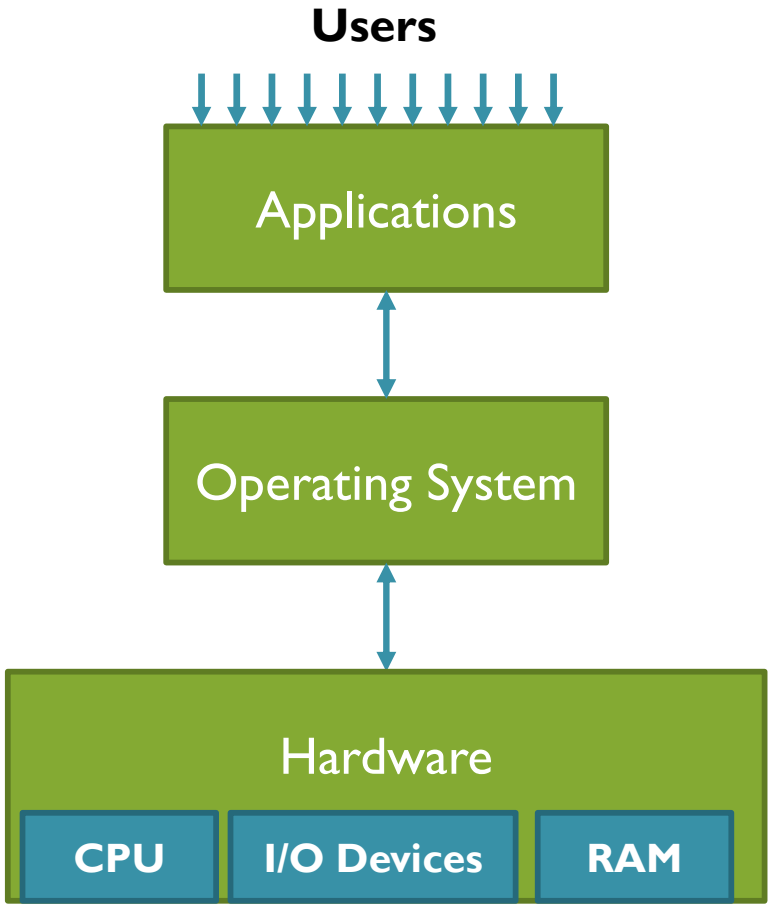
Topics

- About
- Evolution/History of OS
- Types of OS
- Components/Functions of OS
- User and Kernel space

Popular Operating Systems

What is an Operating System

- An Operating System is a program that acts as an interface between a user of a computer and the computer hardware



What if we don't have an OS

- How do you get your program onto the hardware?
- How do you print out the answer?

Once upon a time, users had to Toggle in program in binary and read out the answer from LED's!

Altair 8080

Why do we need an OS?

- Primary Goal – Convenience
- Secondary Goal – Efficiency

Or it could be the other way round!

Operating Systems Goals

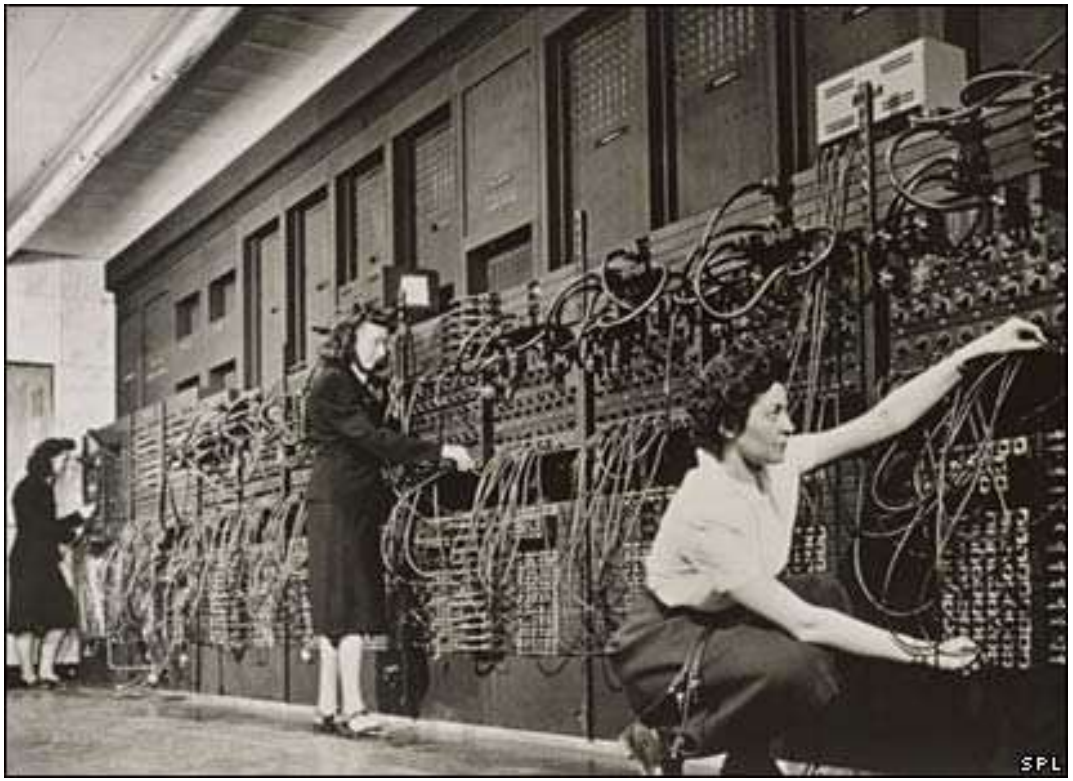
- Make the computer system convenient to use
  - Provide clean, easy-to-use abstractions of physical resources
- Use the computer hardware and resources in an efficient manner
  - Manage protection, isolation, and sharing of resources
- Control/execute user/application programs by providing standard services

Evolution/History of OS

- Bare machine (No Operating System)
- Batch Operating Systems
- Multiprogramming Operating Systems
- Timesharing / Multitasking Operating Systems
- Multiprocessing Operating Systems
- Real Time Operating Systems
- Network OS
- Embedded OS

History

- It all started with computer hardware in about 1940s.



ENIAC 1943

History contd.

- ENIAC (Electronic Numerical Integrator and Computer), at the U.S. Army's Aberdeen Proving Ground in Maryland.
  - built in the 1940s,
  - weighed 30 tons,
  - was eight feet high, three feet deep, and 100 feet long
  - contained over 18,000 vacuum tubes that were cooled by 80 air blowers.

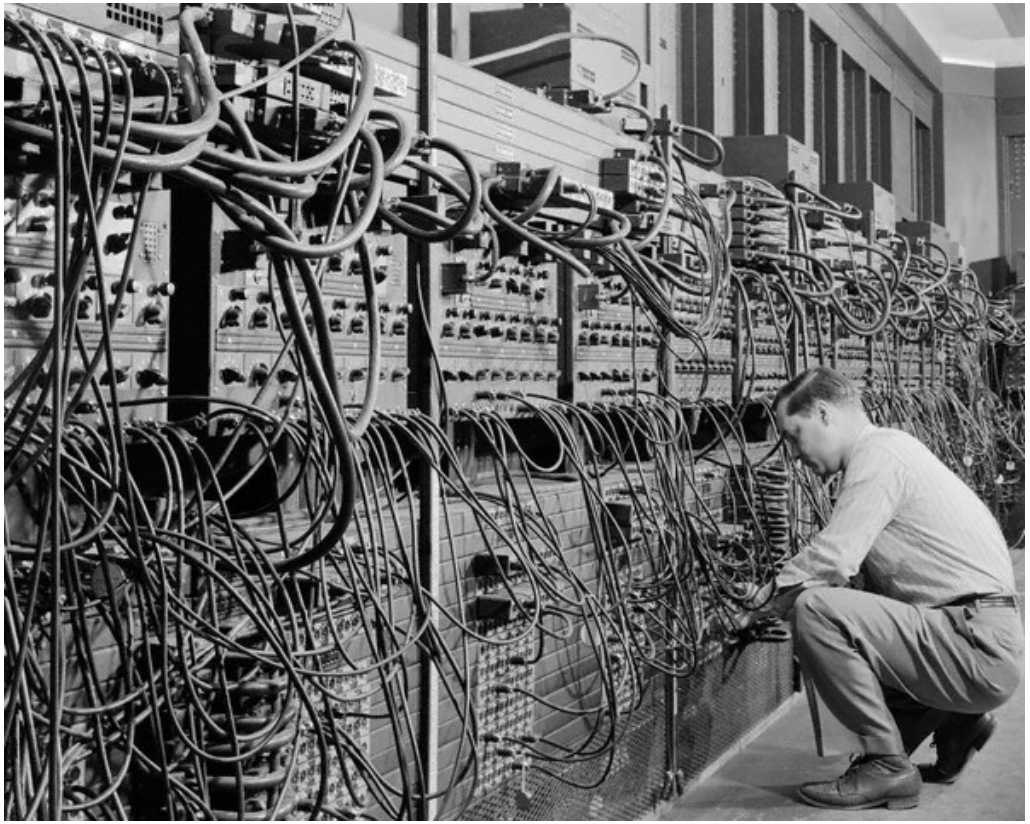
History contd.

- Computers were using vacuum tube technology.

ENIAC's vacuum tubes

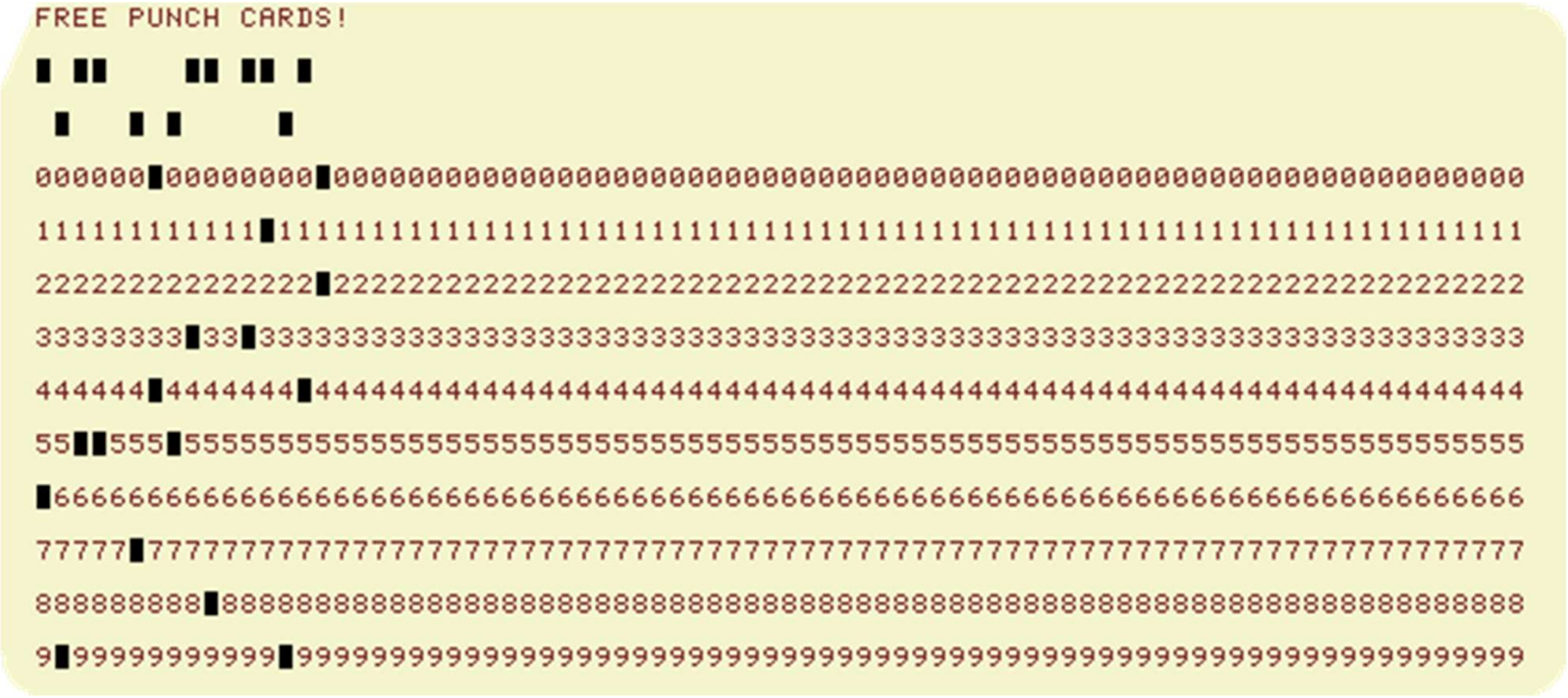
History contd.

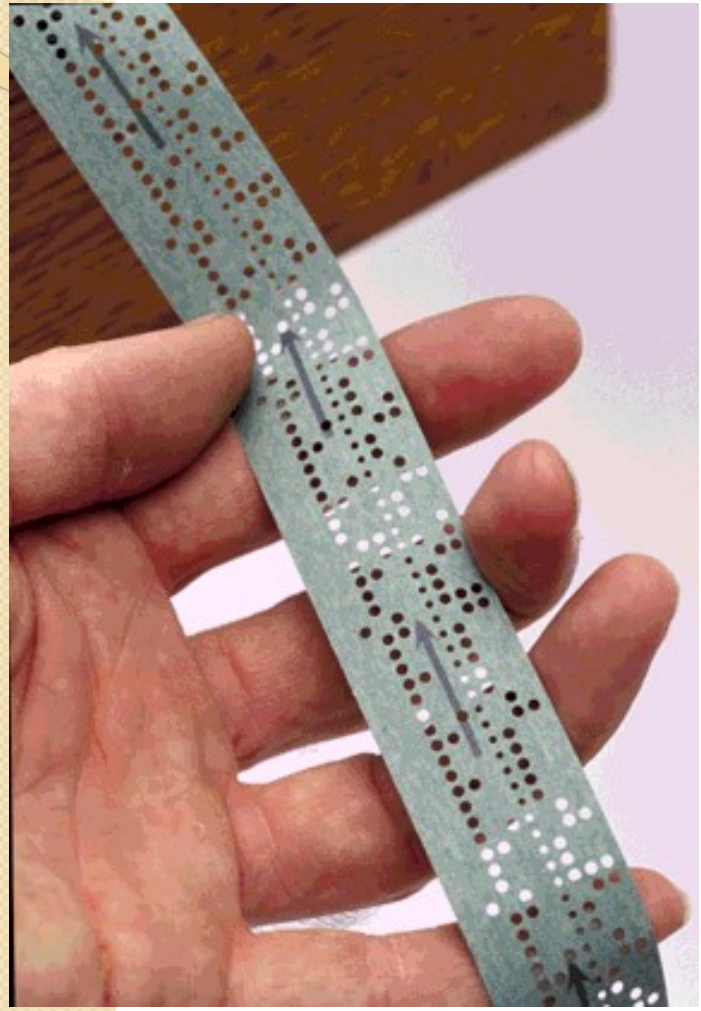
Programs were loaded into memory manually using switches, punched cards, or paper tapes.

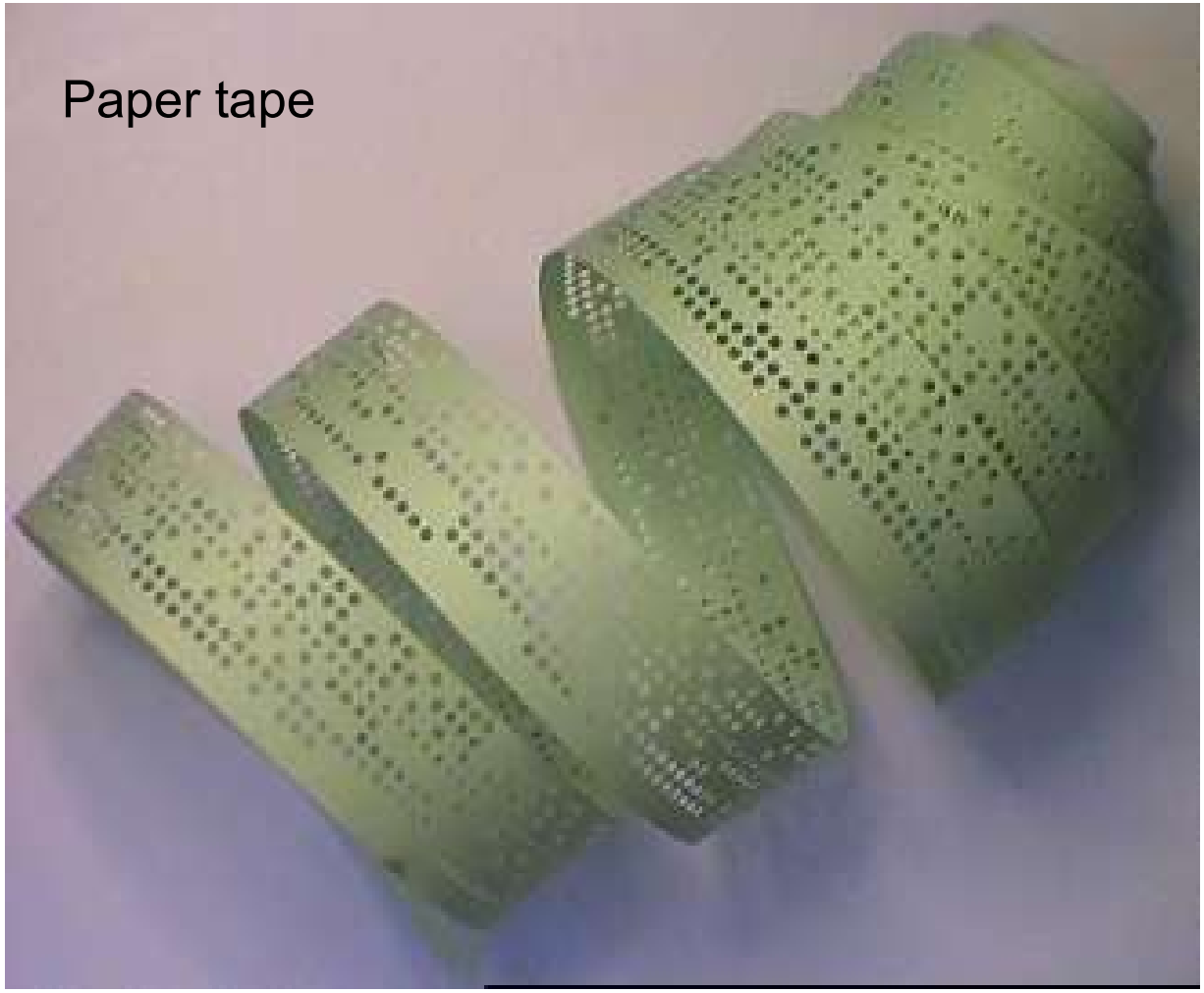


ENIAC : coding by cable connections

Punch Card





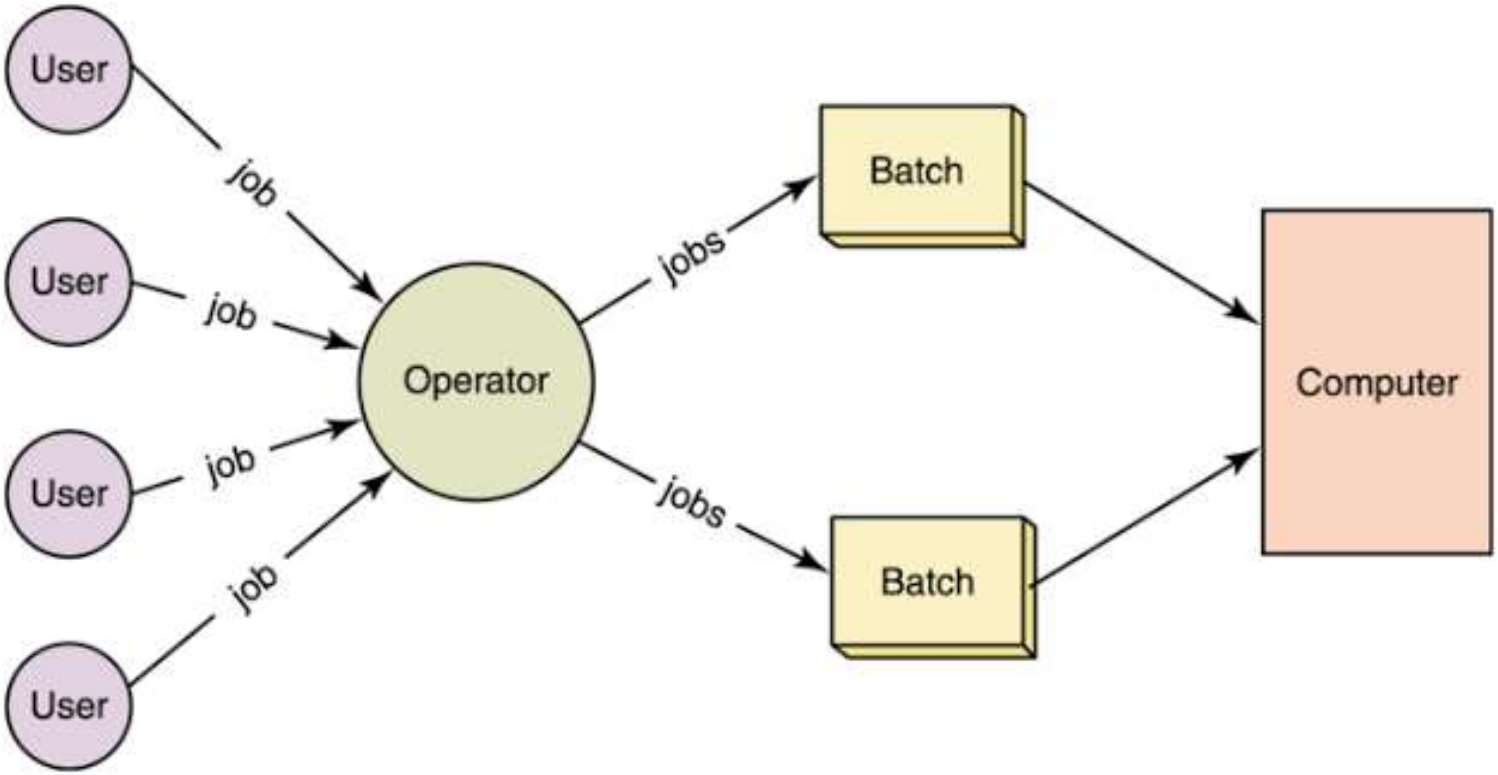

Paper tape

Types of OS

- Batch Operating Systems
- Multiprogramming Operating Systems
- Timesharing / Multitasking Operating Systems
- Multiprocessing Operating Systems
- Real Time Operating Systems
- Network OS
- Embedded OS

Batch Operating Systems

- Users did not interact with computer directly.
- Each user submitted his/her jobs to the operator who in turn fed the programs to the computer.
- User had to come back later to collect the output.



Batch Operating Systems

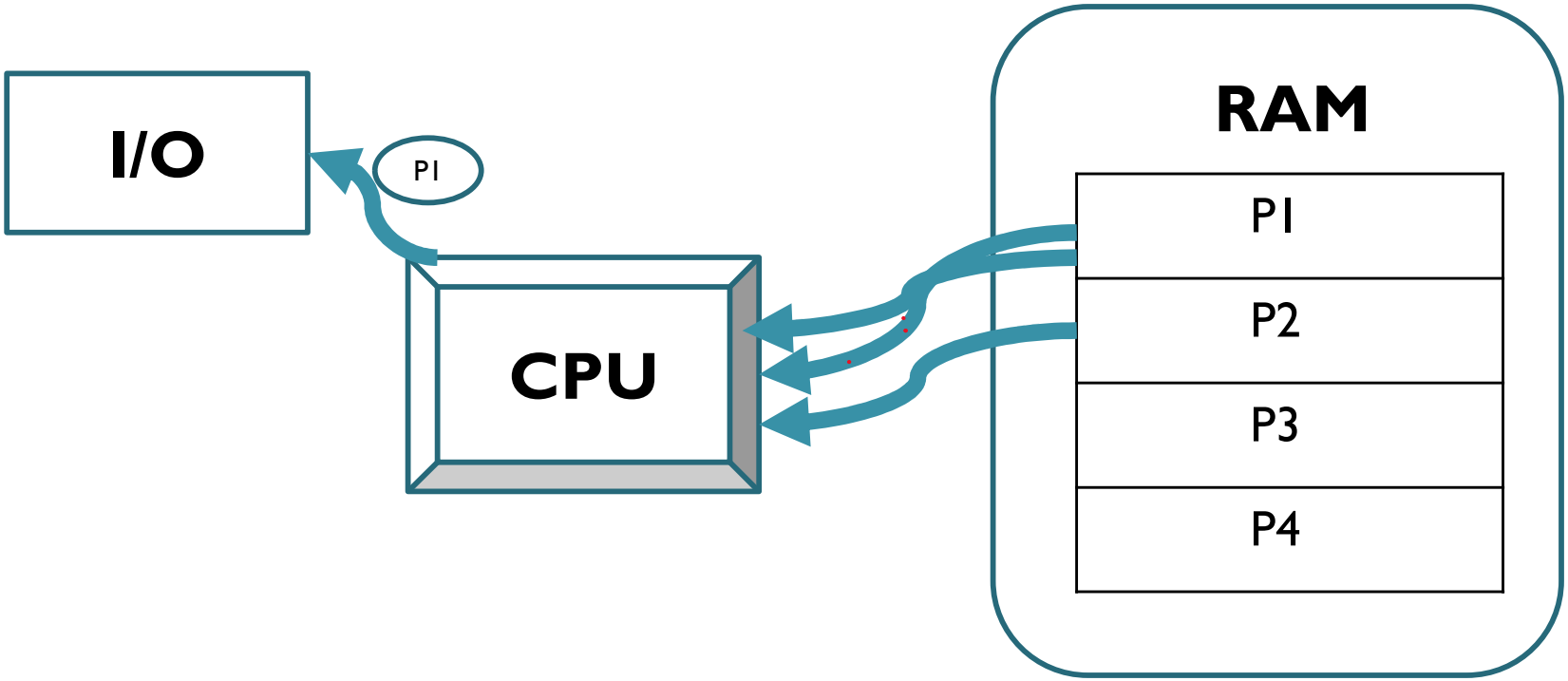
Disadvantages

- Lack of interaction between the user and job
- CPU is often idle
- Difficult to provide the desired priority
- Job Starvation

Multiprogramming OS

- Multiprogramming means sharing of resources between more than one processes.
- CPU time is not wasted, because, while one process moves on some I/O work, the OS picks another process to execute till the current one passes to I/O operation.
- Goal of multiprogramming is to efficiently utilize all of the computing resources.

Multi Programming



Time Sharing Operating System

Time sharing or multitasking is a logical extension of multiprogramming. It is same as multiprogramming with preemption.

- OS switches between user's programs very quickly, generally in round-robin fashion.
- Switching between users is very fast.
- Goal is to give the illusion that each user has his/her own machine.
- Response time is a priority.

Multiprocessing Operating System

- Has multiple processors sharing memory and peripheral devices.
- With this configuration, they have greater computing power and higher reliability.
- Multiprocessor systems are classified into two as tightly-coupled (shared memory) and loosely-coupled (distributed).
- In the tightly-coupled one, each processor is assigned a specific duty but processors work in close association, sharing the same memory.
- In the loosely coupled one, each processor has its own memory and copy of the OS.

Real Time Operating Systems

- A real-time operating system is an operating system that guarantees to process events or data within a stipulated time.

- Two kinds – Hard and soft RTOS

❑ Hard real time

- Strictly follow deadlines, no delay acceptable
- Failure if response time too long.
- Missile launch, air bags in car

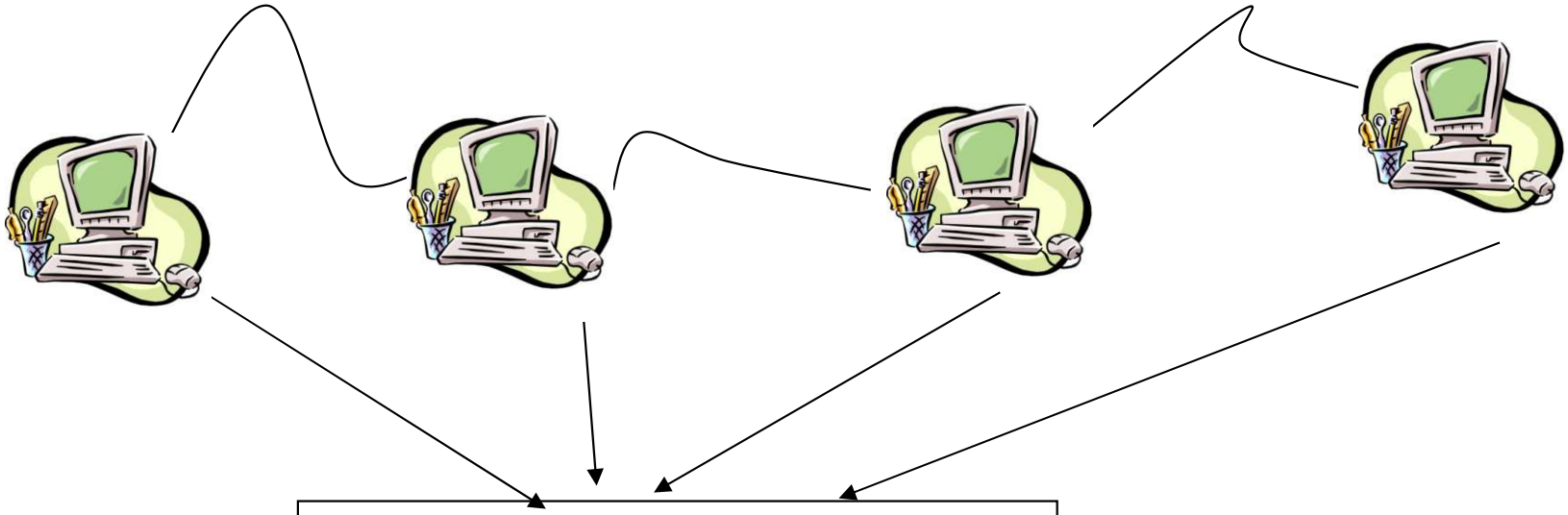
❑Soft

- Less accurate if response time is too long.
- Useful in applications such as multimedia, virtual reality.

Network Operating Systems

- Use of the networks required OSs appropriate for them.
- In **network systems**, each process runs in its own machine but the OS have access to other machines.
- In networks, users are aware of the fact that s/he is working in a network and when information is exchanged.
- The user explicitly handles the transfer of information.
- By this way, file sharing, messaging, etc. became possible

Network OS



Each is a computer having its own CPU, RAM, etc. An OS supporting networks is installed on them.

Embedded Operating System

- Embedded system is a combination of hardware and software designed to do a specific function
- Specifically configured for a certain hardware
- Work on a fixed functionality e.g. microwave, AC, etc. which cannot be changed

Mobile Operating System

- Software that allows smartphones, tablet PCs and other devices to run applications and programs.

- Two of the most widely adopted mobile OS are
  - iPhone's OS, iOS,
  - Google's open source OS, Android

Operating System Components

- Resource Management
- Process Management
- Memory Management
- Storage/Filesystem Management
- Security

Resource Management

- Manages and protects multiple computer resources: CPU, Internal/External memory, files, Applications, Communication channels, etc…
- Handles and allocates resources to multiple users or multiple programs running at the same time and space (e.g., processor time, memory, I/O devices).
- Decides between conflicting requests for efficient and fair resource use (e.g., maximize throughput, minimize response time).
- Load balancing

Process Management

- A process is a program in execution
- Typically system has many processes, some user, some operating system running concurrently on one or more CPUs
- Process needs resources to accomplish its task
- Process termination requires reclaim of any reusable resources
- Concurrency is achieved by multiplexing the CPUs among the processes / threads

Process Management Activities

Complete activity required for managing a process throughout its lifecycle

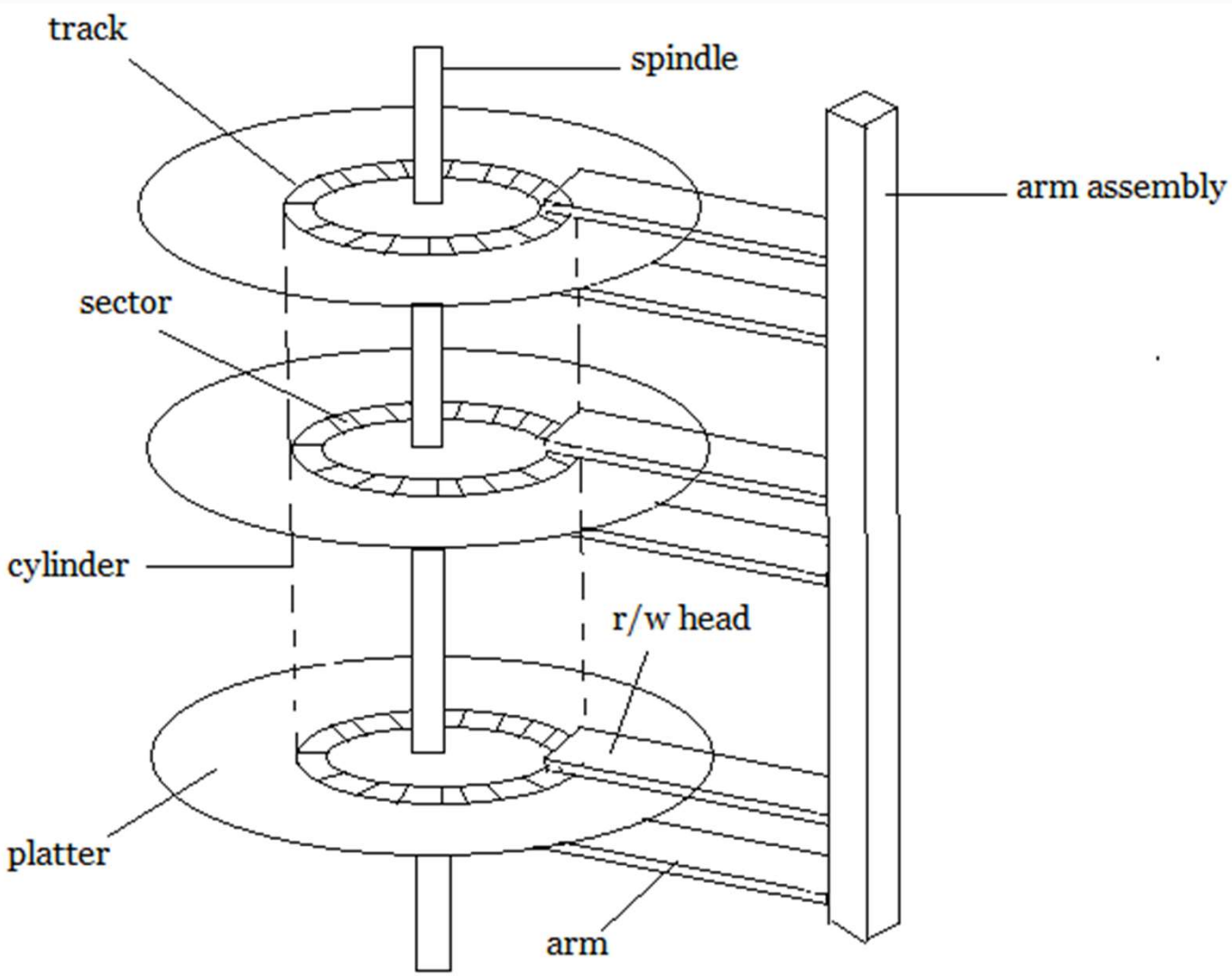
- Allocate resources to processes
- Enable processes to share and exchange information
- Protect the resources of each process from other processes
- Enable synchronization among processes.
- Maintain a data structure for each process
- Execution and Control of processes

Memory Management

- To execute a program all (or part) of the instructions must be in main memory
- All (or part) of the data that is needed by the program must be in memory
- Memory management determines what is in memory and when
  - Optimizing CPU utilization and computer response to users
- Memory management activities:
  - Allocating and deallocating memory space as needed
  - Keeping track of which parts of memory are currently being used and by whom
  - Deciding which processes (or parts thereof) and data to move into and out of memory

Mass - Storage Management

- Usually, disks used to store data that does not fit in main memory or data that must be kept for a "long" period of time
- Proper management is of central importance
- Entire speed of computer operation hinges on disk subsystem and its algorithms
- OS activities
  - Mounting and unmounting
  - Free-space management
  - Storage allocation
  - Disk scheduling
  - Partitioning
  - Protection



Structure of a magnetic disk

Protection and Security

- **Protection** – mechanism for controlling access of processes or users to resources defined by the OS
- **Security** – defense of the system against internal and external attacks
  - Huge range, including denial-of-service, worms, viruses, identity theft, theft of service

Protecting processes from each other

Problem:

How to run multiple applications in such a way that they are protected from one another

Goal:

- Keep User Programs from Crashing each other
- Keep User Programs from Crashing OS

Keep Parts of OS from crashing other parts?

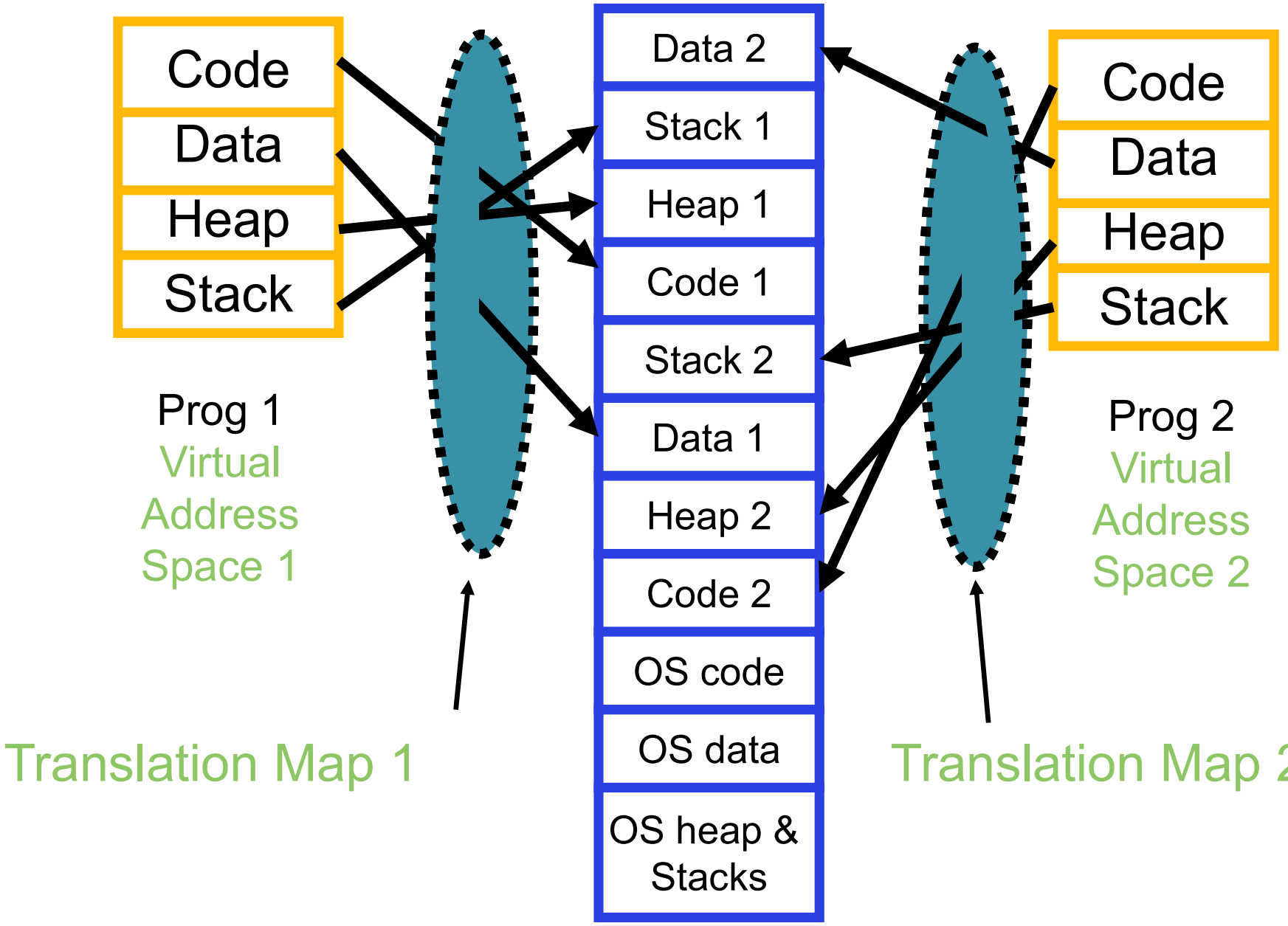
Simple Policy:

Programs are not allowed to read/write memory of other Programs or of Operating System

Mechanisms:

- Address Translation
- Dual Mode Operation

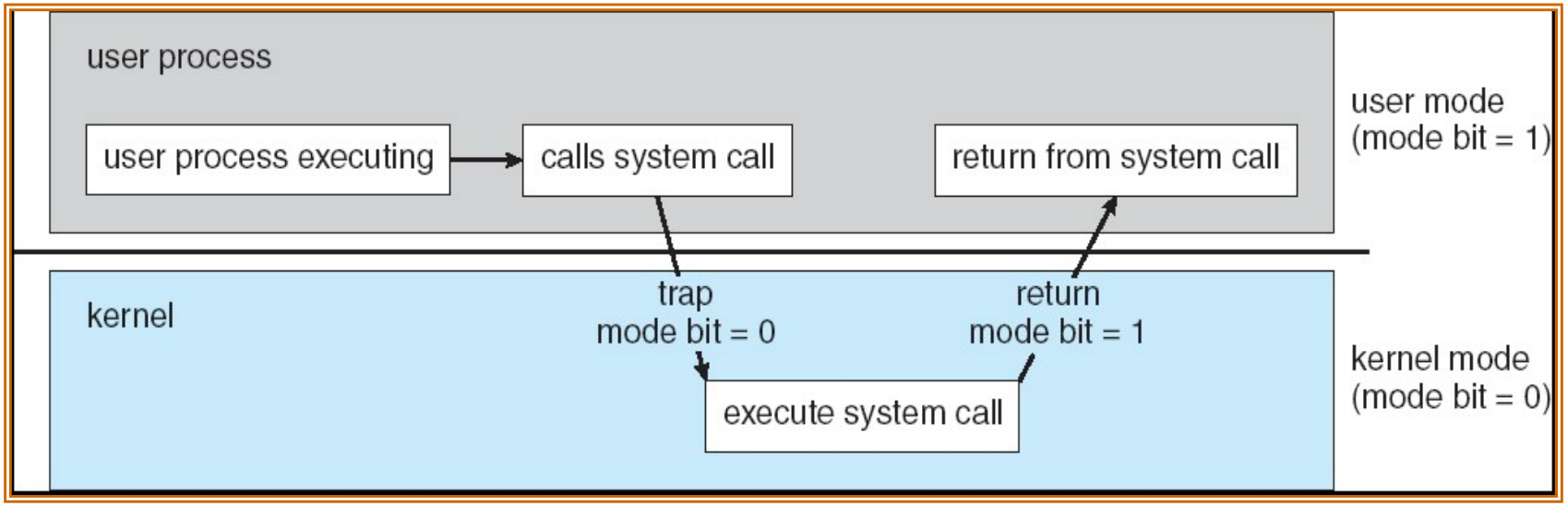
Example of Address Translation



Dual Mode Operation

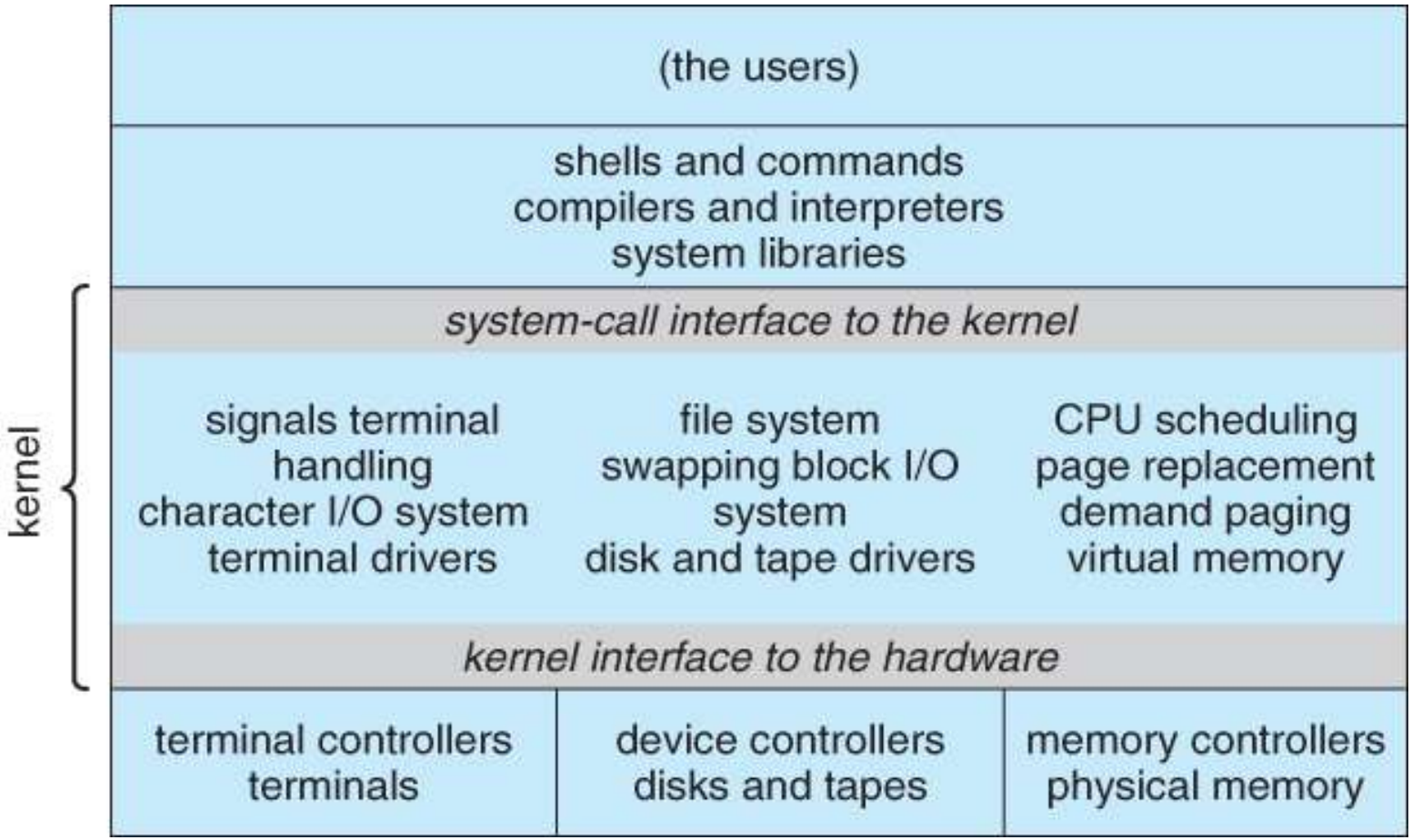
- Two modes:
  - "Kernel" mode (or "supervisor" or "protected")
  - "User" mode: Normal programs executed
- Some instructions/ops prohibited in user mode:
  - Example: cannot modify page tables in user mode

    Attempt to modify ⇒ Exception generated
- Transitions from user mode to kernel mode:
  - System Calls, Interrupts, Other exceptions



The OS Kernel

- The kernel is the central module of an operating system (OS).
- It is the part of the operating system that loads first, and it remains in main memory.
- Consists of everything below the system-call interface and above the physical hardware



User space vs Kernel space

- System memory can be divided into two distinct regions: kernel space and user space.

- Kernel space is where the kernel (i.e., the core of the operating system) executes (i.e., runs) and provides its services.

- User space is that set of memory locations in which user processes (i.e., everything other than the kernel) run.

Kernel Mode

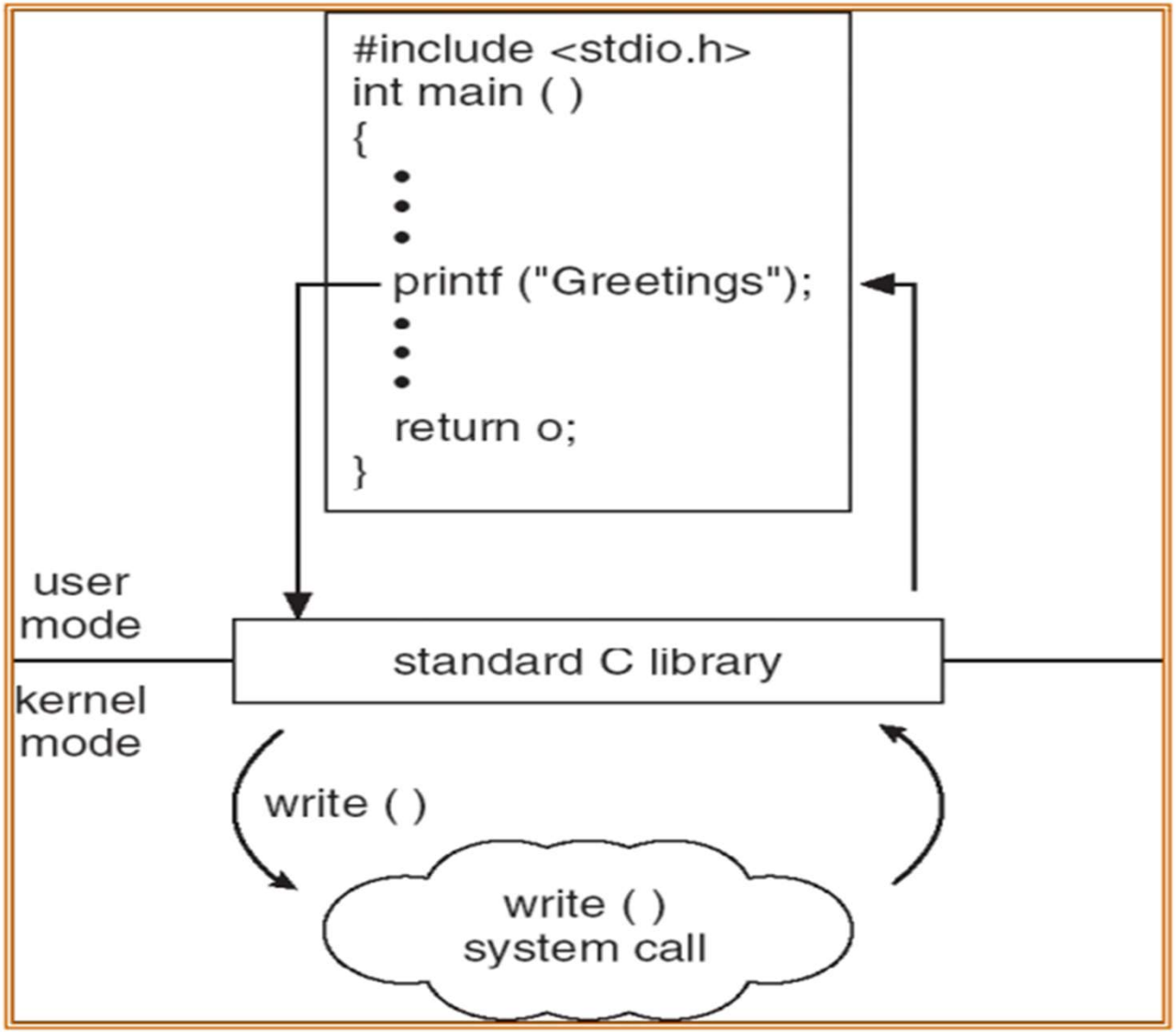
- Also called privileged mode, or as system mode
- Used mainly for Restriction/ Protection from unauthorized user application program
- When the CPU is in kernel mode, it is assumed to be executing trusted software, and thus it can execute any instructions and reference any memory addresses (i.e., locations in memory).

User Mode

- It is a non-privileged mode.
- It is non-privileged in that it is forbidden for processes in this mode to access those portions of memory (i.e., RAM) that have been allocated to the kernel or to other programs.
- User mode is the normal mode of operating for programs.
- Code running in user mode must delegate to system APIs to access hardware or memory.
- Most of the code running on your computer will execute in user mode.

System Call

Example



EXAMPLES OF WINDOWS AND UNIX SYSTEM CALLS

	Windows	Unix
Process Control	CreateProcess()	fork()
	ExitProcess()	exit()
	WaitForSingleObject()	wait()
File Manipulation	CreateFile()	open()
	ReadFile()	read()
	WriteFile()	write()
	CloseHandle()	close()
Device Manipulation	SetConsoleMode()	ioctl()
	ReadConsole()	read()
	WriteConsole()	write()
Information Maintenance	GetCurrentProcessID()	getpid()
	SetTimer()	alarm()
	Sleep()	sleep()
Communication	CreatePipe()	pipe()
	CreateFileMapping()	shmget()
	MapViewOfFile()	mmap()
Protection	SetFileSecurity()	chmod()
	InitlializeSecurityDescriptor()	umask()
	SetSecurityDescriptorGroup()	chown()

Introduction

- Developed by Linus Torvalds
- Initial release in1991
- Used in most of the computers, ranging from super computers to embedded system
- Multi user
- Multi tasking
- Monolithic kernel

FOSS

- Free Open Source Software
- Free – No charge for using it
- Open source – Source code is available and any body can contribute to the development. Organization independent

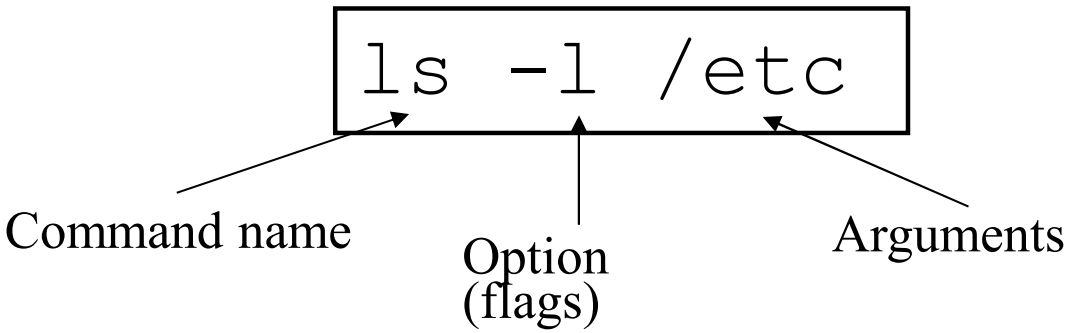
Why Linux

- Growing popularity
- Powerful
  - Runs on multiple hardware platforms
  - Users like its speed and stability
  - No requirement for latest hardware
  - Secure
- Its free
  - Vendors are distributors who package Linux

Linux Distributions

Getting started with Linux

- Logging in
- Commands
- Command structure



Linux File System

- Everything is a file
  - General files
  - Directory files
  - Device files
- No drives
- Peripherals like hard drives, cd rom, printers are also considered files

File Structure

Tree like hierarchical file system

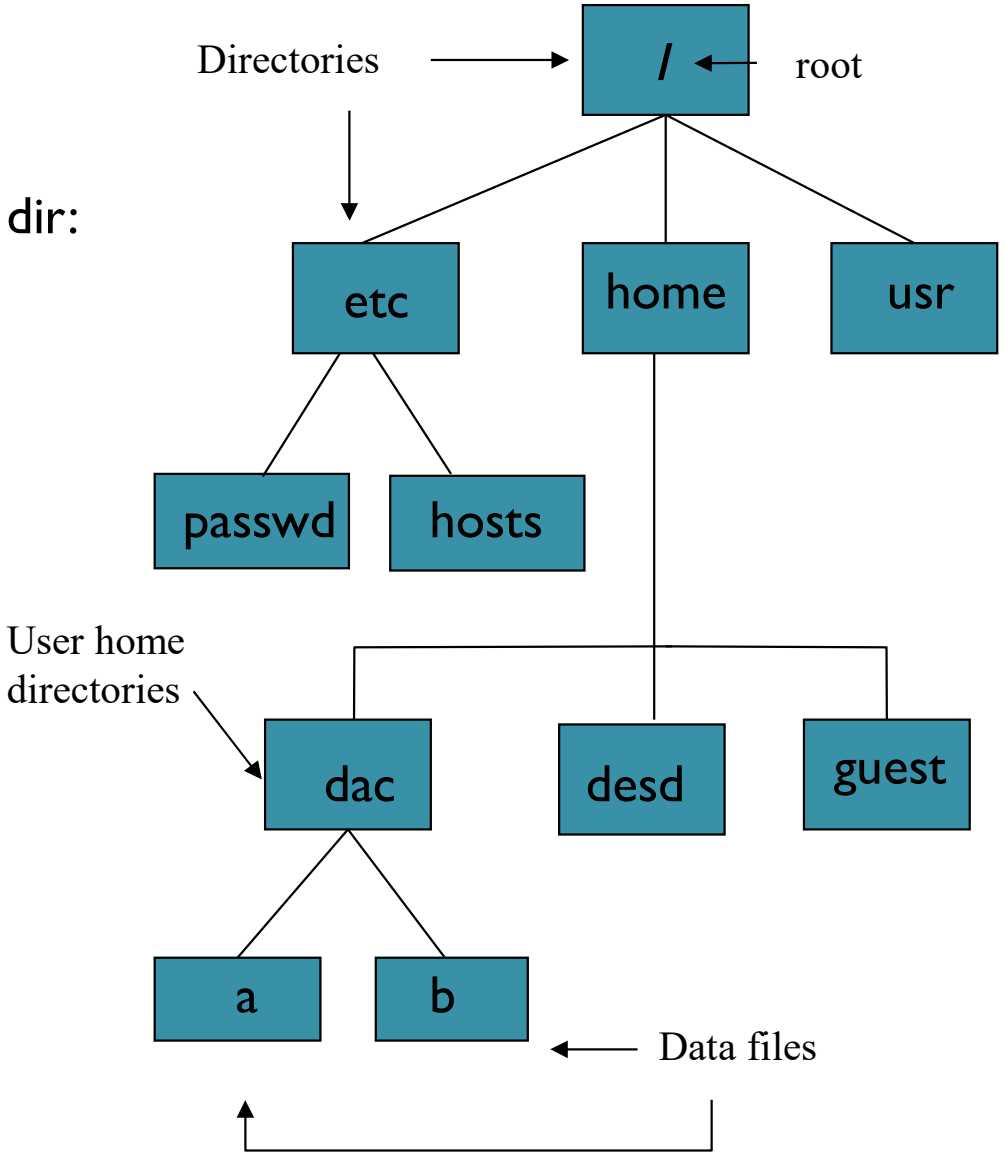
Root is the main directory (denoted with /)

Special files

- **/home**
- **/bin, /usr/bin**
- **/sbin, /usr/sbin**
- **/etc**
- **/lib, /usr/lib**
- **/var**
- **/dev**
- **/tmp**
- **/opt**

File Structure

- Absolute path
  - To access file a from current dir:
    /home/dac/a
- Relative path
  - To access file dac from a:
    ../dac



Linux FS vs Windows FS

Linux vs. Windows File Structure

- In Linux, there are no drives like C:, D:
- Linux files are ordered in a tree structure
- Top hierarchy is /
- Path separator is / not \
- File extensions do not have any meaning

User accounts

- 3 types of user accounts
  - Regular
  - Root
  - Service account
- Root user is super user and has all admin privileges
- For every user, /home/<username> directory is created which is called home directory

Common Commands

- **pwd**
- **cat**
- **echo**
- **man**
- **cd <*dir*>**
- **ls**

Common Commands

Command	Description
cd or cd ~	Navigate to HOME directory
cd ..	Move one level up
cd	To change to a particular directory
cd /	Move to the root directory

ls -l

1st Column	File type and access permissions
2nd Column	# of HardLinks to the File
3rd Column	Owner and the creator of the file
4th Column	Group of the owner
5th Column	File size in Bytes
6th Column	Date and Time
7th Column	Directory or File name

File commands

- **cp <*fromfile*> <*tofile*>**
- **mv <*fromfile*> <*tofile*>**
- **rm <*file*>**
- **mkdir <*newdir*>**
- **rmdir <*dir*>**

Authorization in Linux

- Linux divides authorization in two levels
  - Ownership
  - Permission

- There are 3 user types on a Linux system
  - Owner (user)
  - Group
  - Others

File Permissions

- Every file
  - Is owned by someone
  - Belongs to a group
  - Has certain access permissions for owner, group, and others
- Every user:
  - Has a uid (login name), gid (login group) and membership of a "groups" list:

File Permissions

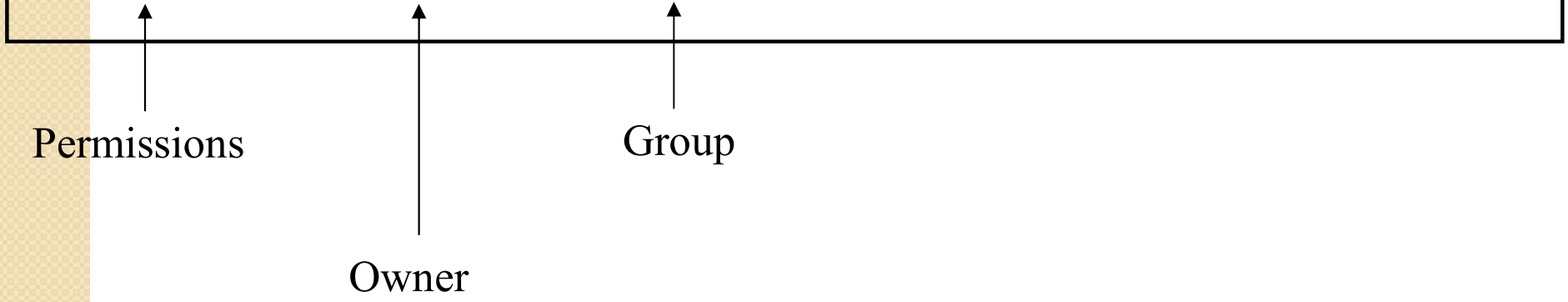
Linux provides three kinds of permissions:

- **Read** - users with read permission may read the file or list the directory (r)
- **Write** - users with write permission may write to the file or new files to the directory (w)
- **Execute** - users with execute permission may execute the file or lookup a specific file within a directory (x)

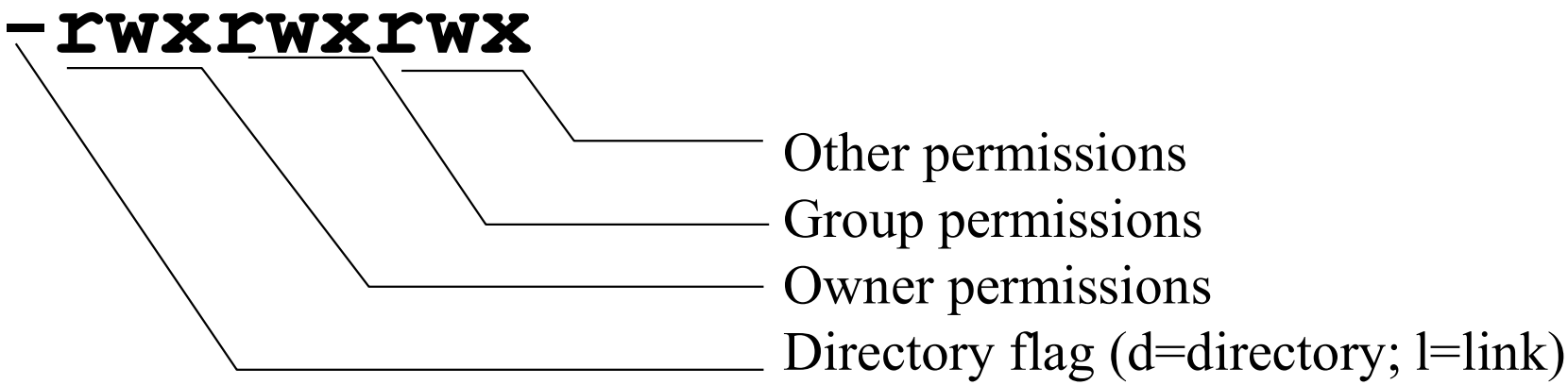
File Permissions

- The long version of a file listing (`ls -l`) will display the file permissions:

```
-rwxrwxr-x   1 rvdheij  rvdheij      5224 Dec 30 03:22 hello
-rw-rw-r--   1 rvdheij  rvdheij       221 Dec 30 03:59 hello.c
-rw-rw-r--   1 rvdheij  rvdheij      1514 Dec 30 03:59 hello.s
drwxrwxr-x   7 rvdheij  rvdheij      1024 Dec 31 14:52 posixuft
```



Interpreting File Permissions



Changing Permissions

- chmod
- chmod permissions filename
- Absolute and symbolic mode

Absolute(Numeric) mode

Number	Permission Type	Symbol
0	No Permission	---
1	Execute	--x
2	Write	-w-
3	Execute + Write	-wx
4	Read	r--
5	Read + Execute	r-x
6	Read +Write	rw-
7	Read + Write + Execute	rwx

Symbolic mode

- To change the permissions for a specific owner

Operator	Description
+	Adds a permission to a file or directory
-	Removes the permission
=	Sets the permission and overrides the permissions set earlier.

User Denotations	
u	user/owner
g	group
o	other
a	all

Changing Ownership

- chown
- chown user <filename>
- chown user:group <filename>

- Groups
  - id
  - /etc/groups
  - chgrp group filename

Summary

- Linux being a multi-user system uses permissions and ownership for security.
- There are three user types on a Linux system viz. User, Group and Other
- Linux divides the file permissions into read, write and execute denoted by r,w, and x
- The permissions on a file can be changed by 'chmod' command which can be further divided into Absolute and Symbolic mode
- The 'chown' command can change the ownership of a file/directory. Use the following commands:
  chown user file or chown user:group file
- The 'chgrp' command can change the group ownership **chrgrp group filename**

Pipes, grep

- Pipes
  - "|" denotes pipe
  - Help combine two or more commands
  - Output of one command serves as input for the next command

- grep
  - Used to find strings and values in a text document
  - Present the result in a format you want
  - grep <search_string>

- Less, pg and more commands are used for dividing a long file into readable bits

Options with grep

Option	Function
-v	Shows all the lines that do not match the searched string
-c	Displays only the count of matching lines
-n	Shows the matching line and its number
-i	Match both (upper and lower) case
-l	Shows just the name of the file with the string

Sort command

- Sorting the contents of a file alphabetically
- sort <filename>

Option	Function
-r	Reverses sorting
-n	Sorts numerically
-f	Case insensitive sorting

More commands

- **wc** command is used to count lines, words and characters, depending on the option used.
  - usage: wc [options] [file name]

- Options:

  -l : Number of lines

  -w: Number of words

  -c: Number of characters

More commands

- **diff**
  - diff filename1 filename2

- **find**
  - find path –name "filename"
  - find / –name "*.log"

Network commands

- For communication with other devices
  - telnet
    - Used to connect to a remote Linux computer and work on it
    - telnet <IP address or hostname>

  - ssh
    - Securely connect to a remote Linux computer
    - More secure than telnet
    - ssh username@<IP address or hostname>

FTP

- FTP is file transfer Protocol for data transfer among computers
- Logging in and establishing secure connection with a remote host
- Upload and download files
- Navigating through directories
- Browsing contents of the directories

FTP

Command	Function
dir	Display files in the current directory of remote computer
cd "dirname"	change directory to "dirname" on remote computer
put file	upload 'file' from local to remote computer
get file	Download 'file' from remote to local computer
quit	Logout

SFTP

- FTP is not secure
  - Data sent is in clear text and not encrypted
- SFTP is Secure File Transfer Protocol
  - Adds a layer of security
  - Data is encrypted
  - Authenticates user and server

More network commands

- finger
  - Display information about the system users.
  - finger <username>
- scp
  - SCP (secure copy) is a command-line utility that allows you to securely copy files and directories between two locations.
  - scp file.txt remote_username@IP:/remote/directory
- ping
  - Check connections with a hostname or IP address
  - ping <IP address or hostname>

System variables – PS1

- **PS1** is the primary prompt string
  - Environment variable which contains the value of the default prompt.
  - For most Linux systems, the defaults values have [\u@\h \W]\$
    - u is the username
    - h is the hostname
    - W is the current working directory
  - To see default value of PS1,
    - echo $PS1
  - To change the value of PS1,
    - export PS1="new value"

```
\a      an ASCII bell character (07)
\d      the date in "Weekday Month Date" format (e.g., "Tue May
        26")
\D{format}
        the format is passed to strftime(3) and the result is
        inserted into the prompt string; an empty format results
        in a locale-specific time representation. The braces are
        required
\e      an ASCII escape character (033)
\h      the hostname up to the first `.'
\H      the hostname
\j      the number of jobs currently managed by the shell
\l      the basename of the shell's terminal device name
\n      newline
\r      carriage return
\s      the name of the shell, the basename of $0 (the portion
        following the final slash)
\t      the current time in 24-hour HH:MM:SS format
\T      the current time in 12-hour HH:MM:SS format
\@      the current time in 12-hour am/pm format
\A      the current time in 24-hour HH:MM format
\u      the username of the current user
\v      the version of bash (e.g., 2.00)
\V      the release of bash, version + patch level (e.g., 2.00.0)
\w      the current working directory, with $HOME abbreviated
        with a tilde (uses the value of the PROMPT_DIRTRIM vari-
        able)
\W      the basename of the current working directory, with $HOME
        abbreviated with a tilde
\!      the history number of this command
\#      the command number of this command
\$      if the effective UID is 0, a #, otherwise a $
\nnn    the character corresponding to the octal number nnn
\\      a backslash
\[      begin a sequence of non-printing characters, which could
        be used to embed a terminal control sequence into the
        prompt
```

System variables – PS2

- If the command is too long to fit in one line, it can be broken down into multiple lines by giving "\" at the end of each line
- **PS2**: environment variable which contains the value the prompt used for a command continuation interpretation.
- The default interactive **PS2** value prompt for a multi-line command is "**>**"
- export PS2="new value"

Some more commands

uname	.. know your machine's characteristics
logout	.. logs off system
cut	.. slitting a file vertically
cat	.. display/create files
wc	.. count lines, words, characters
gzip	.. compressing a file
gunzip	.. uncompressing a file
more	.. views a file, pausing every screenful
less	.. similar to more, more powerful
file	.. show file type
tail	.. show the last few lines of a file
head	.. show the beginning of a file
w	.. shows who is logged on and what they're doing
finger	.. shows more information about a user
df	.. shows disk space available on the system
du	.. shows how much disk space is being used by folders
bc	.. a simple calculator
cal	.. display calendar
date	.. display system date

Redirection

- Input redirection
  - Standard input (stdin) device is keyboard
  - "<" is the input redirection operator
- Output redirection
  - Standard output (stdout)device is screen
  - ">" is the output redirection operator
  - ">>" appends output to an existing file
- Error redirection
  - Error is displayed on screen by default
  - Error redirection is routing the error to a file other than the screen

Regular expressions

- RegEx are a set of characters to check patterns in strings
- Basic regular expressions
  - . Replaces any character
  - * Replaces 0 or more characters
  - ^ Matches start of the string
  - $ Matches end of the string
  - * {n} Matches the preceding character appearing ‘n’ times exactly
- Brace expansion

Shell

- Shell is an active instance of a program (a process) that takes commands typed by the user and calls the OS(using system library routines) to run those commands
- Shell acts as a wrapper around the OS – hence, known by the term shell
- Shell is a special utility that plays several roles – it is a versatile utility
  - Command interpreter
  - Command editor
  - Job controller
  - Programming language interpreter
  - CLI – command line interface to the OS

Different Shells

- Bourne shell(sh)
  - It was developed by Steve Bourne (AT&T) in 1977
- Had many new features and was foundation for many newer derivatives
  - C shell (csh) - 1978
  - Korn shell (ksh) - 1983
  - TENEX C Shell (tcsh) - 1983
  - Bash (bash) -1989

Bash shell features

- Compatible with Bourne shell
- Job control
- History list
- Command-line editing
- Aliases
- Functions
- Arrow keys for command editing
- Control structures for conditional testing and iteration
- Basic debugging and exception handling

Environment Variables

- Dynamic values
- Exist in every OS
- Can be created, edited, saved and deleted
- Change the way software/programs behave
- Eg.
  - PATH
  - USER
  - HOME
- Use **export** command to set the environment variable
- echo $VARIABLE : To display value of a variable

PATH Environment Variable

PATH: The search path for commands. It is a colon-separated list of directories that are searched when you type a command.

Usually, we type in the commands in the following way:

$ ./command

By setting PATH=$PATH:. our working directory is included in the search path for commands, and we simply type:

$ command

If we include the following lines in the ~/.bash_profile:

```
PATH=$PATH:$HOME/bin
export PATH
```

we obtain that the directory /home/userid/bin is included in the search path for commands.

Shell script

- Program which interprets user commands through CLI like terminal
- Shell scripting is writing a series of commands for the shell to execute
- Helps creating complex programs containing conditional statements, loops and functions

Basic Shell Programming

- A script is a file that contains shell commands
  - data structure: variables
  - control structure: sequence, decision, loop

- Shebang line for bash shell script:

  #! /bin/bash

  #! /bin/sh

- to run:
  - make executable: **% chmod +x script**
  - invoke via: **% ./script**

Bash program

- We write a program that copies all files into a directory, and then deletes the directory along with its contents. This can be done with the following commands:

```
$ mkdir temp
$ cp *.log temp
$ rm *.log
```

- Instead of having to type all that interactively on the shell, write a shell program instead:

```
$ cat log_temp.sh
        #!/bin/bash
        # this script copies log files to temp dir
        mkdir temp
        cp *.log temp
        rm *.log
        echo "Log files copied"
```

Shell Metacharacters

Symbol	Meaning
>	Output redirection,
>>	Output redirection
<	Input redirection
*	File substitution wildcard; zero or more characters
?	File substitution wildcard; one character
[ ]	File substitution wildcard; any character between brackets
`cmd`	Command Substitution
$(cmd)	Command Substitution
|	The Pipe (|)
;	Command sequence, Sequences of Commands
||	OR conditional execution
&&	AND conditional execution
( )	Group commands, Sequences of Commands
&	Run command in the background, Background Processes
#	Comment
$	Expand the value of a variable
\	Prevent or escape interpretation of the next character
<<	Input redirection

Variables

- Can use variables as in any programming languages.
- Values are always stored as strings
- Mathematical operators in the shell language convert variables to numbers for calculations.
- No need to declare a variable
- Format for setting a value to a variable:

  Name = Value

- Access the variable by $ symbol
- Rules
  - No space
  - No number in the beginning
  - No $ in name
  - Case sensitive
- Example

```
#!/bin/bash
STR="Hello World!"
echo $STR
```

Variables

- The shell programming language does not type-cast its variables.
- count=0

  count=Sunday

- It is recommended to use a variable for only a single TYPE of data in a script.

- \ is the bash escape character and it preserves the literal value of the next character that follows.
  - $ echo *

Single and double quote

- When assigning character data containing spaces or special characters, the data must be enclosed in either single or double quotes.

- Using double quotes to show a string of characters will allow any variables in the quotes to be resolved

```
$ var="test string"
$ newvar="Value of var is $var"
$ echo $newvar
Value of var is test string
```

- Using single quotes to show a string of characters will not allow variable resolution

```
$ var='test string'
$ newvar='Value of var is $var'
$ echo $newvar
Value of var is $var
```

Command Substitution

- The backquote "`" is different from the single quote "'". It is used for command substitution: `command`

```
$ LIST=`ls`
$ echo $LIST
hello.sh read.sh
```

- We can also perform the command substitution by means of $(command)

```
$ LIST=$(ls)
$ echo $LIST
hello.sh read.sh

$ rm $( find / -name "*.tmp" )
```

Read command

- The read command allows you to prompt for input and store it in a variable.

- Example:

```
#!/bin/bash
echo -n "Enter name of file to delete: "
read file
rm $file
```

- Line 2 prompts for a string that is read in line 3.

Shell parameters

- Positional parameters are assigned from the shell's argument when it is invoked. Positional parameter "N" may be referenced as "${N}", or as "$N" when "N" consists of a single digit.

Parameter	Meaning
$0	Name of the current shell script
$1-$9	Positional parameters 1 through 9
$#	The number of positional parameters
$*	All positional parameters, "$*" is one string
$@	All positional parameters, "$@" is a set of strings
$?	Return status of most recently executed command
$$	Process id of current process

Examples: Command Line Arguments

```
% set blue green red yellow
          $1   $2    $3    $4
% echo $*
blue green red yellow
% echo $#
4
% echo $1
blue
% echo $3 $4
red yellow
```

The 'set' command can be used to assign values to positional parameters

Arithmetic Evaluation

- The let statement can be used to do mathematical functions:

```
$ let X=10+2*7
$ echo $X
24
$ let Y=X+2*4
$ echo $Y
32
```

- An arithmetic expression can be evaluated by $[expression] or $((expression)) or "expr" command

```
$ echo "$((123+20))"
143

$ TEMP=$[123+20]
$ echo "$[123*$TEMP]"
17589
echo $(expr $x + $y )
```

Arithmetic Evaluation

- An arithmetic expression can be evaluated by $[expression] or $((expression)) or "expr" command

```
$ echo "$((123+20))"
143
$ TEMP=$[123+20]
$ echo "$[123*$TEMP]"
17589
echo $(expr $x + $y )
```

For floating point arithmetic operations, we need to use a tool "bc (basic calculator)"

```
$ echo "$x+$y" | bc
```

Arithmetic Evaluation

- Available operators: +, -, /, *, %

- Example : Accept two numbers as input, perform +,-,/,*,% functions on them and print the output.

Solution

```
$ cat arithmetic.sh
#!/bin/bash
echo -n “Enter the first number: ”; read x
echo -n “Enter the second number: ”; read y
add=$(($x + $y))
sub=$(($x - $y))
mul=$(($x * $y))
div=$(($x / $y))
mod=$(($x % $y))
# print out the answers:
echo “Sum: $add”
echo “Difference: $sub”
echo “Product: $mul”
echo “Quotient: $div”
echo “Remainder: $mod”
```

Shell Programming contd.

Prachi Pandey
C-DAC Bangalore

bash control structures

- if-then-else
- case
- loops
  - for
  - while
  - until

If statement

- If conditionals let us decide whether to perform an action or not, this decision is taken by evaluating an expression.

```
if [ expression ];
then
        statements
elif [ expression ];
then
        statements
else
        statements
fi
```

- the elif (else if) and else sections are optional
  - The word **`elif`** stands for "else if"
  - It is part of the if statement and cannot be used by itself

- Put spaces after [and before], and around the operators and operands.

test command

Syntax:

test expression

[expression]

- evaluates ‘expression’ and returns true or false

Example:

```
if test –w "$1"
  then
  echo "file $1 is write-able"
fi
```

Example: if… Statement

```
# The following THREE if-conditions produce the same result

* DOUBLE SQUARE BRACKETS
read -p "Do you want to continue?" reply
if [[ $reply = "y" ]]; then
   echo "You entered " $reply
fi

* SINGLE SQUARE BRACKETS
read -p "Do you want to continue?" reply
if [ $reply = "y" ]; then
   echo "You entered " $reply
fi

* "TEST" COMMAND
read -p "Do you want to continue?" reply
if test $reply = "y"; then
   echo "You entered " $reply
fi
```

Example: if..elif... Statement

```
#!/bin/bash
read –p "Enter Income Amount: " Income
read –p "Enter Expenses Amount: " Expense

Net=$(($Income-$Expense))

if [ $Net -eq 0 ]; then
   echo "Income and Expenses are equal - breakeven."
elif [ $Net -gt 0 ]; then
   echo "Profit of: " $Net
else
   echo "Loss of: " $Net
fi
```

Expressions

- An expression can be: String comparison, Numeric comparison, File operators and Logical operators and it is represented by [expression]:

- String Comparisons:

=	compare if two strings are equal
!=	compare if two strings are not equal
-n	evaluate if string length is greater than zero
-z	evaluate if string length is equal to zero

- Examples:

[s1 = s2]	(true if s1 same as s2, else false)
[s1 != s2]	(true if s1 not same as s2, else false)
[-n s1]	(true if s1 has a length greater then 0, else false)
[-z s2]	(true if s2 has a length of 0, otherwise false)

Expressions

- Number Comparisons:

-eq	compare if two numbers are equal
-ge	compare if one number is greater than or equal to a number
-le	compare if one number is less than or equal to a number
-ne	compare if two numbers are not equal
-gt	compare if one number is greater than another number
-lt	compare if one number is less than another number

- Examples:

[n1 -eq n2]	(true if n1 same as n2, else false)
[n1 -ge n2]	(true if n1 greater then or equal to n2, else false)
[n1 -le n2]	(true if n1 less then or equal to n2, else false)
[n1 -ne n2]	(true if n1 is not same as n2, else false)
[n1 -gt n2]	(true if n1 greater then n2, else false)
[n1 -lt n2]	(true if n1 less then n2, else false)

Examples

```
$ cat user.sh
 #!/bin/bash
        echo -n “Enter your login name: "
        read name
        if [ “$name” = “$USER” ];
        then
                echo “Hello, $name. How are you today ?”
        else
                echo “You are not $USER, so who are you ?”
        fi

$ cat number.sh
#!/bin/bash
        echo -n “Enter a number 1 < x < 10: "
        read num
        if [ “$num” -lt 10 ];       then
                if [ “$num” -gt 1 ]; then
                        echo “$num*$num=$(($num*$num))”
                else
                        echo “Wrong input !”
                fi
        else
                echo “Wrong input !”
        fi
```

Logical Operators

!	negate (NOT) a logical expression
-a or &&	logically AND two logical expressions
-o or ||	logically OR two logical expressions

Note: &&, || must be enclosed within [[ ]]

Example:

```
#!/bin/bash
    echo -n "Enter a number 1 < x < 10:"
    read num
    if [ ["$num" -gt 1 && "$num" -lt 10 ]];
    then
            echo "$num*$num=$(($num*$num))"
    else
            echo "Wrong insertion !"
    fi
```

File Operators

-d check if path given is a directory
-f check if path given is a file
-e check if file name exists
-r check if read permission is set for file or directory
-s check if a file has a length greater than 0
-w check if write permission is set for a file or directory
-x check if execute permission is set for a file or directory

Examples:

[-d fname]	(true if fname is a directory, otherwise false)
[-f fname]	(true if fname is a file, otherwise false)
[-e fname]	(true if fname exists, otherwise false)
[-s fname]	(true if fname length is greater then 0, else false)
[-r fname]	(true if fname has the read permission, else false)
[-w fname]	(true if fname has the write permission, else false)
[-x fname]	(true if fname has the execute permission, else false)

Example

Q. Copy the file /etc/fstab to the current directory if the file exists or else print error message.

Example

Q. Copy the file /etc/fstab to the current directory if the file exists or else print error message.

A.
```
#!/bin/bash
 if [ -f /etc/fstab ];
then
    cp /etc/fstab .
    echo "Done."
else
    echo "This file does not exist."
    exit 1
fi
```

Case Statement

- Used to execute statements based on specific values. Often used in place of an if statement if there are a large number of conditions.
  - Value used can be an expression
  - Each set of statements must be ended by a pair of semicolons;
  - May also contain: **"*", "?", […], [:class:]**
  - **M**ultiple patterns can be listed via "|"
  - **"*)" is used to accept any value not matched with list of values**

```
case $var in
    val1)
            statements;;
    val2)
            statements;;
    *)
            statements;;
    esac
```

Example 1: The case statement

```
$ cat case.sh
    #!/bin/bash
    echo -n "Enter a number 0 < x < 10: "
    read x
    case $x in
            1) echo "Value of x is 1.";;
            2) echo "Value of x is 2.";;
            3) echo "Value of x is 3.";;
            4) echo "Value of x is 4.";;
            5) echo "Value of x is 5.";;
            6) echo "Value of x is 6.";;
            7) echo "Value of x is 7.";;
            8) echo "Value of x is 8.";;
            9) echo "Value of x is 9.";;
            0 | 10) echo "wrong number.";;
            *) echo "Unrecognized value.";;
    esac
```

Example 2: The case Statement

```
#!/bin/bash
echo "Enter Y to see all files including hidden files"
echo "Enter N to see all non-hidden files"
echo "Enter q to quit"

read -p "Enter your choice: " reply

case $reply in
 Y|YES) echo "Displaying all (really…) files"
        ls -a ;;
 N|NO)  echo "Display all non-hidden files..."
        ls ;;
 Q)     exit 0 ;;

 *) echo "Invalid choice!"; exit 1 ;;
esac
```

Example 3: The case Statement

```
#!/bin/bash
ChildRate=3
AdultRate=10
SeniorRate=7
read -p "Enter your age: " age
case $age in
  [1-9]|[1][0-2])   # child, if age 12 and younger
     echo "your rate is" '$'"$ChildRate.00" ;;
     # adult, if age is between 13 and 59 inclusive
  [1][3-9]|[2-5][0-9])
     echo "your rate is" '$'"$AdultRate.00" ;;
  [6-9][0-9])       # senior, if age is 60+
     echo "your rate is" '$'"$SeniorRate.00" ;;
esac
```

The while Loop

The while structure is a looping structure. Used to execute a set of commands while a specified condition is true. The loop terminates as soon as the condition becomes false. If condition never becomes false, loop will never exit.

```
while expression
do
    statements
done
```

$ cat while.sh

```
#!/bin/bash
COUNTER=0
while [ $COUNTER -lt 10 ]
do
   echo The counter is $COUNTER
   let COUNTER=$COUNTER+1
done
```

Until loop

The until structure is very similar to the while structure. The until structure loops until the condition is true. So basically it is "until this condition is true, do this".

```
until [expression]
do
    statements
done
```

Example: counter.sh

```
#!/bin/bash
COUNTER=20
until [ $COUNTER -lt 10 ]
do
  echo $COUNTER
  let COUNTER-=1
done
```

For loop

- The for structure is used when you are looping through a range of variables.

```
for var in list
    do
        statements
    done
```

- Statements are executed with var set to each value in the list.

- Example

```
#!/bin/bash
let sum=0
for num in 1 2 3 4 5
do
        let "sum = $sum + $num"
done
echo $sum
```

For loop

```
#!/bin/bash
for x in paper pencil pen
do
    echo "The value of variable x is: $x"
    sleep 1
done
```

If the list part is left off, var is set to each parameter passed to the script ($1, $2, $3,…)

```
$ cat for1.sh
#!/bin/bash
for x
do
    echo "The value of variable x is: $x"
    sleep 1
done

$ for1.sh arg1 arg2
The value of variable x is: arg1
The value of variable x is: arg2
```

C-like for loop

- An alternative form of the for structure is

```
for (( EXPR1 ; EXPR2 ; EXPR3 ))
do
        statements
done
```

- First, the arithmetic expression EXPR1 is evaluated. EXPR2 is then evaluated repeatedly until it evaluates to 0. Each time EXPR2 is evaluates to a non-zero value, statements are executed and EXPR3 is evaluated.

```
$ cat for2.sh
#!/bin/bash
echo –n "Enter a number: "; read x
let sum=0
for (( i=1 ; $i<$x ; i=$i+1 )) ; do
let "sum = $sum + $i"
done
echo "the sum of the first $x numbers is: $sum"
```

Using arrays with loops

- In the bash shell, we may use arrays. The simplest way to create one is using one of the two subscripts:

```
pet[0]=dog
pet[1]=cat
pet[2]=fish

pet=(dog cat fish)
```

- To extract a value, type ${arrayname[i]}

```
$ echo ${pet[0]}
dog
```

- To extract all the elements, use an asterisk as:

```
echo ${arrayname[*]}
```

- We can combine arrays with loops using a for loop:

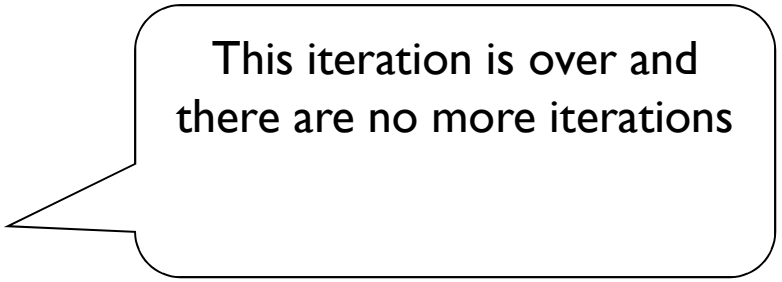
```
for x in ${arrayname[*]}
do
        ...
done
```

break and continue

- Interrupt for, while or until loop
- The break statement
  - transfer control to the statement AFTER the done statement
  - terminate execution of the loop
- The continue statement
  - transfer control to the statement TO the done statement
  - skip the test statements for the current iteration
  - continues execution of the loop

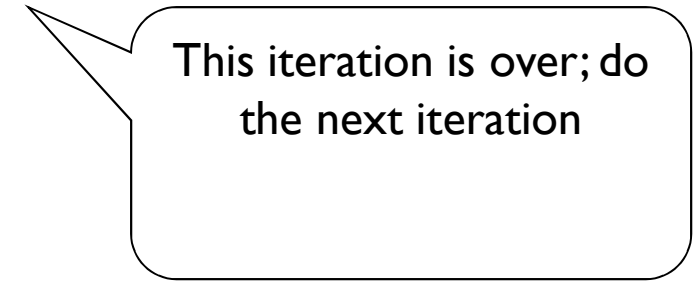
The break command

```
while [ condition ]
do
    cmd-1
    break
    cmd-n
done
echo "done"
```



The continue command

```
while [ condition ]
do
    cmd-1
    continue
    cmd-n
done
echo "done"
```



Example:

```
for index in 1 2 3 4 5 6 7 8 9 10
do
        if [ $index –le 3 ]; then
            echo "continue"
            continue
        fi
        echo $index
        if  [ $index –ge 8 ]; then
            echo "break"
            break
        fi
done
```

What is an Operating System?

- A system software .
- An operating system is a *resource allocator*.
- An Operating System controls (manages) hardware and software.
  - provides support for peripherals such as keyboard, mouse, screen, disk drives, …
  - software applications use the OS to communicate with peripherals.
  - The OS typically manages (starts, stops, pauses, etc) applications.
- -

UNIX/LINUX

What is UNIX?

- The **UNIX** is a multiuser, multitasking operating system.
- operating system was originally developed at Bell Laboratories, in the 1970's, for the Digital Equipment PDP computers.

What is Linux?

- **Linux** is a freely distributed implementation of a UNIX-like kernel.

Overview of the LINUX

- Linux is multitasking, multiuser operating system.
- Created in the 1990s by Linus Torvalds
- Linux is inspired by the UNIX operating system which first appeared in 1969-70, in AT & T Bell Labs.
- conventions behind UNIX also exist in Linux and are central to understanding the basics of the system.

Versions

Examples of Linux Operating Systems, called “distributions”:

- Ubuntu
- Debian
- Fedora
- Redhat
- CentOS
- SuSE

Features of Linux

- Multi-tasking operating system
- Multiuser operating system
- Linux is free
- Linux is portable to any hardware platform
- Linux is secure and versatile:
- Linux is scalable:

Responsibilities of LINUX

The Linux Operating System is responsible for the following functions

- Process management using processes and threads
- Inter-process communication
- Memory management
- Device management using device drivers
- File systems

The Linux subsystem

- **User Applications**

  The set of applications in use on a particular Linux system will be different depending on what the computer system is used for, but typical examples include a word-processing application and a web-browser

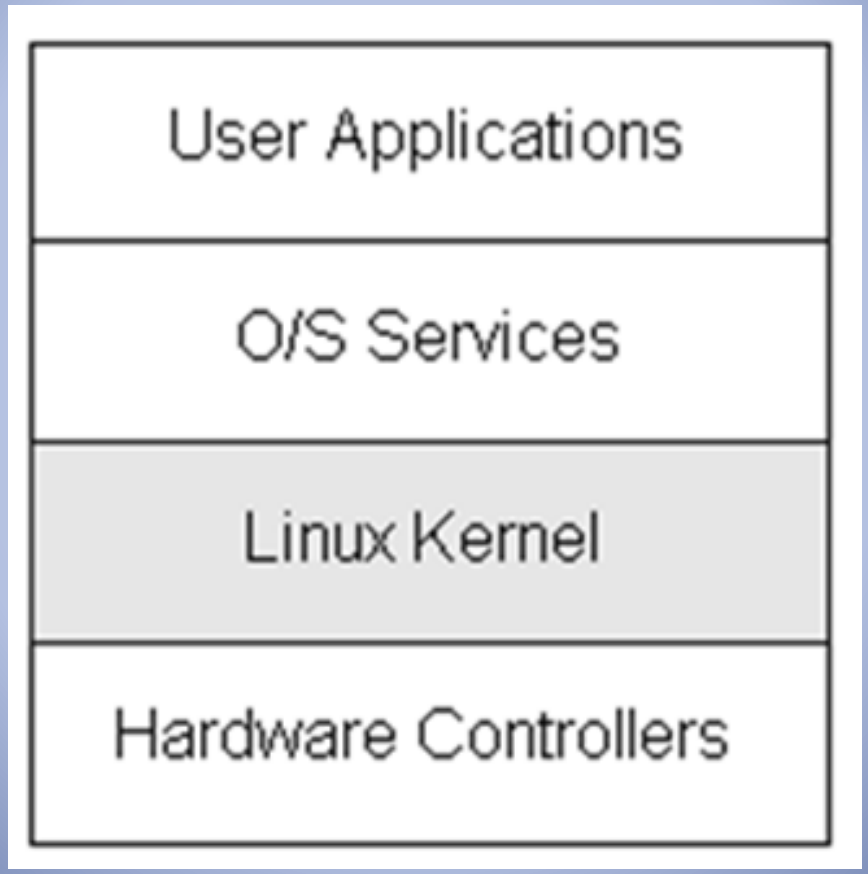
- **O/S Services**

  These are services that are typically considered part of the operating system (a windowing system, command shell, etc.); also, the programming interface to the kernel (compiler tool and library) is included in this subsystem

- **Linux Kernel**

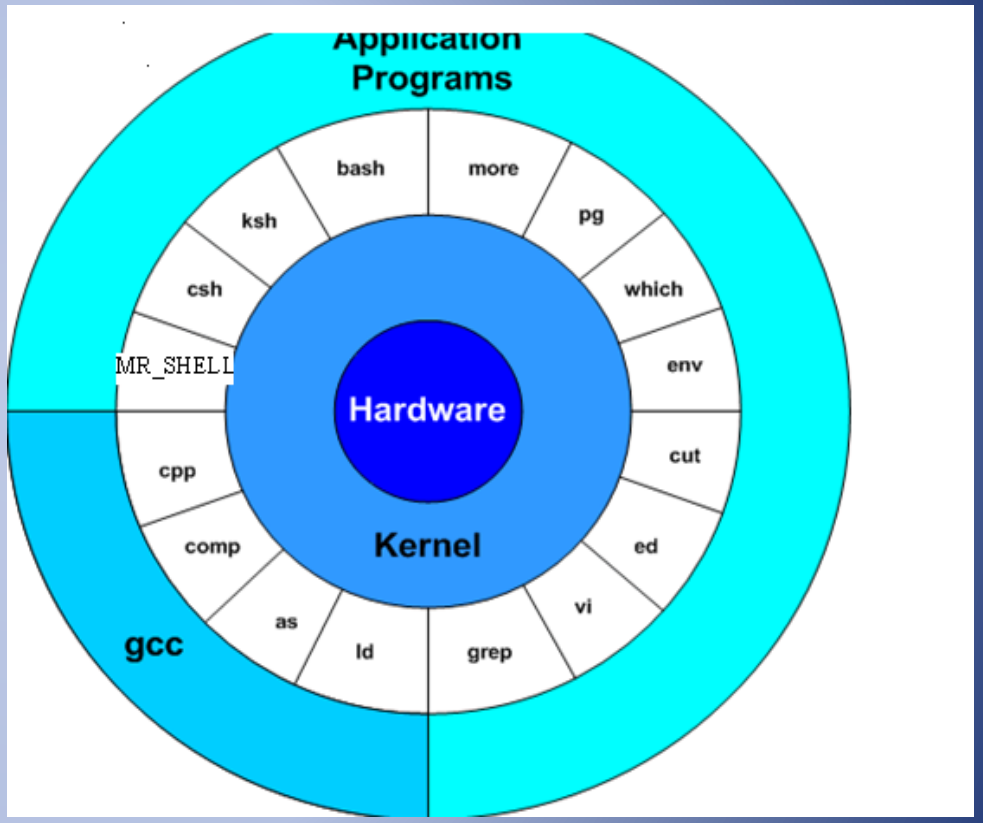
  Kernel is the heart of the operating system. The kernel of Linux is the hub of the operating system. It is loaded into memory when the system is booted and communicates

The LINUX Subsystem

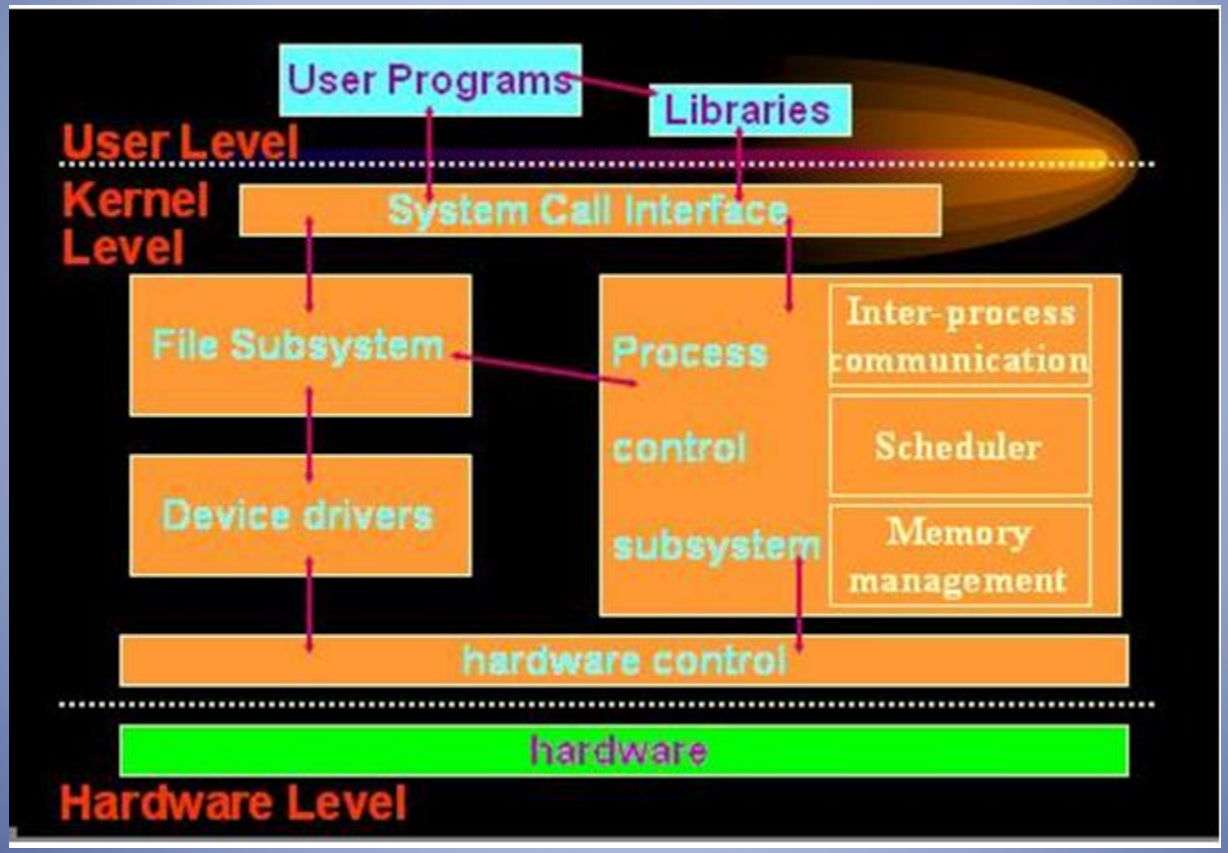


Architecture of Linux

- Kernel schedules tasks.
- Manages data/file access and storage.
- Enforces security mechanisms.
- Performs all hardware access.
- memory and processor management
- sharing the processor among multiple programs



Block Diagram of Kernel



The shell

- The shell acts as an interface between the user and the kernel.
- The shell is a command line interpreter (CLI).
- It interprets the commands the user types in and arranges for them to be carried out.
- The main functions of the shell are:
  - Presents each user with a prompt.
  - Interprets commands types by a user.
  - Executes user commands.
  - Supports a custom environment for each user.
  - It enables users to run application programs

Shell Offerings

- The most common ones are sh, csh, tcsh, ksh, bash, and zsh.
- Sh is the bourne shell. Which is the defaul tshell
- Ksh is very similar in syntax and features as bash however ksh is not free while bash is free. Bash is licensed under the Free Software Foundation.
- csh and tcsh are both based on the c language and makes writing shell scripts easier if you know the language. Both have similar features with tcsh having a few more.
- Zsh is the Z shell. It most closely resembles ksh

Features of LINUX SHELL

- Command line interpretation
- Establishes user environment
- File History generation:
- Input/output redirection
- Pipeline connections
- Programming language:
- Job control:

LINUX VS WINDOWS

- Financial Differences
- Technical Differences
- End-User Differences

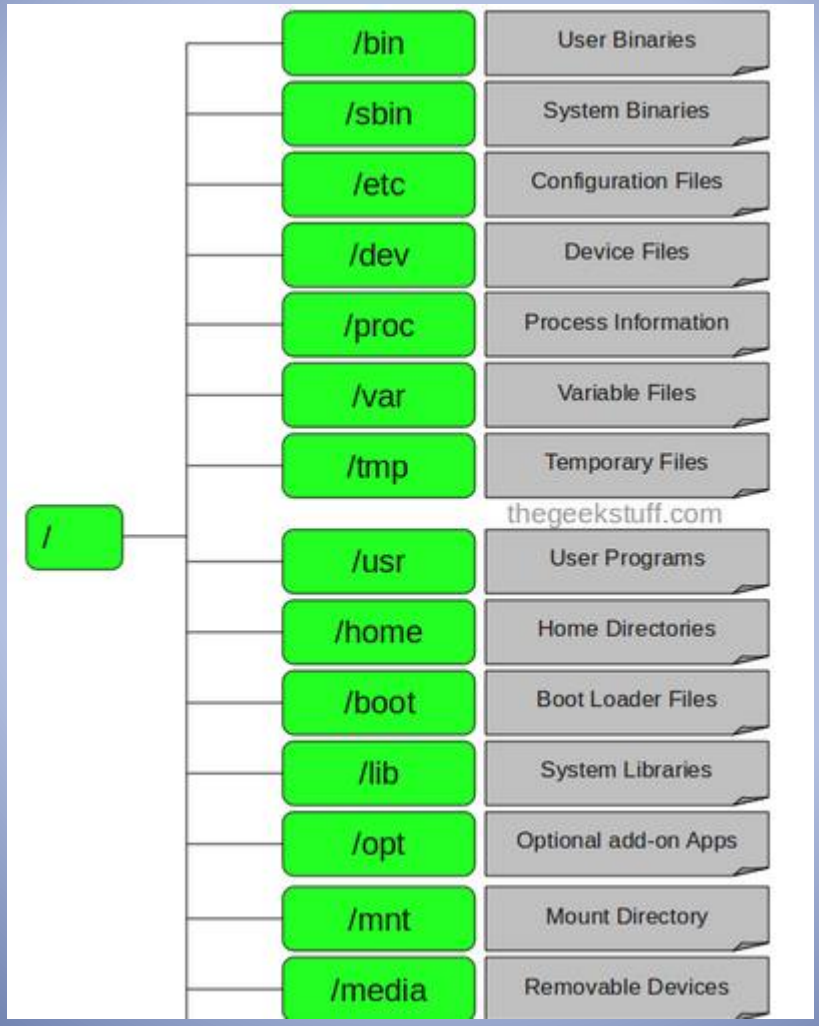
The File system

- A file system is a logical collection of files on a partition or disk. A partition is a container for information and can span an entire hard drive if desired.
- Your hard drive can have various partitions which usually contains only one file system, such as one file system housing the / file system or another containing the /home file system.
- One file system per partition allows for the logical maintenance and management of differing file systems.
- Everything in Unix is considered to be a file, including physical devices such as DVD-ROMs, USB devices, floppy drives, and so forth.

Unix Directory Structure

- Unix uses a hierarchical file system structure, much like an upside-down tree, with root (/) at the base of the file system and all other directories spreading from there.
- A UNIX filesystem is a collection of files and directories that has the following properties:
- It has a root directory (/) that contains other files and directories.
- Each file or directory is uniquely identified by its name, the directory in which it resides, and a unique identifier, typically called an inode.
- By convention, the root directory has an inode number of 2 and the lost+found directory has an inode number of 3. Inode numbers 0 and 1 are not used. File inode numbers can be seen by specifying the -i option to ls command.
- It is self contained. There are no dependencies between one filesystem and any other.
- The directories have specific purposes and generally hold the same types of information for easily locating files. Following are the directories that exist on the major versions of Unix:

The LINUX File System



The Root Directory

- **< / > Root**
- The root directory. The starting point of your directory structure. This is where the Linux system begins. Every other file and directory on your system is under the root directory. Usually the root directory contains only subdirectories, so it's a bad idea to store single files directly under root.
- Every single file and directory starts from the root directory.
- Only root user has write privilege under this directory.
- Don't confuse the *root directory* with the root user account, root password (which obviously is the root user's password) or root user's home directory.

BIN Directory

/bin – User Binaries

- Contains binary executables.
- Common linux commands you need to use in single-user modes are located under this directory.
- Commands used by all the users of the system are located here.
- For example: ps, ls, ping, grep, cp.

/sbin – System Binaries

- Just like /bin, /sbin also contains binary executables.
- But, the linux commands located under this directory are used typically by system aministrator, for system maintenance purpose.
- For example: iptables, reboot, fdisk, ifconfig, swapon

/etc

- **< /etc >**
- The configuration files for the Linux system. Most of these files are text files and can be edited by hand. Some interesting stuff in this directory:

  /etc/inittab

  A text file that describes what processes are started at system bootup and during normal operation. For example, here you can determine if you want the X Window System to start automatically at bootup, and configure what happens when a user presses Ctrl+Alt+Del

  /etc/fstab

  This file contains descriptive information about the various file systems and their mount points, like floppies, cdroms, and so on.

  /etc/passwd

  A file that contains various pieces of information for each user account. This is where the users are defined.

/user

- **< /usr >**
- This directory contains user applications and a variety of other things for them, like their source codes, and pictures, docs, or config files they use. /usr is the largest directory on a Linux system, and some people like to have it on a separate partition. Some interesting stuff in /usr:
- /usr/doc
  Documentation for the user apps, in many file formats.
- /usr/share
  Config files and graphics for many user apps.
- /usr/src
  Source code files for the system's software, including the Linux kernel.
- /usr/include
  Header files for the C compiler. The header files define structures and constants that are needed for building most standard programs. A subdirectory under /usr/include contains headers for the C++ compiler.
- **< /usr/local >**
- This is where you install apps and other files for use on the local machine. If your machine is a part of a network, the /usr directory may physically be on another machine and can be shared by many networked Linux workstations. On this kind of a network, the /usr/local directory contains only stuff that is not supposed to be used on many machines and is intended for use at the local machine only.

/dev & /proc

/dev – Device Files

- Contains device files.
- These include terminal devices, usb, or any device attached to the system.
- For example: /dev/tty1, /dev/usbmon0

/proc – Process Information

- Contains information about system process.
- This is a pseudo filesystem contains information about running process. For example: /proc/{pid} directory contains information about the process with that particular pid.
- This is a virtual filesystem with text information about system resources. For example: /proc/uptime

/var & /tmp

/var – Variable Files

- var stands for variable files.
- Content of the files that are expected to grow can be found under this directory.
- This includes — system log files (/var/log); packages and database files (/var/lib); emails (/var/mail); print queues (/var/spool); lock files (/var/lock); temp files needed across reboots (/var/tmp);

/tmp – Temporary Files

- Directory that contains temporary files created by system and users.
- Files under this directory are deleted when system is rebooted

- /lib – System Libraries
- Contains library files that supports the binaries located under /bin and /sbin
- Library filenames are either ld* or lib*.so.*
- For example: ld-2.11.1.so, libncurses.so.5.7

/opt – Optional add-on Applications

- opt stands for optional.
- Contains add-on applications from individual vendors.
- add-on applications should be installed under either /opt/ or /opt/ sub-directory.

/mnt – Mount Directory

- Temporary mount directory where sysadmins can mount filesystems.

/media – Removable Media Devices

- Temporary mount directory for removable devices.
- For examples, /media/cdrom for CD-ROM; /media/floppy for floppy drives; /media/cdrecorder for CD writer

/srv – Service Data

- srv stands for service.
- Contains server specific services related data.
- For example, /srv/cvs contains CVS related data.

- LINUX COMMANDS

- Type command
- To check a command is internal or external
- Internal commands are the built in commands of the shell. Which means that when you execute an internal command, no process will be launched to execute the command. Therefore the speed of executing an internal command will be very high. Example – cd,pwd,echotc.

  External commands are those command which are stored as a separate binaries. Shell starts separate sub-process to execute them. Most external commands are stored in the form of binaries in /bin directory. To execute external command shell check $PATH variable . If command present in the location mentioned in $PATH variable shell will execute it , otherwise it will give error. example – ls,mv,cat etc.

- Uname
- The Uname command lists the name of the current system you are logged into
- Uname –s gives the os name
- Uname –r gives the version of os
- Uname –x gives the detail

man

Man ual command.

man man This is help command, and will explains you about online manual pages you can also use man in conjunction with any command to learn more about that command for example.

- man ls will explain about the ls command and how you can use it.
- man -k pattern command will search for the pattern in given command.

pwd

- **Pwd** command.
  pwd command will print your home directory on screen, pwd means present working directory.

calendar

Calendar command

calendar command reads your calendar file and displays only lines with current day.
For example in your calendar file if you have this

- 12/20 Test new software.
- 1/15 Test newly developed 3270 product.
- 1/20 Install memory on HP 9000 machine.
- On dec 20th the first line will be displayed. you can use this command with your crontab file or in your login files.

clear

Clear command

- clear command clears the screen and puts cursor at beginning of first line.

Tty

- **Tty** command
  Tty command will display your terminal.
  Syntax is
  tty *options*
- Options
- -l will print the synchronous line number.
- -s will return only the codes: 0 (a terminal), 1 (not a terminal), 2 (invalid options) (good for scripts)

File management commands

- In UNIX there are three basic types of files:
- **Ordinary Files:** An ordinary file is a file on the system that contains data, text, or program instructions. In this tutorial, you look at working with ordinary files.
- **Directories:** Directories store both special and ordinary files. For users familiar with Windows or Mac OS, UNIX directories are equivalent to folders.
- **Special Files:** Some special files provide access to hardware such as hard drives, CD-ROM drives, modems, and Ethernet adapters. Other special files are similar to aliases or shortcuts and enable you to access a single file using different names.

File Management

- cat
- cd
- Cp
- file
- Head
- Tail
- ln
- Ls
- mkdir
- More
- Mv
- pwd
- Rcp
- Rm
- rmdir
- wc.

ls

- **Ls** command
  ls command is most widely used command and it displays the contents of directory.
- **options**
- ls will list all the files in your home directory, this command has many options.
- ls -l will list all the file names, permissions, group, etc in long format.
- ls -a will list all the files including hidden files that start with . .
- ls -lt will list all files names based on the time of creation, newer files bring first.
- ls -Fxwill list files and directory names will be followed by slash.
- ls -Rwill lists all the files and files in the all the directories, recursively.
- ls -R | more will list all the files and files in all the directories, one page at a time.

Mkdir/cd

- **Mkdir** command.

  mkdir cdac

  This will create new directory, i.e. here cdac directory is created.
- **Cd** command.
  cd cdac
- This will change directory from current directory to cdac directory.
  Use pwd to check your current directory and ls to see if cdac directory is there or not.

cat

- Cat command
  cat cal.txt cat command displays the contents of a file here **cal.txt** on screen (or standard out).

Creating a file

- CAT command is used to create a file
- Example

$ Cat > file name

Enter your text

Press ctrl+d, then the contents will be saved to that file

- To see the contents of that file

$ cat filename

- To append some text to that existing file

$ cat >> file name

Enter text you want to enter

Press ctrl+d ,then the text will be appended to that file

Concatenate two files

- You can concatenate two file in to another file

$ Cat file1 file2 > file 3

- Here the contents of file 1 and file2 will be concatented and the contents will be saved in file3
- You can append the contents of one file or two file into another file

$ cat file1>> file2

$cat file1 file2 >> file3

Head/tail

- **Head** command.
  head filename by default will display the first 10 lines of a file.
  If you want first 50 lines you can use head -50 filename or for 37 lines head -37 filename and so forth.
- **Tail** command.
  tail filename by default will display the last 10 lines of a file.
  If you want last 50 lines then you can use tail -50 filename.

more

- **More** command. more command will display a page at a time and then wait for input which is spacebar. For example if you have a file which is 500 lines and you want to read it all. So you can use
- more filename

WC

- **Wc** command
  wc command counts the characters, words or lines in a file depending upon the option.
- **Options**
- wc -l filename will print total number of lines in a file.
- wc -w filename will print total number of words in a file.
- wc -c filename will print total number of characters in a file.

cp

- **Cp** command.
- cp command copies a file. If I want to copy a file named oldfile in a current directory to a file named newfile in a current directory.
- cp oldfile newfile
- If I want to copy oldfile to other directory for example /tmp then
  cp oldfile /tmp/newfile.
- Useful options available with cp are **-p and -r** . -p options preserves the modification time and permissions, -r recursively copy a directory and its files, duplicating the tree structure.

rcp

- **Rcp** command.
  rcp command will copy files between two unix systems and works just like cp command (-p and -i options too).
  For example you are on a unix system that is called *Cheetah* and want to copy a file which is in current directory to a system that is called *lion* in */usr/john/* directory then you can use rcp command
  rcp filename lion:/usr/john
  You will also need permissions between the two machines. For more infor type man rcp at command line.

mv

- **Mv** command.
  mv command is used to move a file from one directory to another directory or to rename a file.
- **Some examples:**
- mv oldfile newfile will rename oldfile to newfile.
- mv -i oldfile newfile with confirmation prompt.
- mv -f oldfile newfile will force the rename even if target file exists.
- mv * destination dir will move all the files in current directory to destination directory.

rm

- **Rm** command.
  To delete files use rm command.
- **Options:**
- rm oldfile will delete file named oldfile.
- rm -f option will remove write-protected files without prompting.
- rm -r option will delete the entire directory as well as all the subdirectories, very dangerous command.

rmdir

- **Rmdir** command.
  rmdir command will remove directory or directories
- **Options:**
- rm -r directory_name will remove all files even if directory is not empty.
- rmdir sandeep is how you use it to remove sandeep directory.
- rmdir -p will remove directories and any parent directories that are empty.
- rmdir -s will suppress standard error messages caused by -p.

Creating a file

- CAT command is used to create a file
- Example

$ Cat > file name

Enter your text

Press ctrl+d, then the contents will be saved to that file

- To see the contents of that file

$ cat filename

- To append some text to that existing file

$ cat >> file name

Enter text you want to enter

Press ctrl+d ,then the text will be appended to that file

Concatenate two files

- You can concatenate two file in to another file

$ Cat file1 file2 > file 3

- Here the contents of file 1 and file2 will be concatented and the contents will be saved in file3
- You can append the contents of one file or two file into another file

$ cat file1>> file2

$cat file1 file2 >> file3

Question 1

- The commonly used UNIX commands like date, ls, cat, etc. are stored in

A. /dev directory
B. /bin and /usr/bin directories
C. /tmp directory
D. /unix directory

Answer 1

- /bin and /usr/bin directories

Q2

- The seventh field of /etc/password is
- password
- login
- shell
- home

A2

- shell

Q3

- Which command is used to display the device name of the terminal you are using?
-
- A. who
- B. ls
- C. tty
- D. stty
- E. None of the above

A3

- . tty

Q4

- Which command is used to set terminal IO characteristic?
- tty
- ctty
- ptty
- stty

A4

- stty

Q5

- Which command is used to change directory to the file beginning with a 'p'?
- A. cd p
- B. cd p?
- C. cd p*
- D. cd [p]
- E. None of the above

A5

- cd p*

Q6

- . Write the command to display the current date in the form dd/mm/yyyy.
- date +%d/%m/%Y
- date +”%d/%m/%Y”
- date +/%d/%m/20%y
- date +”/%d/%m/20%y”

A6

- date +%d/%m/%Y

Q7

- Which command is used to print a file?
- A. print
- B. prn
- C. pg
- D. lp
- E. None of the above

A7

- lp

Q8

- Which of the following commands is used to get directory one level up?
-
- A. cd
- B. cd ..
- C. cd/
- D. chdir
- E. None of the above

A8

- cd ..

Q9

Which command is used to delete all files in the current directory as well as all files and sub-directories in its subdirectories?

-
- A. rm *
- B. rm -r *
- C. rm all
- D. rm *.*
- E. None of the above

A9

- rm -r *

Q10

- Which of the following commands is used to change the working directory?
- A. cd
- B. changedir
- C. chdir
- D. cdir
- E. None of the above

A10

- cd

Q11

- Which of the following commands is used to view your file 24 lines at a time?
-
- A. pg
- B. cat
- C. lp
- D. /p
- E. None of the above

A11

- A. pg

Q12

-) Which command is used to display the end of the file?
-
- A. head - r
- B. tail
- C. eof
- D. bof
- E. None of the above
-

A12

- tail

Q13

- Which command is used to locate all the .profile files in the system?
- A. ls profile
- B. find /-name profile -print
- **C.** cd /.profile
- D. l -u .profile
- E. None of the above

A13

- B. find /-name profile -print

Q14

- Which command is used to move all files to the bin sub-directory of the parent directory?
- A. mv *.* /bin/
- B. mv * /bin/*
- C. mv * ../bin
- D. mv * ../bin *.*
- E. None of the above

A14

- . mv * ../bin

Q15

- The command used to remove the directory is;
-
- A. rmdir
- B. rd
- C. remove
- D. rdir
- E. None of the above

A15

- rmdir

Q16

- Which of the following commands is used to remove files?
-
- A. erase
- B. delete
- C. rm
- D. dm
- E. . None of the above

A16

- rm

Ls command

- The ls command is used to list all the files and dirctories in the cuurrent directory.
- The ls –l is to list 7 attributes of a file.
- Example:

Total 108 (it indicates total 108 blocks are occupied by these files on disk,each block consisting of 512 bytes/1024 in linux)

-rw-r- -r- - 1 siddhi root 1066……mar 11 11:05 a.txt

1 .Permission: Here the first column shows the type of permission associated with each file like Read ,write,Execute.

the first character you may found is – which means this is a ordinary file..you may see a d in case of dir and prog files

2. Link: The second column indicates the no. of links associated with the file. A link count greater than 1 means the file has more than one name ,though the physical copy is only one..

3.Owenership: When you create a file you are automatically the owner of that file. The third column shows that siddhi is the owner of that file.

4.Group ownership: When you open a user account , the system administrator also assigns the user to some groups .The 4^{th} column represents the group owner of the file.

5. File size: The 5th column displays the size of the file in bytes i.e, the amount of data it contains.
6. Last modification time: The 6, 7, 8th column represents the last modified month date and time.
7. FileName: The last column displays the filenames arranged in ASCII.

Listing directory attributes

- To see the attributes of the directory rather than its contents
- Ls -ld
- Ls –ld dir

drwxr-xr-x 2 siddhi root 512 mar 11 12:06

File ownership

- In the file attribute list ,the 3rd column represents the ownership i.e, you are the owner of the file.
- Group name is seen in the 4th col.
- Several users may belong to the same group.
- People working on a project are generally assigned a common group.
- The privileges of the group are set by the owner of the file not by the group member.
- When the system admin creates a user account, he assigns two parameters
- The user-id(UID)-both its name and numeric representation
- The group-id(GID)- both its name and numeric representation
- The file /etc.passwd maintains the UID(both name and no), the GID(only number). The file /etc/group contains the GID(both name and number)

id command

- To know your own UID and GID without viewing /etc/passwd and /etc/group use the id command

File Permission

- LINUX follows a three tired protection system to determine access rights of files.

1.Read

2. Write

3. Execute

Changing file permission(chmod command)

[root@localhost ~]# ls -l stu3.txt

-rw-r--r-- 1 root root 72 Mar 12 12:07 stu3.txt

[root@localhost ~]#

Whenever a file is created by default the file doesnot have execute permission.

To do that permission of the file need to be changed.

This is done through chmod command

chmod

- The chmod (change mod) is used to set the permission of one or more files for all categories(user/group).
- It can be run only by the user and the superuser.
- The command can be used in two ways

1.Relative permission: by specifying the changes to the current permissions

2. Absolute permission: by specifying the final permission

Relative permission

- In relative manner, chmod only changes the permission specified in the command line and leaves other permissions unchanged
- Syntax:

Chmod category operartion permission filenames(s)

- User category(user,group,others)
- The operation to be performed(Assign or remove a permission)
- The type of permission(read,write,execute)

Abbreviations used by chmod

Category	Operation	Permission
u(user)	+ (assign permission)	r(read)
g(group)	-(remove permission)	w(write)
o(others)	=(absolute permission)	x(execute)
a(all)		

Example

- [root@localhost ~]# ls -l stu3.txt

-rw-r--r-- 1 root root 72 Mar 12 12:07 stu3.txt

- [root@localhost ~]# chmod u+x stu3.txt
- [root@localhost ~]# ls -l stu3.txt

-rwxr--r-- 1 root root 72 Mar 12 12:07 stu3.txt

- [root@localhost ~]#
- [root@localhost ~]# chmod ugo+x stu2.txt
- [root@localhost ~]# ls -l stu2.txt
- -rwxr-xr-x 1 root root 36 Mar 12 12:21 stu2.txt
- [root@localhost ~]#

Absolute permission

- The syntax used in chmod is a string of three octal numbers.The octal digits have value 0-7
- Chmod uses three digit string as expression.
- Here we have three categories and three permissions
- Categories(user,group.other)
- Permission(read,write,execute)

Absolute permission

Binary	octal	permission	Significance
000	o	---	No Permission
001	1	--x	Executable only
010	2	-w-	Writable only
011	3	-wx	Writable and Executable
100	4	r--	readable
101	5	r-x	Readable and Executable
110	6	rw-	Readable and Writable
111	7	rwx	Readable and Writable andExecutable

Example

- Suppose you want to assign read and write permission to all three categories.
- Without octal number

Chmod a+rw filename

- With octal number

Chmod 666 filename

Here 6 indicates read and write(4+2)

Ls –l filename

Setting default File permission- umask

- When you create a file or a directory, it has a default set of permissions.
- These are determined by the value of umask variable defined in /etc/profile file. Umask displays the default mask permission.
- By default each file is assigned 666 permission (rw-rw-rw-)and a directory and an executable file has 777 permission (rwxrwxrwx)..
- The value assigned to umask is subtracted from the default value to generate new default permission.
- $ umask
- 022
- If the value is subtracted from the default it generates 644(666-022)(-rwr—r--)

Changing file ownership

- There are two commands meant to change the ownership of a file or directory
- Chown(change owner)
- Chgrp(change group)

chown

- **Chown** command.
  chown command to change ownership of a file or directory to one or more users.
  Syntax is
  chown *options newowner files*
- Options
- -R will recursively descend through the directory, including subdirectories and symbolic links.

example

- Ls –l note

-rwxr---rw 1 abc root

- Chown xyz note
- Ls –l note

-rwxr—rw xyz root

chgrp

- Changing group owner
- **Chgrp** command.
  chgrp command is used to change the group of a file or directory.
  You must own the file or be a superuser.
  chgrp [options] newgroup *files* is syntax of chgrp.
  Newgroup is either a group Id or a group name located in **/etc/group** .
- Options:
- -h will change the group on symbolic links.
- -R recursively descend through directory changing group of all files and subdirectories.

Touch command

- The *touch* command is the easiest way to create new, empty files. It is also used to change the *timestamps* (i.e., dates and times of the most recent access and modification) on existing files and directories.
- touch's syntax is
- touch [option] file_name(s)
- When used without any options, touch creates new files for any file names that are provided as *arguments* (i.e., input data) if files with such names do not already exist. Touch can create any number of files simultaneously.
- Thus, for example, the following command would create three new, empty files named *file1*, *file2* and *file3*:
- touch file1 file2 file3
- A nice feature of touch is that, in contrast to some commands such as *cp* (which is used to copy files and directories) and *mv* (which is used to move or rename files and directories), it does not automatically overwrite (i.e., erase the contents of) existing files with the same name. Rather, it merely changes the last access times for such files to the current time.
- Several of touch's options are specifically designed to allow the user to change the timestamps for files. For example, the *-a* option changes only the access time, while the *-m* option changes only the modification time. The use of both of these options together changes both the access and modification times to the current time, for example:
- touch -am file3

example

- [root@localhost ~]# touch a1.txt
- [root@localhost ~]# ls -l a1.txt
- -rw-r--r-- 1 root root 0 Mar 12 15:42 a1.txt
- [root@localhost ~]# touch -a a1.txt
- [root@localhost ~]# ls -l a1.txt
- -rw-r--r-- 1 root root 0 Mar 12 15:42 a1.txt
- [root@localhost ~]# cat >a1.txt

dsf

sdfs

- [root@localhost ~]# touch -am a1.txt
- [root@localhost ~]# ls -l a1.txt
- -rw-r--r-- 1 root root 9 Mar 12 15:43 a1.txt
- [root@localhost ~]#
- [root@localhost ~]#

Echo command

- Echo is an shell built in command.
- Example
- Echo hello
- Echo ‘hello’
- Echo “hello”
- A=5
- Echo &a

- Echo uses ‘\’character to emphasize a character,that is treat it special
- Echo ’enter your name: \c’ (puts the cursor immediately after the argument ,instead of a new line)
- Output-Enter your name: $_
- \t: produces a tab
- \n : produces newline
- \b: produces backspace

Echo for editing

- Echo “\033[1m hello”
- This will print hello in bold
- 4m-underlined
- 5m blinking

Comparison and Searching

- find ,diff,dircmp,cmp, grep,

Find

- The Linux **find command** is a very useful and handy command to search for files from the command line.
- It can be used to find files based on various search criteria like permissions, user ownership, modification date/time, size etc.
- The Find command repeatedly examines a directory tree to look for file matching some criteria,then takes appropriate action on the selected files.

- Syntax:
-
- Find path-list selection-criteria action

Examples of find

1. List all files in current and sub directories

$ find and find .

2. **List all files in root directory recursively**

$ find / (press ctrl+c to come back anytime)

3. **The following command will look for files in the test directory in the current directory. Lists out all files by default.**

$ find ./test

./test

-/test/abc.txt

./test/subdir

./test/subdir/how.php

./test/cool.php

4. The following command searches for files by their name.

$ find ./test -name "abc.txt“

./test/abc.txt

5. We can also use wildcards

$ find ./test -name "*.php"

./test/subdir/how.php

./test/cool.php

6. **Ignore the case.** It is often useful to ignore the case when searching for file names. To ignore the case, just use the "iname" option instead of the "name" option.

$ find ./test -iname "*.Php" ./test/subdir/how.php ./test/cool.php

-maxdepth

7. Limit depth of directory traversal

- The find command by default travels down the entire directory tree recursively, which is time and resource consuming.
- However the depth of directory traversal can be specified.
- For example we don't want to go more than 2 or 3 levels down in the sub directories. This is done using the maxdepth option.
- **$ find ./test -maxdepth 2 -name "*.php"**
- ./test/subdir/how.php
- ./test/cool.php
- **$ find ./test -maxdepth 1 -name *.php**
- ./test/cool.php
- The second example uses maxdepth of 1, which means it will not go lower than 1 level deep, either only in the current directory.

! (Not)

8. Invert match

It is also possible to search for files that do no match a given name or pattern. This is helpful when we know which files to exclude from the search.

$ find ./test -not -name "*.php"

./test ./test/abc.txt ./test/subdir

So in the above example we found all files that do not have the extension of php, either non-php files.

The find command also supports the exclamation mark in place of not.

find ./test ! -name "*.php"

multiple search criterias

9. **Combine multiple search criterias**

- It is possible to use multiple criterias when specifying name and inverting. For example
- **$ find ./test -name 'abc*' ! -name '*.php'**
- ./test/abc.txt
- ./test/abc
- The above find command looks for files that begin with abc in their names and do not have a php extension.
- This is an example of how powerful search expressions can be build with the find command.
- **OR operator**
- When using multiple name criterias, the find command would combine them with AND operator, which means that only those files which satisfy all criterias will be matched.
- However if we need to perform an OR based matching then the find command has the "o" switch.
- **$ find -name '*.php' -o -name '*.txt‘**
- ./abc.txt
- ./subdir/how.php
- ./abc.php
- ./cool.php
- The above command search for files ending in either the php extension or the txt extension.

only files or only directories

10.Search only files or only directories

- Sometimes we want to find only files or only directories with a given name. Find can do this easily as well.
- **$ find ./test -name abc***
- ./test/abc.txt
- ./test/abc
- **Only files**
- **$ find ./test -type f -name "abc*"**
- ./test/abc.txt
- **Only directories**
- **$ find ./test -type d -name "abc*"**
- ./test/abc
- Quite useful and handy!

Hidden files

11.Find hidden files

- Hidden files on linux begin with a period. So its easy to mention that in the name criteria and list all hidden files.
- **$ find ~ -type f -name ".*"**

readonly files

12. Find readonly files in etc directory

- Find all Read Only files.
- **$ find /etc -maxdepth 1 -perm /u=r**
- /etc /etc/thunderbird
- /etc/brltty
- /etc/dkms
- /etc/phpmyadmin

files belonging to particular user

13. Find files belonging to particular user root

- To find all or single file called tecmint.txt under /root directory of owner root.

$ find . -user bob

.

./abc.txt

./abc ./subdir

./subdir/how.php

./abc.php

We could also specify the name of the file or any name related criteria along with user criteria

$ find . -user bob -name '*.php'

Its very easy to see, how we can build up criteria after criteria to narrow down our search for matching files.

modification date and time

14. Find files modified N days back

To find all the files which are modified 50 days back.

```
# find / -mtime  50
```

15..Find files accessed in last N days

Find all files that were accessed in the last 50 days.

```
# find / -atime 50
```

16. Find files modified in a range of days

Find all files that were modified between 50 to 100 days ago.

```
# find / -mtime +50 –mtime -100
```

17. Find files changed in last N minutes.

Find files modified within the last 1 hour.

```
$ find /home/bob -cmin -60
```

18. Files modified in last hour

To find all the files which are modified in last 1 hour.

```
# find / -mmin -60
```

19. Find Accessed Files in Last 1 Hour

To find all the files which are accessed in last 1 hour.

```
# find / -amin -60
```

Search files and directories based on size

20. Find files of given size

To find all 50MB files, use.

$find / -size 50M

21. Find files in a size range

To find all the files which are greater than 50MB and less than 100MB.

$ find / -size +50M -size -100M

Diff Command

diff [options] file1 file2

The UNIX diff command compares the contents of two text files and outputs a list of differences.

diff command will compare the two files and print out the differences between.

It tells you which lines in one file have to be changed to make the two files identical.

Here I have two ascii text files. fileone and file two.

Contents of fileone are

This is first file

this is second line

this is third line

this is different as;lkdjf

this is not different

filetwo contains

This is first file

this is second line

this is third line

this is different xxxxxxxas;lkdjf

this is not different

diff fileone filetwo

A greater-than or less-than symbol appears at the beginning of each line. "<" means that the text appears in file1, and ">" indicates that it comes from file2.

This will give following output

4c4

< this is different as;lkdjf

> this is different xxxxxxxas;lkdjf

- **# diff file1 file2**
  1,5c1,5
  < this is line 1 The UNIX diff command is used to compare (find the differences) between two files.
  < this is line 2 This line demonstrates how the diff command handles white space
  < this is line 3 if (a == b)
  < this is line 4 THE DIFF COMMAND IS HELPFUL WHEN COMPARING SOURCE CODE FILES
  < this is line 5

  > this is line 1
  > this is line 2 This line demonstrates how the diff command handles white space
  > this is line 3 if(a==b)
  > this is line 4 The diff Command is Helpful When Comparing Source Code Files
  > this is line 5 The UNIX diff command is used to compare (find the differences) between two files. Before listing lines of text, this tool shows you how to eliminate all of the differences. It supplies Ed line editor commands, such as "1,5c1,5." This means that you could make the files match by modifying lines one through five. The letter "c" stands for "change." Diff's directions may also contain "a" for "append" or "d" for "delete."

This command has a variety of helpful options. If you use the "-b" setting, it skips over minor spacing differences. This UNIX diff example shows that "-b" instructs the system to ignore extra spaces and tabs:

- **# diff -b file1 file2**
  1c1
  < this is line 1 The UNIX diff command is used to compare (find the differences) between two files.

  > this is line 1
  3,5c3,5
  < this is line 3 if (a == b)
  < this is line 4 THE DIFF COMMAND IS HELPFUL WHEN COMPARING SOURCE CODE FILES
  < this is line 5

  > this is line 3 if(a==b)
  > this is line 4 The diff Command is Helpful When Comparing Source Code Files
  > this is line 5 The UNIX diff command is used to compare (find the differences) between two files

Example

- To disregard case differences, add the "-i" option. You can use it to check for mistakes after converting an uppercase document to mixed-case characters. It works the same way as it does with the UNIX grep command:

  # diff -i file1 file2
  1,3c1,3
  < this is line 1 The UNIX diff command is used to compare (find the differences) between two files.
  < this is line 2 This line demonstrates how the diff command handles white space
  < this is line 3 if (a == b)

  > this is line 1
  > this is line 2 This line demonstrates how the diff command handles white space
  > this is line 3 if(a==b)
  5c5
  < this is line 5

  > this is line 5 The UNIX diff command is used to compare (find the differences) between two files.

Cmp & dircmp command

Cmp command.
cmp command compares the two files. For exmaple I have two different files fileone and filetwo.
cmp fileone filetwo will give me

fileone filetwo differ: char 80, line 4

if I run cmp command on similar files nothing is returned.
-s command can be used to return exit codes. i.e. return 0 if files are identical, 1 if files are different, 2 if files are inaccessible.
This following command prints a message 'no changes' if files are same
cmp -s fileone file1 && echo 'no changes' .

no changes

Dircmp Command.
dircmp command compares two directories. If i have two directories in my home directory named dirone and dirtwo and each has 5-10 files in it. Then
dircmp dirone dirtwo will return this

Dec 9 16:06 1997 dirone only and dirtwo only Page 1

./cal.txt ./fourth.txt
./dohazaar.txt ./rmt.txt
./four.txt ./te.txt
./junk.txt ./third.txt

Grep command

grep *string filename(s)* --- looks for the string in the files.

This can be useful a lot of purposes, e.g. finding the right file among many, figuring out which is the right version of something, and even doing serious corpus work.

. **grep <str><files>**

Grep cont..

Options:

-b option will precede each line with its block number.

-c option will only print the count of matched lines.

-i ignores uppercase and lowercase distinctions.

-l lists filenames but not matched lines.

other associated commands with grep are egrep and fgrep.

egrep typically runs faster.

for more information type man egrep or man fgrep in your system.

Text processing

Commands

cut,paste, sort, uniq,awk,sed,vi.

Cut command

cut command selects a list of columns or fields from one or more files.
Option -c is for columns and -f for fields. It is entered as
cut options [files]
for example if a file named testfile contains

this is firstline

this is secondline

this is thirdline

Examples:
cut -c1,4 testfile

will print this to standard output (screen)

ts

ts

ts

It is printing columns 1 and 4 of this file which contains t and s (part of this).

Options:

-c list cut the column positions identified in list.

-f list will cut the fields identified in list.

Paste command

paste command merge the lines of one or more files into vertical columns separated by a tab.
for example if a file named testfile contains

this is firstline

and a file named testfile2 contains

this is testfile2

then running this command
paste testfile testfile2 > outputfile
will put this into outputfile

this is firstline this is testfile2

it contains contents of both files in columns.
who | paste - - will list users in two columns.

Options:

-d'char' separate columns with char instead of a tab.

-s merge subsequent lines from one file.

Sort command

sort command sort the lines of a file or files, in alphabetical order. for example if you have a file named testfile with these contents

zzz

aaa

1234

yuer

wer

qww

wwe

Then running sort testfile will give us output of

1234

aaa

qww

wer

wwe

yuer

zzz

options

-b ignores leading spaces and tabs.

-c checks whether files are already sorted.

-d ignores punctuation.

-i ignores non-printing characters.

-n sorts in arithmetic order.

-o*file* put output in a file.

+m[-m] skips n fields before sorting, and sort upto field position m.

-r reverse the order of sort.

-u identical lines in input file apear only one time in output.

uniq command

uniq command removes duplicate adjacent lines from sorted file while sending one copy of each second file.

Examples

sort names | uniq -d will show which lines appear more than once in names file.

Options:

-c print each line once, counting instances of each.

-d print duplicate lines once, but no unique lines.

-u print only unique lines.

Communications

cu,ftp,login, rlogin,talk,telnet, vacation and write .

Cu command

cu command is used for communications over a modem or direct line with another Unix system.
Syntax is
cu *options destination*

Options

-b*n* process lines using n-bit characters (7 or 8).

-c*name* Search UUCP's device file and select local area network that matches name.

-d Prints diagnostics.

-e sends even parity data to remote system

-l*line* communicate on this device (line=/dev/tty001, etc)

-n prompts for a telephone number.

-s*n* set transmission rate to n(e.g 1200,2400,9600, BPS)

Destination

telno is the telephone number of the modem to connect to.

system is call the system known to uucp.

aadr is an address specific to LAN.

ftp command

ftp command is used to execute ftp protocol using which files are transferred over two systems.
Syntax is
ftp *options hostname*

options

-d enable debugging.

-g disable filename globbing.

-i turn off interactive prompts.

-v verbose on. show all responses from remote server.

ftp hostname by default will connect you to the system, you must have a login id to be able to transfer the files. Two types of files can be transferred, ASCII or Binary. bin at ftp> prompt will set the transfer to binary. Practice FTP by ftping to nic.funet.fi loggin in as **anomymous** with password being your e-mail address.

Login command

login command invokes a login session to a Unix system, which then authenticates the login to a system. System prompts you to enter userid and password.

rlogin

- rlogin command is used to log on to remote Unix systems, user must have permissions on both systems as well as same userid, or an id defined in .rhosts file. Syntax is
  rlogin *options host*

options

- -8 will allow 8 bit data to pass, instead of 7-bit data.
- -e *c* will let you use escape character c.
- -l *user* will let you to login as user to remote host, instead of same as local host.

Talk command

talk command is used to invoke talk program available on all unix system which lets two users exchange information back and forth in real time. Syntax is
talk userid@hostname

telnet

- Telnet command invokes a telnet protocol which lets you log on to different unix, vms or any machine connected over TCP/IP protocol, IPx protocol or otherwise. Syntax is
  telnet hostname

vacation

- vacation command is used when you are out of office. It returns a mail message to sender announcing that you are on vacation. to disable this feature, type **mail -F " "** .
  syntax is
  vacation *options*
- Options
- -d will append the date to the logfile.
- -F *user* will forward mail to user when unable to send mail to mailfile.
- -l *logfile* will record in the logfile the names of senders who received automatic reply.
- -m *mailfile* will save received messages in mailfile.

Write command

- **Write** command will initiate an interactive conversation with user. Syntax is
  write *user tty*

Storage Command

Zip ,unzip ,cpio,dump,pack, tar, mt.

Gzip & gunzip command

gzip *filename* --- compresses files, so that they take up much less space. Usually text files compress to about half their original size, but it depends very much on the size of the file and the nature of the contents. There are other tools for this purpose, too (e.g. **compress**), but gzip usually gives the highest compression rate. Gzip produces files with the ending '.gz' appended to the original filename.

gunzip *filename* --- uncompresses files compressed by gzip.

tar

tar command creates an archive of files into a single file.

Tar copies and restore files to a tape or any storage media. Synopsis of tar is

tar [options] [file]

Examples:

tar cvf /dev/rmt/0 /bin /usr/bin creates an archive of /bin and /usr/bin, and store on the tape in /dev/rmt0.

tar tvf /dev/rmt0 will list the tape's content in a /dev/rmt0 drive.

tar cvf - 'find . -print' > backup.tar will creates an archive of current directory and store it in file **backup.tar.**

printing

Lpr command

lpr *filename* --- print. Use the -P option to specify the printer name if you want to use a printer other than your default printer. For example, if you want to print double-sided, use 'lpr -Pvalkyr-d', or if you're at CSLI, you may want to use 'lpr -Pcord115-d'. See 'help printers' for more information about printers and their locations.

lpq

lpq --- check out the printer queue, e.g. to get the number needed for removal, or to see how many other files will be printed before yours will come out

Lprm command

lprm *jobnumber* --- remove something from the printer queue. You can find the job number by using lpq. Theoretically you also have to specify a printer name, but this isn't necessary as long as you use your default printer in the department.

Genscript

genscript --- converts plain text files into postscript for printing, and gives you some options for formatting. Consider making an alias like **alias ecop 'genscript -2 -r \!* | lpr -h -Pvalkyr'** to print two pages on one piece of paper.

About other people

W command

w --- tells you who's logged in, and what they're doing. Especially useful: the 'idle' part. This allows you to see whether they're actually sitting there typing away at their keyboards right at the moment.

Who command

who --- tells you who's logged on, and where they're coming from. Useful if you're looking for someone who's actually physically in the same building as you, or in some other particular location.

whoami

whoami --- returns your username. Sounds useless, but isn't. You may need to find out who it is who forgot to log out somewhere, and make sure *you* have logged out.

finger

finger *username* --- gives you lots of information about that user, e.g. when they last read their mail and whether they're logged in. Often people put other practical information, such as phone numbers and addresses, in a file called **.plan**. This information is also displayed by 'finger'.

last

last -1 ***username*** --- tells you when the user last logged on and off and from where. Without any options, **last** will give you a list of everyone's logins.

Additional commands

- **talk *username*** --- lets you have a (typed) conversation with another user
- **write *username*** --- lets you exchange one-line messages with another user
- **elm** --- lets you send e-mail messages to people around the world (and, of course, read them). It's not the only mailer you can use, but the one we recommend.

passwd

- **passwd** --- lets you change your password, which you should do regularly (at least once a year

Ps command

- **ps -u *yourusername*** --- lists your processes. Contains lots of information about them, including the process ID, which you need if you have to kill a process. Normally, when you have been kicked out of a dialin session or have otherwise managed to get yourself disconnected abruptly, this list will contain the processes you need to kill. Those may include the shell (tcsh or whatever you're using), and anything you were running, for example emacs or elm. Be careful not to kill your current shell - the one with the number closer to the one of the ps command you're currently running. But if it happens, don't panic. Just try again

kill

- **kill *PID*** --- kills (ends) the processes with the ID you gave. This works only for your own processes, of course. Get the ID by using **ps**. If the process doesn't 'die' properly, use the option -9. But attempt without that option first, because it doesn't give the process a chance to finish possibly important business before dying. You may need to kill processes for example if your modem connection was interrupted and you didn't get logged out properly, which sometimes happens.

du

- **du *filename*** --- shows the disk usage of the files and directories in *filename* (without argument the current directory is used).
- **du -s** gives only a total.

date

- **date** --- shows the current date and time.

Top command

- **top command** displays processor activity of your Linux box and also displays tasks managed by kernel in real-time. It'll show processor and memory are being used and other information like running processes.