澳 門 理 工 學 院
Instituto Politécnico de Macau
Macao Polytechnic Institute

COMP 225
Network and System Administration

Notes #1: Linux General

K. L. Eddie Law, PhD

Macao Polytechnic Institute
School of Applied Sciences
Academic Year 2020-2021, 2nd Semester

Topics

- Computers and operating systems
- Introduction to Unix
- History of Unix
- What is Linux?
  - OS structure
  - File system
- Linux distributions
- The shell
- Users, groups, and permissions

Hardware and Software

- The history of computers
  - (timeline events) https://www.livescience.com/20718-computer-history.html
  - (timeline and pictures) https://www.computerhistory.org/timeline/computers/
- From hardware specific operating system to generic operating system platforms

History of Unix

- History - https://www.computerworld.com/article/2524555/operating-systems-timeline-40-years-of-unix.html
- First Version was created in AT&T Bell Labs in 1969
  - Bell Labs were in Lucent after splitting from AT&T
  - Lucent was acquired by Alcatel during tech bubble in early 2000's
- Some famous Bell Labs programmers worked on this project
  - Ken Thompson, Dennis Ritchie, Rudd Canaday, and Doug McIlroy designed and implemented the first version of the Unix File System on a PDP-7 along with a few utilities
  - The name Unix was given by Brian Kernighan

History of Unix (cont'd)

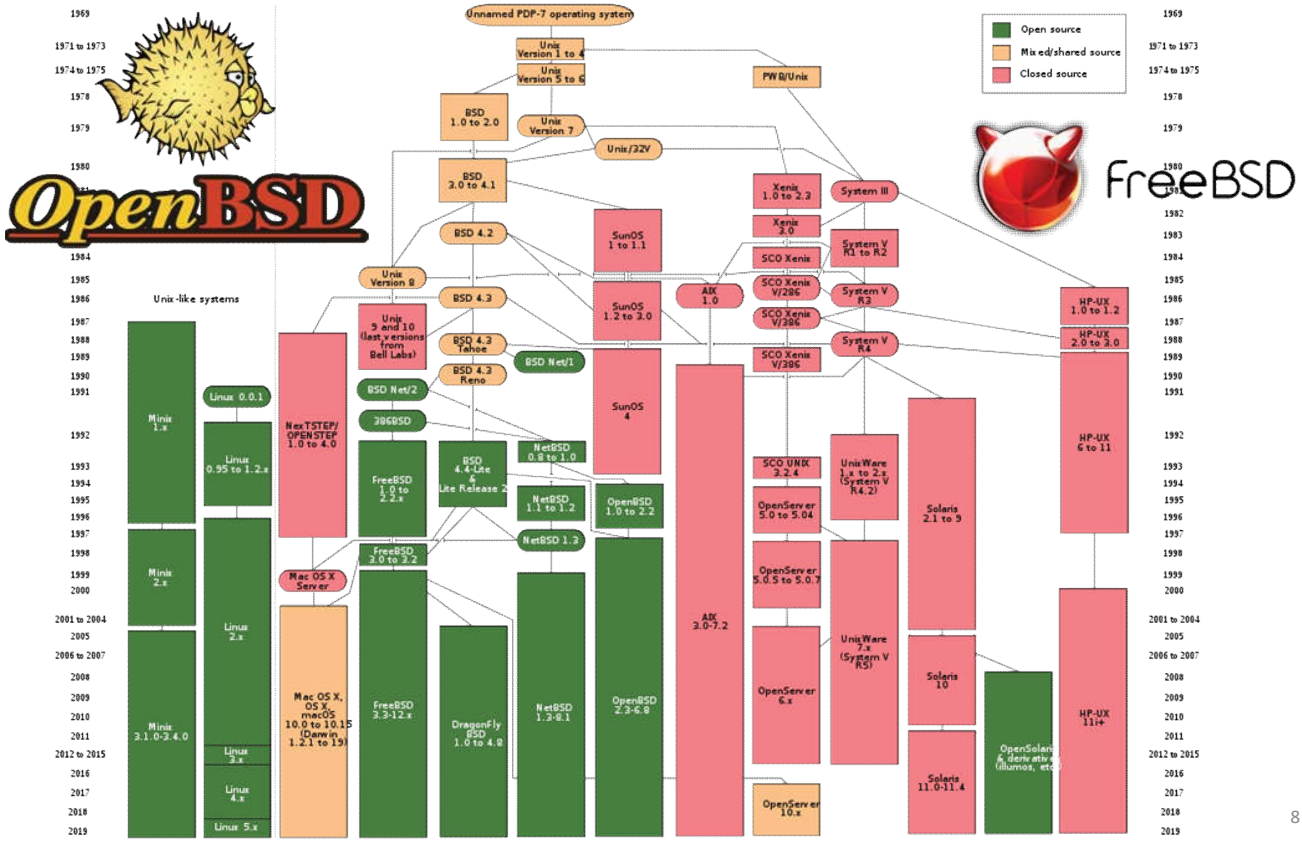
- 00:00:00 Hours, Jan 1, 1970 was the time zero for Unix, also called as an *epoch*
- 1973 Unix was re-written mostly in C, a new language developed at that time by Dennis Ritchie
- Being written in this high-level language (comparing to machine languages) greatly decreased the effort needed to port it to new machines

Introduction to Unix

- Unix is a multi-user, multi-tasking operating system
- Many users may log into a system simultaneously, each running many programs
- The kernel (OS) is responsible for keeping processes and users separated and regulating accesses to system resources, e.g., hardware (e.g., CPU), memory, disk and other I/O devices

Evolution

- 1977: There were about 500 Unix sites world-wide
- 1980: BSD 4.1 (Berkeley Software Development) at UCB
- 1983: SunOS, BSD 4.2, System V
- 1988: AT&T and Sun Microsystems jointly develop System V Release 4 (SVR4), which was later developed into UnixWare and Solaris 2
- 1991: Linux was created by Linus Torvalds



Linux

- A small Unix-like OS called MINUX was created by Andrew Tanenbaum for education purpose
- Triggered by the MINUX, Linus Torvalds, then a Finnish graduate student, started a personal project, the Linux operating system in 1991
- Kernel version 1.0 was released in 1994 and today the most recent stable version is 5.x (as of 2021)
- Developed under the GNU General Public License , the source code for Linux is freely available to everyone
- Today, Linux is maintained by Linus Torvalds with contributions from thousands of developers around the world

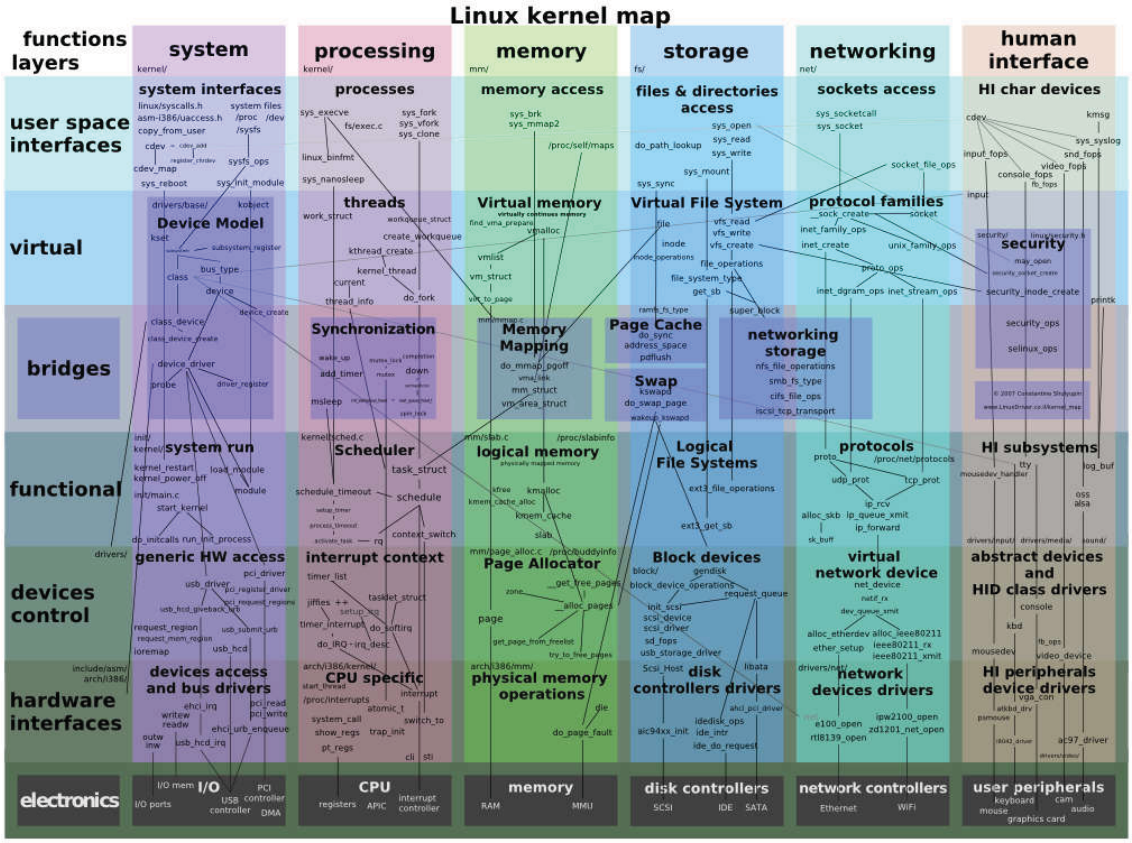
Linux Distributions



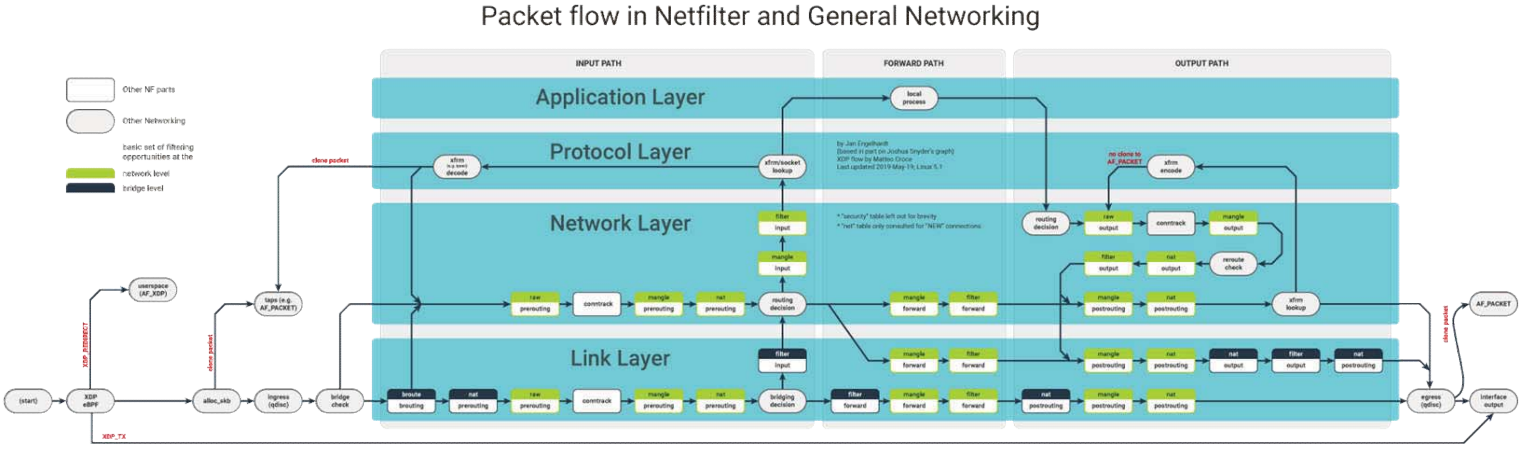



- Debian - https://www.debian.org/ (non-profitable community distribution)
- Ubuntu - https://www.ubuntu.com/ (based on Debian)
- Linux Mint - (based on Debian)
- Red Hat Enterprise Linux (RHEL) - https://www.redhat.com/
  - Red Hat owned by IBM
- Fedora - https://fedora.redhat.com/ (developed by RH)
- CentOS - https://www.centos.com/ (based on RHEL)
- SUSE - https://www.suse.com/ (based on)

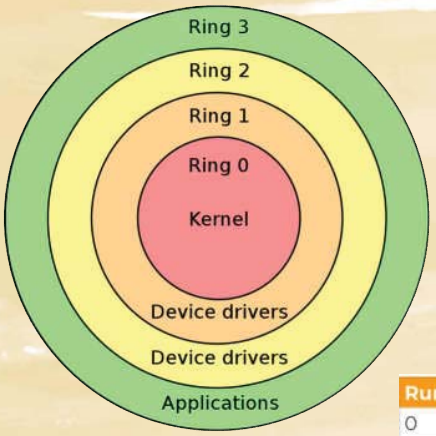

Linux kernel

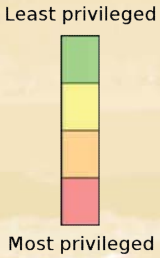


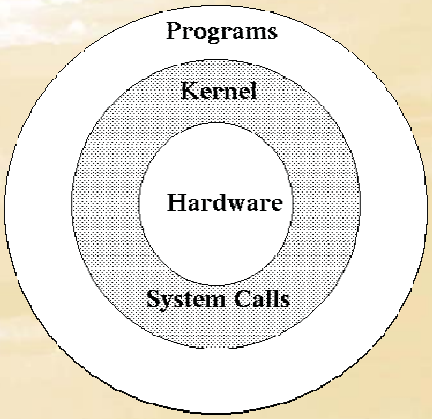
Example: Networking - Packet Loss



Kernel Structure and User Levels







Run Level	Mode	Action
0	Halt	Shuts down system
1	Single-User Mode	Does not configure network interfaces
2	Multi-User Mode	Does not configure network interfaces.
3	Multi-User Mode with Networking	Starts the system normally.
4	Undefined	Not used/User-definable
5	X11	As runlevel 3 + display manager(X)
6	Reboot	Reboots the system

Go Test Drive a Linux OS

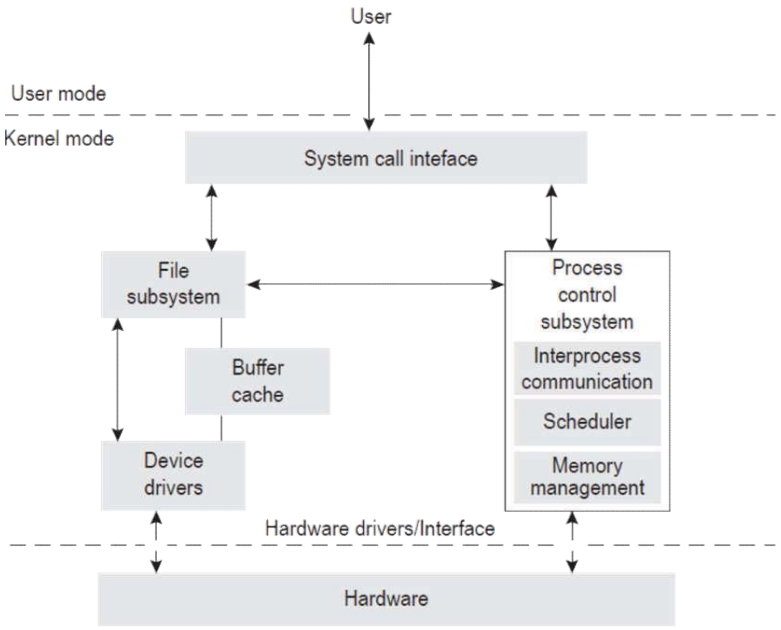
- Install Oracle VirtualBox
  - https://www.virtualbox.org/wiki/Downloads
- Select your host system and click download
  - Can do a check sum if there would be any download errors
- Also download the VM Extension Pack (same version number)
  - Remember to remove the older versioned extension pack (if installed) before installing the newer version
- After installing VirtualBox, download the Ubuntu or Fedora images
  - File sizes are smaller for server images than the desktop images
- **Let's do it now!**

Basic Requirement on Virtual Machine

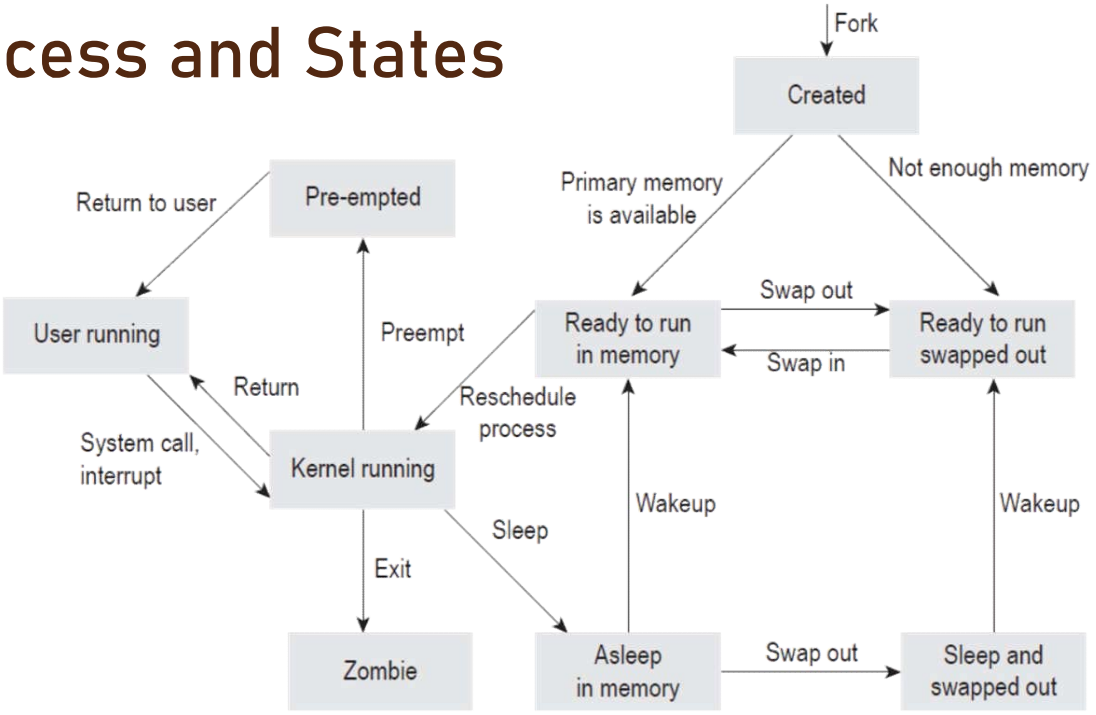
- CPU virtualization support (Intel VT-x or AMD-V)
- High-speed Internet
- 30-40 GB of available disk space
- At least 4-8 GB of system memory
- USB thumb drive larger than 8 GB

A Quick Overview of Linux OS

- User / kernel modes
- System call interface
- File structure



Process and States

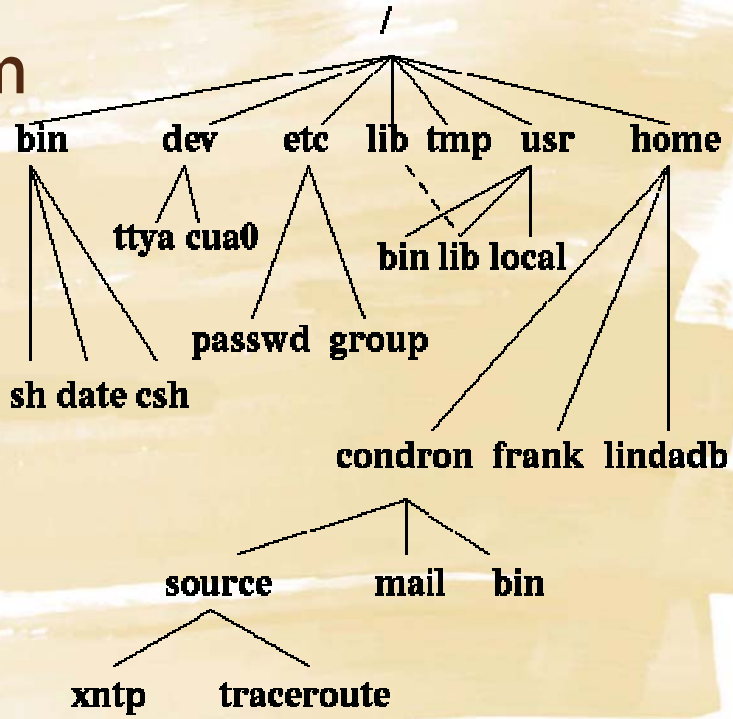


Linux Typical File Structure

Directory	Use	Subdirectories (If Any)
/bin	User executable programs	None
/boot	Linux boot programs including boot loader program (grub in most cases)	Boot loaders in subdirectories
/dev	Physical devices are treated as files, stored in this directory	Various to organize specific types of devices, see Table 3.10
/etc	Configuration files and scripts	Numerous
/home	User home directory space	One subdirectory per user
/lib	Libraries used by the kernel and programs stored in /bin, /sbin	modules, security, udev, etc
/lost+found	Recovered files from disk errors due to an unexpected shutdown of the system	None
/media	Initially empty, removable media can be mounted here	None
/mnt	Mount point for temporary file systems	None
/opt	Initially empty, this optional directory can be used to install some application software	Application software subdirectories (if any)
/proc	Kernel records of running process information	Subdirectories created dynamically of each running process
/root	The system administrator's home directory	None, or subdirectories for profile storage such as .config, .gconf, .mozilla
/sbin	Like /bin, system executable programs, but these are typically used by system administrators, not users	None
/tmp	Temporary storage for running processes	None initially
/usr	Application software files and executables	bin, etc, include, lib, local, sbin, share, src, tmp
/var	Data files that grow over time (printer spool files, email repositories, log files, etc.)	Varies by system, includes cache, log, mail, spool, www

A Typical File System

- Usually draw a file system an inverted tree structure
- The **root directory**, denoted by /, is at the top and work down through sub-directories underneath it



File System (cont'd)

- Each node is either a file or a directory of files
- A directory further can contain other files and directories
- Specify a file or directory by its path name, either the full, or absolute, path name or the one relative to a location
- The full path name must starts with the root, /, and follows the branches of the file system, each separated by /, until you reach the desired file, e.g.:
  - `/home/condron/source/xntp`

File System (cont'd)

- A relative path name specifies the path relative to another, usually the current working directory that you are at
- Two special directories :
  - `.` the current directory
  - `..` the parent of the current directory
- If currently at `/home/frank`, specify the path above in a relative fashion is
  - `../condron/source/xntp`
- This indicates that firstly goes up one directory level, then goes down through the `condron` directory, then the `source` directory and then to `xntp`

File System (cont'd)

- `/` The root of all directories on the system; all other directories are subdirectories of this directory, either directly or through other subdirectories
- `/bin` Essential tools and other programs (or binaries)
- `/dev` Files representing the system's various hardware devices, e.g., use the file "`/dev/cdrom`" to access the CD-ROM drive
- `/etc` Miscellaneous system configuration files, startup files, etc

File System (cont'd)

- /home The home directories for all of the system's users
- `/lib` Essential system library files used by tools in "`/bin`"
- `/proc` Files that give information about current system processes
- `/root` The super-user's home directory, whose username is root

(In the past, the home directory for the super-user was simply "/"; later, "`/root`" was adopted for this purpose to reduce clutter in "/")

File System (cont'd)

- `/sbin` Essential system administrator tools, or system binaries
- `/tmp` Temporary files
- `/usr` Subdirectories with files related to user tools and applications

Inodes

- Every directory and file is listed in its parent directory
- For the root directory, the parent is itself
- A directory is a file that contains a table listing the files contained within it, giving file names to the **inode** (index node) numbers in the list
- The information about all the files and directories is maintained in the ***INODE TABLE***
- An ***inode*** is an entry in the table containing information about a file (metadata) including file permissions, UID, GID, size, time stamp, pointers to files data blocks on the disk etc

Users and Groups

- In Linux, there is a concept of user and an associated group
- System determines whether or not a user or group can access a file or program based on the permissions assigned to them
- Apart from all the users, there is a special user called Super User or the root (the user) which has permission to access any file and directory

Access Permissions

- There are three permissions for any file, directory or application program
- The following lists the symbols used to denote each, along with a brief description:
  - ***r***: Indicates that a given category of user can read a file
  - ***w:*** Indicates that a given category of user can write to a file
  - ***x:*** Indicates that a given category of user can execute the file

File Ownerships and Permissions

- Each of the three permissions are assigned to three defined categories of users
- The categories are:
  - owner: The owner of the file or application
  - group: The group that owns the file or application
  - others: All users with access to the system

Showing File Ownerships

- `$ ls` is used to list the contents of a directory
- If the command `ls` is written with parameter `-l` then the command lists contents of the working directory with details

```
$ ls -l
```

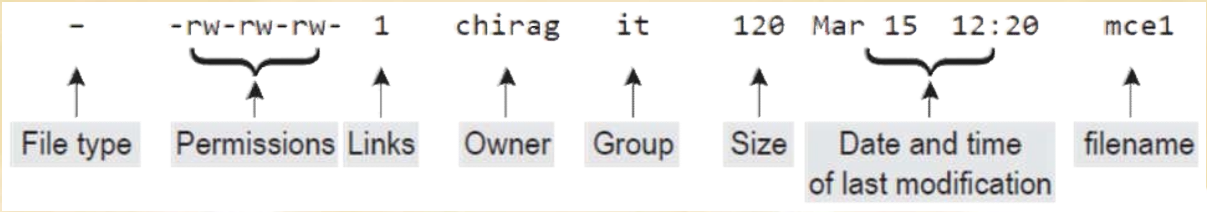

List Files

- One can easily view the permissions for a file by invoking a long format listing using the command

```
$ ls -l
```

- For instance, if the user juan creates an executable file named `test`, the output of the command `$ ls -l test` would look like this

```
-rwxrwxr-x 1 juan student 0 Sep 26 12:25 test
```



File Permissions

- Permissions for this file are listed are listed at the start of the line, starting with ***rwx***
- This first set of symbols define owner access
- The next set of ***rwx*** symbols define group access
- The last set of symbols defining access permitted for all other users

File Permissions (cont'd)

- This listing indicates that the file is readable, writable, and executable by the user who owns the file (user juan) as well as the group owning the file (which is a group named student)
- The file is also world-readable and world-executable, but not world-writable

Some Commands – Examples

- `$ cd try_it`

  Changes the directory to try_it
- `$ pwd`

  Prints present working directory (e.g. /home/smith/try_it)
- `$ cd ..`
  - Move up one directory
- `$ pwd`
  - Prints /home/smith

Some Commands (cont'd)

- `$ cd /home`

  Set the absolute path
- `$ pwd`

  Prints "/home"
- `$ cd`

  Returned to the user's home directory
- `$ pwd`

  Print "/home/elaw"

Some Commands (cont'd)

- `$ mkdir my_dir`
  - Makes a new directory named "my_dir" (the path is given relative) as a subdirectory of the current directory
- `$ rmdir your_dir`
  - Removes directory your_dir if it is empty

Some Commands (cont'd)

- `$ cp file_1 file_2`
  - Copies file_1 to file_2. The both files must be in the same working directory. If they are in various directories, the path must be given.
- `$ mv file_1 file_2`
  - Moves file_1 to file_2, files must be in the same working directory.
  - For setting in different directories, the path(s) must be given
  - The file_1 is removed from the disk.
  - This command is also suitable for moving directories.

Removing Files

- `$ rm file_a`
  - Removes the file_a from the system
  - Can use *wildcard*, e.g.,
- `$ rm h*c`
  - Remove all files beginning with h and ending with c which are in working directory.
  - If writing
- `$ rm *`
  - Erases all files from your working directory.

Changing Ownership and Group

- `$ chown <owner> <file/directory name>`
  - Ownership of a file or directory can be changed with command
- `$ chgrp <group> <file/directory name>`
  - Group of a file or directory can be changed with command

About Read Write and Execute

- `$ chmod -R ### <filename or directory>`
  - Permissions of a file can be changed with the `chmod` command
  - `-R` is optional and when used with directories will traverse all the sub-directories of the target directory changing ALL the permissions to ###
- The ### indicates 3 octal numbers
- Convert an octal number to binary number to represent the "rwx" permissions discussed before
  - 1 – permitted
  - 0 – prohibited

Setting *rwx*

- The octal number #'s can be:
- 0 = Nothing
  1 = Execute
  2 = Write
  3 = Execute & Write? (2 + 1)
  4 = Read
  5 = Execute & Read (4 + 1)
  6 = Read & Write (4 + 2)
  7 = Execute & Read & Write (4 + 2 + 1)

Practice – Go to the Installed VM

- Login
- Find the present Directory
- Write the root directory structure
- Write a few commands available in /bin and /sbin directory
- Find your directory
- Write the permissions of your home directory
- Create a new Directory test in your directory
- Copy the file /etc/resolv.conf in your directory
- Rename the test directory to testing
- Delete the testing directory
- Change the permissions of your directory to 700
- Change the permissions of /tmp directory to 700

澳 門 理 工 學 院
Instituto Politécnico de Macau
Macao Polytechnic Institute

COMP 225 Network and System Administration

Notes #2: Shell Scripting

K. L. Eddie Law, PhD

Macao Polytechnic Institute
School of Applied Sciences
Academic Year 2020-2021, 2nd Semester

Topics

- Typical kernel starting up process
- Bash
- More commands
- Shell scripting

Typical Kernel Starting Procedure

- The initialization process starts by initializing the peripherals it needs, typically the serial port, network, and, maybe, USB
- If the auto boot sequence is not interrupted, it usually copies the kernel from Flash to RAM and starts it executing
- The Linux kernel, a compressed image, decompresses itself into the appropriate location in RAM
- It then initializes all of the kernel subsystems, such as memory management and drivers for all of the peripheral devices in the system (must initialize the devices to suit its own requirements)

Typical Startup Operations (cont'd)

- During initialization process, the kernel spews out a large number of messages describing what it is doing
- Next it mounts the root file system. Up to this point, the kernel has been running in kernel space. Finally it starts the `init` process, which makes the transition to user space
- The last thing the kernel boot does is start a process with PID 1 (`/sbin/init`). This then becomes the ultimate parent of every other process in the system
- `$ ps aux | less`

  (press "q" to quit)

Systemd

- ***systemd***
  - Check if `systemd` is running `$ ps -p 1`
  - The latest design of a kernel initialization mechanism encompassing something like 900 files
  - Manages and acts on objects called ***units***
  - Its most common type of units is "***service***," represented by a file that ends in `.service` (check those `*.service` files in `/lib/systemd/system`)
  - Open up, for example, `iscsid.service`; if there is a parameter `WorkingDirectory`, then it indicates the location of support files for the executable, which is identified by `ExecStart`

Systemd

- Use `systemctl` command to manage `systemd`, it understands

start <unit>	Start the specified unit(s)
stop <unit>	Stop the specified unit(s)
restart <unit>	Restart the specified unit(s). If not running they will be started
reload <unit>	Ask specified unit(s) to reload their configuration files
status <unit>	Display the status of the specified unit(s)
enable <unit>	Create symlinks to allow unit(s) to be automatically started at boot
disable<unit>	Remove the symlinks that cause the unit(s) to be started at boot
deamon-reload	Reloads the systemd manager's configuration. Run this any time you make a change to a systemd file

- First, try command `systemctl` with no arguments to see the display

Shell – The Interface

- Several shell programs in common use, all serve the same basic purpose, yet differ in details of syntax and features
  - ***Bourne Again SHell*** – `bash`, a "reincarnation" of the Bourne shell, `sh`, written by Stephen Bourne at Bell Labs for Unix 7; the default on most Linux distributions; should use it unless there is a good reason to switch to another shell; `bash` was by Brian Fox in 1989
  - ***Korn Shell*** – `ksh`, developed by David Korn at Bell Labs in the early 1980s, more advanced programming facilities than `bash`, but nevertheless maintains backward compatibility
  - ***Tenex C Shell*** – `tcsh`, a successor to the C shell, `csh` that was itself a predecessor to the Bourne shell. Tenex was an OS that inspired some of the features of `tcsh`
  - ***Z Shell*** – `zsh`, described as an extended Bourne shell with a large number of improvements, including some of the most useful features of `bash`, `ksh`, and `tcsh`

Using Bash Shell

- Some hidden configuration files at user's home directory
  - `.profile` – executes whenever the user logs in (no matter which shell is used)
  - `.bash_profile` – executes whenever user logs in to a bash shell
  - `.bashrc` – executes whenever a new bash shell is started
- User edits these files and put individually defined items in these files
  - E.g., if using only Bash, user's defined variables can be put in `.bashrc` file
- Any changes made in `.bashrc` can take effect immediately if executing following command

```
$ source ~/.bashrc
```

About Shell Commands

- A few commands were experienced, let's refresh
- A command with other info on the same line are run together, e.g.,

  `$ ls -la`
  - There are two entries on this "***line of script***"
  - Since $ is for prompting the user, the first input entry is "`ls`" ← the input command (represented by $0 in bash)
  - The second entry is "`-la`" ← the first argument to the command (represented by $1 in bash)
  - For each input line, Bash can intake numerous parameters, but only $0 to $9 values (see how we use them later on…)

More Shell Commands (1)

- `$ cat <filename>`
  - Displays a text File, the name of this command is derived from "concatenate," which means to join together, one after the other, e.g.,

    `$ cat test1 test2 > test3`
  - If both `test1` and `test2` have data content, they are concatenated and saved in file `test3`. The new file will be generated, if it does not exist

    `$ cat test1 test2 >> test3`
  - Append both `test1` and `test2` to the end of file `test3`
- `$ [ less | more ] <textfile>`
  - Displays the content of the file on screen, the command `less` permits to use arrow keys to scroll up and down
  - Press the key "q" to exit from the utility

Shell Commands (2)

- `$ whoami`
  - Who you are
- `$ logname`
  - who you are logged in as
- `$ id`
  - Info about you and your groups
- `$ groups`
  - Info about groups
- `$ uname -a`
  - Info about the operating system
- `$ hostname`
  - Info about the name of the computer

Shell Commands (3)

- `$ date`
  - Displays the date
- `$ cal`
  - The text calendar (can check out command `ncal`), can run such as
    `$ cal 2_digit_month 4_digit_year`
  - For example, `$ cal 01 2020`
- `$ uptime`
  - Since the time that the computer was booted up
- `$ wc <filename>`
  - Counts number of words or lines in a file
    `$ wc -l <fn>` counts lines
    `$ wc -w <fn>` counts words

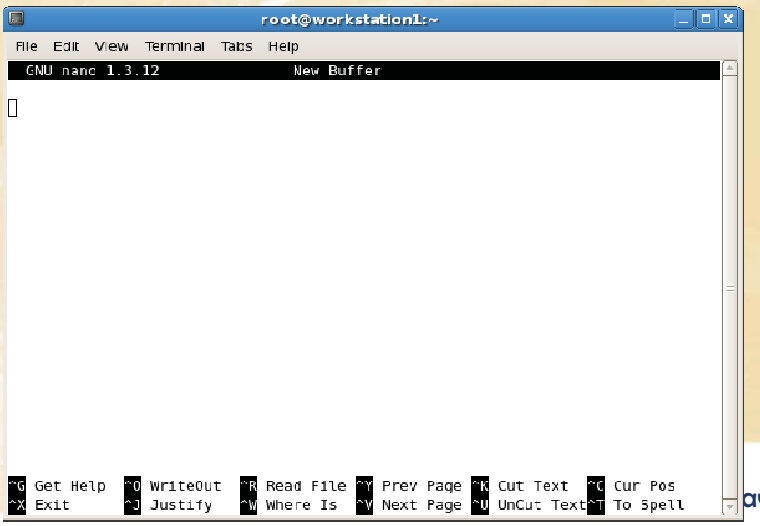
Shell Commands (4)

- `$ file <filename>`
  - Provides information about a file type
- `$ stat <filename>`
  - Provides more detailed information about a file
- `$ tail –x <filename>`
  - Displays the last *x* lines from a file
- `$ head –x <filename>`
  - Displays the first *x* lines from a file
- `$ clear`
  - Clears the screen

More on commands later!!

Text Editors on Terminals

- `$ vi` (or `vim`) `<filename>`
  - A bit complicated for beginners… hence
- `$ nano <filename>`



Nano

- Available options are listed at the bottom bar, the ^ is the “control” key, commonly used keys
  - `cntl-x` = exit, will prompt you to save if you have changes
  - `cntl-o` = save
  - `cntl-r` = insert file
  - `cntl-k` = delete a line
  - `cntl-u` = undelete previous contiguous deletions
- Note: the editor `nano` is available in Red Hat systems on rescue disk
- Let’s try it out **now**!!

What is Shell Scripting?

- We have been doing Linux commands on input prompts interactively in a terminal
- Can we run many commands altogether in one shot?
- Putting all these lines of scripts together into one file, and the file is a ***script*** file
  - Bash is a scripting language
  - Scripts can automate numerous administrative works

What to Cover in Shell Scripting?

- Basics of Linux shell scripting
  - Definition of shell scripts
  - Uses of shell scripts
- Writing shell scripts
  - Control statements
  - Command lines

Running a Script File

- `$ echo ls > lsfile`
  - `echo` is a command to show data on screen, but "`>`" redirects it into a file named `lsfile`
- Run the file with

  `$ . lsfile` (at least a space between a "dot" and the filename)

  `$ source lsfile`

  `$ bash lsfile`
  - What did you see? They work!?
- Try `$ ./lsfile`
  - Seeing anything??

Running a Script File (cont'd)

- Make a script file running by itself
- begin the first line with a #! and a shell command (full path of shell executable)

```
$ which bash
```

  - Gives the full path of the executable bash
  - Add it as the first line of script in the file, i.e.,

```
#!/bin/bash
```

  - Using a text editor to add it the first line in file "`lsfile`", then run it with

```
$ ./lsfile
```


Running a Script File (cont'd)

- In fact, it was not the issue of having the line "#!..." but the mode of the file, make it executable
  - `$ chmod +x lsfile`
  - `$ ./lsfile`
- If no shell (`#!`) is specified in the script file, the default is to choose the current executing shell
- Different scripting languages may act differently
  - Hence the `#!` line is important
  - The `bash` is popular
- **Recalling**: a script doesn't need to be executable if it is an input argument to the `bash` command

Let's Do More Complicated Scripts

Why Writing Script Files?

- To avoid repetition:
  - If you do a sequence of steps with standard commands over and over, why not do it all with just one command?
  - Or in other words, store all these commands in a file and execute them one by one
- To automate difficult tasks:
  - Many commands have subtle and difficult options that you don't like to remember or figure out every time

Environment Macro Variables

- There are environment variables, run

  ```
  $ env
  ```

  - Shows all variables in system

    ```
    $ echo $PATH
    ```

  - Showing the $PATH variable in Linux (the %PATH% in Windows)

    ```
    $ echo $HOME
    ```

  - Showing the home directory of a user

Example

- If running out of disk space
- If there are files no longer useful anymore, then removing them with

```
rm -rf $HOME/.config/menus
rm -f $HOME/.config/mime*
rm -f $HOME/.config/user*
rm -f $HOME/.gtkrc*
rm -rf $HOME/.cinnamon
```

Remark: these files may not be in our systems, example only

Example (cont'd)

- Put all these line commands into a shell script, called `myscript`
- Add the **#!** on the first line in file

```
elaw@s1:~$ cat myscript
#!/bin/bash
rm -rf $HOME/.config/menus
rm -f $HOME/.config/mime*
rm -f $HOME/.config/user*
rm -f $HOME/.gtkrc*
rm -rf $HOME/.cinnamon
```


Example (cont'd)

- Run the script with
- Step 1: make it executable

  ```
  $ chmod u+x myscript
  ```

- Step 2: run it

  ```
  $ ./myscript
  ```

- Each line of the script is processed in order

Shell Variables

- Declare and assign value to a variable:

  `varname=varvalue`

  - `varname` is the name of the variable
  - `varvalue` is the value of the variable
  - No spaces on both sides of the "="
- Make it an environment variable, using "`export`"

  `export varname=varvalue`

Shell Variables (cont'd)

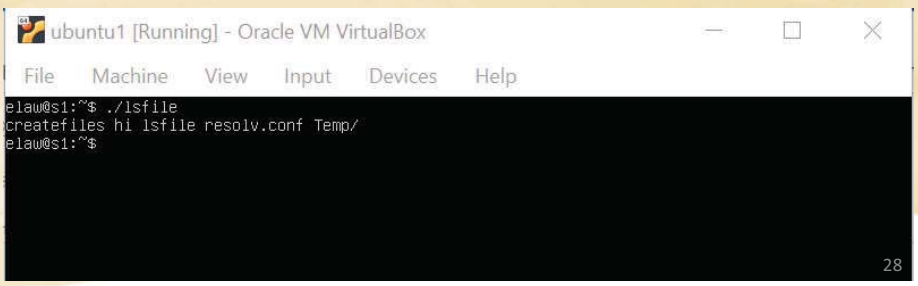
`` and the $( ) operators

A command

- Use a text editor to type in

```
#!/bin/bash
filelist=`ls -F`
echo $filelist
```

- `` backquotes, i.e., usually the key on the left of the number 1
- Backquotes deliver the execution output of a command enclosed
- The output is assigned to the variable `filelist`
- Run it with $ ./lsfile



Evaluate Expressions

$(( )) and the $[ ] expression operators

- Two expressions for evaluations – calculate results
  - The $(( )) expression operator
  - The **expr** command
- Examples:

```
elaw@s1:~$ value=$((1+2))
elaw@s1:~$ echo $value
3
elaw@s1:~$
```

Space

```
elaw@s1:~$ value=`expr 1 + 2`
elaw@s1:~$ echo $value
3
elaw@s1:~$
```


Notes 1

- Why do we need the **expr** command or **$((…))** expansion operator ?
- Example

```
elaw@s1:~$ value=1+2
elaw@s1:~$ echo $value
1+2
elaw@s1:~$
```

NOTES: 1+2 is copied as is into *value* and not the result of the expression, to get the result, we need **$((…))** or **expr**

Notes 2

- Variables as arguments

```
elaw@s1:~$ count=5
elaw@s1:~$ count=$(($count+1))
elaw@s1:~$ echo $count
6
elaw@s1:~$
```

NOTES: *count* is replaced with its value by the shell !

Notes on expr and $((…))

- expr or $((…)) supports the following operators:
  - Arithmetic operators: +,-,*,/,%
  - Comparison operators: <, <=, ==, !=, >=, >
  - Boolean/logical operators: &, |
  - Parentheses: (,)
  - Precedence is the same as C, Java
- N.B. for expr, the multiplication * may have to be added with an escape \ character

Control Statements

- Without control statements, execution within a shell scripts flows from one statement to the next in succession
- Control statements control the flow of execution in a programming language
- 3 most common types of control statements:
  - **Conditionals**: if/then/else, case, ...
  - **Loop statements**: while, for, until, do, ...
  - **Branch statements**: subroutine calls (good programming practice), goto (usage not recommended)
- Semi-colon (;) indicates the end of a line, but usually not added (remark: no ; immediately after keywords)

for Loops

- for loops allow repetition of a command for a specific set of values
- Syntax:

```
for var in value1 value2 ...
do
    command_set
done
```

- command_set is executed with each value of var, i.e., the sequence of (value1, value2, ...)

Notes on the for Loop

- Example: listing all files in a directory

```
#!/bin/bash
for i in *
do
    echo $i
done
```

NOTES: * is a wildcard that stands for all files in the current directory, and `for` will go through each value in *, which are all the files and `$i` is the filename

Notes on the for Loop

- Output of the example

```
elaw@s1:~$ chmod u+x listfiles
elaw@s1:~$ ./listfiles
a
b
c
html
listfiles
elaw@s1:~$
```

Conditionals

- Conditionals are used to "test" something
  - In Java or C, usually test if a Boolean expression is true or false
  - In bash shell script, the only thing you can test is whether or not a command is "***successful***"
- Every well behaved command returns back a **return code**
  - ***0*** if it was ***successful***
  - Non-zero if it was unsuccessful (actually 1..255)
  - Different from Java or C

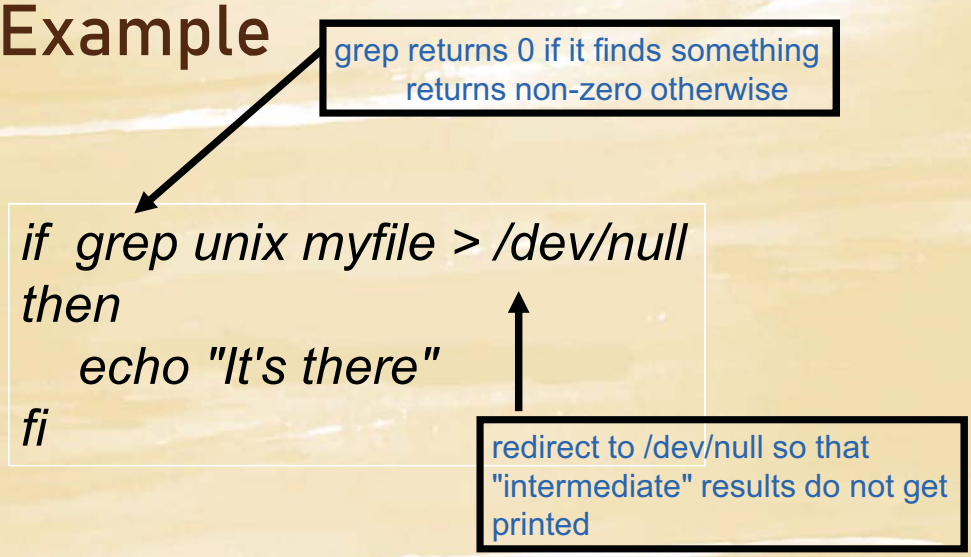
The `if` Command

- Simple form:

```
if decision_command_1
then
    command_set_1
fi
```

- The importance of having "`then`"
  - Each line of a shell script is treated as one command
  - The "`then`" is a command in itself
  - Though it is part of the "`if`" structure, it is treated separately

The `if` Example



For **grep**, the option **-i** ignore upper and lowercases

Using `else` with `if` Expression

- Example

```
#!/bin/bash
if grep "UNIX" myfile > /dev/null
then
    echo  UNIX occurs in myfile
else
    echo  No!
    echo  UNIX does not occur in myfile
fi
```

Using `elif` with `if` Expression

- Example

```
#!/bin/bash
if grep "UNIX" myfile > /dev/null
then
    echo  UNIX occurs in myfile
elif grep “DOS” myfile > /dev/null
then
    echo DOS appears in myfile not UNIX
else
    echo  nobody is here in myfile
fi
```


Using : in Shell Scripts

- Sometimes, we do not want a statement to do anything
- In this case, a colon ‘:’ can be used

```
if grep UNIX myfile > /dev/null
then
:
fi
```

- Does not do anything when the word “UNIX” is found in `myfile`

The `test` Command

- For checking validity, 3 common types of checking
  - Check on files
  - Check on strings
  - Check on integers

The `test` on Files

- **`test -f file`**: does file exist and is not a directory?
- **`test -d file`**: does file exist and is a directory?
- **`test -x file`**: does file exist and is executable?
- **`test -s file`**: does file exist and is longer than 0 bytes?

The test on Files: Example

```
#!/bin/bash
count=0
for i in *; do
if test -x $i
then
    count= $(($count+1))
fi
done
echo Total of $count files executable
```

NOTE: ***$count+1*** serves the purpose of count++

The test on Strings

- **test -z string**: is string of length 0?
- **test string1 = string2**: does string1 equal string2?
- **test string1 != string2**: not equal?

The test on Strings: Example

```
#!/bin/bash
if test -z $REMOTEHOST
then
:
else
    DISPLAY="$REMOTEHOST:0“
    export DISPLAY
fi
```

NOTES: This example tests to see if the value of REMOTEHOST is a string of length > 0 or not, and then sets the DISPLAY to the appropriate value.

The test on Integers

- **test int1 -eq int2**: is int1 equal to int2?
- **test int1 -ne int2**: is int1 not equal to int2?
- **test int1 -lt int2**: is int1 less than to int2?
- **test int1 -gt int2**: is int1 greater than to int2?
- **test int1 -le int2**: is int1 less than or equal to int2?
- **test int1 -ge int2**: is int1 greater than or equal to int2?
- Alternatively, can use **((…))** to test mathematical expressions
  - **((int1 == int2))**
  - **((int1 != int2))**
  - …

The test on Integers: Example

```
#!/bin/bash
smallest=10000
for i in 5 8 19 8 7 3
do
    if test $i -lt $smallest
    then
        smallest=$i
    fi
done
echo $smallest
```

NOTES: This program calculates the smallest among the numbers 5, 8, 19, 8, 3.

Alias of test: [ ]

- The **[ ]** is the alias of the **test** command
- Expression to test is in the brackets, and each bracket must be surrounded by spaces

```
#!/bin/bash
smallest=10000
for i in 5 8 19 8 7 3
do
    if [ $i -lt $smallest ]
    then
        smallest=$i
    fi
done
echo $smallest
```

The while Loop

- **while** loops repeat statements as long as the next Unix command is successful
- Works similar to the **while** loop in Java or C

```
#!/bin/bash
i=1
sum=0
while [ $i -le 100 ]
do
    sum=$(($sum+$i))
    i=$(($i+1))
done
echo The sum is $sum.
```

NOTES: The value of i is tested in the while to see if it is less than or equal to 100.

The until Loop

- **until** loops repeat statements until the next Unix command is successful
- Works similar to the **do-while** loop in C

```
#!/bin/bash
x=1
until [ $x -gt 3 ]
do
    echo x = $x
    x=$(($x+1))
done
```

NOTES: The value of x is tested in the until to see if it is greater than 3

Command Line Arguments

- Shell scripts would not be very useful if we could not pass arguments to them on the command line
- Without arguments, restricts the usefulness of the script
- Parameters to any program, e.g.,

```
$ ls -t foo
```

- The '**-t**' and **foo** are parameters to the program **ls**
- This command line has three parameters: **ls**, **-t** and **foo**.

NOTES: command is also part of the command line

Command Line Arguments (cont'd)

- Shell script arguments are "numbered" from left to right
  - **$1** - first argument after command.
  - **$2** - second argument after command.
  - ... up to **$9**
- They are called "positional parameters"
- Their **position** in the command line determines their value

Command Line Arguments (cont'd)

- Ex: Find out if a string appears in file
- Run **`$ mystr string file`**

```
elaw@s1:~$ cat myTime
#!/bin/bash
grep $1 $2
elaw@s1:~$ ./myTime DateTime myTime.py
From DateTime import DataTime
elaw@s1:~$
```

NOTES: $1 has value *DateTime* and $2 has value *myTime.py*

Command Line Arguments (cont'd)

- Other variables related to arguments:
  - **$0** → Name of the command running
  - **$*** → All the arguments (even if there are more than 9)
  - **$#** → The number of arguments
- Example using these special variables

```
elaw@s1:~$ cat cmd_line
#! /bin/bash
echo "$0 = name of the command"
echo "$* = list of arguments"
echo "$# = total number of arguments"
elaw@s1:~$
```

Command Line Arguments (cont'd)

- Example output

```
elaw@s1:~$ ./cmd_line
./cmd_line = name of the command
0 = total number of arguments

elaw@s1:~$ ./cmd_line 1 2 3 4 5
./cmd_line = name of the command
1 2 3 4 5 = list of arguments
5 = total number of arguments

elaw@s1:~$
```


More on the bash Variables

- 3 basic types of variables in a shell script
- Positional variables: $1, $2, $3, …, $9
- Keyword variables: e.g., $PATH, $HOME, and anything else we may define
- Special variables:
  - $! ← return process id of last background process to finish
  - $? ← return status of last foreground process to finish
  - $$ ← the process id of the current shell
  - There are many more others that we can find out about using `$ man sh`

The case Command

- Statement ends with the `esac` keyword
- Use | to separate multiple patterns
- The) terminates a pattern list
- Clause must be terminated with **;;**
- If no matches, the return status is zero; otherwise, the return status is the exit status of the executed commands
- For default case, use the wildcard asterisk symbol (*) as a final pattern to match the rest

```
case EXPRESSION in
  PATTERN_1)
    STATEMENTS
    ;;
  PATTERN_2)
    STATEMENTS
    ;;
  PATTERN_N)
    STATEMENTS
    ;;
  *)
    STATEMENTS
    ;;
esac
```


Reading Input

- All this while, we have talked about shell scripts that do useful work and write some output
- What about reading input???
- Using the **`read`** command
  - Reads one line of input and assigns it to variables given as arguments
  - Data type of variable does not matter, shell has no concept of data types

Notes on *read*

- Syntax:
  - **read var1, var2, var3 ….**
  - Reads a line of input from standard input.
  - Assigns first word to var1, second word to var2, ...
  - The last variable gets any excess words on the line.

Reading Input (cont'd)

- Syntax:
  - **`read var1, var2, var3 …`**
  - Reads a line of input from standard input
  - Assigns first word to **`var1`**, second word to **`var2`**, ...
  - The last variable gets any excess words on the line
- Example:

```
elaw@s1:~$ read var1 var2 var3
this is to test the read
elaw@s1:~$ echo $var1
this
elaw@s1:~$ echo $var2
is
elaw@s1:~$ echo $var3
to test the read
elaw@s1:~$
```

NOTES: var3 has the rest of the string "to test the read"

Remark

- We have done a lot of works on shell scripting today!!

澳門理工學院
Instituto Politécnico de Macau
Macao Polytechnic Institute

COMP 225 Network and System Administration

Notes #3: Utility

K. L. Eddie Law, PhD

Macao Polytechnic Institute
School of Applied Sciences
Academic Year 2020-2021, 2nd Semester

Some Commands Briefly Discussed

- uname -a
- systemctl
- ps aux
- ls -la
- cat
- less | more
- whoami
- logname
- id
- groups
- hostname
- date
- cal
- uptime
- wc
- file
- stat
- head
- tail
- clear
- echo
- chmod
- chown
- chgrp
- mkdir
- rmdir
- cd
- rm
- pwd
- touch
- which

Some Reserved Keywords – Scripting

- if
- elif
- else
- then
- fi
- case
- esac
- while
- until
- do
- done
- for
- !
- {
- }

Objectives

- Learn some basic UNIX/Linux utilities
- The basic of `grep` command
- Use the `dd` utility to copy and convert files
- Monitor hard disk usage
- Use system status utilities
- Monitor and manage processes
- Check the spelling of text in a document
- Use the `cmp` command to compare the contents of two files
- Format text to create and use a man page

UNIX/Linux Utilities

- Utilities for
  - Creating and managing files
  - Producing reports
  - Monitoring and maintaining the system
  - Executing programs
  - Recovering from a range of errors
- New utilities are updated/edited/added daily to make Linux run more efficiently
  - Of course, some are deprecated or removed

UNIX/Linux Utilities (cont'd)

- Eight major areas:
  - File processing
  - System status
  - Networking
  - Communications
  - Security
  - Programming
  - Source code management
  - Miscellaneous

File Processing Utilities

Command	Brief Description of Function
awk	Processes files
cat	Displays files (and is used with other tools to concatenate files)
cmp	Compares two files
comm	Compares sorted files, and show differences
cp	Copies files
cpio	Copies and backups files in a archive
cut	Selects characters or fields from input lines
dd	Copies and converts input records
diff	Compares two text files, and shows differences
file	Displays the file type
find	Finds files within file tree
fmt	Formats text files for displaying
grep	Matches patterns in files, for line filtering, word search, etc.
gzip	Compresses or decompresses files
ispell	Checks one or more files for spelling errors
ln	Creates a link to a file

Command	Brief Description of Function
lpr	Sends a file to a printer or print device
man	Displays documentation for commands
mkfs	Builds a Linux/UNIX file system
mount	Mounts file systems or devices
od	Formats and displays data from a file in octal, hexadecimal, or ASCII format
paste	Concatenates file horizontally
pr	Formats text files for printing, and displays them
sed	Edits streams (non-interactive)
sort	Sorts or merges files
tail	Displays the last lines of files (default: last 10 lines)
tar	Copies and backs up files to a tape archive
tr	Translates or deletes characters from standard input and writes results to standard output
uniq	Displays unique lines, or reports repeated lines
whereis	Locates information about a specific file

The man Command

- An interface to the system reference manuals
- There might be different "section" page numbers for each command
- `$ man stat`
- And try `$ info stat`

Page Number	Description
1	User commands
2	System calls
3	C library functions
4	Devices and special files
5	File formats and conventions
6	Games
7	Miscellaneous
8	System admin tools and daemons

The ln Command*

- Soft link
  - `$ ln -s original newlink`
- What is the inode number?
  - `$ ls -li`
- Hard link
  - `ln original newlink`
- What are the inode and link numbers?
  - `$ ls -li`

The tar Command

- `tar` stands for tape archiver
- Used as a powerful backup and restore utility
- Most Linux files are downloaded as `.tar` files
- E.g., extraction of a `tar` file

```
tar -xvf file.tar
   -x extracts files
   -v verbose
   -f filename
```

- More on it later…

Two Files

- $ cat file1

```
Brian has a dog
Byron has 2 dogs
Ellen has a cat
Elden has a snake
Louise has two dogs
Ruby has two puppies and one cat
```

- $ cat file2

```
Brian has a dog
Byron has two dogs

Ellen has a cat
Elden has a snake
```


Comparing Files

- Use the cmp utility to compare the contents of two files, and report the first difference between them
- The cmp command displays the position and line number of this difference
  - `$ cmp file1 file2`
- If there are no differences, the cmp command displays nothing

The `uniq` Command

- Runs on a single file to searching for consecutive duplicate lines
- `uniq` removes any such duplicates, does not overwrite the file, but outputs a file without the duplicates, can simply redirect to a new file

```
$ uniq file.txt > filewithoutduplicates.txt
```

- Since only compares adjacent lines, cannot find duplicate lines that do not appear one after the other

`uniq` (cont'd)

- Options:
  - Output file without duplicates, or
  - (`-D`) output the found duplicates, or
  - (`-c`) count the number of duplicates
  - Can ignore (`-i`) case or
  - (`-s #`) skip over characters

The `diff` Command*

- Compares the differences between two files, in general
- For each line in the first file but not in second, output preceded by a '<'
- For every line in second file but not in first, output preceded by a '>'
- For each line or group of lines that differ, a summary is provided indicating how the two differ
  - If file 1 had a line, not in file 2, then the line had to be deleted to match file 2
  - If file 2 had a line that not in file 1, suggested that the line had to be added
  - If two corresponding lines between the two files did not match, suggested to change the line
  - Indicated by letters '**a**' for added, '**d**' for deleted, and '**c**' for changed; if 3a5,6 ⇒ at line 3 of the first file, we had to add lines 5 to 6 of the second file

The `diff` Command (cont'd)

- If a file does not exist, it responds with an error
- If first filename is a directory, `diff` finds file in the directory whose name matches that of the second file
- If `diff` is provided with two directories, it compares all pairs of files who share the same names in both directories
- Option `-i` ignores upper and lower case letters; `-w`, `-B`, `-E`, ignore all white space, blank lines, tab expansions, respectively
- To operate on more than two files, add option --from-file=
  - If comparing file1 to all of file2, file3, file4, and file5, should use

```
$ diff --from-file=file1 file2 file3 file4 file5
```

  - Output of the comparisons is separated by --- to indicate that the next file is being compared

The diff and patch Commands

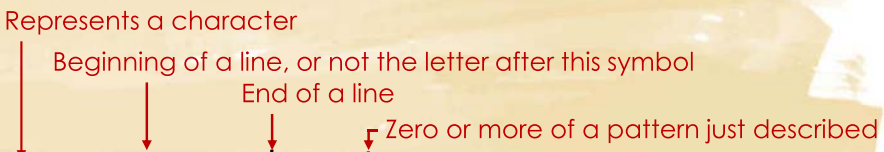
- Given two `file1` and `file2`
  - `$ diff file1 file2 > patchfile`
  - `$ path file1 < patchfile`
- Now both `file1` and `file2` are identical

grep (Global Regular Expression Print)*

- To search for words "`cat`" in the file "`file1`"

```
$ grep cat file1
$ grep cat < file1
$ cat file1 | grep cat
```

- Some ***metacharacters***: `.` `^` `$` `*` etc.
  - Search the metacharacter symbol, use an escape "\"
  - `?` have meaning in shell, not for `grep`



- Character classes: delineated with `[` and `]` symbols
  - `[ai]` matches an a or an `i`, both are in lowercase
  - `[a-z0-9]` matches the lowercase letters a through z, and `0` to 9

grep (cont'd)

- Regular expression can be complicated
- Options
  - `-i` ignores cases
  - `-v` complement lines instead of selected lines, -v "^$" removes blank lines
  - `-c` counts number of **lines** matching
  - `-l` shows names of files that contains the pattern, check out all files with "*"
  - `-w` shows results with exact word matching, no partial matching
  - `-b` prints the byte offsets from the beginning of the file
  - `-n` prints the line offsets from the beginning of the file
  - `-o` only characters that match
  - `-E` use extended regular expression (equal to use `egrep`)

The `find` Command*

- A powerful tool, the format is
  `$ find [directory] [options]`
  - where directory is the starting point in the file system for the search to begin
- Let consider the simple case here
- E.g., search for files with .conf extensions under /etc directory
  `$ find /etc -name "*.conf" -print`
  - Many new implementations don't need "`-print`" anymore
- Option: `-type [b|c|d|p|f|l|s]`
  - where b (block), c (character), d (directory), p (pipe), f (regular file), l (symbolic link), s (socket)
- Test it out: `$ find ~ -type f -exec wc -l {} \;`

System Status Utilities

Command	Brief Description of Function
date	Sets and displays date and time
df	Displays the amount of free space remaining on disk
du	Summarizes file space usage
file	Determines file type (e.g., script, executable, ASCII, etc.)
free	Displays amount of free and used memory in the system
kill	Terminates a running process
pgrep	Returns the process IDs that match the process name
pkill, killall	Kills a process, given its name
ps	Displays process status by process identification number and name
pstree	Visualizes processes, and displays them in tree format
renice	Changes the nice value of an already running process
sleep	Suspends process execution for a specified time
top	Dynamically displays the status of processes in real time, focusing on those processes that are using the most CPU resources
uname	Shows information about the operating system
vmstat	Shows information about virtual memory use
w	Displays detailed information about the users who are logged in
who	Displays brief information about the users who are logged in

The `finger` and `sleep` command

- The `finger` command was used to find out information about users
  - **Unsafe**, usually not installed by default
  - `$ finger username`
    - Displays information about the user including username, full name, home directory, last login time, shell,etc.
- The `sleep` command:
  - Suspends the execution of the process in number of milliseconds

Network Utilities

Command	Brief Description of Function
dig	Performs DNS lookups and displays the answers from name servers
ftp	Transfers files over a network (not safe)
ifconfig	Set up a wired network interface (old command)
ip	Set up network interface (new command)
netstat	Shows network connection information (old)
nfsstat	Shows statistics for Network File System (NFS) activity
nmap	Checks the opened port on the server
nslookup	Queries Internet domain name server (DNS)
ping	Polls another network station (using TCP/IP)
rcp	Remotely copies a file from a network computer
rlogin	Logs in to a remote computer (not safe)
route	Displays routing table information
rsh	Executes commands on a remote computer
scp	Secure transfers files between a local host and a remote host or between two remote hosts
sftp	Transfers files securely over a network connection
ssh	Enables secure connection to SSH server on a remote machine

Some Network Utility Commands

- `dig` – gets IP address from a domain name address, replacing the `nslookup` command

```
$ dig www.yahoo.com
```

- `ifconfig` – sets up a wired network interface card (obsolete in Debian/Ubuntu)
- `ip` – sets up network interfaces, can be used to troubleshoot networking

```
$ ip link
$ ip addr
```

Network Utility Commands (cont'd)

- `netplan` – backend-agnostic network configuration tool in YAML
- `netstat` – shows network connection information (not popular now)
- `ping` – establishes connectivity to a remote device (try it)
- `route` – displays routing table information (being replaced by `ip route`)
  - Can be Installed with `$ sudo apt install net-tools`

Other Network Utility Commands

- Most of them require installations

1. Overall bandwidth - `nload`, `bmon`, `slurm`, `bwm-ng`, `cbm`, `speedometer`, `netload`
2. Overall bandwidth (batch style output) - `vnstat`, `ifstat`, `dstat`, `collectl`
3. Bandwidth per socket connection - `iftop`, `iptraf`, `tcptrack`, `pktstat`, `netwatch`, `trafshow`
4. Bandwidth per process - `nethogs`

- We will come back more on the IP networking later!

Communications Utilities (Insecure)

- wall – sends a message to all logged-in users.
- mesg n – denies any real-time messages
- mesg y – accepts any real-time messages
- write – sends a message to a user
- mail – sends e-mail
- talk – allows users to simultaneously 'chat' with other logged in users

Command	Brief Description of Function
mail	Sends email messages
mesg	Denies (mesg n) or accept (mesg y) messages
talk	Lets users simultaneously type messages to each other (unsafe)
wall	Sends a message to all logged in users (who have permissions set to receive messages)
write	Sends a message to another use

Security Utilities

- Obsolete: `ipchains`
- Up-and-coming but not yet ready: `bpfilter`

Table 8-5 Security utilities

Command	Brief Description of Function
chgrp	Changes the group associated with a file or the file's group ownership
chmod	Changes the access permissions of a file or directory
chown	Changes the owner of a file

Table 8-5 Security utilities (continued)

Command	Brief Description of Function
ipchains	Manages a firewall and packet filtering (do not use if you are using *iptables* instead)
iptables	Manages a firewall and packet filtering (do not use if you are using *ipchains* instead)
passwd	Changes a password

More on it later →

Programming and Source Code Management Utilities

- Linux is written in C programming language
- Get compilers with `$ sudo apt install build-essential`

Table 8-6 Programming utilities

Command	Brief Description of Function
configure	Configures program source code automatically
g++	Compiles a C++ program
gcc	Compiles a C program
make	Maintains program source code
patch	Updates source code

Table 8-7 Source code management utilities **(fyi only, obsolete)**

Command	Brief Description of Function
ci	Creates changes in Revision Control Systems (RCS)
co	Retrieves an unencoded revision of an RCS file
cvs	Manages concurrent access to files in a hierarchy
rcs	Creates or changes the attributes of an RCS file
rlog	Prints a summary of the history of an RCS file

Revisit Permission Management

- Not the encryption security
- Recalling the commands …
- `chmod` …
  - Changes the access permissions of a file or directory
- `chown` [newOwner] [file]
  - Changes the owner of a file or directory
- `chgrp` [newGroup] [file]
  - Changes the default group associated with a file

POSIX File Permission Management

- Design issues with typical Linux file permission system
  - Only one user owner and one group owner
  - Different to change with the inheritance nature
- Try

```
$ getfacl file1
```

- Example:

```
$ setfacl -m g:root:rwx file1
$ getfacl file1
```

- Just let you get a feel of the concept of ACL (Access Control List) only

Miscellaneous Utilities

Table 8-8 Miscellaneous utilities

Command	Brief Description of Function
at	Executes a command or script at a specified time
atq	Shows the jobs (commands or scripts) already scheduled to run
atrm	Enables you to remove a job (command or script) that is scheduled to run
batch	Runs a command or script, and is really a subset of the *at* command that takes you to the *at>* prompt, if you type only *batch* (in Fedora and Red Hat Enterprise Linux, a command or script is run when the system load is at an acceptable level)
cal	Displays a calendar for a month or year
cd	Changes to a directory
crontab	Schedules a command to run at a preset time
expr	Evaluates expressions (used for arithmetic and string manipulations)
fsck	Checks and fixes problems on a file system (repairs damage)
printenv	Prints environment variables
tee	Clones output stream to one or more files
tr	Replaces specified characters (a translation filter)
tty	Displays terminal path name
xargs	Converts standard output of one command into arguments for another

Using the dd Command

- Copy a file and change the format of the destination file
- Options to handle copies when other methods are inappropriate such as when the format of the destination file needs to be altered
  - E.g., ASCII to EBCDIC, uppercase to lowercase, etc.
- An advantage to using the dd command over cp is that all users, not just the administrator, can copy files to and from the drive without mounting it
- Beware: no ways to restore if overwritten

Using the dd Command (cont'd)

- Be careful in using dd command

```
# dd if=/dev/sda of=/dev/sdb                (backup the entire harddisk)
# dd if=/dev/hda1 of=~/partition.img        (backup a partition)
# dd if=/dev/hda of=~/hdadisk.img           (create a hard disk image)
# dd if=hdadisk.img of=/dev/hdb             (restore using a hard disk image)
# dd if=/dev/cdrom of=cd.iso bs=2048        (create CDROM backup, 2048 block size)
# dd if=ubuntu-20.10.iso of=/dev/sdb bs=4M status=progress oflag=sync
```

- Commonly used options:
  - `if=input_file`
  - `of=output_file`
  - `conv=ascii` (converts destination format to ASCII)
  - `conv=lcase` (converts uppercase to lower)

Checking Hard Disk Usage

- To maintain adequate hard disk free space, use these strategies:
  - Be vigilant against running dangerously low on free space by using the **df** command
  - Watch for conspicuous consumption using the **du** command
  - Follow a routine schedule for "garbage" collection and removal by using the **find** and **rm** commands

Using the df (disk free) Utility

- The df utility reports on the status of 1024-byte blocks that are allocated, used, and available and the mount point
- Options:
  - -h human readable form
  - -k sizes in kilobytes

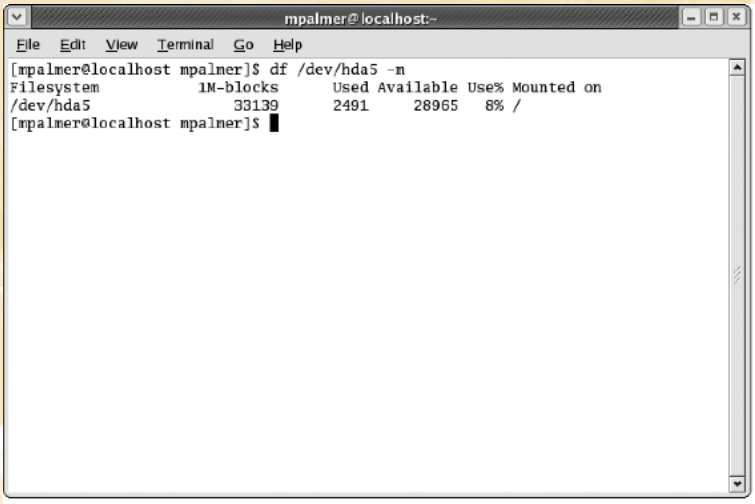


Figure 8-2 Viewing information for one file system in megabytes

Using the du (disk usage) Utility

- The du utility summarizes disk usage, expressed in 1024-byte blocks (default) or by the number of bytes (-b option)
- Remark: 512-byte block for POSIXLY-CORRECT
- Options:
  - `-a` displays info for files/dirs
  - `-c` creates an ending total
  - `-b` displays in bytes
  - `-h` human readable

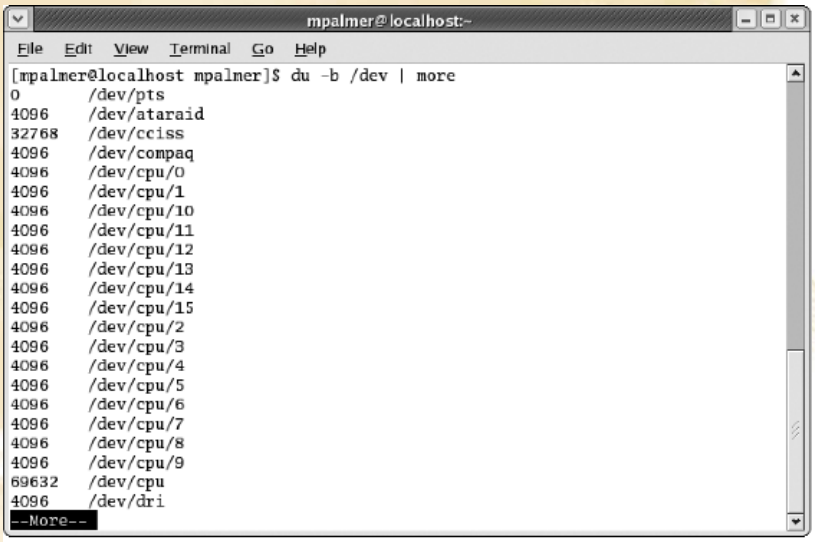


Figure 8-3 Viewing *du* information for the /dev directory

Eddie Law

Removing Garbage Files

- Garbage files are temporary files that lose their usefulness after several days
- Two examples of garbage files are core files (named `core`) and `a.out` files
- Use the `find` command to assist you in locating these files and the `rm` command to remove them
  - On the next slide, **`find`** is used to remove garbage files, and
  - The **`-exec rm {} \;`** option tells Linux to **`rm`** all files found **`{}`** by the command

Eddie Law

Removing Garbage Files (cont'd)

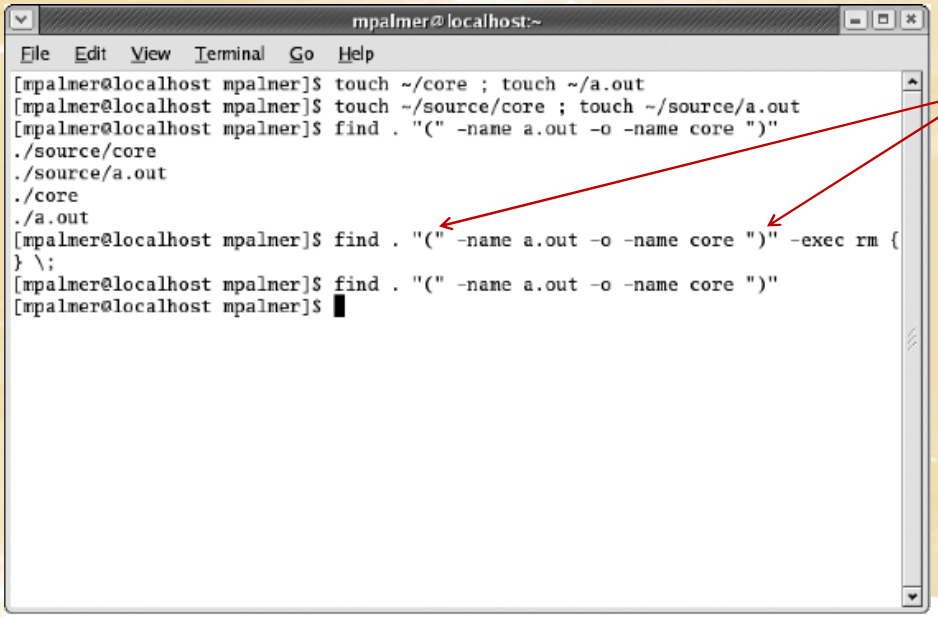


Figure 8-10 Using the *find* command to delete garbage files

die Law

Using System Status Utilities

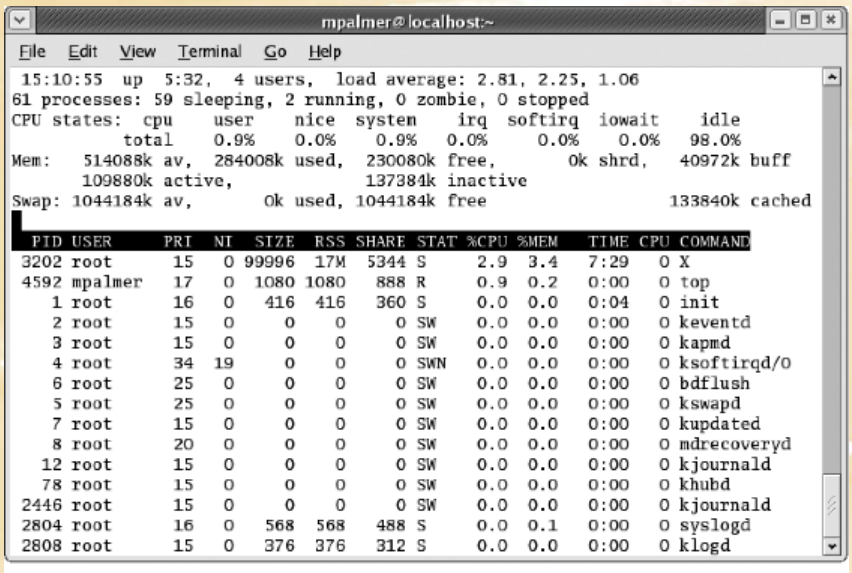
- System status commands reflect the system's performance
- System engineers primarily use the data related to system status
- Good to know how to obtain and store relevant information to send to system administrator and tune-up specialists

Eddie Law

Using the top Command

- One of the most effective utilities for auditing system performance is the top command
- The top command displays a listing of the most CPU-intensive tasks in real time
- Updates every five seconds by default
- Press "q" to quit it

Using the top Command (cont'd)



The top utility run without any options specified

Figure 8-11 Sample *top* display

Using the `uptime` Command

- Uptime tells you how long a system has been running since the last time it was booted
- Displays current time, how long the system has been up, number of users on the system, and the load average for 1, 5, and 15 minutes

Eddie Law

Using the `free` Command (vmstat)

- The `free` utility displays the amount of free and used memory in the system
- Options
  - `-b` bytes
  - `-m` megabytes
  - `-g` gigabytes
  - `-t` totals

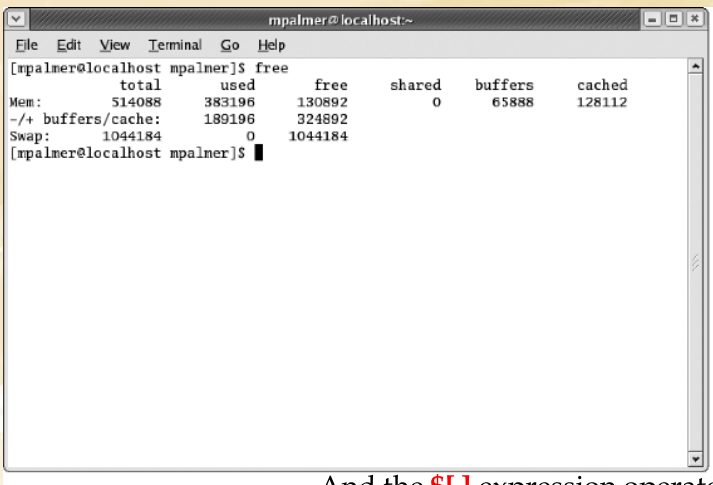


Figure 8-4 Using the *free* command to monitor memory and swap usage

And the $[ ] expression operator

And the $[ ] expression operator

Eddie Law

Forwarding top and free Output

- When problems arise with performance, may need to forward top and free output to support person
- Use redirection (>) to store outputs in files

```
$ top n 3 > topdata
```


Managing Processes

- A process is identified through a unique number called a process id (PID)
- Unix/Linux offer utilities to run, monitor, and kill processes using PIDs

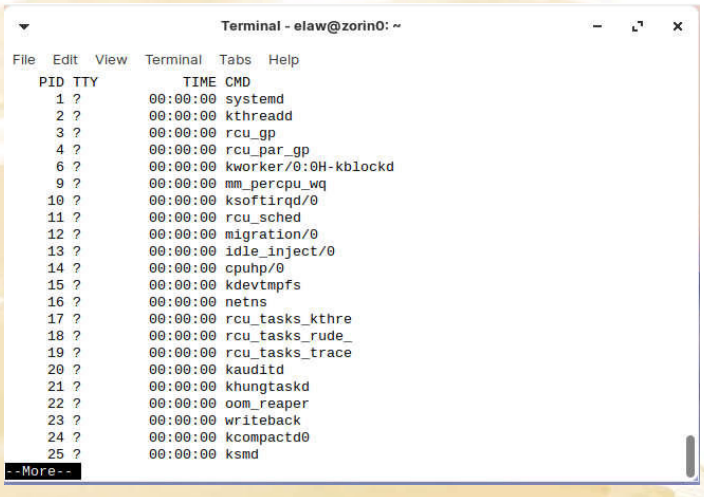
Running Processes in the Background

- Run a process in the background while working with another program in the foreground
- To run a program in the background, append the & character to end of the startup command, e.g.,

```
$ top &
```


Monitoring Processes

- The `ps` command with the `-A` option shows a list of all system processes currently running
- Command `$ ps -aux` displays all processes running in the system
- Command `$ ps f` displays processes in parent-child relationships



Killing Processes

- Administrator with root privileges can kill any user's processes
- User can kill own processes
- Use the `kill` command with the pid of the process
- Use $ `kill -9` (the sure kill) to stop a process that doesn't respond to an initial `kill` command
- If I have started executing a program (p1) that is running infinitely, I may kill that process with the following steps:

  $ ps

  (Assume the PID number of process p1 is 608)

  $ kill 608

Eddie Law

Checking the Spelling of a Document

- `ispell` scans a document, displays errors on the screen and suggests alternative spellings

Install ispell with
`$ sudo apt install ispell`

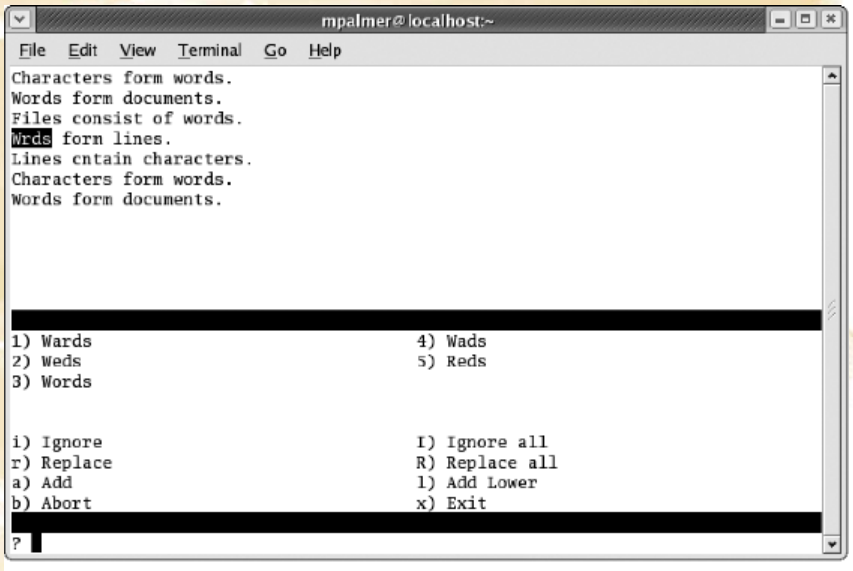


Figure 8-6 Checking the spelling in a document with *ispell*

Eddie Law

Summary

- UNIX/Linux utilities are classified into eight major functional areas
- Utility programs are "commands": executed by entering names on the command line
- `grep` and regular expression
- `dd` command options allow it to handle copies when other copying methods fail
- `df` checks and reports on free disk space
- `du` checks for disk usage
- Use `find` to retrieve temporary files and use `rm` to remove them

Summary (cont'd)

- Run a program in the background by appending & to the end of a command
- `top` and `free` provide views of the "internals" of the system that can be redirected to a file for system tune-up
- `ps` displays all running processes
- `kill` terminates a specific process
- `ispell` scans for spelling errors (requires installation)


QA

澳 門 理 工 學 院
Instituto Politécnico de Macau
Macao Polytechnic Institute

COMP 225 Network and System Administration

Notes #4: User Management

K. L. Eddie Law, PhD

Macao Polytechnic Institute
School of Applied Sciences
Academic Year 2020-2021, 2nd Semester

Topics

- Tab completion
- Users and groups
- Access and file permissions

Tab Completion

- "bash" has a shortcut that completes filenames for you
  - Start typing a path
  - Hit tab once…
- it searches and finds a file that matches
- If there are more than one match it will do nothing, if hit it a second time, then it will show all matches

Filename Globing

***** – 0 or more characters

? – any 1 character

[] – matches characters in brackets

Example: **[abc]**

Example

- Consider the following directory listing

  file1.txt file2.txt file.txt coolgame coolpictures
  vacation.txt poolpictures oolgame

- What files will the following commands match?

  ls *.txt
  ls file?.txt
  ls [c]ool*
  ls file*.*
  ls ?ool*
  ls ?????.txt

User Creation and Management

.

On Users and Groups

- Different commands for adding/deleting users/groups
- Binary executables
  - `useradd`
  - `userdel`
  - `groupadd`
  - `groupdel`
- Perl scripts (more user friendly)
  - `adduser`
  - `deluser`
  - `addgroup`
  - `delgroup`

useradd

- Linux is a multi-user system
  - A special user called **`root`** has unlimited rights
  - Normal users are "un-privileged" and their rights are limited on the system
- System administrator need to be very comfortable with creating and managing users and groups
- For simplicity, as root, run for user creation

```
$ sudo adduser newUserName
```

- Traditionally, run

```
$ sudo useradd newUserName
$ sudo password newUserName
$ sudo mkdir /home/newUserName (if needed)
```

useradd (cont'd)

useradd [**-c** *name_field]* [**–d** *home_dir*] [**–e** *expire_date*] [**–g** *group_id*] [**–s** *shell*] [-p password_hash] username

passwd *username*

usermod

- change user information once the account has been created

usermod [**–c** *name_field*] [**–d** *home_dir*] [**–g** *group_id*] [**–l** *username*] [**–s** *shell*] [**–L**] [**-U**] *username*

- For example, if there is a group called "students", run the following to add user "frank" to the "students" group

```
$ sudo usermod -aG students frank
```

/etc/passwd

- The local users are defined in the file
- For the format of the file, fields separated by ":"

  `username:password:uid:gid:name:homedir:shell`

  where
  - username – username
  - password – x in most instances
  - uid – user id, a unique user identifier number
  - gid – group id, defines the primary group
  - homedir – personal space for users account
  - shell – the users shell

/etc/shadow

- A very important file on Unix that stores users password information
- Similar to the /etc/passwd file, each line is for one user
  - Username
  - Password hash
  - Last password change
  - Days until password can be changed again
  - Days before password expires (must be changed)
  - Days warning before password expires
  - Days after password expires that account is disabled
  - Date when account expires
  - Reserved

chfn & chsh

- File /etc/passwd contains the user configuration information
- No permissions for normal users to edit this file
- There are special programs on the system that lets a user change their shell and their name entries
  - Change name and other info

    `$ chfn`
  - Change the shell

    `$ chsh`

Programs to alter password settings

$ passwd *username* – change a users password

$ chage [options] *username* – change password policies

-l or **--list**

-E or **--expiredate** *YYYY-MM-DD*

-m or **--mindays** *number_of_days*

-M or **--maxdays** *number_of_days*

whoami, logname, id, groups

$ whoami

- displays you who you currently are

$ logname

- displays who you logged in as

$ id

- displays information about your user

$ groups

- Displays information about your group memberships

Groups

- Create groups and assign users to groups
- Access files and resources can be shared among multiple people working on the same project.
- Group information is in /etc/group, with format

```
group_name:password:group_id:[username[,]…]
```

Managing Groups

 # groupadd [options] *group_name*

-g or --gid *group_id*

 # groupdel *group_name*

 # groups *username*

 # usermod --append --groups *group1[,group2…]* **username**

Note: always use the --append option; if not, the system will reset the user to ONLY be in the groups typed in the command; therefore, could accidentally remove a user from old groups!

Eddie Law

userdel

- To delete a user in the system

userdel [**-r**] *username*

-r deletes data in users home directory and mailspool

su

- A command to switch between users
- Good security practice
- Switch from one “normal user” to another (password required)
  - (for Red Hat, Fedora) `$ su -l username`
- Can become root from a “normal user”, user password needed
  - (for Red Hat, Fedora) `$ su -`
  - (for Ubuntu –root login disabled) `$ sudo su -`
- If you are already root you can become any other user without a password

Secure Shell (SSH)

- Permits us to log in a remote computer
- Apart from using “su”, we can use “ssh” to log in local computer too
- ssh is a secure replacement for the legacy “telnet” program

```
$ ssh computerName -l username
```

- ssh requires that an ssh daemon (sshd) be running on the remote host, also need the password of the user for logging in

.ssh Directory

- The `.ssh` directory holds important `ssh` files, e.g.,
  - `id_rsa` - users rsa private key
  - `id_rsa.pub` - user rsa public key
  - `id_dsa` - users dsa private key
  - `id_dsa.pub` - user dsa public key
  - `authorized_keys` - users allowed to login with using digital signatures
  - `known_hosts` - known hosts and keys

A Bit More on File Permissions

Default Permissions

- The `umask` command allows a user to change the default permissions for new any file/directory
- `umask`
  - The actual permissions are “default” permissions
  - REMOVES the specified bits from the system’s default creation permissions
- In general (for ubuntu: 0002, for fedora: 0022)
  - System default for files = rw-rw-r--
  - System default for directories = rwxrwxr-x
- To check the current umask, `$ umask`
- To change the umask, `$ umask new_removal_mask`

File Attributes

- Linux also has file attributes, they are not permissions
- Rarely used
- Lists the file attributes with

  `$ lsattr`
  - Usually shows “-e” the regular extent file system

  `$ sudo chattr +i filename`
  - Add attribute to a file, make it static, cannot be removed

  `$ sudo chattr -i filename`
  - Remove the “+i” attribute from a file

Remarks

- On users and groups
- Using ssh for remote login
- A bit more on file permissions

澳 門 理 工 學 院
Instituto Politécnico de Macau
Macao Polytechnic Institute

COMP 225 Network and System Administration

Notes #5: File System, Storage and Backup

K. L. Eddie Law, PhD

Macao Polytechnic Institute
School of Applied Sciences
Academic Year 2020-2021, 2nd Semester

Topics

- Understanding basic disk systems designs
- Managing server storage system and backup are vital
- Associated basic Linux file systems briefly discussed
- Types of backups – pros and cons
- Practicing backup and restore in Linux
  - Archival utility: tar
  - Task scheduler: crontab

Basic Storage Systems

- Magnetic drive
  - Disk specifications
    - Dimensions
    - Capacity
    - RPM (revolutions per minute)
    - IOPS (input/output per second)
    - Seek time and latency
    - Hot swappable
- SSD (solid state drive)
  - No moving parts
  - Quiet, faster, and more durable

Storage Technologies

- DAS: direct-attached storage, traditional hard drives
- NAS: network-attached storage
- SAN: storage area network (high speed sophisticated independent network)
- JBOD: just a bunch of disks (to form one large volume)

Capacity Planning Considerations

- OS growth
  - Patches
  - Service packs
  - Log files
  - Temp files
- Data growth
  - Customer data
  - Archived data
  - Recovery data

Mitigation Strategies

- Disk quotas
  - Soft quotas – users get alerts
- Compression
  - Loss of performance
  - For backups and archived data
- Regular cleanup
- Routine archival

Compress – Uncompress

- {`gzip`, `gunzip`} filename(.gz)
- {`bzip2`, `bunzip2`} filename(.bz2)
- {`xz`, `unxz`} filename(.xz)
- `zip` filename.zip filename, and `unzip` filename.zip

Legacy System

- Boot with a BIOS
- Master boot record (MBR)
- Four real primary partitions
- Accessible size up to 2 TB
- One extended partition permitted
- Logical partitions in extended partition

- MBR partition numbering

Partition Type	Partition Numbers
Primary or extended partitions	1–4
Logical partitions	5–11

Modern Systems

- Boot with a Unified Extensible Firmware Interface (UEFI)
- Use GUID (Globally Unique Identifier) Partition Tables (GPTs)
- Unlimited number of partitions (commercial products may offer up to 128 in general)
- For current implementation, e.g., for Microsoft, the accessible size of 18 exabytes partition (1 EB = 10^{18} bytes)
- The theoretical upper limit is 9.4 zettabytes partition (1 ZB = 10^{21} bytes)

Checking Hard Drives

- Hard drives are block devices

```
elaw@zorin:~$ cat /proc/partitions
major minor  #blocks  name

  11        0      59724 sr0
   8        0   25165824 sda
   8        1   25163776 sda1
elaw@zorin:~$ lsblk
NAME   MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda      8:0    0   24G  0 disk
└─sda1   8:1    0   24G  0 part /
sr0     11:0    1 58.3M  0 rom
elaw@zorin:~$
```

- Can also try out

```
$ df -h
$ sudo fdisk -l
```

Commands – Partition Tools

- Legacy systems
  - ***fdisk***
- Example, to list partitions
  - `$ sudo fdisk -l`
  - `$ sudo fdisk -l /dev/sda`
- Modern systems
  - ***gdisk***
  - ***parted***
- Run `gdisk`
  - `$ sudo gdisk /dev/sda`

Typical Linux File System

(Remark: logical volume manager (LVM) not discussed)

- Typical Linux file system formatting: Ext2/3/4
- Advantages with ext4
  - Improved file system size
  - Larger number of files
  - Support for solid-state disks
  - Journal checksumming
- Reformat the entire drive, e.g.,

```
$ sudo mkfs -t ext4 /dev/sda1
$ sudo lsblk -f
```

Typical Linux File Systems

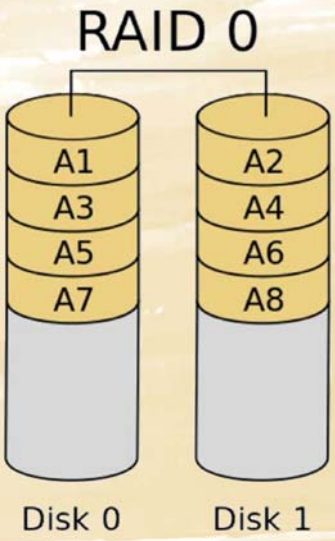
FS	Maximum FS Size	Maximum File Size	Notes
ext2	16–32 TiB	2 TiB	Not journalized
ext3	16–32 TiB	2 TiB	ext2 with a journal
ext4	1 EiB	16 TiB	Supports solid-state disks, larger disks, robust
XFS	8 EiB	8 EiB	Cannot be shrunk, supports snapshots
Btrfs	16 EiB	16 EiB	Supports automatic defragmentation, copy-on-write, RAID, subvolumes, online data correction, snapshots

RAID

Redundant Array of Independent Disks

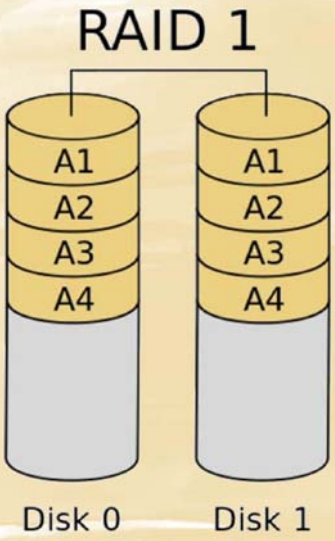
RAID 0

- Disk striping
- No fault tolerance
- Minimum two disks
- Increased read performance
- 100% drive space utilization

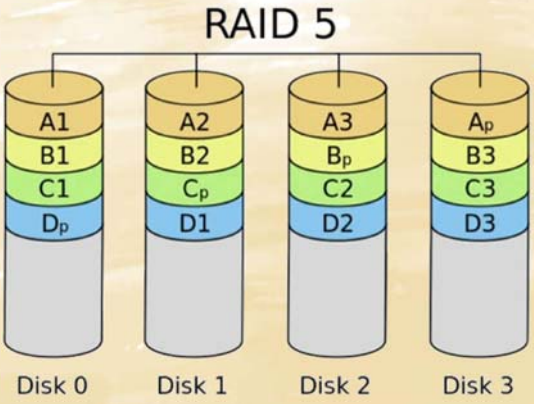


RAID 1

- Disk mirroring
- Exactly two disks
- Increased read performance
- 50% drive space utilization
- Provides fault tolerance in of single-drive failure



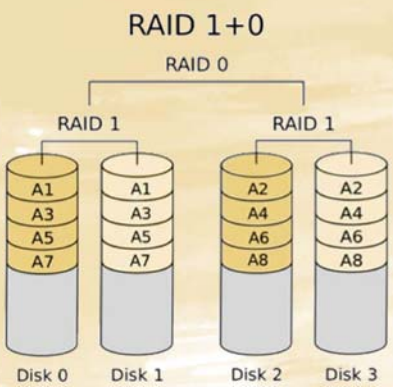
RAID 5

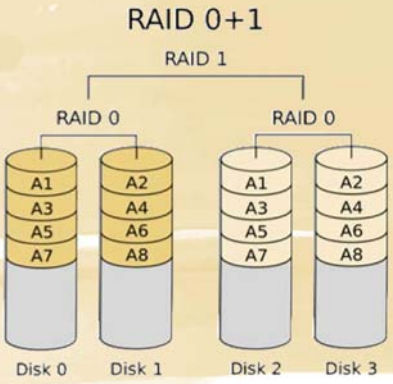


- Disk striping with parity
- Minimum three disks
- Increased read performance
- Efficient disk space utilization
  - 75% disk space utilization if 4 disks are used
  - 90% disk space utilization if 10 disks are used
- Provides fault tolerance in case of single-drive failure

Putting RAIDs Together

- RAID 1+0 (or RAID 10)
  - Two or more mirrors that are striped
- RAID 0+1 (or RAID 01)
  - Two stripes that are mirrored





Disasters and Backup

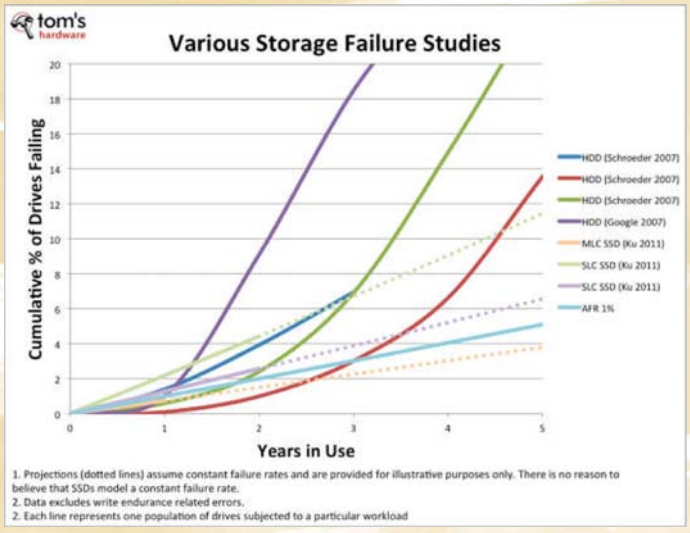
Disaster Recovery

- Business continuity plan
- Disaster recovery plan
- Business impact analysis

Replication Methods

- Disk to disk (disk mirroring)
- Server to server (failover clustering)
- Site to site
- Types of sites
  - Hot site
  - Cold site
  - Warm site

Why Backing up Files?



- With "backup and restore" operations, can protect data from loss
- Backup offers more than one copy of system files
- Purpose of restoration is to recover data that is temporarily unavailable due to some unexpected event

Is It Good to Do Backup?

- Backup is not free
- No backup is risky
- Nowadays, in fact, cost of backing up is diminishing, value of data is growing

Factors on Making Backups

- Determine which data is critical
  - Recovery point objective (RPO) – How much data can be lost? An hour, a week of data?
- Determine frequency and types of backups to be used
  - Recovery time objective (RTO) – How fast should data be recovered? Can we continue to operate without recovering data for a day, a week…?
- Determine categories of data, and schedule the backups accordingly

Factors on Making Backups (cont'd)

- Determine which data is static and which is dynamic
  - Some OS installations are changed infrequently, i.e., few backups required
  - Some applications may require continuous backups, e.g., online market places
  - Understand the changing state of your client's data to determine an appropriate backup schedule
- Determine appropriate backup storage media:
  - CD/DVD
  - Tape
  - Hard Disk
  - Online

On File System Backup

- Backing up user and system files on a single-user Linux system is a good routine operation of an ordinary user
- For complex multi-user systems, it is ***a necessary procedure*** for anyone responsible for the administration of the system
- As a system administrator, an easy-to-remember set of considerations is in the form of ***How***, ***What***, ***Why***, ***When***, ***Where***, and ***Who***?

Questions on System Backup

- ***"How"*** – the commands, utilities, applications, or combination of hardware and software to accomplish the backup and archive; the strategies such as incrementally, in a rolling fashion, or across the entire file system structure totally, etc.
- ***"What"*** – the selected data for backup, such as user files, user account files, certain kinds of documents, the whole disk drive, multiple disk drives, subset or all of system files, etc.
- ***"Why"*** – the decisions on the relative importance of *"What"* for backing up

Questions on System Backup (cont'd)

- ***"When"*** – how often to backup, e.g., hourly, daily, once a week, once a month, or at what time to save a particular file, etc.
- ***"Where"*** – the locations of the backup data, e.g., local disk, Cloud storage, a USB thumb drive, another computer or Network Attached Storage (NAS), etc. manually or automatically, etc.
- ***"Who"*** – the backup is carried out by a person, an automated software, or an automated process through the Cloud vendors, etc.

Proper Backup Procedure

- Choose your application
- Scheduling
- Implementation
- Inventory (content and media)
- Verify
- Automate
- Secure

Backup Choices

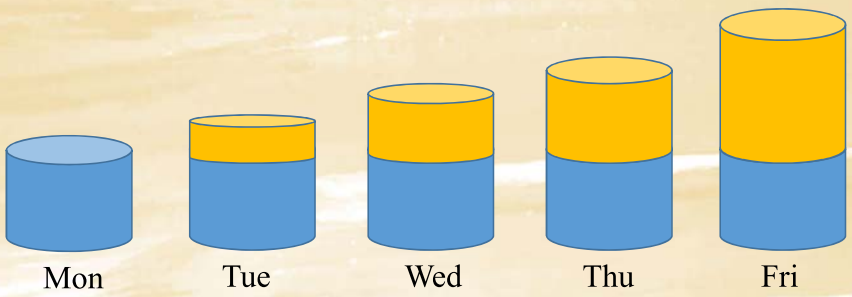
- **Full or normal backup** – all data is backed up and the ***archive bit*** is reset
- **Copy backup** – all data is backed up, but the *archive bit* is not reset
- **Incremental backup** – all data that has been changed since the last full or incremental backup is backed up, and the ***archive bit*** is reset
- **Differential backup** – all data that has been changed since the last full backup is backed up, and the ***archive bit*** is not reset
- Among file's attributes (or metadata), one bit called "***archive flag***"
  - This flag informs the backup program about which files need backing up
  - 0 for has been backed up recently; 1 for needs to be backed up

Full Backup

- Pros
  - Provides a complete copy of all files
  - Easy to manage
  - Done less frequently than other types of backups due to cost and resource requirements, e.g., monthly, quarterly, semi-annually, annually
- Cons
  - Usually requires more media space than either differential or incremental
  - Takes a long duration to recover the full backup to a new disk

Differential Backup

- Copy modified files since the last ***full backup***
- Differential backups grow with time, they can eventually grow larger than the last full backup
- Scheduled more frequently than a full backups: Weekly, monthly

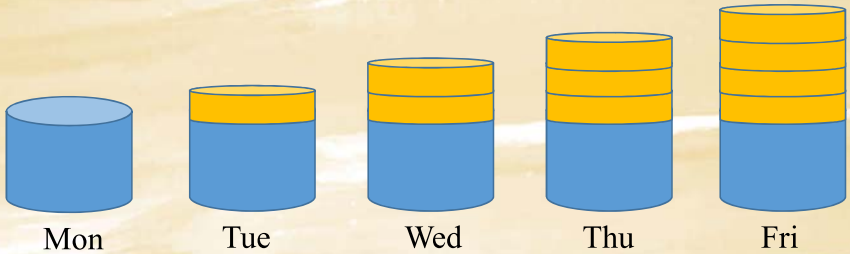


Differential Backup (cont'd)

- Pros
  - Redundancy
  - In general, takes up less time and space than a full backup
  - If the differential backup grows to the size of the last full backup, then schedule a new full backup
- Cons
  - Redundancy – potentially many unneeded copies of the same data
  - Subsequent differentials take longer and use more media space

Incremental Backups

- A backup of what has changed since using any type of backup the last time
- Frequency of incremental backups depends on the client needs: weekly, daily, hourly, continuously



Incremental Backups (cont'd)

- Pros
  - Keeps a revision history of actively changing files
  - Fastest backup type
  - Uses the least amount of media to complete a single backup
- Cons
  - Much more difficult to manage
  - Failure in the chain of backup?

Backup Strategies

- Full backups only
  - Slowest backup, fastest recovery
- Full and incremental backups
  - Once a week a full backup, each weekday an incremental backup
  - Quick backup daily, could be lengthy recovery
- Full and differential backups
  - Once a week a full backup, each weekday a differential backups
  - Backup and recovery times in-between the other two

Example of Scheduling

- Full backup twice per year
- Differential each first Saturday morning of each month that is not scheduled for a full backup
- Incremental each Saturday morning that is not scheduled for a Full or Differential

Backup Considerations

- Backups slow down service
  - Files should be write-locked during backup
- Avoid doing backups during peak service hours
- Schedule during early AM hours on the weekend and holidays

- Other schedule considerations
  - Consider completing a backup in conjunction with and before any major system changes are scheduled

Backup Automation

- Automation reduces human errors
- Many pre-packaged applications include automatic scheduling
- Linux/Unix backup scripts can be submitted using the `cron` utility
- Logs can be kept in `/var/log`, and e-mail can be sent to the admin

Data Compression

- Risks – if the media is damaged, recovery may be difficult or impossible
- Lossy
  - Some data tolerates degradation (loss of information)
- No-loss
  - Some data should not be compressed. Know your data!

Restoration of Data

- Common reasons for restores
  - Accidental file deletion
  - Disk failure
  - Disaster recovery: fire, flood, earthquake, hacker attack, sabotage, terrorist attack, etc.

Files and Times

- Three different times for each file (in Linux)
  - *mtime* - modification time; this value is changed when the content of the file is changed
  - *atime* - access time; the value of this is changed when the file is accessed. The atime can change when a backup utility or script read the file as well as when a use reads the file
  - *ctime* - change time; the value is updated whenever the attributes of the file change. This can be ownership or permissions
- Note: file system backups change *atime* while raw device backups will not. If implementing incremental or differential backups, this is important

Choose your Backup Tools: Examples

- Many commercial apps are available for different OSes
- Linux/Unix

Linux File Backup Facilities

Backup Facility	Description
tar	Command and options to pack a file or a directory hierarchy as an ordinary disk file for backup, archiving, or moving to another location or system. gtar is the Gnu version
cpio	Less popular than tar, but with much of the same functionality
rsync	A disk space-efficient command to copy files and directories
dd	A simple and abbreviated backup utility
zfs snapshot	Built-in commands and options in zfs that offer a variety of backup modes
Script files	Administrator or user-written shell scripts or other programming language backup systems, that can use all of the earlier commands in them
3rd party software	Many products, both local and online. Two examples that are most significant for ordinary use are Clonezilla and Filezilla

`tar` - Linux Backup Utility

- `tar` (tape archiver) is a powerful backup and restore utility
- Most Linux files are downloaded as compressed .tar files

Common Options for tar

Full optional operation name

The short form

-c --create	creates a new archive
-v --verbose	lists the files being processed
-x --extract	extracts/restores the archived file
-r --append	adds the single file or directory to the archive
-p –preserve-permissions	extracts information about file permissions
-f --file	specifies the name of archive file or device location
--acls	enables POSIX ACLs that the directory has
--xattrs	enables extended attributes support

Data Compression for tar

- There are different compression algorithms, the popular ones are
  - -z --gzip --gunzip : compress or decompress using gzip function
  - -j --bzip2 : compress or decompress using bzip2 function
  - -J --xz : compress or decompress using xz function
- Keeping all original attributes (including security setup in Ses//Linux)
  - --xattrs

Examples

- The simplest command to create an archive file from a directory

```
$ tar --create --verbose --file archive_name.tar
directory_name
```

- Backup an entire computer

```
$ sudo tar -cvpzf backup.tar.gz --exclude=/mnt /
```

- Backup up content of a web site excluding the video files

```
$ sudo tar -cvpzf wwwbackup.tar.gz
--exclude=/var/www/video /var/www
```

- Restore files

```
$ sudo tar -xvpzf wwwbackup.tar.gz -C /recover
```

Change to directory

Job Scheduling with crontab

- Many administrative tasks must be done frequently and regularly
  - Rotating log files, backing up data
- In Linux, running jobs at regular intervals is managed by cron facility
  - Consists of the crond daemon and a set of tables describing what work is to be done and with what frequency
  - The daemon wakes up every minute and checks the crontabs to determine what needs to be done
  - The crond daemon is usually started by the init process at system startup
- Use the crontab command

Job Scheduling

- How often to do a job?
- Many administrative tasks must be done frequently and regularly, e.g.,
  - Rotating log files, backing up data
- In Linux, running jobs at regular intervals can be managed with the `cron` (Ubuntu) or `crond` (Red Hat) facility
  - A `crond` daemon and a set of tables describing what work to do and its frequency
  - The daemon wakes up every minute and checks the `crontabs` to check what to do
  - The `crond` daemon starts when the system boots and runs as long as the system is up

Use the `crontab` Command

- A `cron` configuration file is "`crontab`" which we call it ***cron table***
- Containing lists of command lines and times at which they are to be invoked
- `Crontabs` for individual users are stored under `/var/spool/cron` (Linux) or `/var/cron/tabs` (FreeBSD)
- There is at most one `crontab` file per user

Editing `crontab`

- To create or edit a `crontab`, use the `crontab` command with the option `-e` (for "edit")

```
$ sudo crontab -e
```

- We can select our editor of choice using the command

```
$ sudo select-editor
```

  Or it may ask you upon creating a `cron` table at the first time

- Each `crontab` entry contains six fields
  - Comments are introduced with a pound sign (#) in the first column of a line
  - Each non-comment line contains ***six fields*** and represents one command

minute hour dom month weekday command

The Fields

1. Minute of the hour (0-59)
2. Hour of the day (0-23)
3. Day of the month (dom) (1-31)
4. Month of the year (1-12)
5. Day of the week (0-6 for Sun, Mon, Tue, Wed, Thu, Fri, Sat)
6. String to be executed by bash

- For each time-related field
  - ***A star***, which matches everything
  - ***A single integer***, which matches exactly
  - ***Two integers separated by a dash***, matching a range of values
  - ***A range followed by a slash and a step value***, e.g., 1-10/2
  - ***A comma-separate list of integers or ranges***, matching any value

Examples

- `45 10 * * 1-5 [a bash command]`
  - "10:45 a.m., Monday through Friday."
- `0,20,40 22-23 * 7 Fri-Sat /home/ian/mycrontest.sh`
  - Runs `mycrontest.sh` shell script at 10 pm, 10:20 pm, 10:40 pm, 11 pm, 11:20 pm, 11:40 pm, in July on Fridays and Saturdays

Automatic Backup

- Combining `crontab` and `tar` to make an automatic backup
- Back up www web site files every minute (crazy!)

```
* * * * * sudo tar –cvpzf /backupfolder/wwwbackup.tar.gz
/var/www
```

- Back up www web site files at 3 am on Tuesdays

```
0 3 * * 2 sudo tar –cvpzf /backupfolder/wwwbackup.tar.gz
/var/www
```

Remarks

- Discussed backup and “crontab”
- Run command “`at`” is for scheduling single event
- Logical volume group – good for scalable file systems (not discussed though)

澳門理工學院
Instituto Politécnico de Macau
Macao Polytechnic Institute

COMP 225 Network and System Administration

Notes #6: Network Basics

K. L. Eddie Law, PhD

Macao Polytechnic Institute
School of Applied Sciences
Academic Year 2020-2021, 2nd Semester

Topics

- Networking models and the Internet
- International Standard Organization
- Layered architectures

Some Definitions

- **Computer network**: a collection of computers that are connected in such a way that data can be exchanged among any two computers
- **Intranet**: within an enterprise, a collection of networks that are mostly interconnected by ***switches*** and/or ***routers***
- **Internet**: a world wide internetwork operates with different IP-based (Internet Protocol) technologies, standardized through the Internet Administration Board (IAB), and a multitude of low-level (e.g., hardware level) communication technologies

Why Internetworking?

- Standalone computing devices might have limited usefulness
- Solution: networking and internetworking
  - Interconnecting and collaborating numerous units for communications and offer better service solutions

An Internetwork

- No single networking technology best for all needs:
  - Heterogeneity is inevitable
  - With heterogeneous network technologies, come different types of physical network designs
  - Create protocol standards to make resulting system ubiquitous
  - With recommended **standards**, the resulting networks created is the ***Internet***

Benefits of Computer Networks

- Resource Sharing: include hardware devices such disk storage and printers and software (data + programs)
- Higher Reliability: obtained by duplicating devices and replicating data
- Incremental and cheaper growth
- Distributed processing: client/server (e.g. web browser/web server), clustering, and Cloud Computing
- Promote communication among network users through resource sharing

Design Goals of Networking

- Two computers exchange data with the scenarios such as
- They may be located in the same room or separated by thousands of miles
- They are from different manufacturers and run different operating systems
- They may use different byte/bit ordering in multi-byte data and may use different character encodings for text data
- Design must allow for continuing use of existing technology while embracing new technology

Organizations for the Internet

- **IAB** (Internet Architecture Board) – manages ISOC (Internet Society)
- **IETF** (Internet Engineering Task Force) – standards creation body under ISOC
  - The RFC Editor, IESG (Steering Groups) and IETF Working Groups are responsible for reviewing and publishing new standards
- **ICANN**: Internet Corporation for Assigned Names and Numbers
  - For IP address space allocation, protocol parameter assignment, domain name system management, and root server system management functions
  - **IANA**: Internet Assigned Number Authority - information on domain names and application numbers
- **InterNIC**: Keeps information about domain-name registration

Request for Comments (RFCs) at IETF

- IAB maintains a list of RFCs for different protocol suites
- Possible assigned protocol states: Standard, Draft Standard, Proposed Standard, Experimental, Informational, Historic
- Possible protocol status: Required, Recommended, Elective, Limited Use, Not Recommended
- Freely Available: **https://www.ietf.org**
- New protocol proposals presented through *Internet Drafts (IDs)*
- A thoroughly reviewed ID can become an RFC, or
  - Update or obsoleted (when a new RFC number issued for it), or
  - Simply disappeared if no updates or failed to become RFC

Some Internet Standards

When a protocol is standardized, it is assigned a *Standard Number (STD)*, with references associated RFCs

Very important Internet standards, e.g.:

- STD 1 -- Internet Official Protocol Standards
  - State and status of each Internet protocol or standard
  - Issued by the IAB approximately quarterly
- STD 2 -- Assigned Internet Numbers
- STD 3 -- Host Requirements
- STD 4 -- Router Requirements
  - Requirements for IPv4 Internet gateway (router) software
  - RFC 1812 -- Requirements for IPv4 Routers

Standards and Protocols

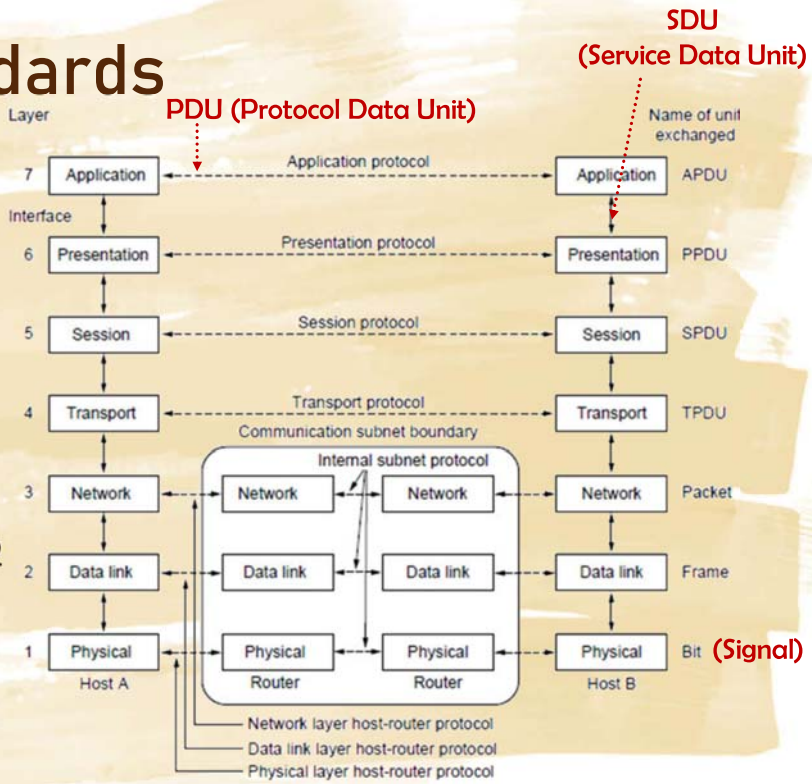
- A standard is an agreed-upon specification for some types of product or service
  - Examples: A4 paper size, 2-feet florescent light, serial (RS-232C) and parallel port interfaces
- A protocol is an agreement (contract) between two (or more) parties to conduct a joint task
  - Examples: two persons handshaking, traffic light

The Beginning of the Internet

- The Advanced Research Projects Agency Network (ARPANET), by the US Department of Defense, was the precursor to today's Internet
- In 1969, ARPANET interconnected *the University of California Los Angeles* (*UCLA*), the Augmentation Research Center at Stanford Research Institute (SRI), the University of California Santa Barbara (UCSB), and the University of Utah
  - *FYI, Prof.* ***Leonard Kleinrock*** *at UCLA is a Honorary Professor with School of Applied Sciences at MPI*
- The University of California at Berkeley (UCB) incorporated the TCP/IP software protocol in their BSD Unix
- Nowadays, ubiquitous across the Earth planet

International Standards Organization

- ISO – another international standard organization
  - Mainly for telecommunication standards
- OSI (Open System Interconnection) 7-layered model
  - ISO layer 2 can be divided into 2 sublayers
  - LLC (logical link control)
  - MAC (medium access control)



Principles of Layering

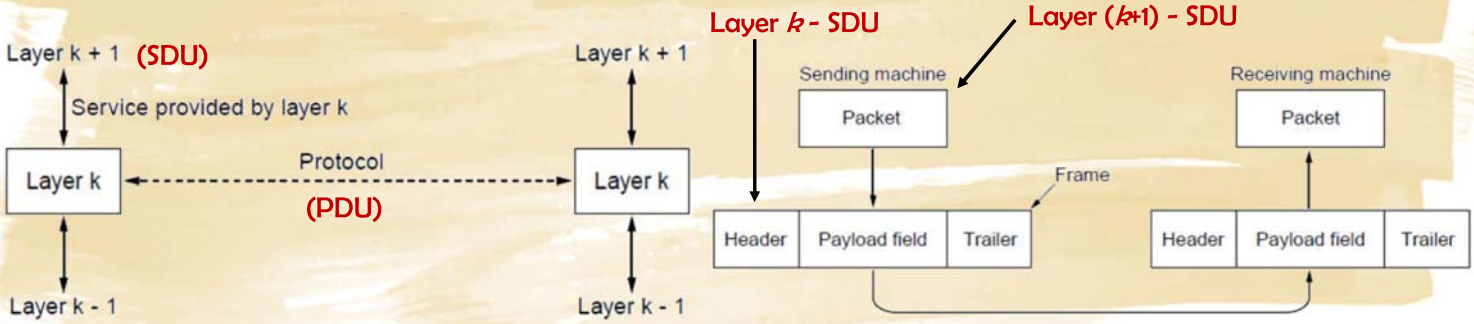
- Each layer should represent a different abstraction level, wherever needed
- Each layer should define well defined functions
- Information flow between layers should be minimum
- Number of layers should be optimum, not too many not too few

Principles of Layering (cont'd)

- Layered Protocols are designed such that the ***layer n at the destination*** receives exactly the same object sent by the ***layer n at the source***
- Protocols are standards that specify how data is represented when being transferred from one machine to another
- Protocols specify how the transfer occurs, how errors are detected, and how acknowledgements are passed

Interface Services between Layers

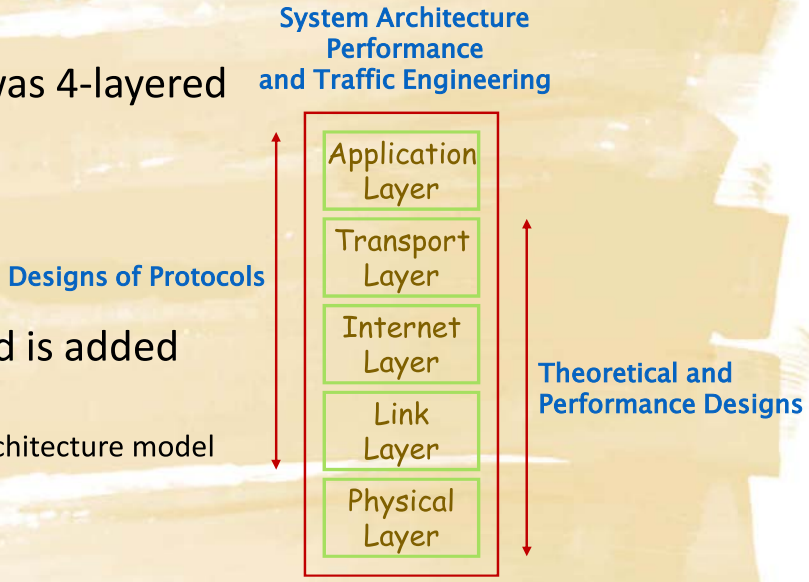
- Encapsulation
  - Data sending down the protocol stack
  - Each layer adds to the data by prepending headers
  - Only the layer 2 SDU, trailer is appended; and layer 2 SDU is generally called "frame"



SDU (Service Data Unit)

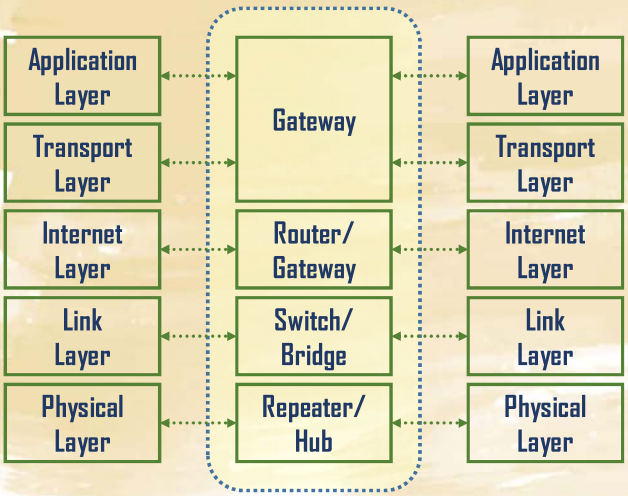
5-Layered TCPIP Model for the Internet

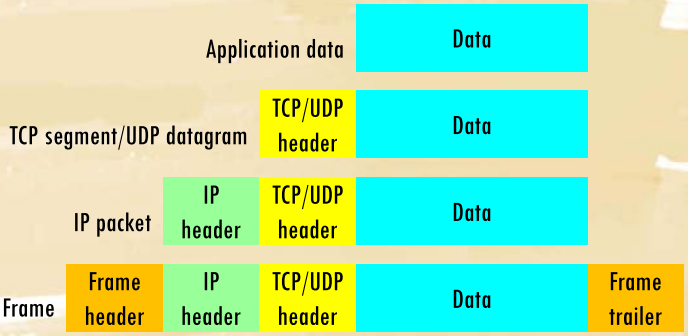
- Originally, the Internet model was 4-layered
  - Application
  - Transport
  - Internet ≡ ISO's Network Layer
  - Link ≅ ISO's Data Link Layer
- For completion, physical layered is added
  - The 5-layered model
    - The generally accepted Internet architecture model



Device Components and Packet Naming

- "Packet" sometime is a generic term used across the layers





Physical Layer

- Interfaces translated upper layered frames into analog signals over the physical medium channel
  - Channel coding – signal streams belong to different users or organizations
  - Different protocols have different channel coding designs, e.g., OFDM (Orthogonal Frequency Division Multiplexing)
  - Channel capacity (bits/second) is associated with the Shannon Theorem
  - Signal modulation and demodulation – specifies bit to signal encoding such as voltage levels and duration for bit 0 and bit 1

Types of Networks

- Body/Personal area networks (BANs/PANs)
- Local-area networks (LANs)
- Metropolitan-area networks (MANs)
- Wide-area networks (WANs)
- The classification could be based on
  - Geographical area?
  - Number of users?
  - Number of computing devices?
  - Equipment cost?
  - Etc.

Typical Classification of Networks by Distances

1 m	Square meter	Personal area network
10 m	Room	Local area network
100 m	Building	
1 km	Campus	
10 km	City	Metropolitan area network
100 km	Country	Wide area network
1000 km	Continent	
10,000 km	Planet	The Internet

The TCP/IP Model

- TCP/IP (Transmission Control Protocol/Internet Protocol)
- TCP/IP is the common name of the protocol suite used on the Internet
  - Simpler than ISO-OSI Model
  - Provides an elegant solution to world wide data communication
  - The TCP flow control designs were successful disregard the amount of traffic jamming a connection (started by Van Jacobson)
  - Majority of the wireline protocols used TCP and IP protocols
  - Applications interface with TCP layer to communicate with other peer applications

The TCP/IP Model (cont'd)

- Some newer wireless technologies for IoT less likely adopt TCP today
  - E.g., BLE (Bluetooth Low Energy)
- IETF are open standards, freely available, and independent of any hardware platforms

TCP/IP Features

- Work independent of any network hardware
  - TCP/IP works on many types of networks (Ethernet, Token Ring, X.25, dial-up)
  - TCP/IP works independent of the scales of the networks: LANs, WANs, etc.
- A common IP addressing scheme
  - Every host on the Internet has a unique address
- Standardized high-level protocols for world wide available network services

Network Communication Protocols

- Generally, a **network communication protocol** (at the application or lower layers) specifies the format and meaning of messages that are exchanged between two peer entities at the same layer
- A network protocol specifies the control header and its interpretation
- Examples:
  - **Application Layer Protocol**: HTTP is used between web browser and server
  - **Internet Layer Protocol**: Internet Protocol (IP)
  - **Link Layer Protocol**: Point-to-Point Protocol (PPP) - used between two routers connected by phone lines

Link Layer

- Through the network address to frame address mapping, this layer encapsulates an incoming **packet** into a new **frame**, and sends it through a specific outgoing port/link
- A transmitted frame is
  - Usually appended with a trailer which contains checksum for quick error detection
  - Delimited with start and end markers (block of bits)
- Channel access control and frame retransmissions may be carried out due to corruption or frame loss, especially, over shared media or broadcast channels

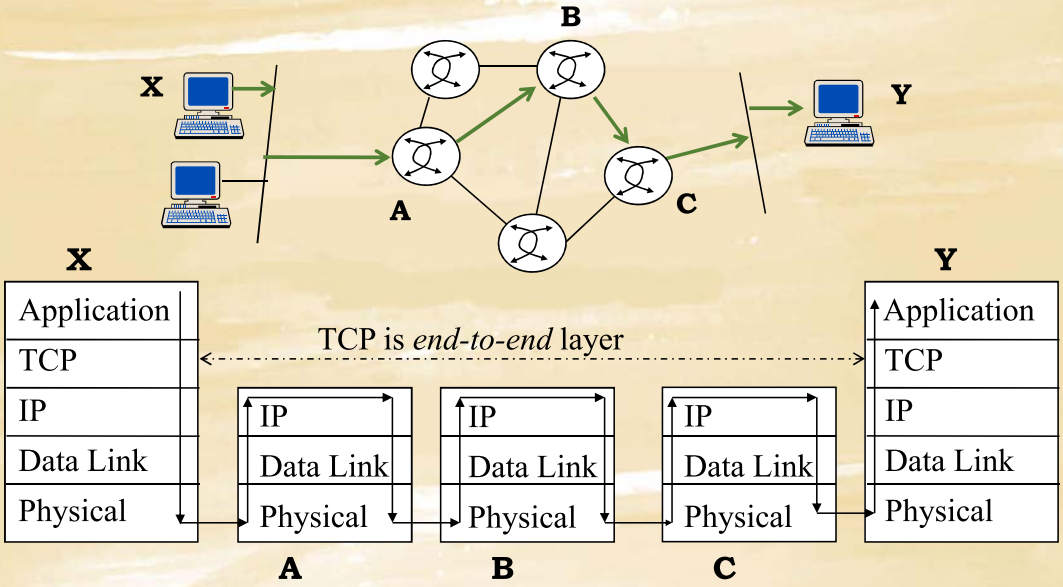
Link Layer – MAC Standards and Technologies

- Ethernet is the most widely used wired hardware
  - 802.3 standard: CSMA/CD (Collision Sense Multiple Access/ Collision Detect)
  - Ethernet (*inventor: Robert Metcalfe*) was a shared wire technology developed originally as 1 or 10 Mbps by Digital, Intel and Xerox
  - 100 Mbps, 1 Gbps, 10 Gbps Ethernet cards widely available
  - 60 Gbps cards coming soon
- WiFi is the most widely used wireless interface
  - Modelled on Ethernet, WiFi is IEEE 802.11 standard based on CSMA/CA (CA: Collision Avoidance)
  - Bandwidth for wireless links expands rapidly recently

Internet Layer

- All hosts (are supposed) to have unique Internet Protocol (IP) addresses on Earth
- IP enables end-to-end connections from sources to destinations through IP routing
- Routing of packets over the Internet
- The layer encapsulates incoming TCP segments into a new IP packet
- Provides software abstraction independent of communication hardware (addresses at link layer are usually permanently fixed in hardware, network layer address are dynamically configured through software)

IP-Layer Packet Routing



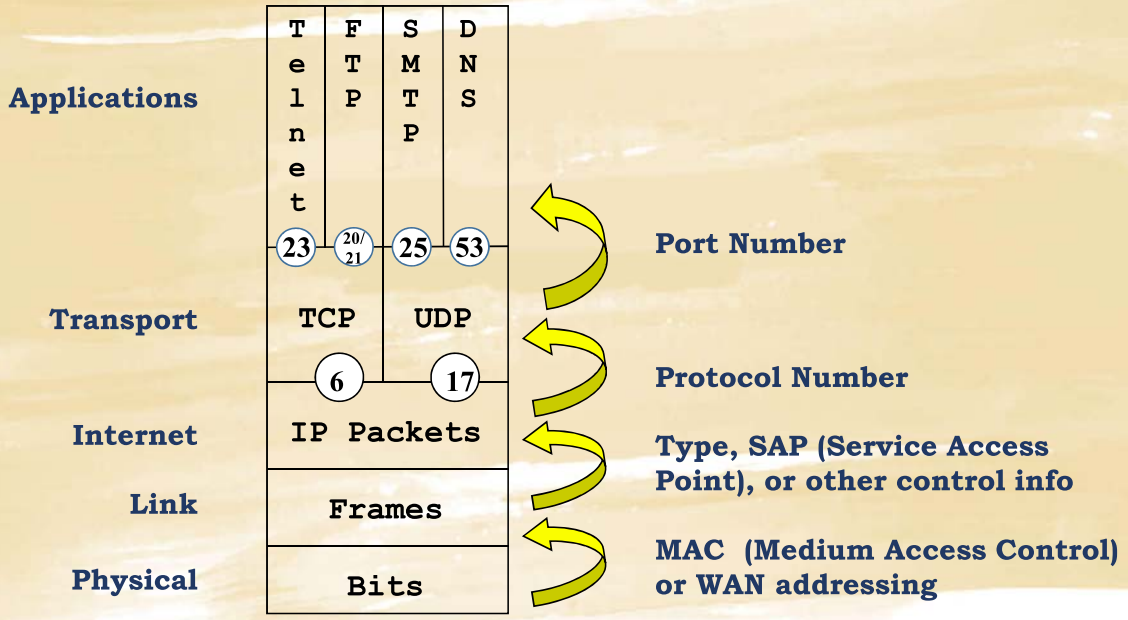
IP Layer – Summary

- Heart of TCP/IP
  - Provides basic packet delivery service on which TCP/IP networks are built
  - All IP packets are also called datagrams because of its original connectionless designs – i.e., packets are accepted to arrive out of order or using different network paths
- Main functions
  - Defines packet format, basic unit of transmission on the Internet
  - Provides Internet addressing
  - Routing of packets/datagrams

Transport Layer

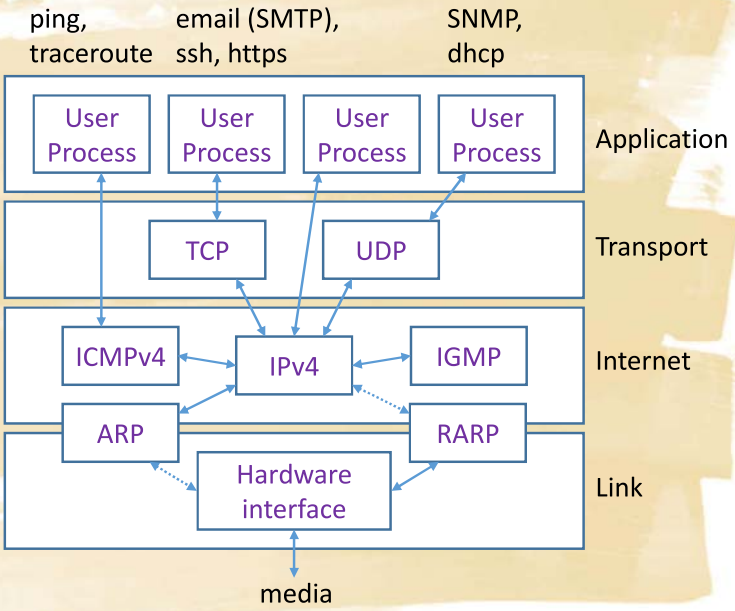
- IP for end-to-end device connection
- Transport layer permits application-to-application communications
- Two popular protocols:
  - TCP (Transmission Control Protocol) uses flow control mechanisms, and sophisticated state machines for connections for re-ordering out-of-order packet arrivals before forwarding to higher layer
  - UDP (User Datagram Protocol) – no flow control, a simple application-oriented connectionless technology

Layer Decapsulation: Examples



Application Layer

- Software programs use the Transport Layer protocols for delivering data messages
- Examples (text-based – unsecured applications)
  - Telnet: Network Terminal Protocol
  - FTP: File Transfer Protocol
  - SMTP: Simple Mail Transfer Protocol
  - DNS: Domain Name Service
  - HTTP: World Wide Web



Remarks

- Covered the concept of layered architecture
- Popular 5-layered TCPIP model


QA

澳 門 理 工 學 院
Instituto Politécnico de Macau
Macao Polytechnic Institute

COMP 225 Network and System Administration

Notes #7: IP Addressing

K. L. Eddie Law, PhD

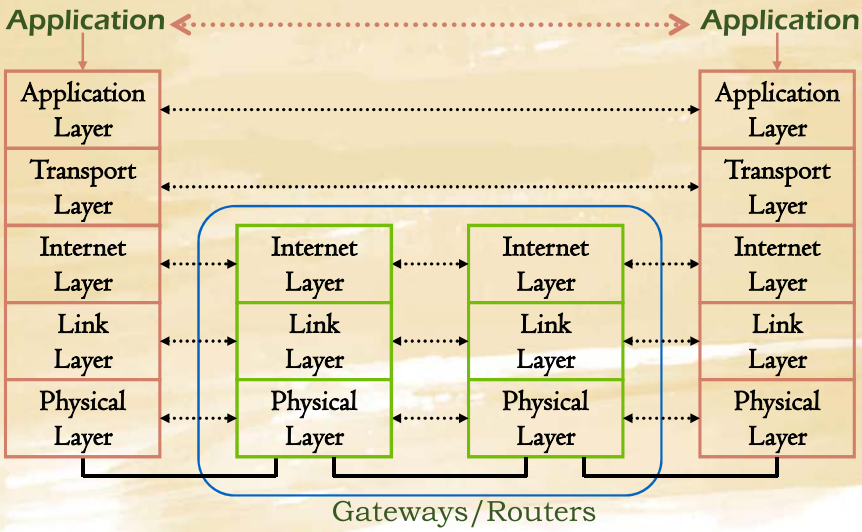
Macao Polytechnic Institute
School of Applied Sciences
Academic Year 2020-2021, 2nd Semester

To Cover

- Internet Protocol (IP) naming and addressing
- Format and basic operations
- Address resolution
- IP packet control

Internet Model

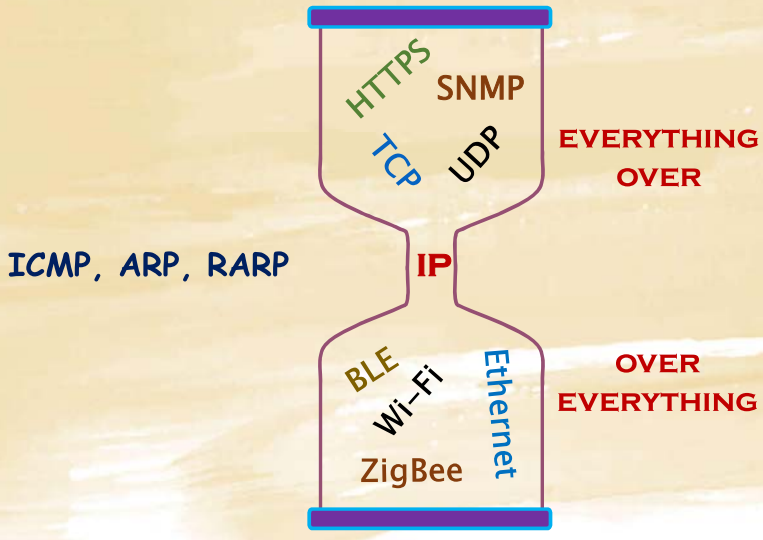
- The traditionally five-layer model for the Internet



Eddie Law

TCP/IP Protocol Suite

- The layered architecture is like an hourglass model

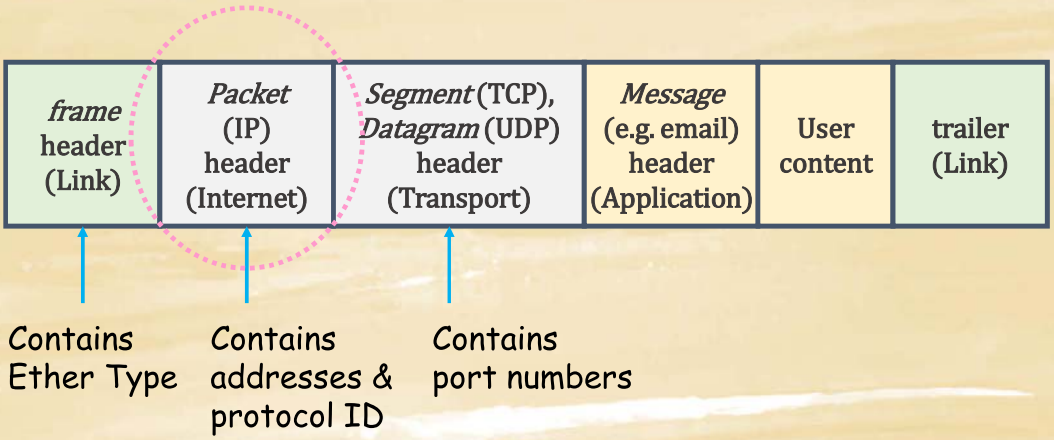


Eddie Law

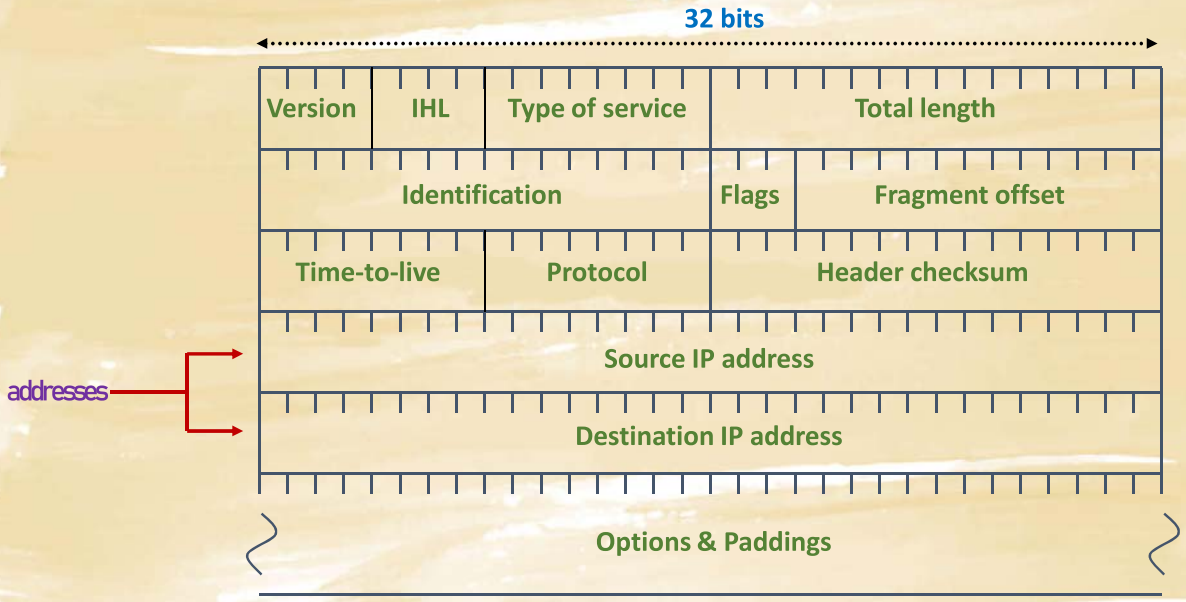
The Internet Layer

- Basic functions:
  - Addressing
  - Routing with sub-networks resolution and forwarding!!
- Traditionally
  - Connectionless
  - Serving best effort traffic: unreliable and no performance guarantees
- Today, in addition to best effect traffic
  - Traffic type classification
  - QoS routing

Typical Structure of a Frame



IPv4 Header Format



IP Addresses

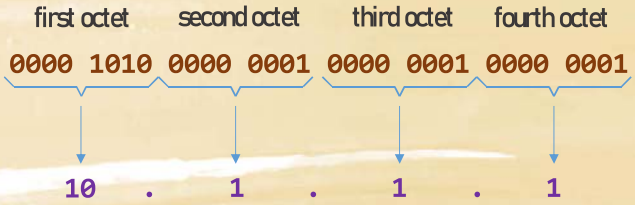
- An network ID reveals the location of an entity on the Internet, and routers know how to route a packet there
- Should be unique to identify each end host
- For host-to-host connections:
  - 32-bit IPv4 addresses:
    4,294,967,296 or 2^{32} (about 4 trillion) # of addresses
  - 128-bit IPv6 addresses:
    340,282,366,920,938,463,463,374,607,431,768,211,456 or 2^{128} (about 340 undecillion) # of addresses

IP Address Assignment

- The Internet Assigned Number Authority (IANA)
  - Assigns IPv4 addresses to the five Regional Internet Registries (RIRs) in /8 address blocks; IANA is also responsible for allocating IPv6 address space to RIRs
- Five RIRs are
  - African Network Information Centre (AFRINIC): Africa
  - American Registry for Internet Numbers (ARIN): United States, Canada, some parts of the Caribbean region, and Antarctica
  - Asia-Pacific Network Information Centre (APNIC): Asia, Australia, New Zealand, and neighboring countries
  - Latin America and Caribbean Network Information Centre (LACNIC): Central America, South America, and most of the Caribbean region
  - Réseaux IP Européens (RIPE) Network Coordination Centre: Europe, the Middle East, and Central Asia

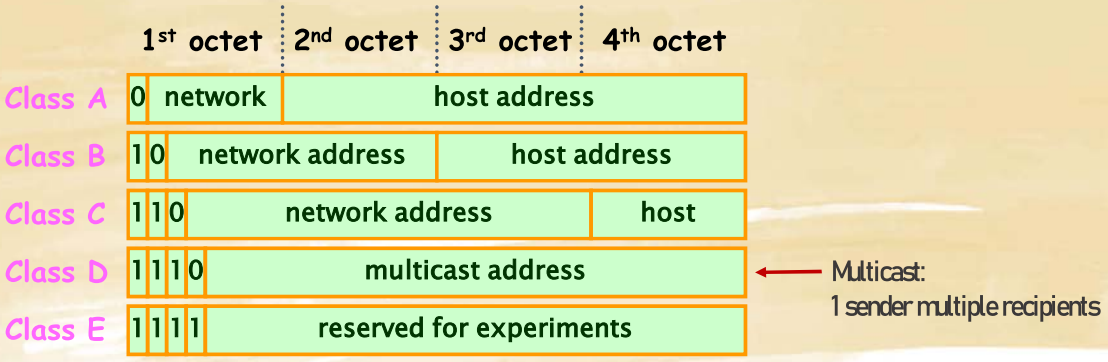
IPv4

- Starting from IPv4 – since early 1980s
- Latest version: IETF RFC 791, Sept. 1981
- 32-bit address space represented in dotted-decimal notation
- Dotted decimal notation: *x.y.z.n*
  - where *x*, *y*, *z*, *n* ∈ {0,..,255}
  - e.g. 128.100.10.2



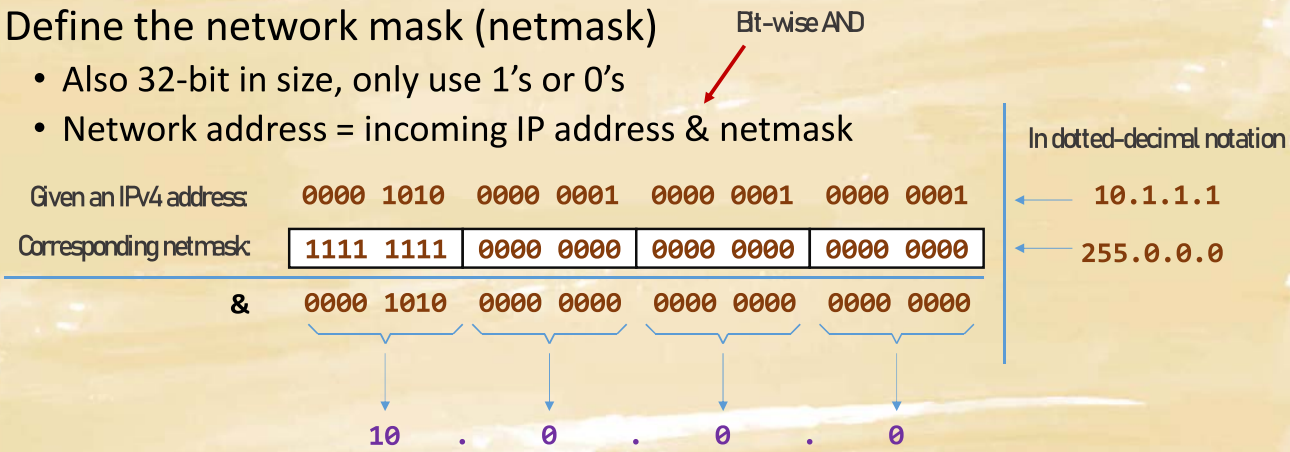
Address Classes (IPv4)

- Initially and historically, IPv4 address was classified into two parts
  - Network address and host address
- There were 5 classes, and only unicast classes *A* to *C* are mostly used



How to Identify the Network Address?

- Define the network mask (netmask)
  - Also 32-bit in size, only use 1's or 0's
  - Network address = incoming IP address & netmask



- Result: 10.0.0.0 – class A address

Old Way to Determine Netmask

- The initial few bits of the first octet of the incoming IP packet determines both the network address and netmask

Class		Size of Network Address	Netmask
A	**1st bit = 0**	8-bit	255.0.0.0
B	**1st two bits = 10**	16-bit	255.255.0.0
C	**1st three bits = 110**	24-bit	255.255.255.0

- The first octet of class A address is from 0000 0000 to 0111 1111
- If in decimal numbers, from 0 to 127

Then

- Class A (**0**.x.x.x to **127**.x.x.x)
  - Network mask **255.0.0.0** (the leftmost 8 bits are network address)
  - 16,777,124 hosts for each class A network
- Class B (**128**.x.x.x to **191**.x.x.x)
  - Network mask **255.255.0.0**
  - 65,534 hosts for each class B network
- Class C (**192**.x.x.x to **223**.x.x.x)
  - Network mask **255.255.255.0**
  - 254 hosts for each class C network
- Remarks: class D (**224**.x.x.x to **239**.x.x.x) for multicast; class E not used… :_(

24-bit, 2^{24} = 16,777,126

2 fewer hosts, why? Explain later…

Problems with Classful Designs

- Difficult to find one organization which needs a class A address (more than 16 million hosts)
- Even for one class B network address, how often we can find a company using up to 60,000 host computers
- Problems:

  ⇒ Inflexible for different network sizes

  ⇒ ***Concept of subnets*** created for handling smaller network sizes
- Moreover, IP address exhausted rapidly if giving out one class A network address easily

In fact, all IPv4 addresses exhausted in 2011

Subnetting – Resizing a Network

- Example: If a service provider has a class B network to give, e.g., 136.28.0.0 with netmask 255.255.0.0. Then a company *X* requests only 8 IP host addresses, then what can service provider do?
- Normally, 3 bits are enough for 8 addresses, but IPv4 addressing has two special address designs
  - One for naming the "network address"
  - One for broadcast address in the network

  ← 2 fewer hosts, why? Explain later...
- E.g., gives out 136.28.1.0 with **subnet mask 255.255.255.240**, this is a class B address but the network size is smaller than a class C address

Subnetting (cont'd)

- Netmask no longer based on class A, B or C
  - E.g., 255.255.255.240 ≡ 1111 1111 . 1111 1111 . 1111 1111 . 1111 0000, there are 28 one's
- A simpler notation is

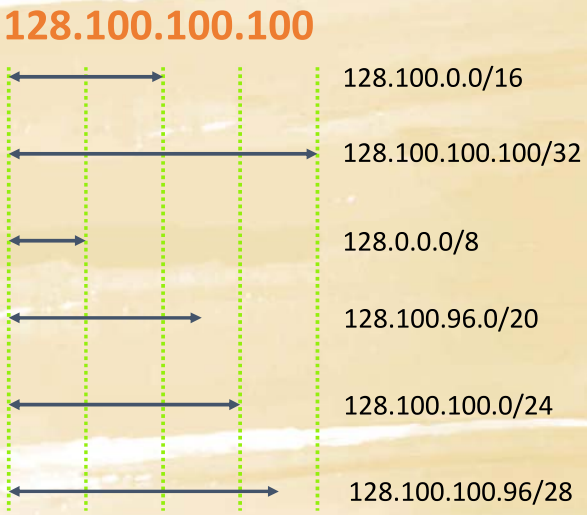
  network address / size of netmask ← CIDR (Classless Inter-Domain Routing) VLSM (Variable Length Subnet Mask)
- Hence, the last example is 136.28.1.0/28

2 Fewer Host Addresses

- For the last example, this network address is 136.28.1.0/28
  - There are 4 bits for assigning host address, ***but***
  - The 136.28.1.0 is the **network name** (the network address) – should not be assigned to any host
  - For broadcasting, all bits for host address are set to 1's, i.e., 136.28.1.15 is the **broadcast address** – cannot be assigned to any host
- Therefore, given *n* bits for host addresses, we have $(2^n - 2)$ for host address assignments
  - If *n* = 8, we have 254 host addresses

Summary on IPv4 Addressing

- Natural class B network
  mask 255.255.0.0
- Host address
  mask 255.255.255.255
- Supernetted range of Class B's
  mask 255.0.0.0
- Class B with 4 bits subnetting
  mask 255.255.240.0
- Class B with 8 bits subnetting
  mask 255.255.255.0
- Class B with 12 bits subnetting
  mask 255.255.255.240



Reserved Private IPv4 Addresses

- Reserved private addresses are critical in extending life of IPv4
- **10**.x.x.x ⇒ private class A networks
- **172.16**.x.x to **176.31**.x.x ⇒ private class B networks
- **192.168**.x.x ⇒ private class C networks

How to Extend the Life of IPv4?

- NAT – network address translation for private addresses
  - Private IP addresses can be used for those local networks setting behind a firewall or NAT (Network Address Translation) device
  - Of course, any networks not connected to the Internet can use any IPv4 addresses
- Use no classes, e.g., CIDR (classless inter-domain routing)
- ***Variable Length Submit Mask*** (VLSM)
  - Notation: ***x.y.z.n/m***
  - Network address: *x.y.z.n*
  - Length of netmask: *m*

Special Addresses

- **0.0.0.0**

⇒ Unknown local host address, this network host

⇒ This **consumes one Class A network address**, sorry :_(

- **255.255.255.255**

⇒ Local LAN broadcast address, **cannot cross routers**; Class E, erh??

- **127.0.0.1** ⇒ local host loopback

⇒ This **consumes another Class A network address** too, sorry :_(

  - Never leaves local computer
  - Other 127.x.x.x addresses are rarely used, but Linux (Ubuntu) sets 127.0.1.1, and many container technology makes uses of these unused addresses

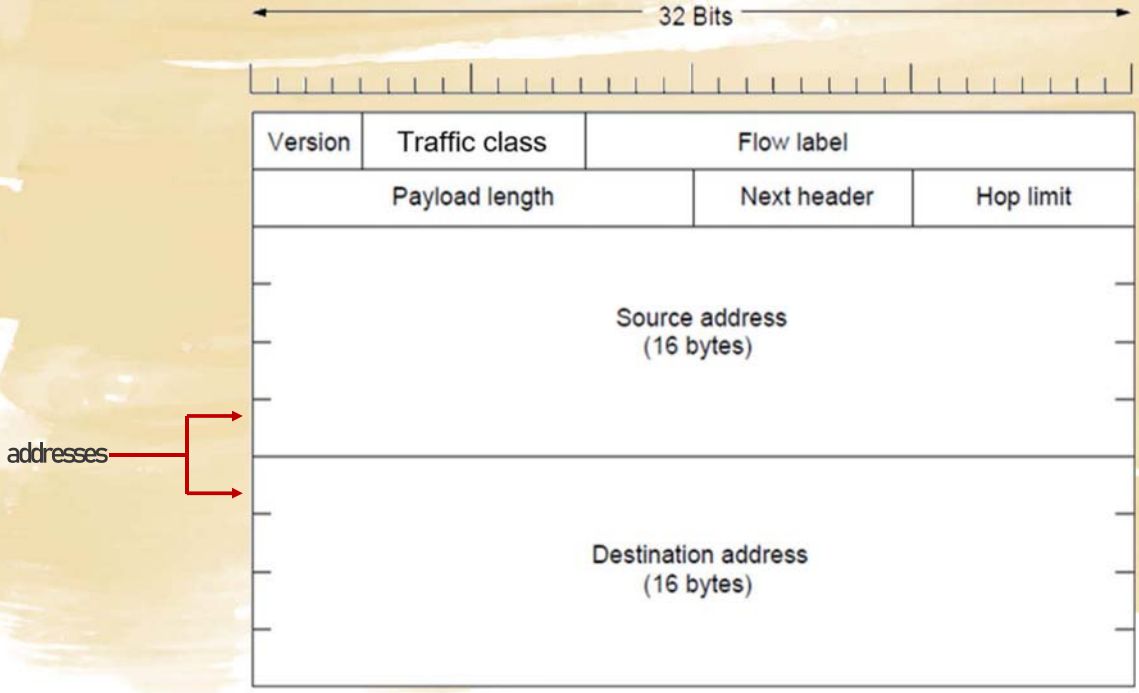
IPv6 Arriving…

- All IPv4 was exhausted around 2011
- But "short-term" solutions to IPv4 address exhaustion: CIDR and NAT with private IPv4 addresses
  - CIDR (Classless Inter-Domain Routing)
  - NAT (Network Address Translation)
  - Private addresses
- Long-term solution: IPv6

IPv6

- Developed mid to late 1990s
- S. Deering, R. Hinden, Internet Protocol, Version 6 (IPv6) Specification, IETF RFC 2460, Dec. 1998
- 128-bit address space, gives 340 undecillion addresses (undecillion = 10^{36})
  - About 10 nonillion per person! (nonillion = 10^{30})
  - 655,570,793,348,866,943,898,599 addresses per square meter of the Earth's surface
- Representation in colon-hexadecimal numbers
  - No dotted-decimal anymore

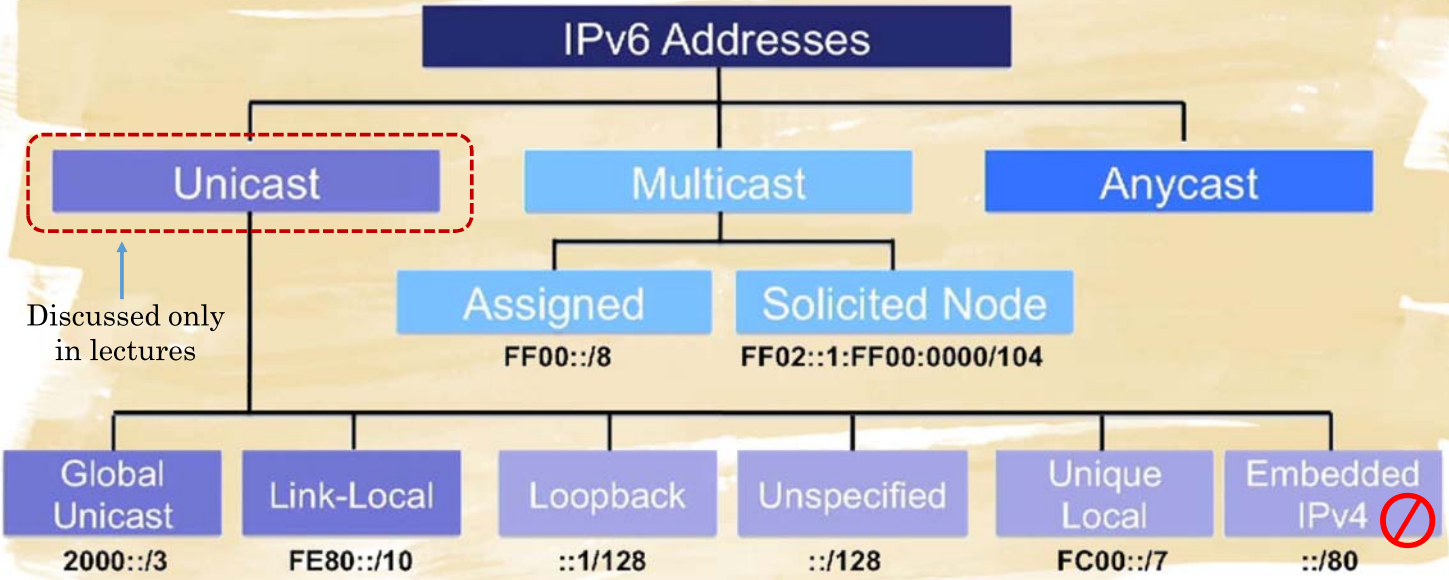
IPv6 Header



Features of IPv6 More than Addresses

- Larger address space
- Stateless address autoconfiguration
- End-to-end reachability without using private addresses and NAT
- Better mobility support
- Peer-to-peer networking easier to create and maintain, and services such as VoIP and Quality of Service (QoS) become more robust
- Fixed sized IPv6 header, no fragmentation IPv6 packets

Types of IPv6 Addresses



Hexadecimal Numbers

- Base-16
- 16 digits - {0, 1, 2, …, 8, 9, A, B, C, D, E, F}
- One hex number represents a 4-bit number

Dec	Hex	Binary	Dec	Hex	Binary
0	0	0000	8	8	1000
1	1	0001	9	9	1001
2	2	0010	10	A	1010
3	3	0011	11	B	1011
4	4	0100	12	C	1100
5	5	0101	13	D	1101
6	6	0110	14	E	1110
7	7	0111	15	F	1111

IPv6 Address Notation

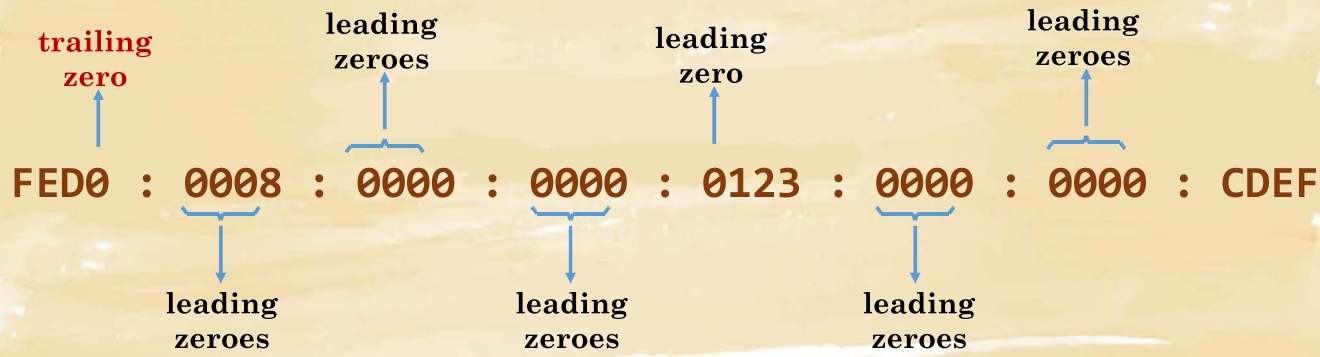
- Each hex represents 4 bits (0000 to 1111 in binary)
- 128 bits = 32 × 4 bits, i.e., 32 concatenated 4-bit units
- This 4-bit unit sometime is called nibble
- Separate each 16 bits (4 units of 4 bits, i.e., 4 hex numbers) by colon except the beginning and ending colons
- Example

FEDC : BA98 : 7654 : 3210 : 0123 : 4567 : 89AB : CDEF

16-bit 16-bit 16-bit 16-bit 16-bit 16-bit 16-bit 16-bit

Two Rules Compressing IPv6 Addresses

- **Rule 1**: omitting leading 0s (no trailing 0 to avoid ambiguity)



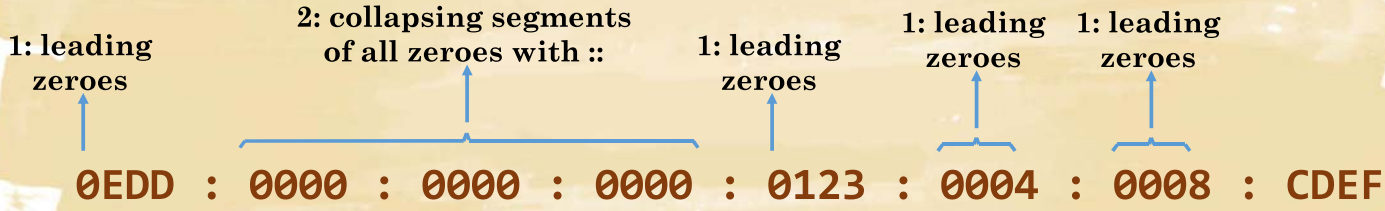
- After rule 1, we have FED0 : 8 : 0 : 0 : 123 : 0 : 0 : CDEF

2 Rules on IPv6 Addresses (cont'd)

- **Rule 2**: double colons (::) replacing any single, contiguous string one or more 16-bit segments consisting of all zeroes
- On rule 2:
  - Double colons can be applied ***only once*** for an address
  - (RFC 5952) If more than one such string, then only the longest string of zeroes must be replaced
  - (RFC 5952) If two strings of identical lengths, then the first string of 0's should use the :: representation

2 Rules on IPv6 Addresses (cont'd)

- Combining both rules



- Result: `EDD :: 123 : 4 : 8 : CDEF`

2 Rules on IPv6 Addresses (cont'd)

- If more then one segment that has the longest number of segments of zeroes and they are equal in lengths…
- From RFC 5952, always the first one selected
- Given

  `FED0 : 0008 : 0000 : 0000 : 0123 : 0000 : 0000 : CDEF`

- We have the final representation

  `FED0:8::123::CDEF` ✗

  `FED0:8:0:0:123::CDEF` ✗

  `FED0:8::123:0:0:CDEF` ✓

Subnet Mask

- Prefix and prefix length are used in IPv6
- All zeroes and all ones interface ID can be used in IPv6
- That is, there are prefix lengths for setting network addresses
- Prefix examples
  - `2001:DB8:1::/48`
  - `2001:DB8:CAFE:1234::/62`
- IPv6 device address examples
  - `2001:DB8:CAFE::99:2/48`
  - `2001:DB8:CAFE:1::100/64`

IP Addressing

- IPv6 source
  - Always a unicast address (GUA or link-local)
- IPv6 destination
  - Unicast, multicast, or anycast address
  - IPv6 has no broadcast addresses

Unicast Addresses Assignments

- Global Unicast Address (GUA)
  - Host address identification on the Internet
  - Globally unique and routable (similar to public IPv4 addresses)
  - Address from 2000::/3 (usually showing at least the first hextet)
    - Check in binary, the initial 3-bit setting is always 001….
  - 2000::/3 (fyi, from IETF, usually /48 assigned for Global Routing Prefix)
  - 2001:DB8::/32 (RFC 2839 / RFC 6890) addresses reserved for documentation
  - An interface may get more than one IPv6 address
  - Keeping the terminologies from IPv4: “prefix,” “prefix length,” “interface ID”
- That’s it, but in fact, there are more on address allocation issues…

Unicast Addresses Assignments (cont'd)

- Link-local Unicast
  - Unique only on the link (the Link layer – local area network, subnet, etc.)
  - Not routable off the link
  - From FE80::/10 to FEBF::/10
  - Created through MAC addresses or randomly generated numbers
  - An IPv6 device must have at least one link-local address -
    - A host communicates to the IPv6 network before obtaining its GUA
    - Router's link-local address is used by hosts as the default gateway address
    - Adjacent routers to exchange routing updates

Unicast Addresses Assignments (cont'd)

- Loopback address
  - ::1/128
  - Same function as the 127.0.0.1 in IPv4
  - Used by a node to send back IPv6 packets to itself, typically used for testing the TCP/IP stack
  - Not routable
- Unspecified Address
  - :: (all 0's)
  - Indicates the absence, or anonymity of an IPv6 address (e.g., for router solicitation)
  - Used as a source IPv6 address during duplicate address detection process

Unicast Addresses Assignments (cont'd)

- Unique Local Address (ULA)
  - FC00::/7 (first hextet: FC00::/7 to FDFF::/7)
  - Should not be routable on the Internet
  - Similar to RFC 1918 IPv4 addresses (private addresses) but not meant to be translated to a global unicast (for security purposes)
  - Used in more limited area such as within a site or devices inaccessible from the global Internet
  - For the first hextet 1111 110*x* (*x* = local flag bit)

    (for *x*=0) FC00::/8 - /48 prefix assigned using RFC 4193 algorithm (dormant)

    (for *x*=1) FD00::/8 - /48 prefix assigned locally

Remark: Site local addresses (FEC0::/10) was deprecated

Unicast Addresses Assignments (cont'd)

- Embedded IPv4 Address (was deprecated)
  - Was used by dual-stack devices that support both IPv4 and IPv6
  - Other transition methods used when required to send IPv6 packets over IPv4-only networks, such as tunneling or NAT64

Summary

- Both the IPv4 and IPv6 addressing schemes are covered
- IPv4 addresses all exhausted
  - But still living quite well today
- IPv6 too abundant and generously assigned at the current stage
  - GUA
  - Link-local
  - ULA

澳 門 理 工 學 院
Instituto Politécnico de Macau
Macao Polytechnic Institute

COMP 225 Network and System Administration

Notes #8: Linux Commands on Network Cards

K. L. Eddie Law, PhD

Macao Polytechnic Institute
School of Applied Sciences
Academic Year 2020-2021, 2nd Semester

Network Interface Cards

- Computers nowadays are set with at least one network card
- Easily to add a few more if deemed needed
- Run `ping` to test if network card working and connecting to the network

```
$ ping www.google.com
```

- `ping` belongs to ICMP messages

IP Commands

- Evolving commands to set IP addresses for network interfaces
- The old time favorites were
  - `ifconfig`
  - `netstat`
  - `route`
- These commands can still be re-installed on Ubuntu again using

```
$ sudo apt install net-tools
```


IP Assignment Tools

- The new "ip" tool is adopted by Ubuntu and Red Hat (in fact, Red Hat supports both tools)

Legacy Tools	New Tools
ifconfig	ip addr
netstat	ss, ip route, ip -s link, ip maddr
route	ip route
arp	ip neighbor
iptunnel	ip tunnel

To Check Network Interfaces

- For the Link layer hardware interfaces, usually
  - Wired – Ethernet card, its name usually starts with an "e..."
  - Wireless – wifi card, its name usually starts with an "w..."
  - These address are the MAC (Medium Access Control) addresses
- To check the names of all hardware interfaces, we can use either

```
$ ip link [show | list]
$ ip addr [show | list]
```

- Both commands show all found network interfaces, the word "show" and "`list`" are interchangeable; but in these cases, in fact, both are redundant

MAC Addresses

- Hardware interfaces have names, and MAC addresses
  - 6-byte in size, similar to IPv6, uses colon-hexadecimal notation
  - One byte then a colon, and so on
  - No shorten notations, as those available for IPv6
- Device name and the associated MAC address can be found with

```
$ ip link
```

  - For example

Name of the interface

```
1: lo: <LOOPBACK,UP,LOWER)UP> mtu 65535 qdisc noqueue state…
   link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00:00
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER,UP> mtu 1500 qdisc…
   link/ether 08:00:27:a7:6e:d2 brd ff:ff:ff:ff:ff:ff
```

MAC address

The IP Address

- Through the command
  ```
  $ ip addr
  ```
  - We may get

```
1: lo: <LOOPBACK,UP,LOWER)UP> mtu 65535 qdisc noqueue state…
   link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00:00:00
   inet 127.0.0.1/8 scope host lo
      valid_lft forever preferred_lft forever
   inet6 ::1/128 scope host
      valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER,UP> mtu 1500 qdisc…
   link/ether 08:00:27:a7:6e:d2 brd ff:ff:ff:ff:ff:ff
   inet 10.0.2.15/24 brd 10.0.2.255 scope global dynamic enp0s3
      valid_lft 76196sec preferred_lft 76196sec
   inet6 fe80::a00:27ff:fea7:6ed2/64 scope link
      valid_lft forever preferred_lft forever
```

IPv4 address

IPv6 address

Find "up" Devices or Find One Device

- Sometimes, only listing those some interfaces that are up
  ```
  $ ip link list up
  $ ip addr show up
  ```
- Like to get information about one interface only, e.g.,
  ```
  $ ip link list enp0s3
  ```
  ← **My interface card only**
  ```
  $ ip -4 addr show enp0s3
  ```
  ← **IPv4 address only**
  ```
  $ ip -6 addr show enp0s3
  ```
  ← **IPv6 address only**
- For hostname, can use
  ```
  $ hostname
  $ hostnamectl
  ```

To Change the State of an Interface

- Change the setting of an interface
  ```
  $ sudo ip link set dev {device name} {up | down}
  ```
- E.g., turn off the enp0s3 interface
  ```
  $ sudo ip link set dev enp0s3 down
  $ ip link list
  $ ip link show up
  ```
- Ok, enough fun, put it back up
  ```
  $ sudo ip link set dev enp0s3 up
  ```


The Command "ip addr"

- $ sudo ip addr {add|change|replace|del} IPADDR dev {device name}
  - Where IPADDR is typical IP address with netmask, support CIDR notation
  - Broadcast address not set in this command! (different from the old "ifconfig")
- E.g., add an IPv4 and associated broadcast address to enp0s3
  ```
  $ sudo ip addr add 10.0.0.2/24 broadcast 10.0.0.255 dev enp0s3
  ```

```
elaw@s1:~$ sudo ip addr add 10.0.0.2/24 broadcast 10.0.0.255 dev enp0s3
elaw@s1:~$ ip addr list enp0s3
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 100
0
    link/ether 08:00:27:a7:6e:d2 brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.15/24 brd 10.0.2.255 scope global dynamic enp0s3
       valid_lft 85381sec preferred_lft 85381sec
    inet 10.0.0.2/24 brd 10.0.0.255 scope global enp0s3
       valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:fea7:6ed2/64 scope link
       valid_lft forever preferred_lft forever
```

Other Settings

- Remove the added-on IPv4 address

```
$ sudo ip addr del 10.0.0.2/24 broadcast 10.0.0.255 dev enp0s3
```

- The interface gets IPv4 dynamically; if not, can set it using dhcp
  - `$ sudo dhclient -4 enp0s3`
  - Use "-6" for IPv6
- However, all we have done on screen are only working for the current active session
- All added/changed/modified/ settings do **NOT** survive a system reboot!

Netplan and Renderers

- Netplan has been used since 2019 in Ubuntu 19.04
- The old "`ifupdown`" scripting system was removed
- It indicates rendering software for setting up network addresses
  - `systemd-networkd` ← **Ubuntu server**
  - `NetworkManager` ← **Ubuntu desktop**
- Check if one of them is running

```
$ systemctl status { systemd-networkd | NetworkManager }
```

Netplan: Setting Network Interfaces on Booting

- Go check the file `$ ls /etc/netplan`
  - Suppose it shows a file named `00-installer-config.yaml`
  - If the filename is different, it is ok if it is a "yaml" file
  - on screen could be:

```
network:
  ethernets:
    enp0s3:
      dhcp4: true
  version: 2
```

Quite easy to read, if modifying it, run command `$ sudo netplan apply` to facilitate the changes

Static IP with Netplan: Example

- Configure with the `.yaml` file

```
network:
  version: 2
  renderer: networkd
  ethernets:
    enp0s3:
      dhcp4: false
      addresses: [10.0.2.4/24]
      gateway4: 10.0.2.1
      nameservers:
        search: [example.com, otherdomain]
        addresses: [10.0.2.1, 8.8.8.8]
```

← Or, **NetworkManager**

- Then run `$ sudo netplan apply`

Gateway and Friends

- To get to the Internet , we need a gateway or router
- To find the default gateway

  `$ ip route`

  - And it shows for a host using DHCP service, for example,

```
default via 10.0.2.1 dev enp0s3 proto dhcp src 10.0.2.6 metric 100
10.0.2.0/24 dev enp0s3 proto kernel scope link src 10.0.2.6
10.0.2.1 dev enp0s3 proto dhcp scope link src 10.0.2.6 metric 100
```

- Any neighboring computers around??

  `$ ip neigh show`

  - If any shown, the last field can be STALE, DELAY, or REACHABLE

Routing Table (Not in Exam)

- Following commands are for adding new routes or deleting routes
- For adding:

  "default" is optional, add it to change the default route

  $ sudo ip route add {default} [network/netmask] via [gatewayIP]

  $ sudo ip route add {default} [network/netmask] dev [deviceName]
- For deleting:

  $ sudo ip route del default ← Delete the default route

  $ sudo ip route del [network/netmask] dev [deviceName]


QA

澳門理工學院
Instituto Politécnico de Macau
Macao Polytechnic Institute

COMP 225 Network and System Administration

Notes #9: Secure Shell and Firewall

K. L. Eddie Law, PhD

Macao Polytechnic Institute
School of Applied Sciences
Academic Year 2020-2021, 2nd Semester

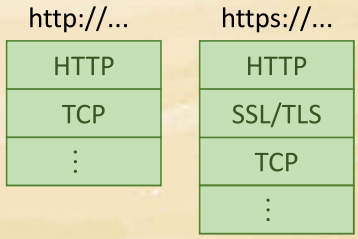
On Security and Protection

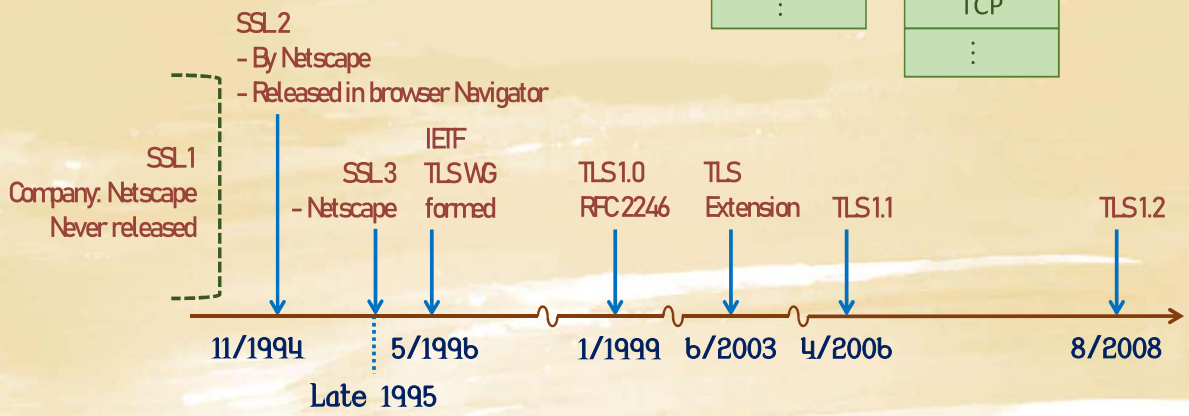
- Network security is extremely important with today's Internet
- For protecting communications, the popular remote login application program is Secure Shell (SSH), and SSH uses TCP as the underlying transport protocol
- For protecting incoming and outgoing traffic, basic firewall-like protection mechanisms are available in Linux
- One of them is called `netfilter`, a packet handling engine, and its command line tool is the

```
$ sudo iptables …
```

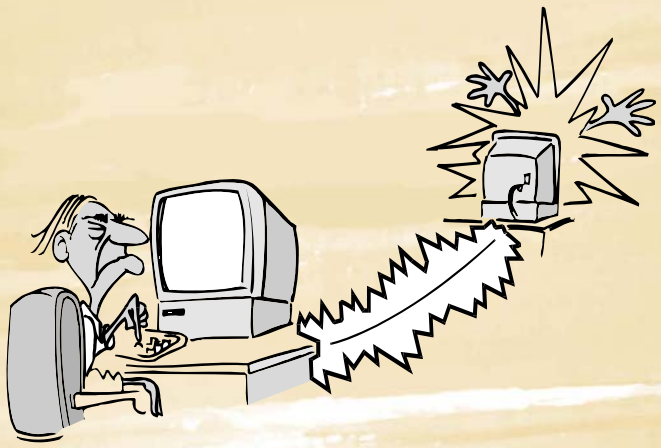
A Word on SSL and TLS

- SSL – Secure Socket Layer
- TLS – Transport Layer Security





Simple Defense – Netfilter and iptables



Low Level Firewalls in Linux Kernels

- Linux 2.0.x – ipfwadm
- Linux 2.2.x – ipchains
- Since Linux 2.4.x – Netfilter and iptables
- Any new systems coming in?
  - `nftables` based on `nft` commands was in since 2014 for Linux 3.13, but failed to get widely used at low level. However… for simple firewall settings…
  - Another newer one, the `bpfilter` from the BSD operating systems, is being seriously considered to be ported over

Implementing Firewalls

- Firewall is a host-based, network-layer, software firewall managed by the `iptables` utility and related kernel-level components
- With `iptables`, e.g.,
  - Create a series of rules for every network packet coming through the Linux system
  - Fine-tune the rules to allow network traffic from one location but not from another
  - These rules essentially make up a network access control list

Other Simplified Firewall Models

- While Fedora, RHEL, and related distributions use the `firewalld` service to provide a more user-friendly way to manage firewall rules
- Ubuntu goes with their UFW (Uncomplicated Firewall), and this UFW acts as a front end for `nftables`

`netfilter/iptables`

- `netfilter` and `iptables` are building blocks of a framework inside the Linux kernel
- The `iptables` utility manages the Linux firewall, called `netfilter`, i.e., the Linux firewall is often referred to as `netfilter/iptables`
- This framework enables packet filtering, network address [and port] translation (NA[P]T) and other packet mangling
- The `iptables` syntax continues to be supported, but for recent Linux releases, `nftables` is actually doing all the work behind the UFW

Functions of iptables

- Stateful packet inspection
  - The firewall keeps track of each connection passing through it, an important feature in the support of VoIP
- Filtering packets based on a MAC interface, IPv4, IPv6
  - Important in WLAN's and similar environments
- Filtering packets based the values of the flags in the TCP header
  - Helpful in preventing attacks using malformed packets and in restricting access
- Network address translation and Port translating NAT/NAPT
  - Building DMZ and more flexible NAT environment to increase security

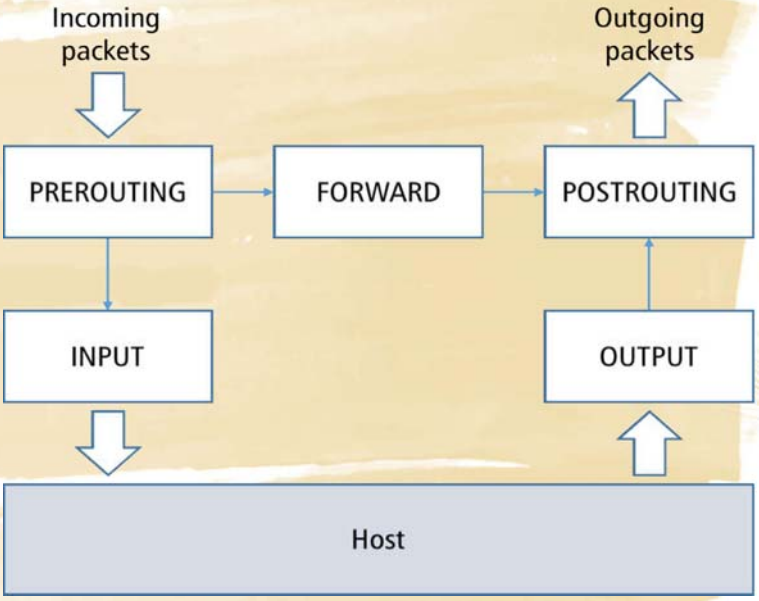
More on iptables

- Source and stateful routing and failover functions
  - Route traffic more efficient and faster than regular IP routers
- System logging of network activities
  - Provides the option of adjusting the level of detail of the reporting
- A rate limiting feature
  - Helps to block some types of denial of service (DoS) attacks
- Packet manipulation (mangling) like altering the ToS/DSCP/ECN bits of the IP header
  - Mark and classify packets dependent on rules, the first step in QoS

Quality of Service (QoS) not covered or tested in this course

Designs of iptables

- `iptables` structures packet examinations through tables
- 4 built-in tables: `filter`, `nat`, `mangle` and `raw` tables
- For all possible traffic flows, 5 chains are shown for packet examining or content editing, e.g., INPUT chain, etc.
- Each table has its own set of chains



Four Tables

- **`filter`**: is the packet-filtering feature of the firewall; in this table, access control decisions are made for packets traveling to, from, and through the Linux system
- **`nat`**: is used for Network Address Translation (NAT), and the rules in NAT table determines the redirection where a packet goes
- **`mangle`**: packets are mangled (modified) according to the rules in the mangle table; but using the mangle table directly is less common and typically done to change how a packet is managed
- **`raw`**: is used to exempt certain network packets from something called connection tracking – this tracking feature is important when using Network Address Translation and virtualization on Linux server

Five Chains

- **INPUT**: Network packets coming into the Linux server
- **FORWARD**: Network packets coming into the Linux server that are to be routed out through another network interface on the server
- **OUTPUT**: Network packets coming out of the Linux server
- **PREROUTING**: Used by, e.g., NAT for modifying network packets when they come into the Linux server
- **POSTROUTING**: Used by, e.g., NAT for modifying network packets before they come out of the Linux server

About Tables and Chains

Chains Available for Each netfilter/iptables Table

Table	Chains Available
filter	INPUT, FORWARD, OUTPUT
nat	PREROUTING, OUTPUT, POSTROUTING
mangle	INPUT, FORWARD, PREROUTING, OUTPUT, POSTROUTING
raw	PREROUTING, OUTPUT

More on 3 Tables and 5 Chains

- The 3 widely used ones – `filter`, `nat`, `mangle` tables
- Table `filter`, for packet filtering, set firewall policy rules in chains
  - **Input** chain: filters packets destined for the firewall
  - **Forward** chain: filters transit packets to/from locations protected by firewall
  - **Output** chain: filters packets originating from the firewall

More on 3 Tables and 5 Chains (cont'd)

- Table `nat`, for network address translation
  - Remember to permit IP forwarding for NAT to work at the interfacing node (traditional way: uncomment the `net.ipv4.ip_forwarding=1` in `/etc/sysctl.conf`, more should be made with `systemd`)
  - Interested in 2 chains
  - **Pre-routing**: NAT packets when destination address need changes
  - **Post-routing**: NAT packets when source address need changes
- Table `mangle`
  - Manipulate QoS bits in TCP header through the **input** and **output** chains, if needed; usually not used by home users

Checking out the iptables

- Check if `iptables` is installed and running
  ```
  $ sudo iptables -L -v
  ```
  - Lists all chains in all tables… could be many… but can also be empty
- FYI, there are many options for `iptables`, going to only discuss some features of iptables (already quite a lot!!)
- Use commands `$ sudo iptables …` to set rules
  - FYI, the rules input in terminal will not be persistent upon rebooting
- But we can write scripts for `iptables` to run at machine boot-up

Switches/Options for iptables

- **-F** flush; deletes all the rules in the selected ***Table***
- **-A** [chain name] append to the end of the named chain
- **-j** [target] jump out to a targeted decision
- **-P** [chain name] default policy for a chain, if needed
- **-D** [chain] [rule #] delete a rule with the order number indicated

Clause	Meaning or possible values
-p *proto*	Matches by protocol: **tcp**, **udp**, or **icmp** or **ANY**
-s *source-ip*	Matches host or network source IP address (CIDR notation is OK)
-d *dest-ip*	Matches host or network destination address
--sport *port#*	Matches by source port (note the double dashes)
--dport *port#*	Matches by destination port (note the double dashes)
--icmp-type *type*	Matches by ICMP type code (note the double dashes)
!	Negates a clause
-t *table*	Specifies the table to which a command applies (default is filter)

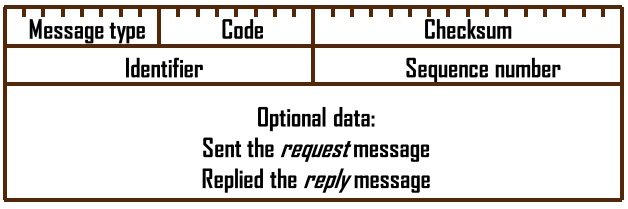
Protocol Switches

- If "-p" is used, we can mark TCP, UDP, **ICMP**

Protocol Switch	Description
-p tcp --sport [source port #]	TCP with source port #, range of ports "**`starting_port:ending_port`**"
-p tcp --dport [destination port #]	TCP with destination port #, range of ports permitted
-p tcp --syn	New TCP connection request with SYN bit set; "**`! --syn`**" SYN bit not set
-p udp --sport [source port #]	UDP with source port #, range of ports permitted
-p udp --dport [destination port #]	UDP with destination port #, range of ports permitted
--icmp-type [type]	Most common types are echo-request or echo-reply, i.e., `ping` commands

Example of ICMP: Ping

- Internet Control Message Protocol (ICMP) has a set of query messages for diagnosing network problems
- One of the message sets is called "ping"
  - `$ ping [IP address]`

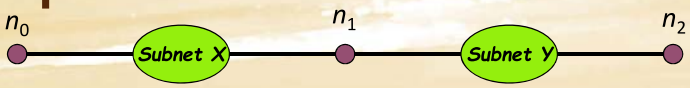


Ping Service

- Uses ICMP **echo request** and **echo reply** messages
- **Determine if a host exists and active in the network**
- An administrator can find out about the state of the resources: no reply means problem exists
- ID is not well-defined, but usually is the process ID
- Message type 8 is for echo request, type 0 is for echo reply
- Can use "ping localhost" to verify the operation of local TCP/IP software

Eddie Law

Ping Example



Send ICMP echo request
(id=700, seq=0, TTL=255)

Send ICMP echo reply
(id=700, seq=0, TTL=255, time=8ms)

Send ICMP echo request
(id=700, seq=1, TTL=255)

Send ICMP echo reply
(id=700, seq=1, TTL=255, time=5ms)

Send ICMP echo request
(id=701, seq=0, TTL=255)

Send ICMP echo reply
(id=701, seq=0, TTL=255, time=20ms)

Eddie Law

Back in iptables

The "-j" : Jump to a Target/Decision

- After matching all conditions in an iptables rule statement, then we can make a decision using the switch "-j"
- A decision target queue must be appended after the "-j" switch
- Commonly used targets are
  - ACCEPT
  - DROP
  - REJECT
  - LOG
  - MASQUERADE

The Targets (1)

- ACCEPT
  - Leaving iptables, the packet is passed to application or the OS for further processing
- DROP
  - Packet is dropped quietly without any further processing
- REJECT
  - Packet is dropped, but an ICMP message is returned to packet sender
  - "`--reject-with [qualifier]`" can be added, where "qualifier" is an ICMP message
- LOG
  - Packet information is sent to syslog daemon for logging, and packet is then checked by next iptables' rule
  - "`--log-prefix 'reason'`" can be added
  - If doing LOG and DROP, then two rules are needed, cannot be integrated into one rule

The Targets (2)

- MASQUERADE
  - The regular NAT (Network Address Translation), the source address is changed to the outgoing IP address of the firewall
  - Port can be changed explicitly through "`{--to-ports [port]{-[port]}}`", or automatically
- SNAT
  - Source NAT – the source address is modified
  - Source address is user-defined, "`--to [IPaddress]{:[port]}`" or a range for selection "`--to [IPaddress{-[IPaddress]}]{:[port]{-[port]}}`"
- DNAT
  - Destination NAT – the destination address is changed
  - "`--to [IPaddress]`"

Using the Protocol Switch

- The "-p" permits us to match specific protocol
- E.g., eth0 is facing the Internet, eth1 is facing an internal machine
  - Accept incoming HTTP traffic

```
$ sudo iptables -A INPUT -i eth0 -p tcp --dport 80 -j ACCEPT
```

  - Accept all new TCP connection request from internal machine

```
$ sudo iptables -A INPUT -i eth1 -p tcp --syn -j ACCEPT
```

  - Accept an UDP datagram from source 10.0.0.1 coming in for destined port 53

```
$ sudo iptables -A INPUT -s 10.0.0.1 -p udp --dport 53 -j ACCEPT
```

  - TCP traffic from anywhere going to 192.168.1.1

```
$ sudo iptables -A INPUT -s 0/0 -i eth0 -d 192.168.1.1 -p tcp -j
ACCEPT
```

- Popular port numbers: SSH (22), HTTP (80), HTTPS (443)), DNS (53)

Any IP addresses

The "-m" : A Sophisticated Setting

- Matching rule with "-m"
- TCP is stateful, for "`-p tcp -m state --state [States]`", we should supply the states of a connection that the iptables shall check
  - Permitted states in TCP: **`NEW`**, **`ESTABLISHED`**, **`RELATED`**, **`INVALID`**

"-m" : A Sophisticated Setting (cont'd)

- For rate control with "`-m limit`"
  - Specifies the maximum average number of matches per second in the forms of /second, /minute, /hour, or /day
  - Abbreviation, example: 3/s is for 3/second
- Following command accepts only one ping request message per second

```
$ sudo iptables -A INPUT -p icmp --icmp-type echo-request
-m limit --limit 1/s -j ACCEPT
```


Setup of NAT

- Make sure the "`ip_forward`" is set in the system
  - Check if content of file /proc/sys/net/ipv4/ip_forward is 1
  - If not, `$ sudo echo 1 > /proc/sys/net/ipv4/ip_forward`
- With the use of "`systemd`" in Debian/Ubuntu for starting up
  - May have to add a file in `/etc/systemd/network` for effecting IPv4 forwarding
  - Some other settings may also be required
  - Read http://manpages.ubuntu.com/manpages/disco/man5/systemd.network.5.html

Setup of NAT (cont'd)

Setting up the NAT for all traffic leaving interface eth0

- Suppose `eth0` facing the Internet, `eth1` facing computer inside organization
- An example, the iptables commands

```
$ sudo iptables -t nat -A POSTROUTING -o eth0 -j
MASQUERADE
$ sudo iptables -A FORWARD -i eth0 -o eth1 –m state
--state RELATED,ESTABLISED -j ACCEPT
$ sudo iptables -A FORWARD -i eth1 -o eth0 -j ACCEPT
```

Permits all traffic from internal computing devices

Permits traffic coming in eth0 and going out at eth1 only if the connection was established or related
⇒ this implies the connection was started by the computer

More Examples

- Allow HTTP traffic for web server over port 80

```
$ sudo iptables -A INPUT -p tcp --dport 80 -i eth0 -j ACCEPT
```

- Allow FTP traffic for FTP daemon over port 21 to service FTP requests

```
$ sudo iptables -A INPUT -p tcp --dport 21 -i eth0 -j ACCEPT
```

- Allow SSH traffic for Secure Shell connections over port 22 to service SSH requests

```
$ sudo iptables -A INPUT -i eth0 -p tcp --dport 22 -j ACCEPT
```

- After applying the rules for the incoming traffic accepted in the **INPUT** chain, then applying a final "catch-all" rule to block those failed to meet any previous rules:

```
$ sudo iptables -A INPUT -p tcp -i eth0 -j DROP
```

- The **catch-all rules** MUST be applied the LAST

Match Criteria for ICMP (ping)

Matches used with `--icmp type`	Description
--icmp-type <type>	The most commonly used types are echo request and echo reply messages

- If allowing `ping` request and reply
- Then configure `iptables` to set firewall to permit sending ICMP echo-requests (pings) and in turn, accepting the expected ICMP echo-replies (with `sudo` before)

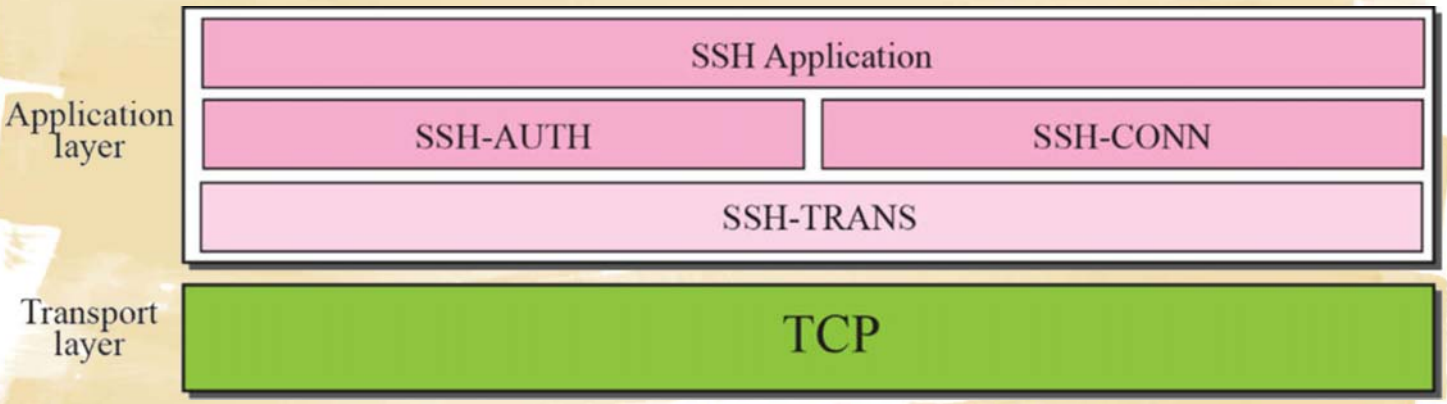
```
iptables -A OUTPUT -p icmp --icmp-type echo-request -j ACCEPT
iptables -A INPUT  -p icmp --icmp-type echo-reply   -j ACCEPT
```


Secure Shell

Communication Protection – SSH

- SSH allows logging in a remote computer (or a local) computer
  ```
  $ ssh [-l username] [computer_name]
  ```
- In fact, can use SSH to log in localhost instead of using command `su`
- SSH is a secure replacement for the legacy text-based "`telnet`"
- SSH requires that an SSH daemon, the server, be running on the remote host. You will also need the password of the user you wish to log in as

Component of SSH



SSH Man in the Middle Warning

- The first time we SSH into a host, we likely see a message similar to the one below

```
The authenticity of host 'localhost (::1)' can't be established.
RSA key fingerprint is 20:d6:36:a1:e7:2f:98:97:58:f5:00:a8:85:3e:9d:58.
Are you sure you want to continue connecting (yes/no)?
```

- SSH uses public key cryptology to add security to the process
- This message is shown because this is the first time seeing this incoming host's public key
- Answering "yes" causes SSH to import this host's public key into the logging in user's `~/.ssh/known_hosts` file

Public Key Infrastructure for SSH

- SSH allows authentication using digital signing, a secure method of proving ones identity
- `$ ssh-keygen`
  - Creates public/private key pairs and stores them in a user's `.ssh` directory
  - On running the `ssh-keygen` command, always prompt for a passphrase
  - The passphrase is **NOT** a password to login to a server; it is a password that is used to encrypt your private key

Public Key Infrastructure for SSH (cont'd)

- `$ ssh-copy-id -i [identity_file] [remote_system]`
  - It copies the public key into the `authorized_keys` file on remote systems, enables you to login those system using public keys encryption rather than your system password
  - E.g., `$ ssh-copy-id -i ~/.ssh/id_rsa.pub testSSH@localhost`

The User's ~/.ssh Directory

- The .ssh directory holds important files for SSH operations
  - **`id_rsa`**: user's private key if rsa is used, keep this key secret!
  - **`id_rsa.pub`**: user's public key if rsa is used; copy this file to `authorized_keys` on machines like to log into in future
  - **`id_dsa`**: user's private key if dsa is used, keep this key secret!
  - **`id_dsa.pub`**: user's public key if dsa is used; copy this file to `authorized_keys` on machines like to log into in future
  - **`known_hosts`**: the hosts and host keys of computers that this user has used SSH to connect to
  - **`authorized_keys`**: grants user's access to log into this account with digital signature authentication; for each public key listed in this file, the associated private key can be used to login to this account

Personal Hygiene: Protection of Private Keys

- **IMPORTANT**: Do NOT allows anyone to access your private keys
- An attacker, gains your private key, can use it to log into other machines without a password, if your associated public key is in the `authorized_keys` file on any other machines
- Also possible for someone to log into your account on this machine if they can insert their own public key into your `authorized_keys` file
- Some administrators put their public keys in the `authorized_keys` file on remote servers. This allows them to use SSH to launch commands on remote computers without a password (via `cron` scripts etc.)

Secure File Transfers

- OpenSSH provides a number of ways to create encrypted remote logins and file transfer connections between clients and servers
- The OpenSSH Secure Copy (`scp`) and Secure FTP (`sftp`) programmes are the secure replacements for traditional text-based FTP

Installing SSH Server Daemon

- For Ubuntu server, SSH server usually is installed, if not, then run
  ```
  $ sudo apt install openssh-server
  ```
- Then start the SSH server daemon, and enable it next upon rebooting
  ```
  $ sudo systemctl start sshd
  $ sudo systemctl enable sshd
  ```
- SSH server and client configuration files can be found at /etc/ssh/sshd_config and /etc/ssh/ssh_config, respectively
- Any changes made in these files, should "restart" the daemon again
  ```
  $ sudo systemctl restart sshd
  $ sudo systemctl status sshd
  ```


Summary

- Some basic security-related commands or tools are introduced
  - SSH – secured communications
  - iptables – for setting simple defense firewall
- If using scripts for running iptables while starting up
  - In general, we should clean up all those tables before adding any rules! For example, the general starting commands are
    ```
    iptables -F
    iptables -P INPUT DROP
    iptables -P FORWARD DROP
    ```
  - Actively running iptables script can be saved using the command "**`iptables-save`**"


QA