Batch: A2	**Roll No.: 16010121045**
Experiment No. 02	
Grade: AA / AB / BB / BC / CC / CD /DD	
Signature of the Staff In-charge with date	

TITLE: Shell Programming and system calls

AIM: To study the shell script and write the program using shell.

Expected Outcome of Experiment:

CO 1. To introduce basic concepts and functions of operating systems.

Books/ Journals/ Websites referred:

1. **Silberschatz A., Galvin P., Gagne G. "Operating Systems Principles", Willey Eight edition.**
2. **William Stallings "Operating Systems" Person, Seventh Edition Edition.**
3. **Sumitabha Das " UNIX Concepts & Applications", McGraw Hill Second Edition.**

Pre Lab/ Prior Concepts:

The shell provides you with an interface to the UNIX system. It gathers input from you and executes programs based on that input. When a program finishes executing, it displays that program's output.

Shell Scripts
The basic concept of a shell script is a list of commands, which are listed in the order of execution. A good shell script will have comments, preceded by a pound sign, #, describing the steps.

Steps to create a Shell Script:
create a file using any text editor say vi, gedit, nano etc
1.$ vi filename

2.Insert the script/ commands in file and save the file to execute the file we need to give execute permission to the file
3.$ chmod 775 filename
4.Now execute the above file using any of following methods:
$ sh filename
OR
$ ./filename

NOTE: Before adding anything to your script, you need to alert the system that a shell script is being started. This is done using the shebang construct. For example –
#!/bin/sh.

Description of the application to be implemented:

1. Write a shell Script that accepts two file names as command line arguments and compare two file contents and check whether contents are same or not. If they are same, then delete second file.

2. Write a shell script that accepts integer and find the factorial of number.

3. Write a shell script for adding users.

4. Write a shell script for counting no of logged in users.

5. Write a shell script for counting no of processes running on system

Implementation details: (printout of code / screen shot)

Code:

```
echo "Enter a number"
read num

fact=1
while [ $num -gt 1 ]
do
fact=$((fact * num)) #fact = fact * num
num=$((num - 1)) #num = num - 1
done

echo $fact
```

Output:

```
❯ bash "/Users/pargatsinghdhanjal/Desktop/Coding/OS/fact.sh"
Enter a number
5
120
```

Code:

```
echo "Enter first file name"
read f1
echo "Enter second file name"
read f2
if cmp -s $f1 $f2
then
echo "Files are same"
rm -rf $f2
else
echo "Files are different"
fi
```

Output:

```
❯ bash "/Users/pargatsinghdhanjal/Desktop/Coding/OS/filesCmp.sh"
Enter first file name
a.txt
Enter second file name
b.txt
Files are
```

Focus folder in explorer (cmd + click)

Code:

```
logged_in_users=$(who)
count=$(echo "$logged_in_users" | wc -l | xargs)
echo "Number of logged-in users: $count"
```

Output:

```
❯ bash "/Users/pargatsinghdhanjal/Desktop/Coding/OS/logUsers.sh"
Number of logged-in users: 1
```

Code:

```
p=$(ps -A)
count=$(echo "$p" | wc -l  | xargs)
echo "Number of processes: $count"
```

Output:

```
❯ bash "/Users/pargatsinghdhanjal/Desktop/Coding/OS/tempCodeRunnerFile.sh"
Number of processes: 367
```

Program for system call

Code:

```
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
int main()
{
    int f = fork();
    if (f < 0)
    {
        printf("Fork failed\n");
    }
    else if (f == 0)
    {
        printf("Child process: %d\n", getpid());
        printf("Parent process: %d\n", getppid());
    }
    else
    {
        printf("Parent process: %d\n", getpid());
        printf("Child process: %d\n", f);
    }
    return 0;
}
```

Output:

```
❯ cd "/Users/pargatsinghdhanjal/
Parent process: 48676
Child process: 48845
Child process: 48845
Parent process: 1
```

Conclusion :

Learnt the to write and execute bash shell scripts.

Post Lab Descriptive Questions

1. What are the different types of commonly used shells on a typical linux system?

On a typical Linux system, there are several commonly used shells. A shell is a command-line interface that allows users to interact with the operating system. Some of the popular shells include:

- **Bash (Bourne-Again Shell)**: This is the default shell for most Linux distributions. It's known for its powerful scripting capabilities and wide usage.
- **Zsh (Z Shell)**: Zsh is an extended version of Bash with additional features and customization options. It offers advanced tab completion and theming capabilities.
- **Fish (Friendly Interactive Shell)**: Fish is designed to be user-friendly and interactive. It has syntax highlighting, autocompletion, and a modern command-line experience.
- **Ksh (Korn Shell)**: Ksh is an older shell with advanced scripting features. There are different variants like ksh88 and ksh93, offering varying levels of functionality.
- **Csh (C Shell)**: Csh has a C-like syntax and features, but it's not as widely used today due to its limitations.

Each of these shells has its own set of features, advantages, and user communities.

2. How do you find out what's your shell?

To find out which shell you are currently using, you can use the **echo** command along with the **$SHELL** environment variable. Open a terminal and enter the following command:

echo $SHELL

This will display the path to the shell you're currently using, such as **/bin/bash**, **/usr/bin/zsh**, etc.

3. List the advantages and disadvantages of shell scripting.

Advantages:

- **Automation**: Shell scripting allows you to automate repetitive tasks, making system administration and maintenance more efficient.
- **Rapid Development**: Shell scripts are generally quick to write and test, making them useful for creating small utilities or scripts on-the-fly.
- **Accessibility**: Shell scripting provides a powerful command-line interface that can be accessed remotely, which is beneficial for remote administration and scripting tasks.
- **Integration**: Shell scripts can easily interact with system utilities and other command-line tools, allowing seamless integration with existing software.
- **Customization**: Shells like Bash offer a range of features like variables, functions, and conditional statements, enabling you to create complex and customized scripts.

Disadvantages:

- **Limited Performance**: For resource-intensive tasks, interpreted shell scripts may have performance issues compared to compiled languages.
- **Complexity**: As scripts grow in size and complexity, maintaining and debugging them can become challenging.
- **Portability**: Shell scripts can be dependent on specific shell features, making them less portable between different shell environments.
- **Security**: Poorly written shell scripts can have security vulnerabilities if not properly sanitized, potentially leading to system compromises.
- **Lack of GUI**: Shell scripts operate in a command-line environment, so they may not be suitable for tasks requiring a graphical user interface.

Despite these disadvantages, shell scripting remains a powerful tool for various system administration and automation tasks, especially when used judiciously and with proper consideration for security and performance.

Date: _____ **Signature of faculty in-charge**