PART 2: LINUX COMMANDS AND FILE SYSTEM

2. Linux Shell

Definition - Command interpreter waiting for user to type commands from keyboard - OS executes commands and shows results back to user - Just another program written on top of OS

Different Shells - Bourne shell (sh): original shell by Steve Bourne - GNU Bourne-again shell (bash): sh with more features - C shell (csh) - Korn shell (ksh)

Used in Course - bash shell only - `$` prompt: bash waiting for command entry

3. Linux Command Structure

Parts of a Command 1. The actual command (required) 2. Command option(s) (optional, start with `-` or `--`) 3. Argument(s) (optional for some, mandatory for others)

Examples

```
$ ls
Output: file1.txt  file2.txt  directory1/
```

Command only, no option or argument

```
$ ls -l
Output: -rw-r--r-- 1 user group 1024 Oct 28 10:05 file1.txt
         -rw-r--r-- 1 user group 2048 Oct 28 10:10 file2.txt
         drwxr-xr-x 2 user group 4096 Oct 28 09:50 directory1
```

Command with option, no argument

```
$ gcc myfile.c
Output: (creates a.out if successful)
```

Command and argument, no option

```
$ gcc -Wall myfile.c
Output: (same as above, with additional warnings)
```

All three present: command, option, and argument

```
$ ls -lar
Output: total 24
        drwxr-xr-x 3 user user 4096 Oct 28 10:15 .
        drwxr-xr-x 4 root root 4096 Oct 27 09:00 ..
        -rw-r--r-- 1 user user 1024 Oct 28 09:50 .bashrc
        -rw-r--r-- 1 user user 2048 Oct 28 10:05 file1.txt
```

Command with multiple options (-l, -a, and -r)

4. Linux Directory Structure

Tree Organization - Root directory: `/` - Subdirectories within directories - Files at leaf nodes - Directory can contain subdirectories and files - Every directory contains: `.` (current) and `..` (parent) - Hidden files/directories: names starting with `.` - Home directory on login: `/home/username`

Example Directory Tree

```
/
  bin/
  boot/
  etc/
  home/
    foobar/
      my_courses/
        SysProgLab/
          Assignments/
          Materials/
          readme.txt
        AdvancedOS/
  lib/
  tmp/
  usr/
    local/
      bin/
      lib/
  var/
```

5. Identifying Files/Directories

Absolute Names (from root `/`)

```
/usr/local/lib/
/usr/local/lib/libstaque.so
/home/foobar/spl/prog/assignments/A1/src/
/home/foobar/spl/prog/assignments/A1/src/Makefile
```

Relative Names (from current directory, assume current is `/home/foobar`)

```
spl/prog/assignments/A2/myprog.c
./spl/prog/assignments/
../artim/SPL/tests/T1/questions.pdf
```

Home Directory Relative Names (using `~`)

```
~/spl/prog/assignments/A3/
~/sad/SPL/doc/T1soln.pdf
~other_user/shared/file.txt
```

6. File and Directory Permissions

Three Types of Users - Owner (u): User who owns the file - Group (g): Users in the file's group - Others (o): All other users

Three Types of Permissions - Read (r) - Write (w) - Execute (x)

Meanings for Files - Read: Can read contents - Write: Can modify contents - Execute: Can run as program

Meanings for Directories - Read: Can read contents (ls command) - With only read permission, cannot access files in directory - Write: Can create new files in directory - Execute: Can go to directory, open/execute files in directory (if you know names) - With only execute permission, cannot see directory contents

Permission Examples

```
rwxr-xr-x
  Owner (u): rwx - read, write, execute
  Group (g): r-x - read, execute
  Others (o): r-x - read, execute

rw-r--r--
  Owner (u): rw- - read, write
  Group (g): r-- - read
  Others (o): r-- - read

rwx------
  Owner (u): rwx - full permissions
  Group (g): --- - no permissions
  Others (o): --- - no permissions
```

PART 3: FILE AND DIRECTORY ORGANIZATION COMMANDS

cd - Change Directory

Syntax

```
cd <dirname>
```

Description Changes current working directory to directory named `<dirname>`. Name can be absolute or relative.

Examples

```
$ pwd
/home/foobar
```

```
$ cd /usr/local/bin
$ pwd
/usr/local/bin

$ cd SysProgLab
$ pwd
/usr/local/bin/SysProgLab

$ cd ..
$ pwd
/usr/local/bin

$ cd ~
$ pwd
/home/foobar

$ cd -
$ pwd
/usr/local/bin
```

Goes back to previous directory

pwd - Print Working Directory

Syntax

`pwd`

Description Shows current working directory (full absolute path).

Examples

```
$ pwd
/home/foobar

$ cd /usr
$ pwd
/usr

$ cd /usr/local/bin
$ pwd
/usr/local/bin
```

mkdir - Make Directory

Syntax

```
mkdir <dirname>
mkdir -p <path/to/nested/directories>
```

Description Creates directory. You need write permission in parent directory.

Option: -p Creates parent directories as needed. Creates entire path if it doesn't exist.

Examples

```
$ mkdir mynewdir
$ ls -l
drwxr-xr-x  2  user  group  4096  Oct 28 10:05  mynewdir
```

```
$ mkdir nested/dir/structure
mkdir: cannot create directory 'nested/dir/structure': No such file or directory
```

```
$ mkdir -p nested/dir/structure
$ pwd
/home/user/nested/dir/structure
```

```
$ mkdir -p my_courses/SysProgLab/Assignments
$ mkdir -p my_courses/SysProgLab/Materials
$ ls -lR my_courses/
my_courses/:
drwxr-xr-x  SysProgLab/

my_courses/SysProgLab/:
drwxr-xr-x  Assignments/
drwxr-xr-x  Materials/
```

rmdir - Remove Directory

Syntax

```
rmdir <dirname>
```

Description Removes directory (must be empty). You need write permission in parent.

Examples

```
$ rmdir emptydir
$ ls
directory1/  directory2/
```

emptydir is gone

```
$ rmdir dirwithfiles
rmdir: failed to remove 'dirwithfiles': Directory not empty
```

cp - Copy File/Directory

Syntax

```
cp <file1> <file2>
cp -r <source_dir> <dest_dir>
cp -f <file> <dest>
```

Options - `-r`: Recursively copy entire directory tree - `-f`: Force overwrite without asking

Examples

```
$ ls
file1.txt  file2.txt

$ cp file1.txt file1_backup.txt
$ ls
file1.txt  file1_backup.txt  file2.txt
```

```
$ cp file1.txt /tmp/
$ ls /tmp/
file1.txt
```

```
$ ls
mydir/  myfile.txt

$ cp -r mydir/ mydir_backup/
$ ls
mydir/  mydir_backup/  myfile.txt

$ ls mydir_backup/
(same contents as mydir/)
```

```
$ cp file1.txt file2.txt file3.txt ./targetdir/
$ ls ./targetdir/
file1.txt  file2.txt  file3.txt
```

mv - Move/Rename File

Syntax

```
mv <file1> <file2>
mv <file> <directory>
mv <dir1> <dir2>
```

Description Moves or renames files. Can move to different directory or rename in same directory.

Examples

```
$ ls
oldname.txt

$ mv oldname.txt newname.txt
$ ls
newname.txt
```

```
$ ls
newname.txt

$ mv newname.txt /tmp/
$ ls /tmp/
newname.txt
```

```
$ cd /tmp
$ ls
newname.txt

$ mv newname.txt .
$ pwd
/tmp
$ ls
newname.txt
```

```
$ ls
file1.txt  file2.txt  file3.txt  targetdir/

$ mv file1.txt file2.txt file3.txt ./targetdir/
$ ls
targetdir/

$ ls ./targetdir/
file1.txt  file2.txt  file3.txt
```

```
$ ls
dir1/  dir2/

$ mv dir1/ newdirname/
$ ls
newdirname/  dir2/
```

rm - Remove File

Syntax

```
rm <file1> <file2> ...
rm -i <file>
rm -r <directory>
rm -d <empty_dir>
```

Options - `-i`: Interactive - asks for confirmation before deletion - `-r`: Recursive - delete entire directory tree - `-d`: Delete empty directory

Examples

```
$ ls
file1.txt  file2.txt  dir1/

$ rm file1.txt
$ ls
file2.txt  dir1/
```

```
$ ls
file2.txt

$ rm -i file2.txt
remove file2.txt? y
$ ls
(empty)
```

```
$ ls
file2.txt

$ rm -i file2.txt
remove file2.txt? n
$ ls
file2.txt
```

File not deleted

```
$ ls
dir1/  dir2/

$ rm -r dir1/
$ ls
dir2/
```

```
$ ls
dir_with_files/  (contains file1.txt and file2.txt)

$ rm -r dir_with_files/
$ ls
(empty)
```

PART 4: LISTING FILES AND DIRECTORIES

ls - List Directory Contents

Syntax

```
ls [options] [file/directory]
```

Options

Option	Meaning
`-l`	Long listing (detailed format)
`-a`	Show hidden files (names starting with .)
`-R`	Recursively list all subdirectories
`-t`	Sort by modification time (newest first)
`-r`	Reverse sorting order
`-d`	List directory itself, not its contents

Examples

```
$ ls
AdvancedOS/  Materials/  readme.txt  SysProgLab/
```

Basic listing

```
$ ls -a
.  ..  .bashrc  .profile  AdvancedOS/  Materials/  readme.txt  SysProgLab/
```

Shows hidden files (`.bashrc`, `.profile`)

```
$ ls -l
total 20
drwxr-xr-x  2  user  group  4096  Oct 28 19:46  AdvancedOS/
drwxr-xr-x  2  user  group  4096  Oct 28 19:48  Materials/
-rw-r--r--  1  user  group    47  Oct 28 19:47  readme.txt
drwxr-xr-x  2  user  group  4096  Oct 28 19:46  SysProgLab/
```

Long detailed listing

```
$ ls -l *.txt
-rw-r--r--  1  user  group  47  Oct 28 19:47  readme.txt
```

List specific pattern

```
$ ls -ld SysProgLab/
drwxr-xr-x  2  user  group  4096  Oct 28 19:46  SysProgLab/
```

List directory itself, not contents

```
$ ls -lR
AdvancedOS/:
total 8
```

```
-rw-r--r-- file1.txt

Materials/:
total 12
-rw-r--r-- list.txt
-rw-r--r-- readme.txt

SysProgLab/:
total 16
-rw-r--r-- assignment1.txt
-rw-r--r-- assignment2.txt
```

Recursive listing

```
$ ls -lart
total 24
-rw-r--r-- 1 user group  47 Oct 28 19:46 readme.txt
drwxr-xr-x 2 user group 4096 Oct 28 19:47 Materials/
drwxr-xr-x 2 user group 4096 Oct 28 19:48 AdvancedOS/
drwxr-xr-x 2 user group 4096 Oct 28 19:50 SysProgLab/
```

Long listing, all files, reverse order, sorted by time (oldest first)

Explanation of ls -l Output

```
-rw-r--r--  1  user  group  47  Oct 28 19:47  readme.txt
```

Component	Meaning
`-`	Type: `-` = regular file, `d` = directory
`rw-r--r--`	Permissions (user-group-others)
`1`	Number of hard links
`user`	Owner username
`group`	Group name
`47`	File size in bytes
`Oct 28 19:47`	Last modification date/time
`readme.txt`	Filename

cat - Concatenate and Print Files

Syntax

```
cat <file1> <file2> ...
```

Description Prints contents of files to screen. Concatenates multiple files.

Examples

```
$ cat readme.txt
This is the readme file.
```

```
It contains important information.
Please read carefully.
```

```
$ cat file1.txt file2.txt
Contents of file1
More contents of file1

Contents of file2
More contents of file2
```

```
$ cat > newfile.txt
hello
world
Ctrl+D
$ cat newfile.txt
hello
world
```

head - Print Beginning of File

Syntax

```
head [options] <file>
```

Options

Option	Meaning
`-n N`	Print first N lines (default 10)
`-c N`	Print first N bytes

Examples

```
$ cat readme.txt
Line 1
Line 2
Line 3
Line 4
Line 5
Line 6
Line 7
Line 8
Line 9
Line 10
Line 11
Line 12

$ head readme.txt
```

```
Line 1
Line 2
Line 3
Line 4
Line 5
Line 6
Line 7
Line 8
Line 9
Line 10
```

```
$ head -n 3 readme.txt
Line 1
Line 2
Line 3
```

```
$ head -c 50 readme.txt
Line 1
Line 2
Line 3
Line 4
Line
```

tail - Print End of File

Syntax

```
tail [options] <file>
```

Options

Option	Meaning
`-n N`	Print last N lines (default 10)
`-c N`	Print last N bytes

Examples

```
$ cat readme.txt
Line 1
Line 2
Line 3
Line 4
Line 5
Line 6
Line 7
Line 8
Line 9
Line 10
Line 11
Line 12

$ tail readme.txt
Line 3
Line 4
Line 5
Line 6
Line 7
Line 8
Line 9
Line 10
Line 11
Line 12
```

```
$ tail -n 3 readme.txt
Line 10
Line 11
Line 12
```

```
$ tail -c 20 readme.txt
e 10
Line 11
Line 12
```

more - Page-by-Page Display

Syntax

```
more <file>
```

Description Shows file one screen at a time. Waits for user interaction.

Navigation Keys - Space: Next page - Enter: Next line - q: Quit - /pattern: Search for pattern - n: Find next search result

Example

```
$ more largefile.txt
(displays first screen)
(press space for next screen)
```

PART 5: FILE COMPARISON AND INFORMATION COMMANDS

wc - Word/Character/Line Count

Syntax

```
wc [options] <file>
```

Options

Option	Meaning
`-l`	Count lines only
`-w`	Count words only
`-c`	Count bytes only
`-m`	Count characters

Default Output: `lines words bytes filename`

Examples

```
$ cat readme.txt
This is line one
This is line two
This is line three

$ wc readme.txt
3 9 47 readme.txt
```

3 lines, 9 words, 47 bytes

```
$ wc -l readme.txt
3 readme.txt
```

```
$ wc -w readme.txt
9 readme.txt
```

```
$ wc -c readme.txt
47 readme.txt
```

```
$ ls *.txt
file1.txt  file2.txt  file3.txt

$ wc -l *.txt
10 file1.txt
15 file2.txt
8 file3.txt
33 total
```

diff - Compare Files Line by Line

Syntax

```
diff [options] <file1> <file2>
```

Options

Option	Meaning
`-y`	Side-by-side comparison
`-u`	Unified format (shows context)

Notation - `<`: Lines in file1 - `>`: Lines in file2 - `d`: delete line - `a`: add line - `c`: change line

Examples

```
$ cat file1.txt
apple
banana
cherry
date

$ cat file2.txt
apple
blueberry
cherry
date

$ diff file1.txt file2.txt
2c2
< banana
---
> blueberry
```

Line 2: change banana to blueberry

```
$ diff -y file1.txt file2.txt
apple                           apple
banana                        |  blueberry
cherry                          cherry
date                            date
```

PART 6: FILE PERMISSIONS

chmod - Change File Permissions

Syntax

```
chmod <mode> <file/directory>
```

Two Methods

1. Symbolic Mode

```
chmod [who][+/-][permission] file
```

- **who**: `u` (user/owner), `g` (group), `o` (others), `a` (all)
- **+/-**: `+` (add), `-` (remove)
- **permission**: `r` (read), `w` (write), `x` (execute)

2. Numeric Mode

```
chmod [octal] file
```

Number	Permission
4	read (r)
2	write (w)
1	execute (x)

Combine for each category: user-group-others

Examples

```
$ ls -l file.txt
-rw-r--r-- file.txt

$ chmod g+x file.txt
$ ls -l file.txt
-rw-r-xr-- file.txt

$ chmod o-r file.txt
$ ls -l file.txt
-rw-r-x--- file.txt

$ chmod 755 file.txt
$ ls -l file.txt
-rwxr-xr-x file.txt

$ chmod 644 file.txt
$ ls -l file.txt
-rw-r--r-- file.txt
```

```
$ chmod 600 file.txt
$ ls -l file.txt
-rw------- file.txt
```

```
$ chmod 700 file.txt
$ ls -l file.txt
-rwx------ file.txt
```

```
$ chmod u+x script.sh
$ ls -l script.sh
-rwxr--r-- script.sh
```

Add execute for owner

```
$ chmod a+w file.txt
$ ls -l file.txt
-rw-rw-rw- file.txt
```

Add write for everyone

Permission Conversion Table

```
rwx = 4+2+1 = 7  (read, write, execute)
rw- = 4+2 = 6    (read, write)
r-x = 4+1 = 5    (read, execute)
r-- = 4 = 4      (read only)
-wx = 2+1 = 3    (write, execute)
-w- = 2 = 2      (write only)
--x = 1 = 1      (execute only)
--- = 0 = 0      (no permissions)

Examples:
755 = rwxr-xr-x  (owner: all, group: rx, others: rx)
644 = rw-r--r--  (owner: rw, group: r, others: r)
700 = rwx------  (owner: all, group: none, others: none)
777 = rwxrwxrwx  (everyone: all permissions)
600 = rw-------  (owner: rw, others: none)
```

PART 7: WILDCARDS AND PATTERN MATCHING

Wildcard Characters

* (Asterisk) - Any Character Sequence

Matches any sequence of characters (including no characters).

Examples

```
$ ls *.txt
file1.txt  file2.txt  readme.txt
```

```
$ ls *.c
program.c  util.c  test.c

$ ls start*
start_file.txt  startup.sh  started.log

$ cat *.txt
```

Prints all .txt files

`?` (Question Mark) - Single Character

Matches exactly ONE character.

Examples

```
$ ls ?.txt
a.txt  b.txt  z.txt

$ ls ???.c
main.c  util.c  foo.c

$ ls file?.txt
file1.txt  file2.txt
```

Does NOT match file10.txt or file.txt

`[...]` (Bracket Expression) - Character Range

Matches any single character within brackets.

Examples

```
$ ls file[1-3].txt
file1.txt  file2.txt  file3.txt

$ ls [a-c].txt
a.txt  b.txt  c.txt

$ ls [[:digit:]]*.txt
0data.txt  1data.txt  9notes.txt

$ ls [[:alpha:]]*
apple.txt  books.pdf  config.sh
```

PART 8: REDIRECTION AND PIPING

Input Redirection (<)

Syntax

```
command < input_file
```

Description Command reads input from file instead of keyboard.

Use Case Instead of typing large inputs every time, put them in file and redirect.

Examples

```
$ cat > input.txt
10
20
30
Ctrl+D

$ ./calculate < input.txt
30
40
50
```

Program reads from input.txt instead of keyboard

```
$ grep "pattern" < file.txt
pattern1 found
pattern2 found
```

Output Redirection (>)

Syntax

```
command > output_file
command >> output_file
```

Description - `>`: Overwrites file if it exists, creates if it doesn't - `>>`: Appends to file if it exists, creates if it doesn't

Examples

```
$ ls -l > file_list.txt
$ cat file_list.txt
total 12
-rw-r--r-- 1 user group 1024 Oct 28 09:50 file1.txt
-rw-r--r-- 1 user group 2048 Oct 28 10:05 file2.txt
drwxr-xr-x 2 user group 4096 Oct 28 10:10 directory1
```

Nothing printed on screen

```
$ echo "First line" > output.txt
$ cat output.txt
First line

$ echo "Second line" >> output.txt
$ cat output.txt
First line
Second line
```

```
$ date > timestamp.txt
$ cat timestamp.txt
Tue Oct 28 10:15:30 IST 2025
```

Piping (|)

Syntax

```
command1 | command2
```

Description Output of command1 becomes input to command2.

Examples

```
$ ls -l | wc -l
15
```

Counts number of lines in ls output

```
$ cat readme.txt | grep "important"
This is an important file
```

```
$ ls -l | grep "\.txt$"
-rw-r--r-- readme.txt
-rw-r--r-- notes.txt
```

```
$ cat data.txt | sort | uniq
apple
banana
cherry
```

```
$ ps aux | grep bash
user  12345  0.0  0.1  4124  2048 pts/0 S+  10:15  0:00 bash
user  12346  0.0  0.1  4124  2048 pts/1 S+  10:20  0:00 bash
```

Complex Piping

```
$ cat input.txt | head -n 5 | wc -l
5
```

Gets first 5 lines, counts them

```
$ ls -l | grep "\.c$" | wc -l
3
```

Counts C source files

PART 9: USING MANPAGES

man Command

Syntax

```
man <command>
man <section> <command>
```

Description Displays manual pages for commands and functions.

Sections

Section	Type
1	Commands (default)
2	System calls
3	Library functions
4	Device files
5	File formats

Examples

```
$ man ls
(opens manual page for ls command)
```

```
$ man 3 printf
(opens manual for printf() function, section 3)
```

```
$ man printf
(opens manual for printf command, section 1)
```

```
$ man -k permission
chgrp (1)           - change group ownership
chmod (1)           - change file mode bits
(searches for pages related to "permission")
```

Navigation in man - Space/Page Down: Next page - b/Page Up: Previous page - q: Quit - /pattern: Search - n: Next search result

---

PART 10: PRACTICE EXERCISES FROM PDF

Exercise 1: Create Directory Structure

Objective: Create directory tree and verify

Directory Structure to Create:

```
my_courses/
  readme.txt
  SysProgLab/
     Assignments/
        Assgn-1.txt
        Assgn-2.txt
     Materials/
        Slides/
     Marks/
  AdvancedOS/
  additional/
      AGT/
```

Commands:

```
$ mkdir -p my_courses/SysProgLab/Assignments
$ mkdir -p my_courses/SysProgLab/Materials/Slides
$ mkdir -p my_courses/SysProgLab/Marks
$ mkdir -p my_courses/AdvancedOS
$ mkdir -p my_courses/additional/AGT

$ echo "readme content" > my_courses/readme.txt
$ echo "Assignment 1" > my_courses/SysProgLab/Assignments/Assgn-1.txt
$ echo "Assignment 2" > my_courses/SysProgLab/Assignments/Assgn-2.txt

$ ls -lR my_courses/
my_courses/:
-rw-r--r-- readme.txt
drwxr-xr-x AdvancedOS/
drwxr-xr-x SysProgLab/
drwxr-xr-x additional/

my_courses/AdvancedOS:
(empty)

my_courses/SysProgLab:
drwxr-xr-x Assignments/
drwxr-xr-x Materials/
drwxr-xr-x Marks/

my_courses/SysProgLab/Assignments:
-rw-r--r-- Assgn-1.txt
-rw-r--r-- Assgn-2.txt

my_courses/SysProgLab/Materials:
```

```
drwxr-xr-x Slides/

my_courses/SysProgLab/Materials/Slides:
(empty)

my_courses/SysProgLab/Marks:
(empty)

my_courses/additional:
drwxr-xr-x AGT/

my_courses/additional/AGT:
(empty)
```

Exercise 2: List Contents with Different Options

```
$ ls my_courses/
AdvancedOS/  SysProgLab/  additional/  readme.txt

$ ls -l my_courses/
drwxr-xr-x 2 user group 4096 Oct 28 10:10 AdvancedOS/
-rw-r--r-- 1 user group   15 Oct 28 10:15 readme.txt
drwxr-xr-x 4 user group 4096 Oct 28 10:20 SysProgLab/
drwxr-xr-x 2 user group 4096 Oct 28 10:25 additional/

$ ls -lR my_courses/ | head -n 20
my_courses/:
drwxr-xr-x 2 user group 4096 Oct 28 10:10 AdvancedOS/
-rw-r--r-- 1 user group   15 Oct 28 10:15 readme.txt
drwxr-xr-x 4 user group 4096 Oct 28 10:20 SysProgLab/
drwxr-xr-x 2 user group 4096 Oct 28 10:25 additional/

my_courses/AdvancedOS:
total 0

my_courses/SysProgLab:
drwxr-xr-x 2 user group 4096 Oct 28 10:20 Assignments/
drwxr-xr-x 2 user group 4096 Oct 28 10:20 Materials/
drwxr-xr-x 2 user group 4096 Oct 28 10:20 Marks/
```

Exercise 3: File Operations

```
$ cp my_courses/readme.txt my_courses/SysProgLab/Materials/readme-copy.txt

$ ls my_courses/SysProgLab/Materials/
readme-copy.txt  Slides/
```

```
$ cat my_courses/readme.txt
readme content

$ head -n 1 my_courses/readme.txt
readme content

$ tail -n 1 my_courses/readme.txt
readme content

$ wc my_courses/SysProgLab/Materials/readme-copy.txt
1 2 15 my_courses/SysProgLab/Materials/readme-copy.txt
```

Exercise 4: File Comparison

```
$ echo "modified content here" >> my_courses/SysProgLab/Materials/readme-copy.txt

$ wc my_courses/SysProgLab/Materials/readme-copy.txt
2 5 36 my_courses/SysProgLab/Materials/readme-copy.txt

$ diff my_courses/readme.txt my_courses/SysProgLab/Materials/readme-copy.txt
1a2
> modified content here
```

Exercise 5: Concatenation and Redirection

```
$ cat my_courses/SysProgLab/Assignments/Assgn-1.txt \
      my_courses/SysProgLab/Assignments/Assgn-2.txt > \
      my_courses/SysProgLab/Materials/Assgns.txt

$ cat my_courses/SysProgLab/Materials/Assgns.txt
Assignment 1
Assignment 2

$ ls -l my_courses | wc -l
5
(output includes total line + 4 items)

$ head -n 1 my_courses/readme.txt
readme content
```

Exercise 6: Copy Directories

```
$ cp -r my_courses/SysProgLab my_courses/SysProgLab-Copy

$ ls my_courses/
AdvancedOS/  SysProgLab/  SysProgLab-Copy/  additional/  readme.txt
```

```
$ ls my_courses/SysProgLab-Copy/
Assignments/  Materials/  Marks/

$ ls my_courses/SysProgLab-Copy/Assignments/
Assgn-1.txt  Assgn-2.txt
```

Exercise 7: Permission Changes

Create test directory

```
$ mkdir my_courses/test_perms

$ ls -ld my_courses/test_perms/
drwxr-xr-x 2 user group 4096 Oct 28 11:00 test_perms/
```

Test Case 1: Only Read and Execute

```
$ chmod u=rx my_courses/test_perms/
$ ls -ld my_courses/test_perms/
dr-xr-xr-x 2 user group 4096 Oct 28 11:00 test_perms/

$ cd my_courses/test_perms/
$ pwd
my_courses/test_perms

$ mkdir newdir
mkdir: cannot create directory 'newdir': Permission denied
```

Can go to directory (execute) and see contents (read), but cannot create files/directories (no write)

Test Case 2: Only Read and Write

```
$ chmod u=rw my_courses/test_perms/
$ ls -ld my_courses/test_perms/
drw-r--r-x 2 user group 4096 Oct 28 11:00 test_perms/

$ cd my_courses/test_perms/
bash: cd: my_courses/test_perms/: Permission denied
```

Cannot go to directory (no execute), so cannot access anything

Test Case 3: Only Read

```
$ chmod u=r my_courses/test_perms/
$ ls -ld my_courses/test_perms/
dr--r--r-x 2 user group 4096 Oct 28 11:00 test_perms/

$ ls my_courses/test_perms/
```

```
(shows contents, but no write or execute)

$ cd my_courses/test_perms/
bash: cd: my_courses/test_perms/: Permission denied
```

Test Case 4: Only Execute

```
$ chmod u=x my_courses/test_perms/
$ ls -ld my_courses/test_perms/
d--x--x--x 2 user group 4096 Oct 28 11:00 test_perms/

$ ls my_courses/test_perms/
ls: cannot open directory 'my_courses/test_perms/': Permission denied
```

Can go to directory but cannot see contents

Test Case 5: Only Write

```
$ chmod u=w my_courses/test_perms/
$ ls -ld my_courses/test_perms/
d-w------- 2 user group 4096 Oct 28 11:00 test_perms/

$ cd my_courses/test_perms/
bash: cd: my_courses/test_perms/: Permission denied

$ ls my_courses/test_perms/
ls: cannot open directory 'my_courses/test_perms/': Permission denied
```

Revert to rwx

```
$ chmod u=rwx my_courses/test_perms/
$ ls -ld my_courses/test_perms/
drwxr--r-x 2 user group 4096 Oct 28 11:00 test_perms/
```

Exercise 8: Numeric Permission Mode

```
$ chmod 755 my_courses/readme.txt
$ ls -l my_courses/readme.txt
-rwxr-xr-x user group readme.txt

$ chmod 644 my_courses/readme.txt
$ ls -l my_courses/readme.txt
-rw-r--r-- user group readme.txt

$ chmod 700 my_courses/additional/
$ ls -ld my_courses/additional/
drwx------ user group additional/

$ chmod 777 my_courses/
$ ls -ld my_courses/
drwxrwxrwx user group my_courses/
```

Exercise 9: C Program with Execute Permission

Create program

```
$ cat > hello.c << EOF
#include <stdio.h>
int main() {
    printf("Hello World\n");
    return 0;
}
EOF

$ gcc hello.c -o hello

$ ls -l hello
-rwxr-xr-x user group hello
```

Execute permission already present (from gcc)

Run program

```
$ ./hello
Hello World
```

Remove execute permission

```
$ chmod u-x hello
$ ls -l hello
-rw-r--r-- user group hello

$ ./hello
bash: ./hello: Permission denied
```

Restore execute permission

```
$ chmod u+x hello
$ ./hello
Hello World
```

PART 11: REQUIRED COMMANDS AND OPTIONS SUMMARY

Commands to Know

File/Directory Organization

```
cd <dirname>
pwd
mkdir <dirname>
mkdir -p <path>
rmdir <dirname>
cp <file1> <file2>
cp -r <source> <dest>
cp -f <file> <dest>
mv <file1> <file2>
rm <file>
rm -i <file>
rm -r <directory>
rm -d <empty_dir>
```

Listing Contents

```
ls
ls -l
ls -a
ls -R
ls -t
ls -r
ls -d
cat <file>
head <file>
head -n N <file>
head -c N <file>
tail <file>
tail -n N <file>
tail -c N <file>
more <file>
```

Permissions

```
chmod u+r <file>
chmod u-w <file>
chmod g+x <file>
chmod o-rwx <file>
chmod a+x <file>
chmod 755 <file>
chmod 644 <file>
chmod 700 <file>
chmod 777 <file>
```

Comparison/Information

```
wc <file>
wc -l <file>
wc -w <file>
wc -c <file>
wc -m <file>
diff <file1> <file2>
diff -y <file1> <file2>
```

Wildcards

```
* (any sequence)
? (single character)
[...] (character range)
```

Redirection

```
< (input redirection)
> (output redirect - overwrite)
>> (output redirect - append)
| (pipe)
```

Manual Pages

```
man <command>
man <section> <command>
man -k <keyword>
```