**Bash and Bash scripts**

1. Where is the bash program located on your system?

Ans: In my system the bash program is in the location ‘**/usr/bin/bash**’

A black background with a black rectangle

Description automatically generated

1. Use the --version option to find out which version you are running. (Add screenshot)

A screen shot of a computer program

Description automatically generated

1. why command bash\_profile is used in Linux?

Ans: In Linux and Unix-like systems, the ‘bash\_profile’ file in a user's home directory is crucial for customizing and configuring the Bash shell. It allows users to set environment variables, define aliases, and personalize their shell experience. When a user logs in, the Bash shell executes the ‘bash\_profile,’ ensuring that their preferred environment is established for each session. This file can also be used for security measures and access control. By using it, users keep their shell customization organized, avoiding clutter in system-wide configuration files like those in ‘/etc.’

1. Which shell configuration files are read when you log in to your system using the graphical user interface and then opening a terminal window?

Ans:

A black background with white text

Description automatically generated

I am using the bash environment. So, the files which are read when I login to my system are:

1. The ‘**~/.bash\_profile’** is a Bash-specific file that gets read when a user logs in, providing them with the ability to personalize their Bash environment, define environment variables, and run custom commands or scripts.
2. The file ‘**~/.bashrc**’ is exclusive to the Bash shell and gets executed when a user launches a new interactive Bash shell, commonly when opening a terminal window in the GUI. It's not typically used during the initial GUI login process. Users frequently use this file to add customizations, define aliases, and set environment variables.
3. The ‘**~/.bash\_logout**’ file is exclusive to the Bash shell and gets executed when a user logs out. In this file, users can include commands for cleaning up or specifying actions to be carried out as part of the logout process.
4. The ‘**/etc/profile**’ file is a configuration file that applies to all users during login. It establishes system-wide settings and environment variables. It commonly references the ‘**/etc/profile.d/**’ directory, which is a location for adding extra scripts to customize the system.
5. What is the .bashrc file in Linux? (Add screenshot of cat .bashrc)

Ans: The ‘**.bashrc**’ file found in Linux is a configuration file that is closely tied to the Bash shell, a widely used command-line interface in numerous Linux distributions. It is situated within a user's home directory and serves the purpose of personalizing and adjusting the way the Bash shell operates for that particular user.

A screenshot of a computer program

Description automatically generated

1. What is an interactive shell and login shell?

Ans: **Interactive Shell:** An interactive shell is a command-line interface that enables users to engage with the computer in real-time by entering commands. It offers a prompt for user input and is commonly utilized for actions like executing commands, exploring the file system, and overseeing processes.

**Login Shell:** A login shell is an interactive shell that begins when a user logs into a computer system. It runs specific configuration files like ‘**~/.bash\_profile’** and establishes the user's environment, serving to initialize their session and guarantee proper configuration.

1. Are the following shells interactive shells? Are they login shells?

* A shell is opened by clicking on the background of your graphical desktop, selecting "Terminal" or such from a menu – **Interactive Shell**
* A shell that you get after issuing the command ssh localhost – **Both Interactive and Login Shell.**
* A shell that you get when logging in to the console in text mode - **Both Interactive and Login Shell.**
* A shell obtained by the command xterm & - **Interactive Shell.**
* A shell opened by the mysystem.sh script – **Not an Interactive nor a Login Shell.**
* A shell that you get on a remote host, for which you didn't have to give the login and/or password because you use SSH and maybe SSH keys - **Both Interactive and Login Shell.**

1. Can you explain why bash does not exit when you type Ctrl+C on the command line?

Ans: Pressing Ctrl+C in Bash doesn't cause the shell to exit because Ctrl+C is meant to interrupt and terminate the currently running process in the terminal, not to shut down the entire shell session. This behavior allows you to halt or stop a specific command or process while leaving the shell session operational for additional commands. To exit the shell itself, you can use either the "exit" command or Ctrl+D (EOF signal).

1. Display directory stack content. (Add screenshot)



1. Display hash commands for your current shell session. (Add screenshot)

A black screen with white text

Description automatically generated

1. How many processes are currently running on your system? Use ps and wc, the first line of output of ps is not a process! (Add screenshot)

Ans: There are 141 processes running in my system currently.

A black screen with white text

Description automatically generated

1. How to display the system hostname? Only the name, nothing more! (Add screenshot)

A black screen with white text

Description automatically generated

1. Write down the properties of good scripts.

Ans: There are many properties of good scripts. I am listing five of the most important properties:

1. **Clarity and Readability**: Good scripts are easy to comprehend. They employ clear and descriptive variable and function names, adhere to a consistent coding style, and incorporate comments to clarify intricate code segments. Readability is crucial for script maintenance and collaboration.
2. **Modularity:** Quality scripts are structured into reusable functions or modules. This enhances code maintainability and simplifies testing and debugging of individual components.
3. **Error Handling:** Effective scripts excel in error management. They anticipate and manage errors gracefully, delivering informative error messages, and, when feasible, recovering from errors or exiting with meaningful error codes.
4. **Efficiency:** Efficient scripts are crafted to execute tasks with minimal resource consumption, like CPU and memory. They steer clear of unnecessary loops, redundant operations, and excessive resource usage to ensure optimal performance.
5. **Documentation:** A well-crafted script comes with clear and concise documentation that elucidates its purpose, usage, inputs, outputs, and any prerequisites or dependencies. Comprehensive documentation simplifies script utilization and comprehension, even for future users or yourself.

These properties contribute to the reliability, maintainability, and usability of scripts, making them more effective tools for automation and other tasks.

**Writing and debugging scripts**

1. Write a script using your favorite editor. The script should display the path to your home directory and the terminal type that you are using. Additionally, it shows all the services started up in runlevel 3 on your system.

A screenshot of a computer

Description automatically generated

1. Add comments in your script.

A screenshot of a computer program

Description automatically generated

1. Add information for the users of your script. (Add screenshot)

A screenshot of a computer program

Description automatically generated

1. Change permissions on your script so that you can run it.

A screenshot of a computer

Description automatically generated

1. Run the script in normal mode and debug mode. It should run without errors.

A screenshot of a computer program

Description automatically generated

1. Make errors in your script: see what happens if you misspell commands, if you leave out the first line or put something unintelligible there, or if you misspell shell variable names or write them in lowercase characters after they have been declared in capitals. Check what the debug comments say about this. (Add screenshot)

A screenshot of a computer program

Description automatically generated

**The Bash environment**

1. Create 3 variables, VAR1, VAR2, and VAR3; initialize them to hold the values "thirteen", "13" and "Happy Birthday" respectively. (Add screenshot)

**A black screen with white text

Description automatically generated**

1. Display the values of all three variables. (Add screenshot)

A screenshot of a computer program

Description automatically generated

1. Are these local or global variables? Explain in brief.

Ans: These are local variables as these variables are used within this script itself and cannot be used outside of this script. They are locally declared and are printed within this script.

1. Remove VAR3.

A screenshot of a computer

Description automatically generated

1. Can you see the two remaining variables in a new terminal window? (Add screenshot)

Ans: When I run the same program after unsetting the variable ‘**VAR3’**, I can see the variables ‘**VAR1**’ and ‘**VAR2**’ as these two variables still hold values.

A black screen with white text

Description automatically generated

1. Edit /etc/profile so that all users are greeted upon login (test this). (Add screenshot)

A screenshot of a computer screen

Description automatically generated

A screen shot of a computer

Description automatically generated

1. Write a script in which you assign two integer values to two variables. The script should calculate the surface of a rectangle that has these proportions. It should be aired with comments and generate elegant output.

A screenshot of a computer program

Description automatically generated

**Write a bash script**

1. Use an if/then/elif/else construct that prints information about the current month. The script should print the number of days in this month and give information about leap years for the entered value of the Year.

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screen shot of a computer

Description automatically generated

1. Give a brief explanation of the Case statement used in Bash scripting. Run the following bash script and explain the output.

Ans: In Bash scripting, the **case** statement is a versatile control structure employed for making decisions based on the value of a variable or expression. It enables you to compare the variable or expression against various patterns or conditions and execute specific code blocks when a match is found, with execution stopping after the first successful match.

In this program, the user will input the day and according to the input cases will be executed. In my output, as today is ‘**Mon’** the first case gets executed, and the output is shown as ‘**Today is a weekday**’. If the user runs this on weekend ‘**Sat or Sun’** the second case will be executed, and the output will be shown as ‘**Today is the weekend**’ and lastly for all other cases the **default case** will be executed which will give the output ‘**date is not recognized**’.

A screenshot of a computer

Description automatically generated

\_\_\_End of Lab\_\_\_