ht-shell-feedback

- **Upcoming Hacker Tools**:
  Data Wrangling, , 6.30pm, Same Location!
  https://hckr.cc/ht-data-signups

- **Happening this Thursday!**:
  Applcations of OpenCV: https://hckr.cc/hs2223-w5

- **Upcoming Friday Hacks**:
  HackRoll 23: More Hacks: https://hckr.cc/links