# **Introduction to Operating Systems**

### **Computer System Components**:

* **Hardware** – CPU, memory, I/O devices.
* **Operating System (OS)** – Manages hardware and software.
* **Application Programs** – Compilers, databases, etc.
* **Users** – People, machines, or other computers.

### **Operating System Basics**

* **OS Definition**: A program that manages system resources and enforces fairness.
  + Provides process scheduling, memory management, file systems, device drivers, and networking.
* **Kernel**:
  + The core component of the OS.
  + Manages processes, memory, file systems, and device access.
  + **Kernel Mode:** Full access to hardware/system resources.
  + **User Mode:** Restricted access; cannot modify system resources directly.

### **OS Structures & Functions**

1. **System Components**:
   * **Process Management**: Creating, deleting, and scheduling processes.
   * **Memory Management**: Allocating and deallocating memory.
   * **File Management**: Creating, deleting, and manipulating files.
   * **I/O System Management**: Handling input/output devices.
   * **Secondary Storage Management**: Managing disks and storage.
   * **Networking**: Communication between computers.
   * **Protection System**: Controlling access to system resources.
   * **Command Interpreter**: Processes commands (shell in UNIX).
2. **System Calls**:
   * Interface between running programs and the OS.
     + Types: **Process control, file management, device management, communication, and information maintenance.**
3. **CPU Scheduling**:
   * **First-Come, First-Served (FCFS):** Processes execute in the order they arrive.
   * **Shortest Job Next (SJN):** Shortest process executes first.
   * **Round Robin (RR):** Each process gets a time slice before moving back to the queue.
   * **Priority Scheduling:** Higher-priority processes execute first.
4. **Virtual Machines**:
   * Allows multiple OS instances on a single machine.
     + **Pros**: OS compatibility, stability, good for testing.
     + **Cons**: Complex and may reduce performance.
5. **System Design Goals**:
   * **User Goals**: Convenience, ease of learning, reliability, and speed.
   * **System Goals**: Easy design, flexibility, efficiency, and error-free operation.

### **UNIX & Linux Basics**

### **UNIX vs. Linux:**

* **UNIX:** Proprietary multi-user OS.
* **Linux:** Open-source UNIX-like OS.
* **Popular Linux distributions:** Ubuntu, Debian, Fedora, CentOS.

### **User Types:**

* **Normal Users:** Restricted access.
* **Super User (root):** Full access.

### **Basic UNIX Commands**

### **User Commands:**

* whoami # Display current username
* uname -a # Show system information
* ls # List directory contents.
* man [command] # Get manual page for a command.
* **>** → Overwrite file (ls > file.txt)
* **>>** → Append to file (echo "New" >> file.txt)
* echo $VAR # Display the contents of a variable
* read VAR # Get standard input and save it to variable VAR.
* # # Designates all text after # on the same line to be comments
* #!/bin/sh # Alert the system that a shell script is being executed.
* ssh user@access # Connect to access as user.
* ssh access # Connect to access as your local username.
* ssh -p port user@access

### **File Management:**

* ls # List directory contents
* mkdir dir # Create directory
* rmdir dir # Remove empty directory
* rm -rf dir # Delete directory with all files
* -d $FILE # Returns true if FILE is a directory.
* -r $FILE # Returns true if FILE is readable.
* -w $FILE # Returns true if FILE is writable.
* -x $FILE # Returns true if FILE is executable.
* -e $FILE # Returns true if FILE exists, even if FILE is a directory.

### **Permissions:** Unix/Linux file permissions

* -rwx # read (4), write (2), execute permissions (1)
* Owner (7) | Group (4) | Others (4)
* chmod 777 file # Give all permissions
* 0 No permissions -rwx
* 1 Execute permission only x
* 2 Write permission only w
* 3 Write and execute permissions only: 2 + 1 = 3 wx
* 4 Read permission only r
* 5 Read and execute permissions only: 4 + 1 = 5 rx
* 6 Read and write permissions only: 4 + 2 = 6 rw
* 7 All permissions: 4 + 2 + 1 = 7
* chown user file # Change file owner
* chgrp group file # Change file group
* chmod g+r myfile # Group can read.
* chmod +x testfile # allow all users to execute the file
* chmod u-w testfile  # forbid the current user from writing or changing the file
* chmod u+wx,g-x,o=rx testfile # simultaneously add write & execute permissions to user, remove execute permission from group, and set the permissions of other users to only read and write.
* chmod 365 testfile  # user (u) gets to write and execute only; group (g), read and write only; others (o), read and execute only.

### **Processes:**

* ps -A # Show all processes
* ps # Show active processes.
* top # Real-time process monitoring
* kill <PID> # Kill process by ID
* killall <name> # Kill processes by name
* kill -9 <PID> # Force kill.

### Unix Management

* **Files** – Creation, deletion, and permissions.
* **Users** – Access control and permissions.
* **Processes** – Running, managing, and terminating processes.

### Files and File Systems

* Unix organizes files in a single directory tree.
* **Two main file types:**
  + Plain files (contain code/data).
  + Directories (contain files and subdirectories).
* **Each file has attributes:**
  + Name, type, owner, size, creation/modification date, group, access rights.

### Navigating Unix

### **Directory Navigation:**

* pwd # Show current directory
* cd dir # Change directory
* cd .. # Move up one level
* cd ~ # Move to home directory

### **Paths:**

* **Absolute Path:** /home/user/dir
* **Relative Path:** cd ../dir

### **File Viewing:**

* cat file # Display file contents
* more file # View file page by page
* less file # Scroll through file

### **Copy/ Move file:**

* $ cp file1 file2 – Copies file1 to file2.
* $ mv file1 file2 – Moves/renames file.
* $ rm file1 – Deletes a file.

### File & Directory Management

* $ ls – Lists files.
* $ ls -a – Includes hidden files.
* $ ls -l – Shows details.
* $ ls -t – Sorts by modification time.
* $ ls -r – Reverses order.

### Disk & Memory Management

### Disk Usage:

* df -h # Show available disk space
* du -sh dir # Show directory size

### Memory Usage:

* $ free -h – Shows memory stats.
* $ cat /proc/meminfo – Displays detailed memory info.

## **System Information Commands**

* uname -a # Displays system details.
* uptime # Shows system uptime.
* date # Shows current date/time.
* cal # Shows current date/month.
* find / -name <file> # Finds a file.
* grep "text" file # Searches for "text" inside file.

### **Shell Programming**

* **Shell**: Interface between user and Linux system.
* **Why use shell scripting?**
  + Really, very fast to write
  + Use once, throw away
  + Powerful library functions
  + Example:
    - show the file list in a directory
    - find a file with a name ABC

### **Variables in Shell**

* **Local Variables**
  + Define: varname=value
  + Retrieve: $varname
  + Example:

$ greetings="Hello"

$ echo $greetings

Hello

* Example:

bash-3.2$ greetings=Hello

bash-3.2$ echo greetings

greetings

bash-3.2$ echo $greetings

Hello

bash-3.2$ echo "greetings"

greetings

bash-3.2$ echo "$greetings"

Hello

bash-3.2$ echo "\$greetings"

$greetings

bash-3.2$ echo "\$$greetings"

$Hello

bash-3.2$ echo \"$greetings\"

"Hello"

bash-3.2$ echo \'$greetings\'

' Hello'

bash-3.2$

* **Environment Variables**
  + $HOME – User home directory.
  + $PATH – Command search path.
  + $SHELL – Current shell.
    - $ /bin/bash –version
  + $USER – Username.
  + View all: env or set.
* **Command-Line Arguments**
  + $0 – Script name.
  + $1, $2, ... – Arguments.
  + $# – Number of arguments.
  + $@ – All arguments as separate words.
  + $\* – All arguments as a single string.

### **Conditional Statements**

* **If Statement**

if [ -f myfile.txt ]; then

echo "File exists"

else

echo "File does not exist"

fi

* **String Comparisons**

[ "$a" = "$b" ] → Equal

[ "$a" != "$b" ] → Not equal

[ -z "$a" ] → Empty string

[ -n "$a" ] → Non-empty string

* **Number Comparisons**

eq → Equal ([ $a -eq $b ])

ne → Not equal

gt → Greater than

lt → Less than

### **Loops**

* **For Loop**

for i in {1..5}; do

echo "Number: $i"

done

* **While Loop**

x=1

while [ $x -le 5 ]; do

echo "Iteration $x"

x=$((x+1))

done

* **Until Loop**

x=5

until [ $x -lt 1 ]; do

echo "Countdown: $x"

x=$((x-1))

done

### **Shell Functions**

* **Define and Call a Function**

my\_function() {

echo "Hello from function"

}

my\_function

* **Pass Arguments to Functions**

my\_function() {

echo "First argument: $1"

echo "Second argument: $2"

}

my\_function "Hello" "World"

### **Case Statement**

case "$var" in

yes) echo "You chose yes!" ;;

no) echo "You chose no!" ;;

\*) echo "Invalid choice" ;;

Esac

### **Arithmetic Operations**

* **Using expr**

a=5

b=10

sum=$(expr $a + $b)

echo $sum

* **Using $(( ))**

sum=$((5 + 10))

echo $sum

* **Using bc for Floating Point**

echo "scale=2; 5/2" | bc

### **Arrays in Shell**

* **Define an Array**

arr=("apple" "banana" "cherry")

* **Access Elements**

echo ${arr[0]} # First element

echo ${arr[@]} # All elements

echo ${#arr[@]} # Array length

* **Loop Through Array**

for fruit in "${arr[@]}"; do

echo "$fruit"

done

### **Shell Scripting Examples**

1. **Basic Script**

#!/bin/sh

echo "Hello, World!"

exit 0

* + Make executable: chmod +x script.sh
  + Run: ./script.sh

1. **Check File Exists**

if [ -f "$1" ]; then

echo "File exists"

else

echo "File not found"

fi

1. **Loop Through Files**

for file in \*.txt; do

echo "Processing $file"

done

# INPUT/OUTPUT REDIRECTION

| **Command** | **Description** | |
| --- | --- | --- |
| echo TEXT | | Display a line of TEXT or the contents of a variable. |
| echo -e TEXT | | Also interprets escape characters in TEXT, e.g. \n → new line, \b → backslash, \t → tab. |
| echo -n TEXT | | Omits trailing newline of TEXT. |
| cmd1 | cmd2 | | | is the pipe character; feeds the output of the command cmd1 and sends it to the command cmd2, e.g. ps aux | grep python3. |
| cmd > file | | Output of cmd is redirected to file. Overwrites pre-existing content of file. |
| cmd >> file | | Output of cmd is appended to file. |
| cmd < file | Input of cmd is read from file. | |

# FILE MANAGEMENT

| **Command** | **Description** |
| --- | --- |
| \* | Wildcard symbol for variable length, e.g. \*.txt refers to all files with the TXT extension. |
| ? | Wildcard symbol referring to a single character, e.g. Doc?.docx can refer to Doc1.docx, DocA.docx, etc. |
| ls | List the names of files and subfolders in the current directory. |
| ls -l | Also show details of each item displayed, such as user permissions and the time/date when the item was last modified. |
| ls -a | Also display hidden files/folders. May be combined with ls -l to form ls -al. |
| ls -t | Sort the files/folders according to the last modified time/date, starting with the most recently modified item. |
| ls X | List the files |
| cd Y | Change directory to Y. Special instances of Y: .  — current directory .. — parent directory |
| cd | To the $HOME directory |
| cd .. | Up one level to enclosing folder / parent directory |
| cmp A B | Compare two files A and B for sameness. No output if A and B are identical, outputs character and line number otherwise. |
| diff A B | Compare two files A and B for differences. Outputs the difference. |
| pwd | Display the path of the current working directory. |
| mkdir X | Make a new directory named X inside the current directory. |
| mv A B | Move a file from path A to path B. Also used for renaming files. mv ./folder1/file.txt ./folder2 |
| cp A B | Copy a file from path A to path B.  Example: cp ./f1/file.txt ./f2/expenses.txt simultaneously copies the file file.txt to the new location with a new name expenses.txt. |
| cp -r Y Z | Recursively copy a directory Y and its contents to Z. If Z exists, copy source Y into it; otherwise, create Z and Y becomes its subdirectory with Y’s contents |
| rm X | Remove (delete) X permanently. |
| rm -f X | Forcibly remove file X without prompts or confirmation |
| rm -rf Y | Forcibly remove directory Y and its contents recursively |
| rmdir Y | Remove a directory Y permanently, provided Y is empty. |
| du -ah | Disk usage in human readable format (KB, MB etc.) |
| du -sh | Total disk usage of the current directory |
| open X | Open X in its default program. |
| open -e X | Opens X in the default text editor (macOS: TextEdit) |
| touch X | Create an empty file X or update the access and modification times of X. |
| cat X | View contents of X. |
| cat -b X | Also display line numbers as well. |
| wc X | Display word count of X. |

# ARRAY

In bash shell: ARRAY\_NAME=(value1 ... valueN)

Accessing array values (zero-indexed, i.e. first element is at [0] not [1]):

| **Array variable** | **Description** |
| --- | --- |
| ${ARRAY\_NAME[index]} | Display the value at [index] of ARRAY\_NAME. |
| ${ARRAY\_NAME[\*]} | Display all values of the array ARRAY\_NAME. |
| ${ARRAY\_NAME[@]} | Same as ${ARRAY\_NAME[\*]}. |

# BASIC OPERATIONS

| **Relational operator** | **Description** |
| --- | --- |
| -eq | Equal to |
| -ne | Not equal to |
| -gt | Greater than |
| -lt | Less than |
| -ge | Greater than or equal to |
| -le | Less than or equal to |

| **Boolean operator** | | **Description** |
| --- | --- | --- |
| ! | | Logical negation / not: inverts true/false condition |
| -o | Logical OR (inclusive): returns true if any one of the operands is true | |
| -a | Logical AND: returns true if all operands are true | |

| **String operator** | **Description** |
| --- | --- |
| = | Returns true if the two operands on both sides of = are equal. |
| != | Returns true if the two operands on both sides of != are not equal. |
| -z $STRING\_VAR | Returns true if $STRING\_VAR is zero in length. |
| -n $STRING\_VAR | Returns true if $STRING\_VAR is not zero in length. |
| [ $STRING\_VAR ] | Returns true if $STRING\_VAR is not the empty string. |

# Shell Script

### **Files in the directory:**

1f

11f

111f

1111f

11111f

longf

.longfille2 ( hidden file because starts with **.** )

1. **echo \*f\***
   * expands to all files that contain **"f"** anywhere in their name.

1f 11f 111f 1111f 11111f longf

1. **echo \*lon\***
   * expands to all files that contain **"lon"** anywhere in their name.

longf

* + .longfille2 is **not** included because hidden files (those starting with .) are ignored unless explicitly included using .\*.

1. **echo .\***
   * This command lists **all hidden files** in the directory, because .\* matches files that start with a dot (.)

. .. .longfille2

* + . (current directory) and .. (parent directory) are included in the output.

1. **echo [1f]\***
   * This pattern [1f]\* matches filenames that **start** with either 1 or f.

1f 11f 111f 1111f 11111f

# **Opportunities to Practice Scripts**

1. Write the running command and  a script to check whether a number that is entered from the keyboard is one digit number , 2 digits number , or 3 or more digits number.

check\_digits() {

echo -n "Enter a number: "

read num

if [[ ! "$num" =~ ^[0-9]+$ ]]; then

echo "Invalid input. Please enter a numeric value."

elif [[ ${#num} -eq 1 ]]; then

echo "The number is a one-digit number."

elif [[ ${#num} -eq 2 ]]; then

echo "The number is a two-digit number."

else

echo "The number has three or more digits." fi

}

2. The purpose of this script is to calculate the amount that will cost a customer when renting a car given the type of car and number of renting days.

Type of car

car   @         $25 per day

van  @         $40 per day

jeep @         $35 per day

bicycle  @  $15 per day

Exit to terminate the program.

Anything else is an invalid option.

calculate\_rental() {

while true; do

echo "Choose a vehicle to rent:"

echo "1. car @ \$25 per day"

echo "2. van @ \$40 per day" echo "3. jeep @ \$35 per day"

echo "4. bicycle @ \$15 per day"

echo "5. Exit"

echo -n "Enter your choice: "

read vehicle

case $vehicle in

"car"|"Car")

rate=25 ;;

"van"|"Van")

rate=40 ;;

"jeep"|"Jeep")

rate=35 ;;

"bicycle"|"Bicycle")

rate=15 ;;

"exit"|"Exit")

echo "Exiting the program."

exit 0 ;;

\*)

echo "Invalid option. Please try again."

continue ;;

1  -  What is the output of the following

**array=( apple bat cat dog elephant frog )**

echo ${array[0]}

apple

echo ${array[@]:0} (Prints **all elements** starting from index 0 (which is the entire array).

apple bat cat dog elephant frog

echo ${array[@]:2}

cat dog elephant frog

echo ${#array[0]} (Prints the **length of the first element)**

5 (apple has 5 characters)

echo ${#array[@]} (Prints the **number of elements in the array**)

6

echo ${array[@]//a/A}

Apple bAt cAt dog elephAnt frog

# **Opportunities to Practice Functions - Shell Script**

**Write script that Add 2 numbers entered from the keyboard. Your script must contain 2 functions:**

* **A function to accept the integers values from the user.**
* **A function to process the continue / exit prompt**

The following is the sample run:

Enter first integer: 10

Enter second integer: 20

The result is:  30

Do you wish to continue? [y]es or [n]o: n

Exiting the script. Have a nice day!

#!/bin/bash

get\_numbers() {

echo -n "Enter first integer: "

read num1

echo -n "Enter second integer: "

read num2

if ! [[ "$num1" =~ ^-?[0-9]+$ && "$num2" =~ ^-?[0-9]+$ ]]; then

echo "Invalid input. Please enter valid integers."

return

fi

sum=$((num1 + num2))

echo "The result is: $sum"

}

continue\_prompt() {

while true; do

echo -n "Do you wish to continue? [y]es or [n]o: "

read choice

case $choice in

[Yy]\*) return 0 ;; # Continue the loop

[Nn]\*) echo "Exiting the script. Have a nice day!"; exit 0 ;;

\*) echo "Invalid input. Please enter 'y' or 'n'." ;;

esac

done

}

# **Random Numbers and Grep Shell Script**

**Assume that you have the following file**

#!/bin/sh

myvar="Hi there"

echo $myvar

echo "$myvar"

echo '$myvar'

echo \$myvar

echo Enter some text

read myvar

echo '$myvar' now equals $myvar

exit 0

**What is the output of the following:**

1. What is the output of the above script if you run the script and enter Hello.

Hi there

Hi there

$myvar

$myvar

Enter some text

Hello

'$myvar' now equals Hello

1. bash-3.2$ grep "echo" sc2.sh (prints all with echo)

echo $myvar

echo "$myvar"

echo '$myvar'

echo \$myvar

echo Enter some text

echo '$myvar' now equals $myvar

1. bash-3.2$ grep -c "echo" sc2.sh (counts number of lines w/ echo)

6

1. bash-3.2$ grep -l "echo" sc2.sh
2. bash-3.2$ grep -v "echo" sc2.sh (all lines that do not contain echo)

myvar="Hi there"

read myvar

exit 0

1. bash-3.2$ grep "e\?ho" sc2.sh (Searches for "ho" optionally preceded by "e")

echo $myvar

echo "$myvar"

echo '$myvar'

echo \$myvar

echo Enter some text

echo '$myvar' now equals $myvar

1. bash-3.2$ grep [[:digit:]] sc2.sh (search for digits)

exit 0

1. bash-3.2$ grep [[:alpha:]] sc2.sh

echo $myvar

echo "$myvar"

echo '$myvar'

echo \$myvar

echo Enter some text

echo '$myvar' now equals $myvar

1. bash-3.2$ grep d[[:space:]] sc2.sh ("d" is followed by a space)

read myvar

**Q2 - Write a Command that Generates a Random  Number between 1 and 139.**

echo $(( RANDOM % 139 + 1 ))

($RANDOM generates a random number between 0 and 32767.)

# **Opportunities to Practice Shell Command**

|  |  |
| --- | --- |
| **Copy the file "foo.txt" into "foo.copy"** | cp foo.txt foo.copy |
| **Rename the file "foo.copy" to "foo2"** | mv foo.copy foo2 |
| **Create a new subdirectory called "new" in the Unix directory** | mkdir new |
| **Delete the file "foo.old"** | rm foo.old |
| **Remove the "new" subdirectory.** | rmdir new  (If the directory is not empty, use *rm -r new* instead.) |
| **Move to the directory above the current one.** | cd .. |
| **Display the content of the file "verylong.seq" on the screen.** | cat verylong.seq |
| **Change the protection of your own file "foo.txt" so that anyone may read it.** | chmod 644 foo.txt |
| **Change the password to your account.** | passwd |
| **What is the command to count lines, words, and characters in a file?** | wc foo.txt |
| **List all files that contain "seq" in the filename.** | ls \*seq\* |
| **List the names of all files that end with "seq" in their filename.** | ls \*seq |
| **Show the current date and time.** | date |
| **Show a calendar of the current month.** | cal |
| **Show a calendar of a specific month/year.** | cal 3 2025  (This example shows the calendar for March 2025.) |