Shell basics

Shell script starts with: #!/bin/bash and follows with commands to run

Variable, Argument and Argument Checking

Store Variables: $Var=... (no space)

- $0, $1, ... – arguments passed to script ($0 is the script name)
- $@ – all arguments passed to script as array
- $# - length of the argument input. xxx.sh aaa aaa, #=2

Redirection

Command	Description	Example
>	Redirects output to a file, Creates one if it does not exist, Overrides if it does	echo hello > myfile.txt
>>	Redirects and appends output to a file	echo world >> myfile.txt
<	Redirects contents from file to stdin of another command	grep hello < myfile.txt
2>	Redirects stderr to a file	grep hi myfile.txt 2> errors.txt

Pipe → Take stdout and use this as stdin for second command

Shell If Statement

```
if [ condition ]
then
    Do something
elif [ condition ]
then
    Do something else
else
    Do something different
fi
```

Shell While Statement

```
while program
do
    echo looping
done
```

Shell for Statement

```
for name in Alice Bob Carroll #name
do
    echo name is $name
done
```

Managing Processes

fork

→ Create a *child* process which is identical to parent process, except for PID, parent process PID, resource metadata..runtime

→ Once forked, both programs continue from the point of the fork system call

Parent: fork returns PID of child

Child: fork returns 0

exec

→ Used to completely replace the currently running process with a different program specified

wait

→ Suspends execution of current process until one of its child processes exits

→ If child exits but parent has not called wait, child becomes a *zombie* process until wait is called

→ Stays in the process table once exited

▸ **zombie process example: PLACEHOLDER**

kill

→ not really "kill" but sending signals to process, Syntax: kill <flag> <pid>

Signal	Description	Code	Example
SIGINT	Signal Interrupt	2	kill -2 22
SIGKILL	Terminate process	9	kill -9 22
SIGSTOP	Stop the process	19	kill -19 22
SIGCONT	Continue the process	18	kill -18 22

```
import os
import time

def main():
    pid = os.fork()

    if pid == 0:  # Child process
        print("Child process: PID =", os.getpid())
        exit(0)  # Exit child process immediately
    else:  # Parent process
        print("Parent process: PID =", os.getpid())
        print("Sleeping for 60 seconds...")
        time.sleep(60)  # Sleep for 1 minute

if __name__ == '__main__':
    main()
```

Memory Management

Memory for processes is stored in the following layout:

- program code instructions: at the bottom of program
- static and global data
  - data available to the whole program and constants often
- **heap - used to store data structures.**
  - eg. objects in OOP code
  - grows upwards to expand
- **Stack -** used to store temporary variables
  - e.g. variables which are used within a function, or function arguments
  - often discarded when not needed
  - grows **downwards** to expand

File System

A file system describes a way to lay out/represent file data on mass storage

Operating systems have file system module, which provides an interface to the namespace to interact with files from mass storage, or other devices

File systems can also "mount" 'files' which display information about the system

Example: /proc directory on Linux.

This means there are many implementations of file systems, optimised for different features such as speed, reliability/redundancy etc

For example: **FAT16, FAT32, exFAT** in the Windows operating system, **ext2, ext4, BtrFS, ZFS** etc for the Linux operating system.

Virtual Memory

- Each process is allocated **virtual memory**
  - The process sees this as a regular, contiguous memory block
  - In reality, the OS maps different parts of this memory to the different locations of RAM/mass storage in which they are stored, as in the diagram
- Why use mass storage for memory?
  - RAM has more limited size than hard drives
  - If some memory required by the program won't be accessed/changed much, it makes sense to temporarily store it in mass storage
- Benefits of virtual memory
  - If process memory needs to be expanded, the page table entry for the process can be adjusted to account for extra memory

Accessing Memory on mass storage

- What happens if a process needs to access memory on mass storage?
  - Request for access will throw an *interrupt* to OS kernel – essentially telling CPU to save what it's doing and handle this request
  - Kernel finds section on disk, and writes it to free area of RAM
    - If there is no free RAM left, the kernel must take some existing block of memory from a process in the RAM, and *swap* it to mass storage
  - Page table updated to account for this changes
- Handy features of virtual memory
  - "shared memory" – processes can have parts of their memory mapped to the same physical block
  - "copy on write" – when a process is forked, both parent and child share the same memory, until it needs to be changed
- Find info about virtual and "real" memory of a process
  - In ps:
    - the VSZ field tells us about the size of the process's virtual memory
    - the RSS field tells us about how much physical RAM a process is using

Scheduling

- Sleeping: (maybe
  - waiting for resource (e.g. file, I/O device) to become available
  - waiting to be executed by CPU
- Priority of process: indication and changing it
  - Range from negative integers (highest priority) to positive integers (lowest priority)
  - On Linux, by default, the priority for processes started by users is 20
  - nice value
    - nice value adds/subtracts a number to a process's priority to modify its default priority
    - nice value is separated from the priority itself
    - Valid nice values typically range from -20 (highest priority) to 19 (lowest priority).
    - Default value is 0 (has a neutral priority)

```
$ nice -n 5 sleep 30
$ renice 10 -p 1203
```

Shell

```
echo -n
# -n flag basically cancel the new line at the end of each line echoed
mkdir
rm #remove
cp #copy
touch #create files
sort #sorting -f, --ignore-case // -r, --reverse // -u, --unique
     # // -s, --stable
ls -l #view metadata with files in the working directory
chmod <group name><+/-><premission>
find "PATH" -type f/d -name ".txt"
grep -i '^[^u]*[aeio][^u]*$' /usr/share/dict/words | sort -r
#regular expression  -i case insensitive
tail #
head # By default, head prints the first 10 lines of each file
-n [number]: Display the first [number] lines of the file.
-c [bytes]: Display the first [bytes] bytes of the file.
-q: Quiet mode. Do not print headers giving file names.
-v: Always print headers giving file names.
cat #reading content

cut -d "," -f 1,4 "yellow_pages.csv"
# -d is like split by ",", -f (LIST) is choosing the outcome of the cut
# -b selecting certain bytes, -c = -b but only chars
```

Boot Sequence

power on	hardware self-checks
BIOS running	in flash memory
MBR loaded	from 1st block of disk
GRUB loaded	also from early part of disk
kernel loaded	started by kernel from file system
init started, PID=1	started by kernel from file system
system processes running	started by init
system operating normally	/

Question 14 1 /

To retrieve the third and fourth lowest bits (i.e. from the right) of the number x, which sequence of bitwise operations should we use in the missing line of the Python code below?

For example, if we are given x = 1110<u>1100</u>, the output should be 11.

```
x = 0b00101100
.....          # what shoul
print("{:b}".format(x))  # pri
```

- x = (x << 2) & 0b11
- x = (x & 0b1100)
- x = (x & 0b1100) >> 2
- no correct choice

```
#!/bin/bash

if [ $# -eq 0 ];
then
    echo "WHAT IS GOING ON"
    exit 1
fi

echo -n "" > file.txt

BASE=$1

if [ $(expr $BASE % 2) -eq 0 ];
then
    BASE=$((BASE+1))
fi

for (( i=0; i<=100; i++ )); do
    echo $(expr $BASE + $i \* 2) >> file.txt
done
```

Emulators, Virtual Machines, Containers

Assembly language → Human readable format(?) which directly represents instructions CPU will execute

- **Emulators** – programs which can run code designed for a different CPU/system architecture, and translate it to run on our own system
  - Name of "emulate" – effectively simulates a completely different system
  - Emulators will take CPU instructions given by a program for a specific system hardware (e.g. x86), and translate it to system instructions for another hardware type (e.g. ARM)
  - Often used to
    - emulate older system architecture to run programs (e.g. Commodore 64, PowerPC)
    - emulate a game console to play console games on a computer
- **Virtual machines** – program which gains limited access to real hardware resources and constructs a **virtual computer** based on this hardware
  - Does **not** emulate system code – so "virtual" system must have same architecture as host computer (so that the same CPU instructions are used)
  - Can choose how much hardware resources to allocate to virtual system
  - Often faster than emulators, as instructions do not have to be translated

Containers

- **Containers** are programs which provide a virtualised namespace for applications to access with an **emulated system call interface**
  - Applications think they are running in a completely different environment, or even OS
  - Because containers only emulate system call interface, applications can access real hardware
    - more lightweight than virtual machine
  - Virtualised namespaces often contain common utilities found on a regular OS:
    - standard command-line programs
    - configuration files
    - libraries

Microservices

- Lightweight services accessible via the Internet
- Provide an API to communicate via a protocol
  - Can be an existing protocol
  - Can be its own custom application-layer protocol as an extension to the protocol stack

API - Application Programming Interface

- a standardised structure and methodology for 2 programs to communicate with each other
- Can standardise communication, e.g. if a client sends some specific data, the server responds with data in a specific format
- Primarily used to send and receive data over the Internet whilst hiding the details of how that data is implemented
- Can be thought of as "function calls", with their own parameters and return results

Implementation of APIs

- **Web protocols are commonly used for communicating over APIs**
- sending data over a web protocol → Make parameters part of URLs (may use structured text encoding to present) : XML / JSON → We all know JSON
- **API aggregators → organisations which collate micro-services accessible via APIs →** Provide template code to integrate communication with micro-service API in your own code

Processes

- **Program**: executable sequence of instructions stored on mass storage
- **Process**: instructions of program stored in memory, and any other data required by program

unix system has many process stored in memory at same time

▸ **For a single CPU, only one program can run at a time**

▸ **UID associated to process stating who started the process, superuser UID = 0**

▾ **Metadata of file can be seen by running ls -l**

type of file (- = normal file, d = directory, l = link b = block file)

permissions (r = readable, w = writable, x = executable (for files)/accessible (for directories)) – listed as [r/-][w/-][x/-] for each group, where - means permission doesn't exist

user permissions | group permissions | permissions for users in other groups

▸ **Changing Permission using chmod command:**

▸ **Namespace**

Packet Switching

- **Packet switching** is a technology developed to overcome shortcomings in circuit-switched communications
  - Rather than being sent along a continuous stream, data is broken down into smaller **packets**
  - Packets travel independently from sender to receiver – they don't have to necessarily follow the same path!
  - Important thing is that all data is transmitted

data stream divided into packets at the source computer, send through routers, then be reconstructed from received packets at destination

- **Reroute → As packets follow different routes around a network, they can be rerouted**
  - cases: congestion on one connection
  - a connection is broken
- **Rerouting gives data sent through packets switching resilience**
- Furthermore, packet switching allows for multiple packets from different sources to be sent over the same connection

LAN, WAN

- **Local area networks (LAN)** – networks that are localised to one specific location, such as a house, business, or university campus — 256 addresses
  - LANs are generally connected with **ethernet** technology (for wired connections), or **Wi-Fi** for wireless networks
    - **Router** used to route internet data correctly to connected devices
    - Routers connect to **modems**, which connect to the outside Internet, and are used to convert between analogue à digital data when downloading, or digital à analogue data when uploading (hence the name: **mo**dulator-**dem**odulator)
      - Most modern home routers are combined with a modem in a single package
- **Wide area networks (WAN)** – networks that connect multiple LANs together to create a larger, interconnected network – i.e. all LANs connected in Sydney, or indeed, the Internet

Protocol

- **Protocol** – set rules/standards which dictate how communication should occur between two devices
  - Many protocols used in network communications
  - Protocols can be divided up into "layers"
    - "Internet Protocol Suite" or "TCP/IP Model" dictates basic, functional layers of protocols
    - OSI model is more of a theoretical (rather than practical) layout of protocol layers
- As data packets move **down** the protocol stack, the lower layers wrap the packet in more information with their own **headers** and **footers**
- As the data packets move **up** the protocol stack, these headers/footers are removed to access data

Example of protocol:

- **Ethernet** – protocol in the "Link" (lowest) layer of internet protocol suite
  - Ethernet packets are used to transfer data from one point to another point in a network (e.g. from one router to another)
  - Packets only exist for this single "hop", before they are unwrapped and wrapped up again to get to the next point
  - Uses **MAC address** to store information on source, and find destination of "hop"
    - A MAC (Media Access Control) address, sometimes referred to as a **hardware** or **physical** address, is a unique, 12-character alphanumeric attribute that is used to identify individual electronic devices on a network. An example of a MAC address is: 00-B0-D0-63-C2-26.
      - Unique address for all devices which connect to the internet
- **Internet Protocol (IP)** – used for communication between any two devices connected to the Internet
  - Header contains destination **address** and source **address**
    - IPv4 address is 4 bytes long, and written in 4 sections (e.g. 129.168.0.1)
    - Difference between **local IP address** and **public IP address**
      - Public IP address is what is used to communicate over the internet
      - Router is able to use IP packet to convert a public IP into a local IP address
  - When a data packet gets to the IP layer, the router finds the next device in which the packet should "hop" to to get to its destination, and puts this in a surrounding ethernet frame
- **Transmission Control Protocol (TCP)** – used in order to check for any errors during data packet transmission that may have occurred by calculating **checksum**
  - Compares calculated checksum to checksum stored in header

IP Routing

- **Internet Protocol (IP)** – used for communication between any two devices connected to the Internet
  - Wraps packets in higher protocol layers, such as
  - Transmission Control Protocol (TCP)
  - User Datagram Protocol (UDP)
  - Internet Control Message Protocol (ICMP)
- **ICMP** – protocol which dictates a simple "echo" request
  - When the intended receiver receives this packet with an echo request, it re-sends the packet back to the sender
  - Used in ping command
  - Has "time to live" (ttl) property, which states maximum number of "hops" packet can take
    - Once maximum number of hops is reached, the receiving device (if it is not the intended receiver) sends the packet back with a "time exceeded" message
    - Used in traceroute command

Structure of domain names

- **Hierarchical** – similar to file system
  - Each text section separated by a dot forms a part of this hierarchy
  - Hierarchy forms a tree
- Example: www.sydney.edu.au›
- "Dot" represents "root" of the tree (not included in domain name)
- Below root is **top-level domain name,** Each "level" represents a "nameserver" that contains information about the level below
  - Used to represent country of registration, e.g. Australia
  - Separates each level into various purposes, e.g. ".com" represents commercial, ".edu" – educational

How does a computer get IP and DNS information?

- **Dynamic Host Configuration Protocol (DHCP)** – commonly used protocol to dynamically assign IP addresses in a LAN based on which IP addresses are currently free
- When a computer connects to a network, it sends a broadcast packet that requests information from a DHCP server, receiving the computer's IP address and what DNS server to use

Within domain names, each "level" has its own authority which manages the namespace

- E.g. sydney.edu.au
  - ".au" domains are country code top level domains (TLD) managed by auDA
  - ".edu" Sections of a domain level authority can become "delegated subzones" to be managed by education Australia
  - "Sydney" finally all Sydney.edu.au websites can be edited, created and managed by Sydney.edu.au (USYD)

Application Protocol

- Recap: **application layer** protocols are highest layers of Internet Protocol Suite
  - Dictates the standards in how individual applications can communicate with each other
    - email software
    - web pages and websites
    - text/video chat services
- Examples:
  - **HyperText Transfer Protocol (HTTP)** – for transmitting web pages
  - **Simple Mail Transfer Protocol (SMTP)** – for sending emails
- These protocols are generally *text-based*
  - Anyone, including yourself, could implement an application layer protocol

HTTP

- When a client wants to fetch a web page, it sends a GET request to the web server, specifically to port 80
- Web server looks in database of web pages and fetches the web page the client wants
- Web server sends the result of the request back, and if successful, the HTML code of the desired webpage

SMTP

- Client determines mail server IP from the DNS MX record (indicating **m**ail **ex**change)
- Connection made to mail server
- Server sends initial status message
- Client responds with request to fetch mail items
- Server responds with mail items

SMTP

- General standard for email messages is **Multipurpose Internet Mail Extensions (MIME)**
- Email follows specific structure as shown:
- MIME can not only carry text data, but other information as well (such as file attachments)

Three key things we want to have in network connections

- **Confidentiality**
  - Interception protection
  - "Message is only readable by the intended party"
- **Integrity**
  - Ensure data has not been tampered with
  - "Message is the same one that was sent"
- **Authenticity**
  - Ensuring we know who the sender is and cannot dispute it is actually them
  - "We know who sent the message"

Basics of sending encrypted message

1. Encrypting message with the encryption function E to produce cyphertext: E(m)="wy@523hDua#9sh" Original = "Hi Bob!"
2. Sending the cyphertext over the internet to ones E(m)="wy@523hDua#9sh"
3. By using the decryption function D to decrypt the cyphertext received: D(E(m))="Hi Bob!"

This theoretically stops messages from being read by an interceptor

- But is there a way Carol can somehow find out the original message?

Sending an encrypted message with a key

1. Alice has a message m she intends to send to Bob m="Hi Bob!"
2. Alice encrypts the message with the encryption function E, using the shared secret key k, to produce cyphertext: E(m,k)="hdd&#hduw)38"
3. Alice sends the cyphertext over the internet to Bob E(m,k)="hdd&#hduw)38"
4. Bob uses the decryption function D and the shared secret key k to decrypt the cyphertext Alice sent him: D(E(m),k)="Hi Bob!"
5. Bob is now able to read the original message m from Alice

Integrity and Public Key Cryptography

- **Hashing**
  - One-way function which calculates checksum representing a **hash** or **message digest** for some data → Can't get original data back from hash
  - A good hash function should have unique values for any input data
    - i.e. it should be very difficult for H(m)=H(n)
  - Example: **SHA-256**
  - If the hash of Bob's message from Alice does not match the hash Alice previously sent, it can be said the massage has been changed or tampered with during transmission
- **Salting**
  - A **salt** is a random string of characters that is generated to go along with hashed data
  - This salt can then be used as part of the hashing function
  - This means that even if a password for 2 different users was the same, if the *salt* for each user is different, the hash is still unique

    Thus, salts prevent:
    - attackers from being able to derive the hashing function from password hashes
    - using a user's compromised hashed password in other locations

    Salt is usually stored along with hashed password

Digital Signatures

- If our cryptography functions D and E can operate in any order,
  - i.e. $D(E(m, k_{pub}), k_{pri}) = m = E(D(m, k_{pri}), k_{pub})$,
- then we can use these to **sign** a message
  - Signing a message involves the sending party sending a **signature** of the message, h, generated by $h = D(m, k_{pri,sender})$
  - The recipient can then check that h is equal to m by checking whether this occurs $m = E(h, k_{pub,sender})$
  - This effectively verifies Alice as the sender, as only Alice has access to her private key

- Public key cryptography is much slower than using shared secret cryptography
- However, we can utilise the best of both methods in order to send data confidentially
- Alice can generate a shared secret key to use for communication with Alice and Bob, and send this key to Bob encrypted with Bob's public key, and signed with Alice's private key
  - Whilst anyone can see Alice and Bob's public keys, the decryption and signature verification remains confidential, as private keys were used in the signing
- Once Bob receives and decrypts/verifies the shared secret key sent by Alice, they can use this key in symmetric encryption to send data

Digital Certificate

- A **digital certificate** is a data structure that certifies a public key is owned by a specific person/organisation
- Certificates are managed by **certificate authorities (CAs)** – trusted organisations which issue certificates
  - Certificates are signed by the private key of the CA, and can be checked using the CA's public key
- Example: HTTPS
  - When a URL beginning with https is requested, the web server sends a public key to the browser along with the signed certificate
  - The certificate is checked with the public key of the CA to verify its authenticity
  - Thus, the public key sent by the web server can be trusted

Cloud Computing

- What is "the cloud"?
  - Using computing infrastructure that's not yours in order store, manage and process data
  - Used rather than relying on a personal computer or local server in an organisation
- Largest Providers include:
  - Amazon Web Services (AWS)
  - Microsoft Azure
  - IBM Cloud
  - Google Cloud
- Physical servers for large organisations are distributed around the world →
  - Closer to users – reduced latency à faster/less-delayed connections
  - Placed in locations with cheap and/or environmentally-friendly power
  - Users of providers can request access to resources that match their needs
    - CPU power
    - RAM
    - Storage
  - Provider then "spins up" virtualised environment for user
    - Can be virtual machine or container

Cloud Computing Service Available

- **Virtual Machines** – have a virtual server set up to use as you need
- **Storage services**
  - Simple storage, e.g. AWS S3 – store any sort of object
  - Backup/archival services – cheaper, but may take a lot of time to retrieve an object required
- **Notification services**
- **Email services** e.g. pre-configured SMTP servers – which we will explore in the lab content!
- **API gateway**
- **Serverless computing**
  - Rather than continuously running a server to wait for some action to occur and then perform something in response, users provide any sort of **programming function** that can be called when needed
  - Function can be called in response to some action (e.g. user performs some transaction)
  - Function can call other functions too
  - When the function is called, a VM/container is quickly generated, executes the function, and is shut down
  - Many languages supported

Window Server

- Program which holds information about elements on the screen as data structures
- Includes:
  - windows
  - window elements (such as scroll bars, close/minimise/maximise buttons)
  - other parts of a user interface

Window server can be broken down into sub-components or programs

- **Window manager** is responsible specifically for displaying windows on the screen
- **Desktop environment** provides overall look-and-feel of interface
  - OSs such as Linux provide a variety of desktop environments

Each program in a window acts as a **client** of the server

- Server communicates with client to dictate styling of common elements (e.g. buttons, scroll bars, etc.)

Android – how does that relate to Unix?

- Kernel android uses is based upon Linux kernel (95% similar)
- Uses its own window server to display programs as **apps**
- Each manufacturer needs to provide its own drivers so that the phone's hardware can interact with the common Android software