DAY 3

Understand Tab Completion

With tab completion, users can quickly complete commands or file names after typing enough at the prompt to make it unique. If the typed characters are not unique, then pressing the **Tab** key twice displays all commands that begin with the typed characters.

```
[user@host ~]$ pasTab+Tab (1)
passwd       paste        pasuspender
[user@host ~]$ passTab (2)
[user@host ~]$ passwd
Changing password for user user.
Current password:
```

(1) Press **Tab** twice.

(2) Press **Tab** once.

Tab completion helps to complete file names when typing them as arguments to commands. Press **Tab** to complete as much of the file name as possible. Pressing **Tab** a second time causes the shell to list all files that the current pattern matches. Type additional characters until the name is unique, and then use tab completion to complete the command.

```
[user@host ~]$ ls /etc/pasTab (1)
[user@host ~]$ ls /etc/passwdTab (2)
passwd   passwd-
```

(1) Press **Tab** once.

(2) Press **Tab** once.

Use the useradd command to create users on the system. The useradd command has many options that might be hard to remember. By using tab completion, you can complete the option name with minimal typing.

```
[root@host ~]# useradd --Tab+Tab (1)
--badnames              --gid                 --no-log-init         --shell
--base-dir              --groups              --non-unique          --skel
--btrfs-subvolume-home  --help                --no-user-group       --system
--comment               --home-dir            --password            --uid
--create-home           --inactive            --prefix              --user-group
--defaults              --key                 --root
--expiredate            --no-create-home      --selinux-user
```

(1) Press **Tab** twice.

To write one command in more than one line, use a backslash character (\), which is referred to as the escape character.

**[user@host ~]$ head -n 3 **
/usr/share/dict/words

1. Which Bash command displays the last five lines of the `/var/log/messages` file?

A ○ `head -n 10 /var/log/messages`

B ○ `tail 10 /var/log/messages`

C ○ `tail -n 5 /var/log/messages`

D ○ `tail -l 10 /var/log/messages`

E ○ `less /var/log/messages`

2. Which Bash shortcut or command separates commands on the same line?

A ○ Pressing **Tab**

B ○ `history`

C ○ `;`

D ○ *`!string`*

E ○ Pressing **Esc**+.

3. Which Bash command is used to change a user's password?

A ○ `password`

B ○ `pass`

C ○ `passwd`

D ○ `usermod`

E ○ `userpassword`

4. Which Bash command is used to display the file type?

A ○ `file`

B ○ `less`

C ○ `cat`

D ○ `history`

E ○ `view`

DISPLAY THE COMMAND HISTORY

The **history** command displays a list of previously executed commands that are prefixed with a command number.

The exclamation point character (!) is a metacharacter to expand previous commands without retyping them.

The **!number** command expands to the command that matches the specified number.

The **!string** command expands to the most recent command that begins with the specified string.

[user@host ~]$ history
...output omitted...
23 clear
24 who
25 pwd
26 ls /etc
27 history

[user@host ~]$!ls

[user@host ~]$!26

ABSOLUTE PATHS AND RELATIVE PATHS

The path of a file or directory specifies its unique file-system location.

Following a file path traverses one or more named subdirectories, which are delimited by a forward slash (/), until the destination is reached.

Directories, also called folders, can contain other files and other subdirectories. Directories are referenced in the same manner as files.

An **absolute path** is a fully qualified name that specifies the exact location of the file in the file-system hierarchy.
It begins at the root (/) directory and includes each subdirectory that must be traversed to reach the specific file.

Every file in a file system has a unique absolute path name, which is recognized with a simple rule: a path name with a forward slash (/) as the first character is an absolute path name.

For example, the absolute path name for the system message log file is **/var/log/messages**.

A **relative path** identifies a unique location and specifies only the necessary path to reach the location from the working directory. Relative path names follow this rule: a path name with anything other than a forward slash as the first character is a relative path name.

For example, relative to the **/var** directory, the message log file is **log/messages**.

Displays the full path name of the current working directory
pwd

Lists directory contents for the specified directory or, if no directory is given, for the current working directory.
ls
ls –l # long listing format
ls –a # all files, including hidden files
ls –R # recursive, to include the contents of all subdirectories

To change your shell's current working directory. If you do not specify any arguments to the command, then it changes to your home directory.
cd /etc

To move up one level to the parent directory, without needing to know the exact parent name.
cd ..

To create a file & updates the time stamp of a file to the current date and time.
touch /tmp/test.txt

Which command returns to your current home directory, assuming that the current working directory is `/tmp` and your home directory is `/home/user`?

A ○ `cd`

B ○ `cd ..`

C ○ `cd .`

D ○ `cd *`

E ○ `cd /home`

Which command changes the working directory to the parent of the current location?

A ○ `cd ~`

B ○ `cd ..`

C ○ `cd ../..`

D ○ `cd -u1`

COMMAND-LINE FILE MANAGEMENT

Creating, copying, moving, and removing files and directories are common operations for a system administrator. Without options, some commands are used to interact with files, or they can manipulate directories with the appropriate set of options.

 # Create Directories

The **mkdir** command creates one or more directories or subdirectories. It takes as an argument a list of paths to the directories that you want to create.

```
mkdir Project
```

 # Copy Files and Directories

The **cp** command copies a file and creates a file either in the current directory or in a different specified directory.

```
cp file1.txt /tmp/file2.txt
cp file1.txt file2.txt Project
```