Operating System

Name： Yin Yuang	STU_NO： 202383930027	Exp： 1-3

Experiment1:Introduction to linux

Experimental Purpose

1.Learn the installation process of the Linux distribution (Ubuntu).
2.Understand the basic concepts of the Shell (bash).
3.Design a C program and compile/run it in the shell using the gcc compiler.

Experimental Content

1.Installation of VirtualBox and Ubuntu

Create a virtual machine using VirtualBox and complete the installation of the Ubuntu system, including disk configuration, memory allocation, and image loading.

2.Ubuntu System Configuration

Change software sources, update the system, and install necessary development tools to ensure the environment is ready for programming tasks.

3.Basic Bash Command Practice

Practice common file management, system inspection, and permission commands to become familiar with basic bash usage.

4.C Program Design and Compilation

Write a simple C program in the Ubuntu shell and use the gcc compiler to compile and execute it, understanding the basic process from source code to executable file.

Experimental Results

1.Basic Bash Command Practice

This module focuses on executing common Bash commands in the terminal to understand shell interaction logic and master directory and file operations.

1.1 Directory Operation Commands

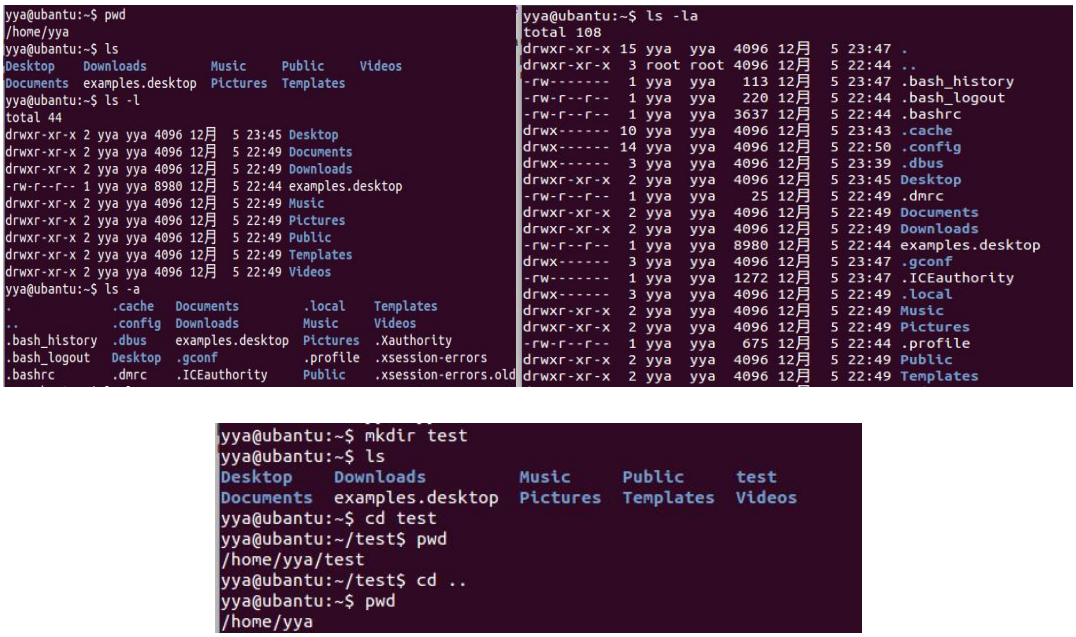
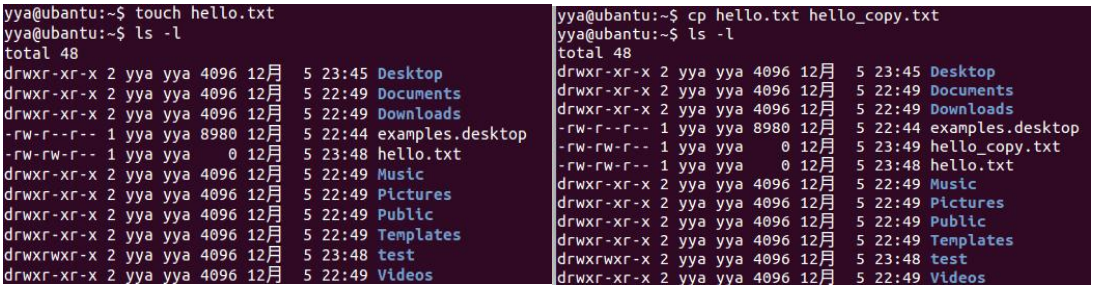


Figure1.Screenshot of Directory Operation Commands

The screenshot records a complete sequence of directory-related commands.

Command	Description
pwd	Display the full path of the current working directory.
ls	List files and directories in the current directory.
ls -l	List files and directories in long format, including permissions, owner, size, and modification time.
ls -a	List all files and directories, including hidden ones.
mkdir	Create a new directory.
cd	Change the current working directory.

1.2 File Operations and Permission Commands



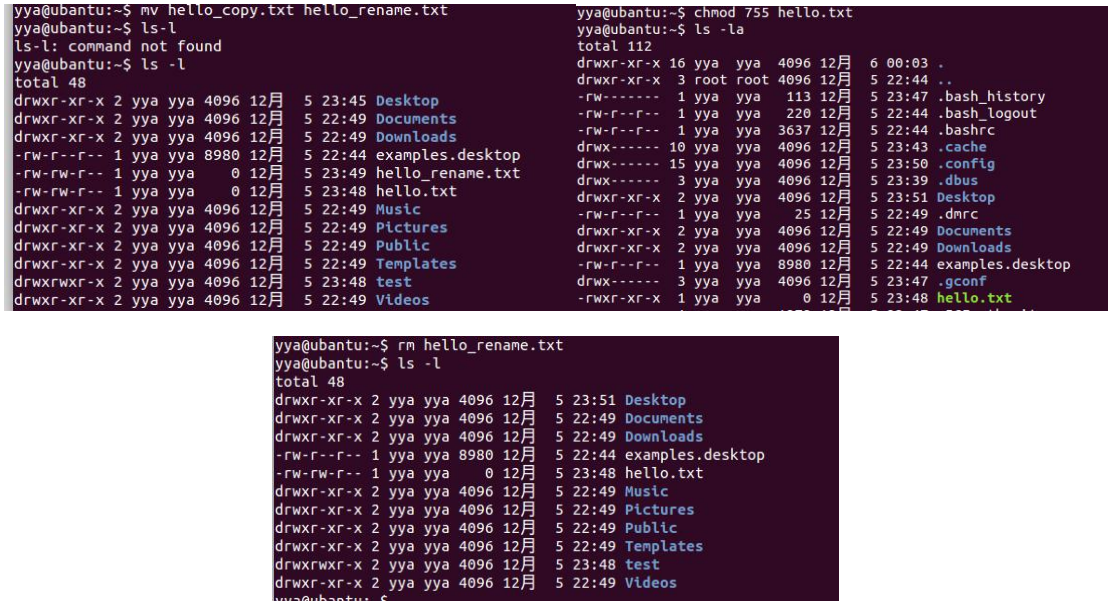


Figure2.Screenshot of File Operations and Permission Commands

The screenshot shows the execution process of file creation, copying, renaming, permission modification, and deletion, as summarized in the following table.

Command	Description & Output
touch hello.txt	Create an empty file named hello.txt.
cp hello.txt hello_copy.txt	Copy hello.txt to hello_copy.txt. ls -l shows both files with size 0 bytes.
mv hello_copy.txt hello_rename.txt	Rename hello_copy.txt to hello_rename.txt. ls -l confirms the new filename.
chmod 755 hello.txt	Change permissions of hello.txt from -rw-r--r-- to -rwxr-xr-x, adding execute permission.
rm hello_rename.txt	Delete hello_rename.txt. ls -l confirms the file is removed.

1.3 System Information Query Commands

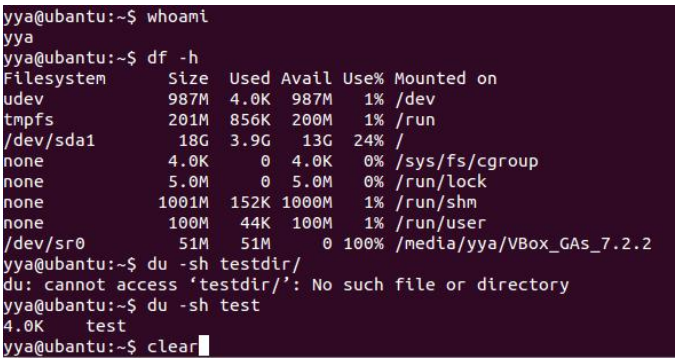


Figure3.Sreenshot of System Information Query Commands

The screenshot presents the process of executing system information query commands in the terminal, with the details summarized in the following table.

Command	Description
whoami	Display the name of the current user.

df -h	Show disk-space usage in human-readable format.
du -sh testdir/	Report the total size of testdir in a summary, human-readable form.
clear	Clear the terminal screen and present a clean prompt for subsequent commands.

2.C Program Development and Compilation

This module covers writing, compiling, and running a C program using gcc, demonstrating the full workflow from source code to executable.

2.1 Writing C Code Using nano

Figure4.Sreenshot of Editing hello.c Using nano

The screenshot shows the nano hello.c editor window. The C program includes:

(1) The header #include <stdio.h>;
(2) The definition of the main() function;
(3) A printf("Hello, Linux!\n"); statement;
(4) return 0:indicating normal program termination.

2.2 Compiling and Running the C Program

Figure5.Screenshot of Process of Compiling and Running the C Program Using gcc

The screenshot shows the complete process of compiling and running a C program in Ubuntu. First, the file hello.c is created and edited using nano. Then, the command gcc hello.c -o hello compiles the program and generates an executable named hello. Using ls -l, the directory listing confirms the presence of the source file and the new executable. Finally, running the program with ./hello outputs:Hello!

Experiment 2:Linux File System

Experimental Purpose

1.Master the three-layer structure of the Linux file system and understand the functions of each layer.
2.Familiarize yourself with key concepts of the Linux file system, such as Journaling, Versioning, and Inode.
3.Learn about common Linux file system types (ext series, JFS, XFS) and their characteristics.
4.Master the classification of Linux files and proficiently use file management commands (listing, creating, displaying, copying, moving, renaming, deleting).
5.Understand the principle of Linux file system permissions, including user permission groups and permission types, and master the use of the chmod command to modify permissions.

Experimental Content

1.Learn the structure of the Linux file system, including the Logical File System, Virtual File System (VFS), and Physical File System, and clarify the functions of each layer.
2.Understand core concepts related to the Linux file system, such as Journaling, Versioning, and Inode.
3.Study common Linux file system types and their characteristics, including ext, ext4, JFS, and XFS.
4.Master the three types of Linux files (Regular Files, Directories, Special Files) and their characteristics.
5.Practice file management commands: use ls, touch, cat, more, less, cp, mv, rm commands to complete file/directory listing, creation, content display, copying, moving, renaming, and deletion operations.
6.Learn Linux file permission mechanisms: understand user permission groups (Owner, Group, All Users) and permission types (Read, Write, Execute), and practice modifying file permissions using the chmod command.

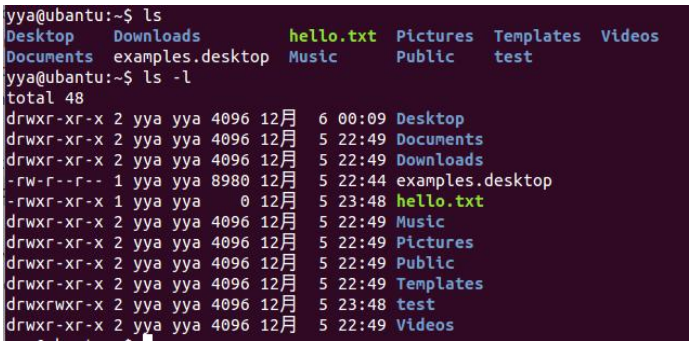
Experimental Results

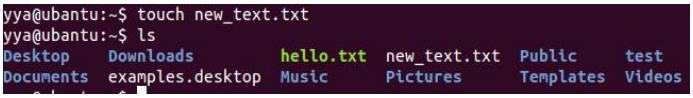
1.File Management Command Practice

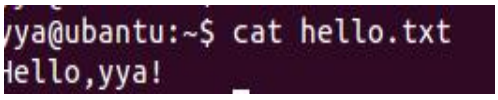
This module focuses on practicing common file management commands in the terminal to proficiently complete file/directory operation tasks.

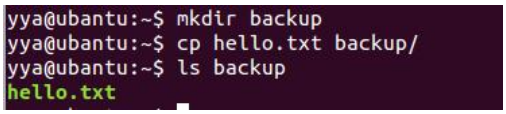
File Type	Characteristics	Creation Command
Regular Files	The most common file type, including text files, images, binary files, etc. Contains ASCII/human-readable text, executable binaries, program data, etc.	touch
Directories	Also known as folders, used to store file names and related information, maintaining a tree-like structure. The root directory is /, with common directories such as /home/ (user home directories), /bin/ (essential user binaries), and /boot/ (static boot files).	mkdir
Special Files	Represent physical devices (e.g., printers) for I/O operations. Displayed like ordinary files/directories in the file system, used for device input/output on UNIX and Linux systems.	System-generated (no manual creation command in basic operations)

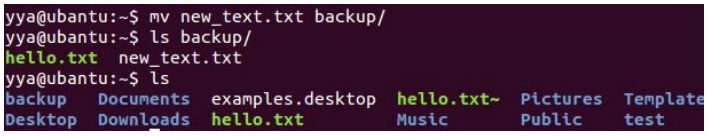
Below are the seven screenshots corresponding to the File Management Operations.











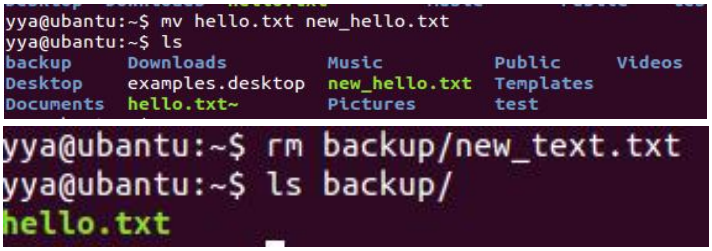


Figure1.Screenshots of the File Management Operations

File Management Operation	Command	Description
Files Listing (Basic)	ls	Lists all files and directories in the current directory. Different file types are displayed in different colors (e.g., directories in dark blue).
Files Listing (Detailed)	ls -l	Returns a detailed listing of files and directories in the current directory, including information such as file owner, group, permissions (access/execute rights for users/groups), size, and modification time.
Creating Files	touch filename	Creates an empty file with the specified filename.
Displaying File Contents	cat filename	Displays the full contents of the filename file. For large files, use more or less to fit the output on the terminal screen (avoids displaying the entire file at once).
Copying a File	cp source/filename destination/	Creates a copy of filename from the source directory to the destination directory. The copied file retains the same name and content as the original.
Moving a File	mv source/filename destination/	Moves filename from the source folder to the destination folder. The file is removed from the source and recreated with the same name and content in the destination.
Renaming a File	mv filename new_filename	Renames filename to new_filename. The original file is removed, and a new file with the same content is created under the new name.
Deleting a File	rm filename	Removes the filename file from the current dir

2. Linux File System Permissions

This module focuses on understanding Linux file permissions and performing permission modification using chmod.

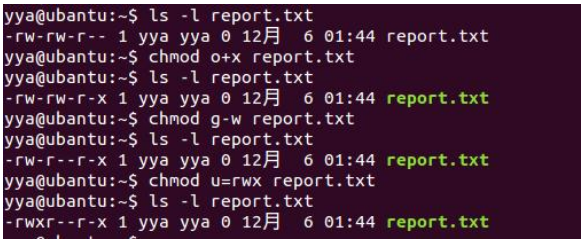


Figure2. Screenshot of Permission Modification with chmod Command

Command	Description & Output
chmod o+x report.txt	Add execute (x) permission for All Users (o) to report.txt. The permission changes from -rw-rw-r-- to -rw-rw-r-x.
chmod g-w report.txt	Remove write (w) permission for Group (g) from report.txt. The permission changes from -rw-rw-r-x to -rw-r--r-x.
chmod u=rwx report.txt	Set the Owner (u) permission of report.txt to read, write, and execute (rwx). The permission changes from -rw-r--r-x to -rwxr--r-x.

Experiment 3:Shell Script

Experimental Purpose

1.Understand the concept, function and common types of Shell scripts .
2.Master the basic structure and execution method of Shell scripts..
3.Learn to define and use scalar variables and array variables in Shell.
4.Familiarize with the usage rules of arithmetic operators, boolean operators and relational operators supported by Shell.
5.Master the writing of decision-making statements (if...else, case...esac) and loop statements (for loop) in Shell scripts.
6.Understand and apply the output redirection function in Shell.

Experimental Content

1.Basic Structure and Execution of Shell Scripts

Understanding Script Structure: A Shell script must include a declaration of the specified interpreter (e.g., #!/bin/sh or #!/bin/bash), comments (starting with #), and a sequence of executable commands.）

2.Definition and Use of Shell Variables

2.1 Scalar Variable Operation

Define scalar variables to store string and numeric values, and practice accessing and outputting variable values.

2.2 Array Variable Operation

Create array variables to store multiple values, and learn to access elements through array indexes.

3.Use of Shell Operators

3.1 Arithmetic Operator Practice

Use common arithmetic operators (addition, subtraction, multiplication, division, modulus, etc.) to perform numerical calculations, and master the escape rule of the multiplication operator.

3.2 Relational and Boolean Operator Practice

Use relational operators to compare numerical sizes and boolean operators to combine conditions, and verify the result of conditional judgments.

4.Shell Flow Control Statements

4.1 if...else Statement Practice

Write multi-condition judgment logic to determine whether the input number is positive, negative or zero.

4.2 for Loop Statement Practice

Implement two loop scenarios—traversing a list of numbers and traversing files matching a specific pattern in the home directory.

5.Output Redirection

5.1 Basic Output Redirection

Redirect the output of system commands to a specified file instead of displaying it on the terminal.

5.2 Overwrite Output Redirection

Verify that the output redirection operation will overwrite the original content of the target file.

Experimental Results

This report summarizes the results of seven shell scripting experiments, covering basic interaction, variables, arrays, arithmetic operations, conditional statements, loops, and output redirection. Each experiment was executed in the Ubuntu terminal, and the results are documented below.

1.Basic Interaction Script (read + variables)

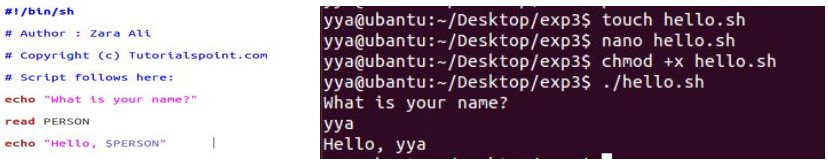


Figure1.Screenshots of hello.sh Code and Execution

2.Variable Definition and Access

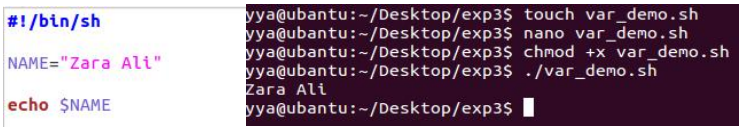


Figure2.Screenshots of var_demo.sh Code and Execution

3.Array Definition and Access

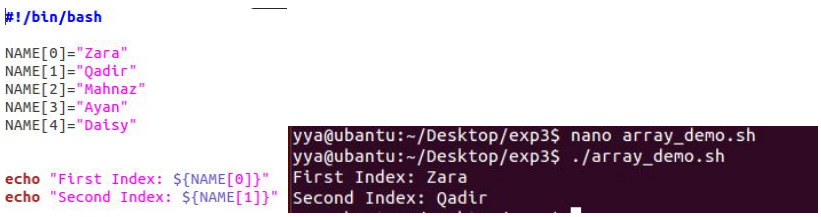
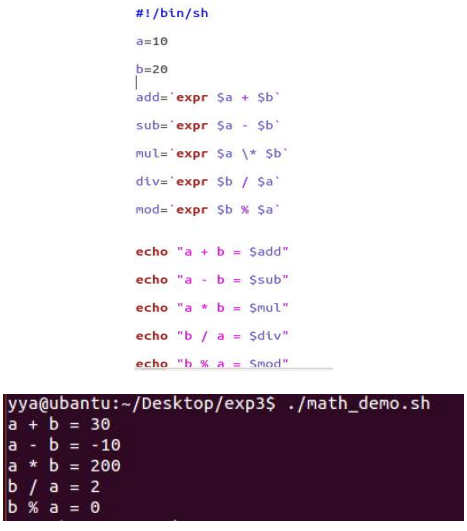


Figure3.Screenshots of var_demo.sh Code and Execution

4.Arithmetic Operations



4.Screenshot of math_demo.sh Code

5.Conditional Statements (if...else)

```
#!/bin/sh

a=10

b=20

# Relational operators: -eq (equal), -ne (not equal), -gt (greater than), -lt
(less than)

if [ $a -eq $b ]; then  # Check if a is equal to b (; then can be split into
separate lines)

    echo "$a is equal to $b"

elif [ $a -lt $b ]; then  # Else check if a is less than b

    echo "$a is less than $b"

else

    echo "$a is greater than $b"

fi  # End the if statement (mandatory)
```

```
yya@ubantu:~/Desktop/exp3$ touch conditional_demo.sh
yya@ubantu:~/Desktop/exp3$ nano conditional_demo.sh
yya@ubantu:~/Desktop/exp3$ chmod +x conditional_demo.sh
yya@ubantu:~/Desktop/exp3$ ./conditional_demo.sh
10 is less than 20
```

Figure5.Screenshot of condition_demo.sh Code

6.for Loop (List & File Traversal)

6.1 List Traversal

```
#!/bin/sh

for var in 0 1 2 3 4 5 6 7 8 9

do

    echo $var

done
```

```
yya@ubantu:~/Desktop/exp3$ touch for_num.sh
yya@ubantu:~/Desktop/exp3$ nano for_num.sh
yya@ubantu:~/Desktop/exp3$ chmod +x for_num.sh
yya@ubantu:~/Desktop/exp3$ ./for_num.sh
yya@ubantu:~/Desktop/exp3$ nano for_num.sh
yya@ubantu:~/Desktop/exp3$ chmod +x for_num.sh
yya@ubantu:~/Desktop/exp3$ ./for_num.sh
0
1
2
3
4
5
6
7
8
9
```

Figure6.Screenshot of for_num.sh Code

6.2 File Traversal

```
#!/bin/sh

for FILE in $HOME/.bash*

do

    echo $FILE

done
```

```
yya@ubantu:~/Desktop/exp3$ touch for_file.sh
yya@ubantu:~/Desktop/exp3$ nano for_file.sh
yya@ubantu:~/Desktop/exp3$ chmod +x for_file.sh
yya@ubantu:~/Desktop/exp3$ ./for_file.sh
/home/yya/.bash_history
/home/yya/.bash_logout
/home/yya/.bashrc
```

Figure7.Screenshot of for_file.sh Code

7.Output Redirection

Understand the core role of output redirection (transfer terminal output to a file), master the use of the basic redirection symbol `>`, and verify its feature of "overwriting the original content of the target file".

```
yya@ubantu:~/Desktop/exp3$ who >users.txt
yya@ubantu:~/Desktop/exp3$ cat users.txt
yya      :0           2025-12-06 11:53 (:0)
yya      pts/0        2025-12-06 12:37 (:0)
yya@ubantu:~/Desktop/exp3$ echo "This is the new added content">users.txt
yya@ubantu:~/Desktop/exp3$ cat users.txt
This is the new added content
```

Figure8.Screenshot of Redirect Commands

Through the practice of 7 core modules, this experiment has comprehensively verified the basic syntax and core functions of Shell scripting: from script structure, variable (scalar and array) operations, to operators, flow control (if condition, for loop) and output redirection, all experimental cases have been successfully executed in the Ubuntu terminal, and the results are completely consistent with expectations.