SECR2043 - OPERATING SYSTEMS (Lab Exercise)

Written by: Dr. Farkhana Muchtar

LAB 4: LINUX FILE SYSTEM AND BASH SCRIPTING

Introduction of File System and Shell Script

1 Overview

To learn and understand the Linux File System and basics of Bash scripting, with a focus on file permissions, different types of file paths, and the creation of symbolic and hard links.

2 File System and File Permissions

2.1 Exercise 1: Directory Navigation: Absolute and Relative Paths

1. **Basic Navigation**

   a. Use the `pwd` command to display the current working directory.
   b. Create a directory named `LabExercise` in your home directory using the command:

   ```
   mkdir ~/LabExercise
   ```

   c. Navigate into the `LabExercise` directory:

   ```
   cd ~/LabExercise
   ```

   d. Use the `pwd` command again to verify that you are in the `LabExercise` directory.
   e. Create a subdirectory named `SubDir1` inside the `LabExercise` directory using the command:

   ```
   mkdir SubDir1
   ```

   f. Navigate into the `SubDir1` directory:

   ```
   cd SubDir1
   ```

   g. Use the `pwd` command to verify that you are in the `SubDir1` directory.
   h. Create another subdirectory named `SubDir2` inside the `SubDir1` directory:

   ```
   mkdir SubDir2
   ```

2. **Path Navigation using Absolute Paths:**

   a. Use the `cd` command to navigate to the `LabExercise` directory using an absolute path:

   ```
   cd ~/LabExercise
   ```

   b. Use the `pwd` command to verify that you are in the `LabExercise` directory.

   c. Use the `cd` command to navigate to the `SubDir1` directory using an absolute path:

   ```
   cd ~/LabExercise/SubDir1
   ```

   d. Use the `pwd` command to verify that you are in the `SubDir1` directory.

   e. Use the `cd` command to navigate back to the `LabExercise` directory using an absolute path:

   ```
   cd ~/LabExercise
   ```

   f. Use the `pwd` command to verify that you are in the `LabExercise` directory.

3. **Path Navigation using Relative Paths:**

   a. Use the `cd` command to navigate to the `SubDir1` directory using a relative path:

   ```
   cd SubDir1
   ```

   b. Use the `pwd` command to verify that you are in the `SubDir1` directory.

   c. Use the `cd` command to navigate to the `SubDir2` directory using a relative path:

   ```
   cd SubDir2
   ```

   d. Use the `pwd` command to verify that you are in the `SubDir2` directory.

   e. Use the `cd` command to navigate back to the parent directory (`SubDir1`) using a relative path:

   ```
   cd ..
   ```

   f. Use the `pwd` command to verify that you are in the `SubDir1` directory.

   g. Use the `cd` command to navigate back to the parent directory (`LabExercise`) using a relative path:

   ```
   cd ..
   ```

   h. Use the `pwd` command to verify that you are in the `LabExercise` directory.

4. **Shortcut and Special Paths:**

   a. Use the `cd` command to navigate to your home directory using the ~ shortcut:

   ```
   cd ~
   ```

   b. Use the `pwd` command to verify that you are in your home directory.
   c. Use the `cd` command to navigate to the previous directory (`LabExercise`) using the .. shortcut:

   ```
   cd LabExercise
   ```

   d. Use the `pwd` command to verify that you are in the `LabExercise` directory.
   e. Use the `cd` command to navigate to the root directory using the / shortcut:

   ```
   cd /
   ```

   f. Use the `pwd` command to verify that you are in the root directory.
   g. Use the `cd` command to navigate.

   ```
   cd
   ```

2.2 Exercise 2: File Permission

File permissions in Linux determine who can read, write, and execute files. There are three types of permissions: **read (r)**, **write (w)**, and **execute (x)**, which can be assigned to three categories of users: **owner (u)**, **group (g)**, and **others (o)**. Here is a breakdown of the permission notations:

1. **Symbolic Notation:**

- **u** represents the owner of the file.
- **g** represents the group that the file belongs to.
- **o** represents other users who are not the owner or part of the group.
- **+** adds the specified permission.
- **-** removes the specified permission.
- **r** represents read permission.
- **w** represents write permission.
- **x** represents execute permission.

2. **Octal Notation:**

   Each permission is represented by a digit in the octal system (base-8), where read is 4, write is 2, and execute is 1.

   The three digits represent permissions for the owner, group, and others, respectively.

   For example, 644 represents read-write for the owner and read-only for the group and others.

SECR2043 - OPERATING SYSTEMS (Lab Exercise)

Written by: Dr. Farkhana Muchtar

By using the chmod command, you can modify file permissions either by specifying the permissions in symbolic notation or by using octal values.

Below are the summary of file permission notation (both symbolic and octal notation).

Permission	**Symbolic Notation**	**Octal Notation**
Read	r	4
Write	w	2
Execute	x	1
Owner	u	-
Group	g	-
Others	o	-
Add Permission	+	-
Remove Permission	-	-

In the Octal Notation column, the numbers 4, 2, and 1 represent the read, write, and execute permissions, respectively. You can use these numbers to form different combinations to represent the desired permissions for the owner, group, and others.

Before we start to use chmod command to change file permission,

i. create dummy files, file.txt, file1.txt, file2.txt, script.sh and reference_file.txt.

```
touch file.txt file1.txt file2.txt script.sh reference_file.txt
```

1. **Changing Permissions with Symbolic Notation:**

   a. Adds executable permission for the owner of the file (u is for user/owner).

   ```
   ls -l file.txt
   chmod u+x file.txt
   ls -l file.txt
   ```

   b. Removes write permission for the group and others (g is for group, o is for others).

   ```
   ls -l file.txt
   chmod go-w file.txt
   ls -l file.txt
   ```

   c. Adds read permission for all (owner, group, and others) (a is for all).

   ```
   ls -l file.txt
   chmod a+r file.txt
   ls -l file.txt
   ```

2. **Changing Permissions with Octal Notation:**

   a. Sets read, write, and execute permissions for the owner, and read and execute permissions for the group and others.

   (Owner: 7 = 4+2+1, Group: 5 = 4+1, Others: 5 = 4+1).

```
ls -l file.txt
chmod 755 file.txt
ls -l file.txt
```

   b. Sets read and write permissions for the owner, and read-only permissions for the group, while removing all permissions for others.

   (Owner: 6 = 4+2, Group: 4 = 4, Others: 0).

```
ls -l file.txt
chmod 640 file.txt
ls -l file.txt
```

3. **Changing Permissions with Reference File:**

   a. Sets the permissions of `file.txt` to match those of `reference_file.txt`.

```
ls -l reference_file.txt file.txt
chmod --reference=reference_file.txt file.txt
ls -l reference_file.txt file.txt
```

4. **Changing Permissions for Multiple Files:**

   a. Sets read and write permissions for the owner, read permissions for the group, and removes all permissions for others for both `file1.txt` and `file2.txt`.

```
ls -l file1.txt file2.txt
chmod u=rw,g=r,o= file1.txt file2.txt
ls -l file1.txt file2.txt
```

   b. Sets read, write, and execute permissions for all (owner, group, and others) for all files with the .txt extension in the current directory.

```
ls -l *.txt
chmod 777 *.txt
ls -l *.txt
```

5. **Changing Permissions Recursively for Directories:**

   a. Adds write permission for the owner and all files and directories within directory recursively.

   ```
   ls -lR LabExercise
   chmod -R u+w LabExercise
   ls -lR LabExercise
   ```

6. **Changing Permissions with Special Modes:**

   a. Adds executable permission for all (owner, group, and others) using the + sign.

   ```
   ls -l script.sh
   chmod +x script.sh
   ls -l script.sh
   ```

   b. Sets the group ID on execution, ensuring new files and directories created within directory inherit the group ownership.

   ```
   ls -l LabExercise
   chmod g+s LabExercise
   ls -l LabExercise
   ```

SECR2043 - OPERATING SYSTEMS (Lab Exercise)

Written by: Dr. Farkhana Muchtar

3 Introduction to Bash Scripting

3.1 Exercise 1: Creating a Bash Script

1. Create a new file named `hello.sh` using the touch command. Open the file in a text editor (e.g., nano or vi)

   ```
   nano hello.sh
   ```

   and add the following lines to the top of the file:

   ```
   #!/bin/bash
   echo "Hello, World!"
   ```

2. Save and close the file. This is your first Bash script!
3. To make this script executable (runnable), give execute permissions to `hello.sh` with the command.

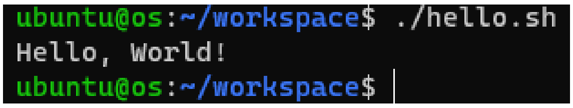
   ```
   chmod u+x hello.sh
   ```

   Run command `ls -l` before and after this command to see the changes.

4. Then, run the script by typing

   ```
   ./hello.sh
   ```

5. You should see `"Hello, World!"` printed to the console.



```
ubuntu@os:~/workspace$ ./hello.sh
Hello, World!
ubuntu@os:~/workspace$
```

3.2 Exercise 2: Using Variables

1. Create a new file named `hello2.sh` using the touch command. Open the file in a text editor (e.g., nano or vi)

   ```
   nano hello2.sh
   ```

   and add the following lines to the top of the file:

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

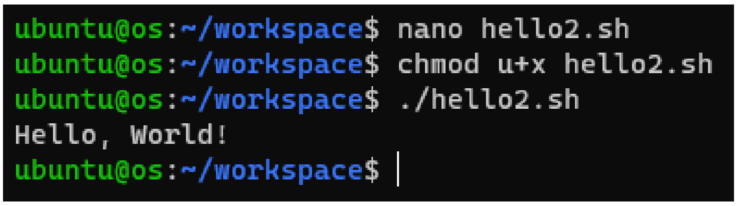
2. Save and close the file.
3. Give execute permissions to `hello2.sh` with the command

   ```
   chmod u+x hello2.sh
   ```

4. Then, run the script by typing.

```
./hello2.sh
```

5. You should see "`Hello, World!`" printed to the console.



```
ubuntu@os:~/workspace$ nano hello2.sh
ubuntu@os:~/workspace$ chmod u+x hello2.sh
ubuntu@os:~/workspace$ ./hello2.sh
Hello, World!
ubuntu@os:~/workspace$
```

3.3 Exercise 3: Using Variables for Arithmetic

1. Create a new file named `variable.sh` using the touch command. Open the file in a text editor (e.g., nano or vi)

```
nano variable.sh
```

and add the following lines to the top of the file:

```
#!/bin/bash

NUMBER1=10
NUMBER2=5
SUM=$((NUMBER1 + NUMBER2))
echo "The sum of $NUMBER1 and $NUMBER2 is: $SUM"
```

2. Save and close the file.
3. Give execute permissions to `variable.sh` with the command

```
chmod u+x variable.sh
```

4. Then, run the script by typing.

```
./variable.sh
```

5. You should see sum of the two numbers, printed to the console.

3.4 Exercise 4: Getting User Input

1. Create a new file named `hello3.sh` using the touch command. Open the file in a text editor (e.g., nano or vi)

   ```
   nano hello3.sh
   ```

   and add the following lines to the top of the file:

   ```
   #!/bin/bash
   echo "What's your name?"
   read NAME
   echo "Hello, $NAME!"
   ```

2. Save and close the file.
3. Give execute permissions to `hello3.sh` with the command

   ```
   chmod u+x hello3.sh
   ```

4. Then, run the script by typing.

   ```
   ./hello3.sh
   ```

5. You should see hellow message, following with name, printed to the console

3.5 Exercise 5: Getting User Input for Arithmetic

1. Create a new file named `input.sh` using the touch command. Open the file in a text editor (e.g., nano or vi)

   ```
   nano input.sh
   ```

   and add the following lines to the top of the file:

   ```
   #!/bin/bash
   echo "Enter the first number:"
   read NUMBER1

   echo "Enter the second number:"
   read NUMBER2

   SUM=$((NUMBER1 + NUMBER2))
   PRODUCT=$((NUMBER1 * NUMBER2))

   echo "The sum of $NUMBER1 and $NUMBER2 is: $SUM"
   echo "The product of $NUMBER1 and $NUMBER2 is: $PRODUCT"
   ```

2. Save and close the file.
3. Give execute permissions to `input.sh` with the command

   ```
   chmod u+x input.sh
   ```

4. Then, run the script by typing.

   ```
   ./input.sh
   ```

5. You should see sum and product of two numbers, printed to the console

3.6 Exercise 6: Conditional Statements

1. Create a new file named `if_statement.sh` using the touch command. Open the file in a text editor (e.g., nano or vi)

   ```
   nano if_statement.sh
   ```

   and add the following lines to the top of the file:

   ```
   #!/bin/bash
   echo "Enter the first number:"
   read NUMBER1

   echo "Enter the second number:"
   read NUMBER2

   SUM=$((NUMBER1 + NUMBER2))
   PRODUCT=$((NUMBER1 * NUMBER2))

   echo "The sum of $NUMBER1 and $NUMBER2 is: $SUM"
   echo "The product of $NUMBER1 and $NUMBER2 is: $PRODUCT"

   if [ $SUM -gt $PRODUCT ]
   then
      echo "The sum is greater than the product."
   elif [ $SUM -lt $PRODUCT ]
   then
      echo "The product is greater than the sum."
   else
      echo "The sum and product are equal."
   fi
   ```

2. Save and close the file.
3. Give execute permissions to `if_statement.sh` with the command

   ```
   chmod u+x if_statement.sh
   ```

4. Then, run the script by typing.

   ```
   ./if_statement.sh
   ```

5. You should see either the status is sum is bigger than product or product is bigger than sum, printed to the console

3.7 Exercise 7: Loops

1. Create a new file named `loop.sh` using the touch command. Open the file in a text editor (e.g., nano or vi)

   ```
   nano loop.sh
   ```

   and add the following lines to the top of the file:

   ```
   #!/bin/bash
   echo "Counting from 1 to 10:"

   for ((i=1; i<=10; i++))
   do
      SQUARE=$((i * i))
      echo "The square of $i is: $SQUARE"
   done
   ```

2. Save and close the file.
3. Give execute permissions to `loop.sh` with the command

   ```
   chmod u+x loop.sh
   ```

4. Then, run the script by typing.

   ```
   ./loop.sh
   ```

5. You should see list of number from 1-10 and square of the numbers, printed to the console.

3.8 Exercise 8: Running Several Linux Commands

1. Create a new file named `file_management.sh` using the touch command. Open the file in a text editor (e.g., nano or vi)

   ```
   nano file_management.sh
   ```

   and add the following lines to the top of the file:

   ```
   #!/bin/bash

   # Create a new directory and navigate into it
   mkdir Exercise; cd Exercise

   # Create subdirectories: SubDir1 and SubDir1/SubSubDir
   mkdir SubDir1; mkdir SubDir1/SubSubDir

   # Navigate to the SubSubDir directory
   cd SubDir1/SubSubDir

   # Echo a message into a text file
   echo "This is some text." > file.txt

   # Display the content of the text file
   cat file.txt

   # Navigate back to the Exercise directory
   cd ../../

   # List the contents of the Exercise directory
   ls

   # Display the current working directory
   pwd
   ```

2. Save and close the file.
3. Give execute permissions to `file_management.sh` with the command

   ```
   chmod u+x file_management.sh
   ```

4. Then, run the script by typing.

   ```
   ./file_management.sh
   ```

5. You should see file management in action, printed to the console.

3.9 Exercise 9: Modify File Permission

1. Create a new file named `file_permission.sh` using the touch command. Open the file in a text editor (e.g., nano or vi)

   ```
   nano file_permission.sh
   ```

   and add the following lines to the top of the file:

   ```
   #!/bin/bash
   echo "Enter the filename:"
   read FILENAME

   echo "Enter the desired permissions (in numeric format):"
   read PERMISSIONS

   chmod $PERMISSIONS $FILENAME

   echo "Permissions for $FILENAME have been set to $PERMISSIONS."
   ```

2. Save and close the file.
3. Give execute permissions to `file_permission.sh` with the command

   ```
   chmod u+x file_permission.sh
   ```

4. Then, run the script by typing.

   ```
   ./file_permission.sh
   ```

5. You should see file permission status, printed to the console.