Linux Curriculum

Linux and Unix Distros

- GNU/Linux vs BSD
    - Linux: created by Linus Torvals
        - Just a kernel
        - Not an operating system
    - GNU: created by Richard Stallman
    - BSD: created by University of California Berkeley
    - GPL vs. BSD License
        - GPL (Linux): If you make a change to the source code and want to distribute it, you have to release your source code. Basically, if you use our code to make more code, you have to make your code open source too.
        - BSD (BSD): if you make a change to the source code and distribute it, you can do whatever you want. MacOS is based on FreeBSD. MacOS is not open source. Windows has even taken some code from BSD.
- Linux Family Tree
- Debian
    - Ubuntu
        - Kubuntu
        - Pop OS
- RedHat
    - CentOS
    - Fedora
    - RedHat
- Arch
- Gentoo
- Slackware
- OpenSUSE
- FreeBSD
- OpenBSD

Installing our Development Environment

- Introduction to VirtualBox and virtualization
- Installing VirtualBox
- Installing a Linux VM
- Installing VirtualBox guest additions

The File System

- Everything is a file!
- File system basics

    - Namespace: a way to name things and organize them in a hierarchy
    - API: set system calls for navigating and manipulating objects
    - Security Models: schemes for protecting, hiding, and sharing things
    - An implementation: software to tie the logical model to the hardware
- Different file systems
    - Ext4
    - XFS
    - UFS
    - ZFS
    - Btrfs
    - NFS
    - FAT32
    - NTFS
- Directory Hierarchy

Pathname	Contents
/bin	Core operating system commands
/boot	Boot loader, kernel, and files needed by the kernel
/compat	On FreeBSD, files and libraries for Linux binary compatibility
/dev	Device entries for disks, printers, pseudo-terminals, etc.
/etc	Critical startup and configuration files
/home	Default home directories for users
/lib	Libraries, shared libraries, and commands used by **/bin** and **/sbin**
/media	Mount points for filesystems on removable media
/mnt	Temporary mount points, mounts for removable media
/opt	Optional software packages (rarely used, for compatibility)
/proc	Information about all running processes
/root	Home directory of the superuser (sometimes just /)
/run	Rendezvous points for running programs (PIDs, sockets, etc.)
/sbin	Core operating system commands[a]
/srv	Files held for distribution through web or other servers
/sys	A plethora of different kernel interfaces (Linux)
/tmp	Temporary files that may disappear between reboots
/usr	Hierarchy of secondary files and commands
/usr/bin	Most commands and executable files
/usr/include	Header files for compiling C programs
/usr/lib	Libraries; also, support files for standard programs
/usr/local	Local software or configuration data; mirrors **/usr**
/usr/sbin	Less essential commands for administration and repair
/usr/share	Items that might be common to multiple systems
/usr/share/man	On-line manual pages
/usr/src	Source code for nonlocal software (not widely used)
/usr/tmp	More temporary space (preserved between reboots)
/var	System-specific data and a few configuration files
/var/adm	Varies: logs, setup records, strange administrative bits
/var/log	System log files
/var/run	Same function as **/run**; now often a symlink
/var/spool	Spooling (that is, storage) directories for printers, mail, etc.
/var/tmp	More temporary space (preserved between reboots)

a. The distinguishing characteristic of **/sbin** was originally that its contents were statically linked and so had fewer dependencies on other parts of the system. These days, all binaries are dynamically linked and there is no real difference between **/bin** and **/sbin**.

- Relative path names vs absolute path names
- File permissions
    - Owner
        - Read: 4
        - Write: 2
        - Execute: 1
    - Group
        - Read: 4
        - Write: 2
        - Execute: 1
    - Everyone else (world)
        - Read: 4
        - Write: 2
        - Execute: 1
    - H**ugo** -> (u, g, o)
    - Setuid bit
    - Sgid bit
    - Sticky bit
    - umask
- File ownership
    - Owner and group attached to a file
- Commands
    - Navigating the file system
        - Cd
        - Ls
        - Pwd
        - Find
    - Interacting with files
        - Cat
        - Echo
        - Rm
            - -r
            - -f (CAREFUL)
        - Cp
        - Touch
        - Mv
        - Chmod
            - Change by numbers
            - Change by a combination of u, g, o and +, -, or =
            - -R
        - Chown
            - Chown user:group
            - -R
        - Diff

        - Wget
        - curl
        - Vim
            - Insert mode
            - Command mode
            - Exiting

Users and Groups

- Root: the user with all the power
- Important Files and Structure of files
    - /etc/passwd
        - Format
            - Login name
            - Encrypted password placeholder
            - UID (user ID)
            - Default GID (group ID)
            - Optional “GECOS” information (full name, office, extension, etc)
            - Home directory
            - Login shell
    - /etc/shadow
        - Can only be accessed by a superuser
        - Format
            - Login name
            - Encrypted password
            - Date of last password change
            - Minimum number of days between password changes
            - Maximum number of days between password changes
            - Number of days in advance to warn users about password expiration
            - Days after password expiration that account is disabled
            - Account expiration date
            - A field reserved for future use which is currently always empty
    - /etc/group
        - Format
            - Group name
            - Encrypted password or a placeholder
            - GID number
            - List of members, separated by commas (be careful not to add spaces)
- Manually adding a new user
    - Edit the passwd file
    - Edit the shadow file
    - Add user to /etc/group

    - Set an initial password
    - Create, chown, and chmod the user's home directory
- Commands
    - whoami
    - Sudo:
        - limited su
        - Runs command as root
        - Consults the /etc/sudoers file
    - Su:
        - switch user
    - Groupadd
        - Add a group
        - Available in all Linux systems
    - Addgroup
        - Wrapper around groupadd
        - Available in Debian systems
    - Adduser
        - A bit more interactive
        - Available on Debian systems
    - Useradd
        - Have to specify all options with command-line options
        - Available on all Linux systems
    - Userdel
        - Available in all Linux systems
    - Deluser
        - Available on Debian systems
    - Usermod
        - -L: lock
        - -U: unlock
        - -aG: add group
        - Locking and unlocking
            - Changes /etc/shadow file
            - May not work for SSH
            - Try setting shell to /usr/sbin/nologin (Debian) or /sbin/nologin (Redhat)

Package Managers and Managing Software

- How they work
    - Sources list
- Different package managers
    - Yum
        - The old package manager for RedHat. DNF is far better
    - Dnf

        - Dnf --version
        - Dnf list all or dnf list <package>
            - --recent lists all recently added packages
        - Dnf check-update or dnf check-update <package>
            - --changelog shows changelog delta
        - Dnf search <package-name>
        - Dnf repolist all
        - Dnf update == dnf upgrade but dnf upgrade is preferred
        - Dnf install <package>
        - Dnf remove <package> removes all versions of that package. To remove a specific package, specify it
    - Apt
        - Takes the best from apt-get and apt-cache and makes it easier to use
    - Apt-get & apt-cache
        - More functionality but also less “nice” to use
- Rpm
- Dpkg
- Other commands
    - which

Processes

- Descriptions
    - Represents a running program
    - Abstraction through which memory, processor time, and I/O resources can be managed and monitored
- Processes basics
    - Components of a process
        - Consists of an address space and a set of data structures within the kernel
            - Address space is a set of memory pages that the kernel has marked for the process’s use
                - Memory pages are units in which memory is managed typically 4 KiB or 8 KiB in size
        - Memory pages of a process contain:
            - Code and libraries that the process is executing
            - Process’s variables
            - Process’s stacks
            - Various other information needed by the kernel while the process is running
        - Important information held in a process (not exhaustive)
            - Process’s address space map
            - Current status of the process (sleeping, stopped, runnable, etc)
            - Execution priority of the process

            - Information about the resources the process has used (CPU, memory, etc)
            - Information about the files and network ports the process has opened
            - Process's signal mask (a record of which signals are blocked)
            - Owner of the process
        - Thread: execution context within a process
            - Process can have one thread or many threads
        - PID: process ID number
            - Unique number for every process
            - Need the ID to modify a process
        - PPID: parent PID
            - New processes come from two steps:
                - Existing process cloning itself (fork)
                - Cloned process changes program running inside of it
            - Existing process = parent; clone = child
            - All processes are created from init or systemd (PID: 1)
        - UID: user who created the process
        - GID: group ID who owns the process
        - Signals
            - Process-level interruptions
                - 9: KILL
- /proc
    - Presents processes like a file system you can interact with
    - Man proc
    - Some of the process information

File	Contents
cgroup	The control groups to which the process belongs
cmd	Command or program the process is executing
cmdline [a]	Complete command line of the process (null-separated)
cwd	Symbolic link to the process's current directory
environ	The process's environment variables (null-separated)
exe	Symbolic link to the file being executed
fd	Subdirectory containing links for each open file descriptor
fdinfo	Subdirectory containing further info for each open file descriptor
maps	Memory mapping information (shared segments, libraries, etc.)
ns	Subdirectory with links to each namespace used by the process.
root	Symbolic link to the process's root directory (set with **chroot**)
stat	General process status information (best decoded with **ps**)
statm	Memory usage information

a. Might be unavailable if the process is swapped out of memory

- Commands
    - Top/htop
    - Ps
        - Ps -A
            - Display every active process on a Linux system in generic format
        - Ps aux
            - Display all processes in BSD format
        - Ps -fU <user>
            - Display all processes by a user
        - Ps -fp <PID>
            - Display process by PID
        - Ps -f --ppid <PPID>
            - Display proces by PPID
        - Ps -e --forest
            - Display process tree
    - Lsof
        - Lsof
            - List all open files
        - Lsof -u <user>
            - List all opened files by user
        - Lsof -i TCP:<port>
            - Find processes running that are using the specified port
            - Can also specify port range
        - Lsof -p 1
    - Kill
        - Kill -9 <PID>
            - Kill process with PID

Services

- Init's job
    - To start every other process at boot
    - Systemd one implementation of an init system
    - System V is another implementation
- Systemd
    - Why people hate it
        - UNIX philosophy: do one thing and do it really well
            - This is because of security and complexity reasons
        - Systemd aims to do a lot of things
            - Manage processes in parallel (systemd)
            - Manage network connections (networkd)
            - Manage log entries (journald)
            - Manage logins (logind)

- What is it?
  - Collection of programs daemons, libraries, technologies, and kernel components
  - 69 different programs in one
  - Think of it like a buffet where you are forced to eat everything
- How systemd works in general
  - Units and unit files: any entity managed by systemd is a unit
    - Defined in a configuration file (unit file)
    - A Unit file for a service specifies:
      - Location of executable file for the daemon
      - Tells systemd how to start and stop the service
      - Identifies any other units the service depends on
  - Unit file directories
    - /usr/lib/systemd/system: main place where packages deposit their unit files during installation
      - Sometimes /lib/systemd/system instead
      - Shouldn't modify these as they are considered stock
    - /etc/systemd/system: local unit files and customizations
    - Files in /etc have highest priority

- Systemctl
  - Systemctl
    - Lists all loaded and active services, sockets, targets, mounts, and devices
  - Systemctl list-units --type=service
    - List all service units
  - Systemctl list-unit-files --type=service
    - List all unit files regardless of whether or not they're active
  - Systemctl status <unit>
    - Show status of unit
  - Most popular subcommands of systemctl

Subcommand	Function
list-unit-files [*pattern*]	Shows installed units; optionally matching *pattern*
enable *unit*	Enables *unit* to activate at boot
disable *unit*	Prevents *unit* from activating at boot
isolate *target*	Changes operating mode to *target*
start *unit*	Activates *unit* immediately
stop *unit*	Deactivates *unit* immediately
restart *unit*	Restarts (or starts, if not running) *unit* immediately
status *unit*	Shows *unit*'s status and recent log entries
kill *pattern*	Sends a signal to units matching *pattern*
reboot	Reboots the computer
daemon-reload	Reloads unit files and **systemd** configuration

- Unit File statuses

State	Meaning
bad	Some kind of problem within **systemd**; usually a bad unit file
disabled	Present, but not configured to start autonomously
enabled	Installed and runnable; will start autonomously
indirect	Disabled, but has peers in `Also` clauses that may be enabled
linked	Unit file available through a symlink
masked	Banished from the **systemd** world from a logical perspective
static	Depended upon by another unit; has no install requirements

- Service
    - Still a usable command on Fedora but I won't cover it

Networking on Linux

- Debian network setting
    - /etc/hostname: set the hostname
    - /etc/network/interfaces: set the ip address, netmast, and default route

```
auto lo enp0s5
iface lo inet loopback
iface enp0s5 inet static
    address 192.168.1.102
    netmask 255.255.255.0
    gateway 192.168.1.254
```

        - Inet = IPv4
            - IPv6 is inet6
        - Static: called a method (method for getting an IP)
            - Static means the method for getting an IP is being directly assigned
        - Address, netmask, and gateway are all required for a static setting
        - DHCP config

```
iface enp0s5 inet dhcp
```

        - Ifup and ifdown commands read this file and bring the interfaces up or down by calling lower-level commands (like ip)
- Redhat
    - /etc/sysconfig/network: Hostname, default route

```
NETWORKING=yes
NETWORKING_IPV6=no
HOSTNAME=redhat.toadranch.com
DOMAINNAME=toadranch.com                ### optional
GATEWAY=192.168.1.254
```

- /etc/sysconfig/network-scripts/ifcfg-<ifname>: Per-interface parameters: IP address, netmast, etc
  - Static IP

```
DEVICE=eth0
IPADDR=192.168.1.13
NETMASK=255.255.255.0
NETWORK=192.168.1.0
BROADCAST=192.168.1.255
MTU=1500
ONBOOT=yes
```

  - DHCP

```
DEVICE=eth0
BOOTPROTO=dhcp
ONBOOT=yes
```

    - After changing something in here, run ifdown <interface-name> followed by ifup <interface-name>
      - Or you can restart every interface with sysctl restart network
- /etc/sysconfig/network-scripts/route-<ifname>: Per-interface routing; arguments to ip route

- Command ifconfig is replaced by ip
- Commands
  - Ip
    - Ip link
      - Show all network devices
    - Ip address (ip a)
      - Show network devices and addresses
    - Ip route
      - Show routing table

        - Man ip-link
            - Show man page for ip link command
        - Man ip-route
            - Show man page for ip route command
        - Ip <subcommand> help
            - Show help message for ip command
        - Ip address add 192.168.1.13/26 broadcast 192.168.1.63 dev enp0s5
            - Set IP address for network device
    - Traceroute
        - Sudo dnf install traceroute -y
        - Traceroute <destination>
    - Tcpdump
        - Tcpdump -i <interface> [port <port>] [-vv]
    - Wireshark & tshark
- Local dns config
    - /etc/resolv.conf

SSH

- Terminal usage
- SSH config file
    - Securing SSH
        - PasswordAuthentication
        - PermitRootLogin
        - Port
        - PermitEmptyPasswords
        - AllowGroups
        - DenyGroups
        - AllowUsers <user1> <user2> …
            - Takes precedence over AllowGroups
        - DenyUsers
        - *The allow/deny directives are processed in the following order: DenyUsers, AllowUsers, DenyGroups, and finally AllowGroups*
- Sshd_config vs ssh_config
    - Sshd_config = SSH daemon config (server process)
    - Ssh_config = SSH client config (used when you ssh to another server)
- Using SSH keys
    - Ssh-keygen
    - Ssh-copy-id -i <public_key> <user>@<remote_host>
- Using an SSH config file
    - Example

```
Host server_name
    Hostname <IP or DNS name>
    User <username>
    Port <SSH-Port>
    IdentityFile path/to/private/key
```

- SCP
    - Scp user@source:/path/to/file/from/source user@target:/path/to/file/from/target
    - Scp -i /path/to/key user@source:/path/to/file/from/source user@target:/path/to/file/from/target
    - Scp -p portuser@source:/path/to/file/from/source user@target:/path/to/file/from/target:qw

Advanced Terminal Usage

- Piping
- Directing output
    - Stdout
    - Stderr
- Refining terminal output
    - Sort
    - Wc
    - Cut
    - Uniq
    - Tee
    - Grep
        - Grep “literal_string” FILE
        - Grep “string” filename*
        - Grep -i “sTrInG” FILE
        - Grep “^string beginning of line” FILE
        - Grep -w “get full word” FILE
        - Grep -r “search for this recursively” FILE
        - Grep -v “display lines that don’t match” FILE
        - Grep -c “count number of matches” FILE
    - Head
    - Tail
    - Sed (stream editor)
        - Sed ‘s/old-text/new-text’ FILENAME
            - Replace first occurrence of old-text
        - Sed ‘s/old-text/new-text/N’ FILENAME
            - Changes the Nth occurrence of old-text
        - Sed ‘s/old-text/new-text/g’ FILENAME
            - Replace all occurrences of old-text

        - Sed ‘s/old-text/new-text/Ng’ FILENAME
            - Replace from Nth occurrence to end of the file of old-text
    - Awk (Aho, Weinberger, and Kernighan) <- developer names
        - Syntax: **awk options 'selection _criteria {action }' input-file > output-file**
        - Example File

```
ajay manager account 45000
sunil clerk account 25000
varun manager sales 50000
amit manager account 47000
tarun peon sales 15000
deepak clerk sales 23000
sunil peon sales 13000
satvik director purchase 80000
```

            - awk '/manager/ {print}' employee.txt
            - awk '{print $1,$4}' employee.txt
            - awk '{print NR,$0}' employee.txt
            - awk '{print $1,$NF}' employee.txt
            - awk 'NR==3, NR==6 {print NR,$0}' employee.txt
        - Built-in AWK variables
            - NR: count of input records (usually lines in a file)
            - NF: count of number of fields in current input record
            - FS: **field** separator character which is used to divide fields on the input line
            - RS: current **record** separator character. Default is the newline character
            - OFS: output fields separator which is the character that separates **fields** when they are outputted
            - ORD: output record separator which separates the output **lines** when outputted

Bash Scripting

- .bashrc
- Setting up your first script
    - You can include bash commands in a bash script straight up
    - .sh file name extension
    - Making it executable
- Variables
    - Assignment: variable=”value”

        - No space in between
    - Referencing
    - Get variable value: $variable
    - Get literal string of variable name: '$variable'
    - Get value of variable as a string: "$variable"

```
dave@howtogeek:~$ echo '$my_name'
$my_name
dave@howtogeek:~$ echo "$my_name"
dave
dave@howtogeek:~$ echo my_name
my_name
dave@howtogeek:~$ echo $my_name
dave
dave@howtogeek:~$
```

    - Special Variables
        - $#: How many command line parameters were passed to the script.
        - $@: All the command line parameters passed to the script.
        - $?: The exit status of the last process to run.
        - $$: The Process ID (PID) of the current script.
        - $USER: The username of the user executing the script.
        - $HOSTNAME: The hostname of the computer running the script.
        - $SECONDS: The number of seconds the script has been running for.
        - $RANDOM: Returns a random number.
        - $LINENO: Returns the current line number of the script.
- Parameter expansion
    - While getting the value of a variable, you can modify it before it gets printed. These parameter expansions allow you to get the length of a variable, certain element of an array, substitute text in the variable, and others
    - Get length of variable: ${#variable}

```
attack@user-VirtualBox-ubuntu:/home/user$ sentence="This is a sentence"
attack@user-VirtualBox-ubuntu:/home/user$ echo ${#sentence}
18
```

    - Get element of array: ${array[<index>]}

```
attack@user-VirtualBox-ubuntu:/home/user$ array0=(one two three four five six)
attack@user-VirtualBox-ubuntu:/home/user$ echo ${array0[0]}
one
attack@user-VirtualBox-ubuntu:/home/user$ echo ${array0[1]}
two
```

    - Substitute text: ${variable/<old_text>/<new_text>}

```
attack@user-VirtualBox-ubuntu:/home/user$ var="Jake Brown"
attack@user-VirtualBox-ubuntu:/home/user$ echo ${var/J/j}
jake Brown
```

- Command line parameters

    - $1 = first parameter
    - $2 = second parameter
    - And so on
- Input
    - Read input from terminal
      echo -n "Proceed? [y/n]: "
      read ans
      echo $ans
    - Read only one character
      read -n 1 ans
- Conditionals
    - Double parenthesis such as **((...))** are used to test an arithmetic expression
    - Double square brackets such as **[[...]]** is the new way to evaluate expressions. It can handle more complex conditions and is less error prone

`[[ -z STRING ]]`	Empty string
`[[ -n STRING ]]`	Not empty string
`[[ STRING == STRING ]]`	Equal
`[[ STRING != STRING ]]`	Not Equal
`[[ NUM -eq NUM ]]`	Equal
`[[ NUM -ne NUM ]]`	Not equal
`[[ NUM -lt NUM ]]`	Less than
`[[ NUM -le NUM ]]`	Less than or equal
`[[ NUM -gt NUM ]]`	Greater than
`[[ NUM -ge NUM ]]`	Greater than or equal
`[[ STRING =~ STRING ]]`	Regexp
`(( NUM < NUM ))`	Numeric conditions

`[[ -o noclobber ]]`	If OPTIONNAME is enabled
`[[ ! EXPR ]]`	Not
`[[ X && Y ]]`	And
`[[ X || Y ]]`	Or
`[[ -e FILE ]]`	Exists
`[[ -r FILE ]]`	Readable
`[[ -h FILE ]]`	Symlink
`[[ -d FILE ]]`	Directory
`[[ -w FILE ]]`	Writable
`[[ -s FILE ]]`	Size is > 0 bytes
`[[ -f FILE ]]`	File
`[[ -x FILE ]]`	Executable
`[[ FILE1 -nt FILE2 ]]`	1 is more recent than 2
`[[ FILE1 -ot FILE2 ]]`	2 is more recent than 1
`[[ FILE1 -ef FILE2 ]]`	Same files

- Single square brackets such as **[...]** is a call to the command **test**
  - View the man page for **test** to see all of the options. Below are some

Operator	Description
! EXPRESSION	The EXPRESSION is false.
-n STRING	The length of STRING is greater than zero.
-z STRING	The lengh of STRING is zero (ie it is empty).
STRING1 = STRING2	STRING1 is equal to STRING2
STRING1 != STRING2	STRING1 is not equal to STRING2
INTEGER1 -eq INTEGER2	INTEGER1 is numerically equal to INTEGER2
INTEGER1 -gt INTEGER2	INTEGER1 is numerically greater than INTEGER2
INTEGER1 -lt INTEGER2	INTEGER1 is numerically less than INTEGER2
-d FILE	FILE exists and is a directory.
-e FILE	FILE exists.
-r FILE	FILE exists and the read permission is granted.
-s FILE	FILE exists and it's size is greater than zero (ie. it is not empty).
-w FILE	FILE exists and the write permission is granted.
-x FILE	FILE exists and the execute permission is granted.

        - Few things to note
            - = is different from -eq. = does string comparison where -eq does numerical comparison
            - FILE above refers to the path (can be absolute or relative)
            - You can experiment with test expressions on the command line
    - Else and elif function how they usually do
    - Boolean Operators:
        - && (and)
        - || (or)
    - Case statements

```
#!/bin/bash
# case example

case $1 in
   start)
      echo starting
      ;;
   stop)
      echo stoping
      ;;
   restart)
      echo restarting
      ;;
   *)
      echo don\'t know
      ;;
esac
```

- Loops
  - Basic for loop

```
for i in /etc/rc.*; do
  echo $i
done
```

  - C-like for loop

```
for ((i = 0 ; i < 100 ; i++)); do
  echo $i
done
```

  - Ranges
    - Without step size

```
for i in {1..5}; do
    echo "Welcome $i"
done
```

    - With step size

```
for i in {5..50..5}; do
    echo "Welcome $i"
done
```

  - Reading lines

```
cat file.txt | while read line; do
  echo $line
done
```

- Never ending for loop

```
while true; do
  ...
done
```

- Functions
  - Defining functions

```
myfunc() {
    echo "hello $1"
}
```

```
# Same as above (alternate syntax)
function myfunc() {
    echo "hello $1"
}
```

```
myfunc "John"
```

  - Returning values

```
myfunc() {
    local myresult='some value'
    echo $myresult
}
```

```
result="$(myfunc)"
```

  - Raising errors

```
myfunc() {
  return 1
}
```

```
if myfunc; then
  echo "success"
else
  echo "failure"
fi
```

- Arguments

`$#`	Number of arguments
`$*`	All arguments
`$@`	All arguments, starting from first
`$1`	First argument
`$_`	Last argument of the previous command

See Special parameters.

- Arrays
  - Defining Arrays

```
Fruits=('Apple' 'Banana' 'Orange')
```

```
Fruits[0]="Apple"
Fruits[1]="Banana"
Fruits[2]="Orange"
```

  - Operations

```
Fruits=("${Fruits[@]}" "Watermelon")    # Push
Fruits+=('Watermelon')                  # Also Push
Fruits=( ${Fruits[@]/Ap*/} )            # Remove by regex match
unset Fruits[2]                         # Remove one item
Fruits=("${Fruits[@]}")                 # Duplicate
Fruits=("${Fruits[@]}" "${Veggies[@]}") # Concatenate
lines=(`cat "logfile"`)                 # Read from file
```

  - Working with arrays

```
echo ${Fruits[0]}           # Element #0
echo ${Fruits[-1]}          # Last element
echo ${Fruits[@]}           # All elements, space-separated
echo ${#Fruits[@]}          # Number of elements
echo ${#Fruits}             # String length of the 1st element
echo ${#Fruits[3]}          # String length of the Nth element
echo ${Fruits[@]:3:2}       # Range (from position 3, length 2)
echo ${!Fruits[@]}          # Keys of all elements, space-separated
```

  - Iteration

```
for i in "${arrayName[@]}"; do
  echo $i
done
```

- Dictionaries
  - Defining

```
declare -A sounds
```

```
sounds[dog]="bark"
sounds[cow]="moo"
sounds[bird]="tweet"
sounds[wolf]="howl"
```

Declares `sound` as a Dictionary object (aka associative array).

  - Working with dictionaries

```
echo ${sounds[dog]} # Dog's sound
echo ${sounds[@]}   # All values
echo ${!sounds[@]}  # All keys
echo ${#sounds[@]}  # Number of elements
unset sounds[dog]   # Delete dog
```

  - Iteration

Iterate over values

```
for val in "${sounds[@]}"; do
  echo $val
done
```

Iterate over keys

```
for key in "${!sounds[@]}"; do
  echo $key
done
```

- Options
  - Options are used to change the way the script behaves as a whole
  - Below are some basic options

```
set -o noclobber  # Avoid overlay files (echo "hi" > foo)
set -o errexit    # Used to exit upon error, avoiding cascading errors
set -o pipefail   # Unveils hidden failures
set -o nounset    # Exposes unset variables
```

Introduction to Server Software

- NGINX
    - Important directories
        - /var/www/html: Default location for web content
        - /etc/nginx: All NGINX configuration files live here
        - /etc/nginx/nginx.conf: Main NGINX configuration file. Changes here modify the NGINX global configuration
        - /etc/nginx/sites-available: Per-site “server blocks”. These files need to be linked to sites-enabled to be run
        - /etc/nginx/sites-enabled: Links to sites-available config files which enable the respective configuration file
        - /var/log/nginx/access.log: All web requests are logged to this file
        - /var/log/nginx/error.log: All NGINX errors are logged to this file
- Apache
    - Important directories in Debian-based systems
        - /var/www/html: Default location for web content
        - /etc/apache2: All Apache configuration files live here
        - /etc/apache2/apache2.conf: Main configuration file. It contains global configurations and is responsible for loading other configuration files
        - /etc/apache2/ports.conf: Specifies the ports that Apache will listen on
        - /etc/apache2/sites-available/: Stores the site-specific configuration files for each virtual host
        - /etc/apache2/sites-enabled/: Links to sites-available configuration files which enable the respective configuration file
        - /var/log/apache2/access.log: All web requests are logged to this file
        - /var/log/apache2/error.log: All Apache errors are logged to this file
- SMB
- BIND9
- Cronjobs
- Firewalls
    - High-level Information
        - IPTables is what lives under everything
        - Debian iptables wrapper: ufw
        - Redhat iptables wrapper: firewalld
    - Iptables
        - Chains: list of rules that apply to a function that are processed in order
            - Input: filter inward bound (ingress) traffic

        - Forward: filter traffic being forwarded to another host
        - Output: filter outbound (egress) traffic
    - Default Chain Behavior: if no rules match, what will it do?
        - Use the command iptables -L
        - Change default behavior: iptables --policy <chain> <behavior>
            - Behavior can be ACCEPT, DROP, or REJECT
                - ACCEPT: Allow the connection
                - DROP: Drop the connection and pretend it never happened. Best if you don't want the source to know you exist. This slows down software that is connecting because the software will continue to retry.
                - REJECT: Don't allow the connection and send back an error. Best if you want to block someone and make sure they know that they got blocked. Because a return message gets sent, software will not be slowed down
    - Connection States
        - Description: We can filter packets based on their state. State refers to was the connection just created? Is this packet a response to a connection that was already created? Etc
        - Different connection states
            - NEW: Packet relating to a new connection
            - ESTABLISHED: Packet that is part of an existing connection
            - RELATED: Packet that is requesting a new connection but is part of an existing connection. For example, FTP uses port 21 to establish a connection, but data is transferred on a different port (typically port 20).
            - INVALID: Packet that cannot be identified in one of the three other states. This could be caused by various types of stealth network probes, or it could mean that you're running out of CONNTRACK entries (which you should also see stated in your logs). Or it may simply be entirely benign.
        - Command examples
            - iptables -A INPUT -p tcp --dport ssh -s 10.10.10.10 -m state --state NEW,ESTABLISHED -j ACCEPT
            - iptables -A OUTPUT -p tcp --sport 22 -d 10.10.10.10 -m state --state ESTABLISHED -j ACCEPT
    - Commands
        - List currently configured iptables rules
            - iptables -L
        - Flush all the currently configured rules

        - iptables -F
    - Append rule to end of chain list
        - Iptables -A [chain] RULE
    - Insert rule in list at specified place
        - Iptables -I [chain] [number] RULE
    - Connections from a single IP address
        - iptables -A INPUT -s 10.10.10.10 -j DROP
    - Connections from a range of IP addresses
        - iptables -A INPUT -s 10.10.10.0/24 -j DROP
        - or
        - iptables -A INPUT -s 10.10.10.0/255.255.255.0 -j DROP
    - Connections to a specific port from specific host
        - iptables -A INPUT -p tcp --dport ssh -s 10.10.10.10 -j DROP
    - Connections to a specific port from anywhere
        - iptables -A INPUT -p tcp --dport ssh -j DROP
- Saving Rules
    - Ubuntu:
        - sudo /sbin/iptables-save
    - Red Hat / CentOS:
        - /sbin/service iptables save
    - Or
        - /etc/init.d/iptables save

System Resource Monitoring and Management

- CPU information
    - cat /proc/cpuinfo
    - lscpu
- RAM information
    - free -h
- Persistent storage information
    - Df
    - Du
    - Lsblk
    - mount

Logging

- Log locations
    - Most logs live in /var/log
    - SSH logs
        - RHEL: /var/log/secure

        - Debian: /var/log/auth.log
- The systemd journal
    - journalctl
- Syslog
    - RHEL: /var/log/messages
    - Debian: /var/log/syslog
- Centralized logging
    - Splunk
    - ELK

Resources

- Bash
    - Bash cheat sheet
    - Bash set command reference
- Regular expressions
    - Engineer Man YouTube video
    - Regex visualization website
- Grep
    - Grep Examples
- Sed
    - Sed Examples
- AWK
    - Why learn AWK
    - AWK Guide
- Iptables
    - Beginner's Guide
- Apache
    - Apache Guide
- NGINX
    - NGINX Guide
- Linux Books
    - UNIX and Linux System Administration Handbook 5th Edition
    - How Linux Works, 2nd Edition: What Every Superuser Should Know
    - Linux Bible 8th Edition