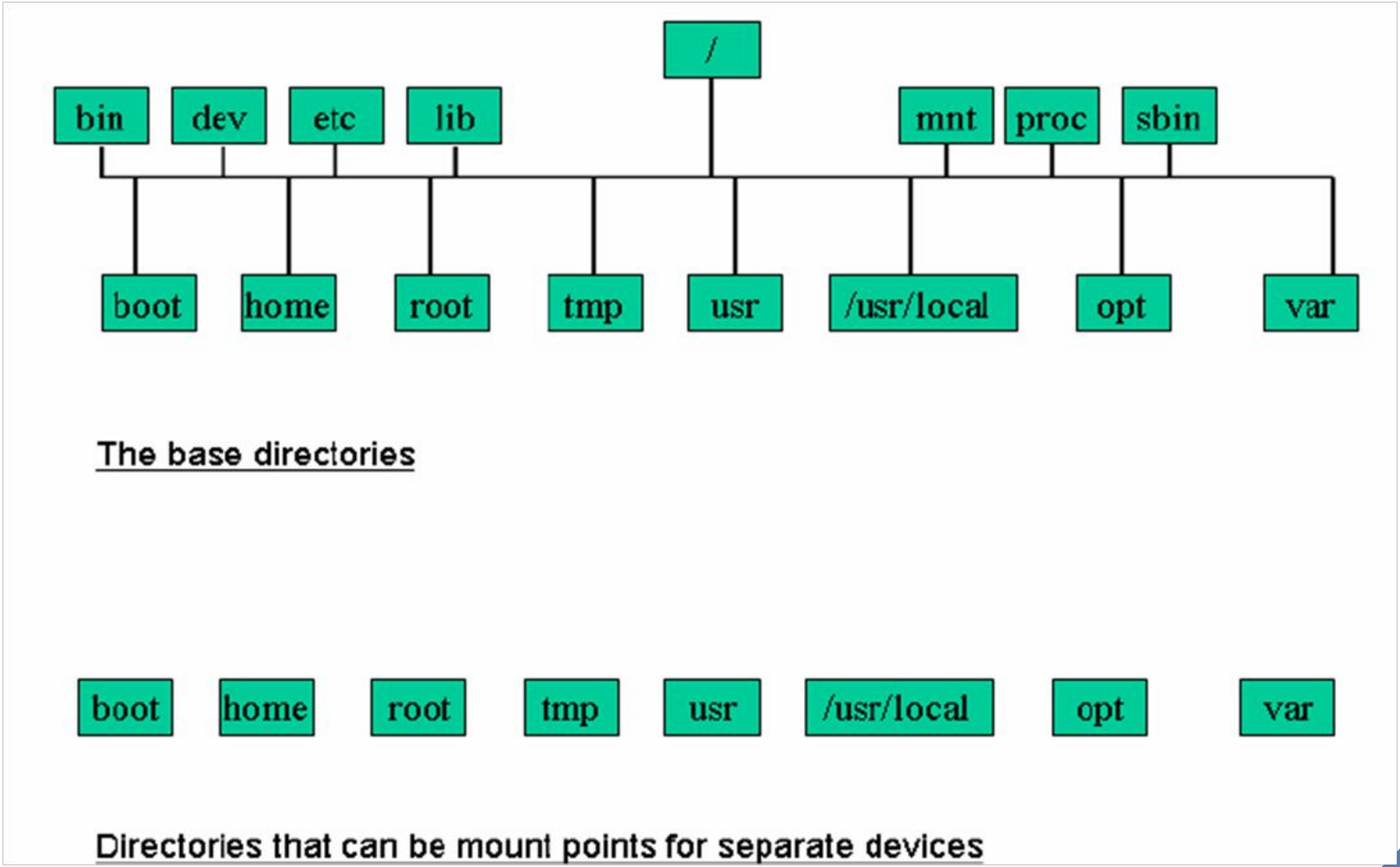
Linux Operating System and Applications

Command Line Basics

Linux Directory Structure



Linux Directory Structure

Directory	Description
`/boot`	Kernel and boot configuration files
`/bin`	Essential user commands (binaries)
`/dev`	Device files (hardware interfaces)
`/etc`	System and application configuration files
`/home`	User home directories
`/lib`	Shared libraries required by binaries
`/mnt`	Mount point for temporary filesystems
`/proc`	Virtual filesystem for process and system info
`/sbin`	System administration commands
`/tmp`	Temporary files
`/usr`	User applications and libraries
`/var`	Variable data (e.g., logs, caches, spool files)

File Naming Conventions in Linux

- **Maximum length** for a single file name is **255 characters** in most Linux file systems (like ext4).
- Linux **allows almost any character** in file names, including **special characters** such as `?`, `-`, `+`, and spaces.
- File names **can include** letters, numbers, dots (`.`), underscores (`_`), and hyphens (`-`).
- File names are **case-sensitive** (`File.txt` ≠ `file.txt`).
- File names **can include extensions**, but they are not required or enforced by the system (e.g., `.txt`, `.sh`).
- File names **can contain spaces**, but it's better to avoid them. Use underscores (`_`) or hyphens (`-`) instead.
- Hidden files (and directories) start with a dot "." Example: `.bash_history`

Linux Directory Paths

Absolute Path

- Starts with `/`
- Full path from root
- Examples: `/`, `/bin`, `/usr`, `/usr/bin`

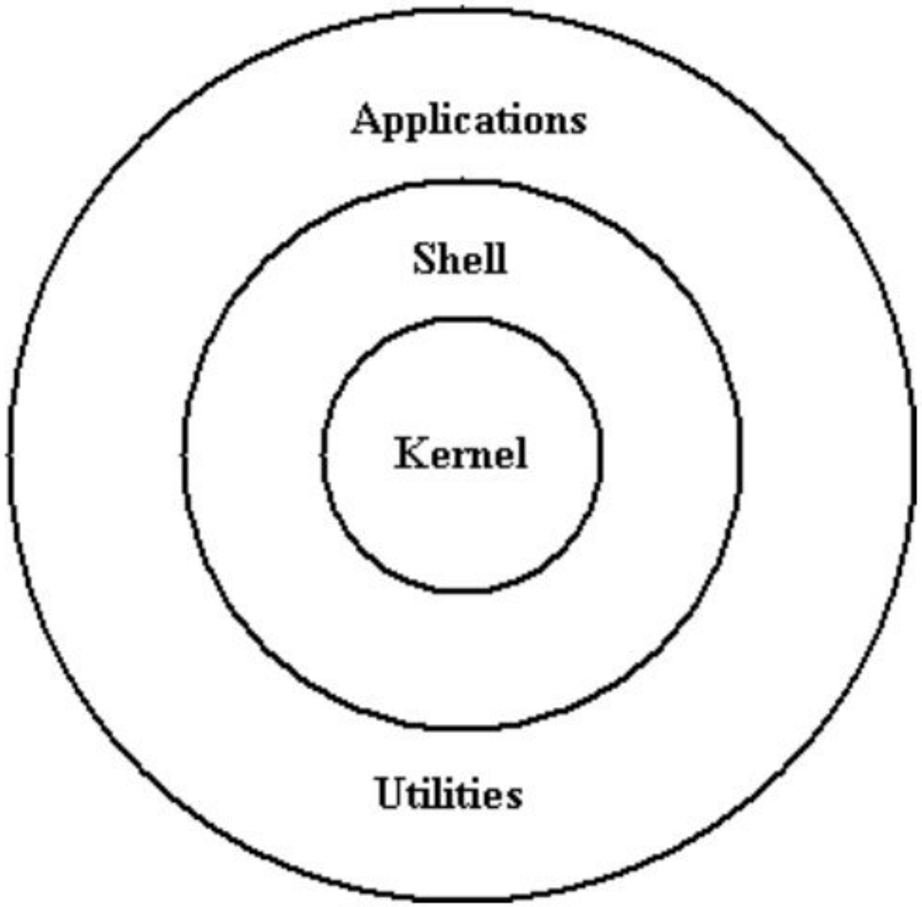
Relative Path

- Does *not* start with `/`
- Relative to current directory
- Examples: `etc/httpd/`, `usr/bin`

Special Notations

- `..` — Parent directory
- `.` — Current directory
- `~` — User’s home directory
- **Example:** If current directory is `/etc`, the relative path to `/etc/vsftp.conf` is `./vsftp.conf`

Unix/Linux System Architecture



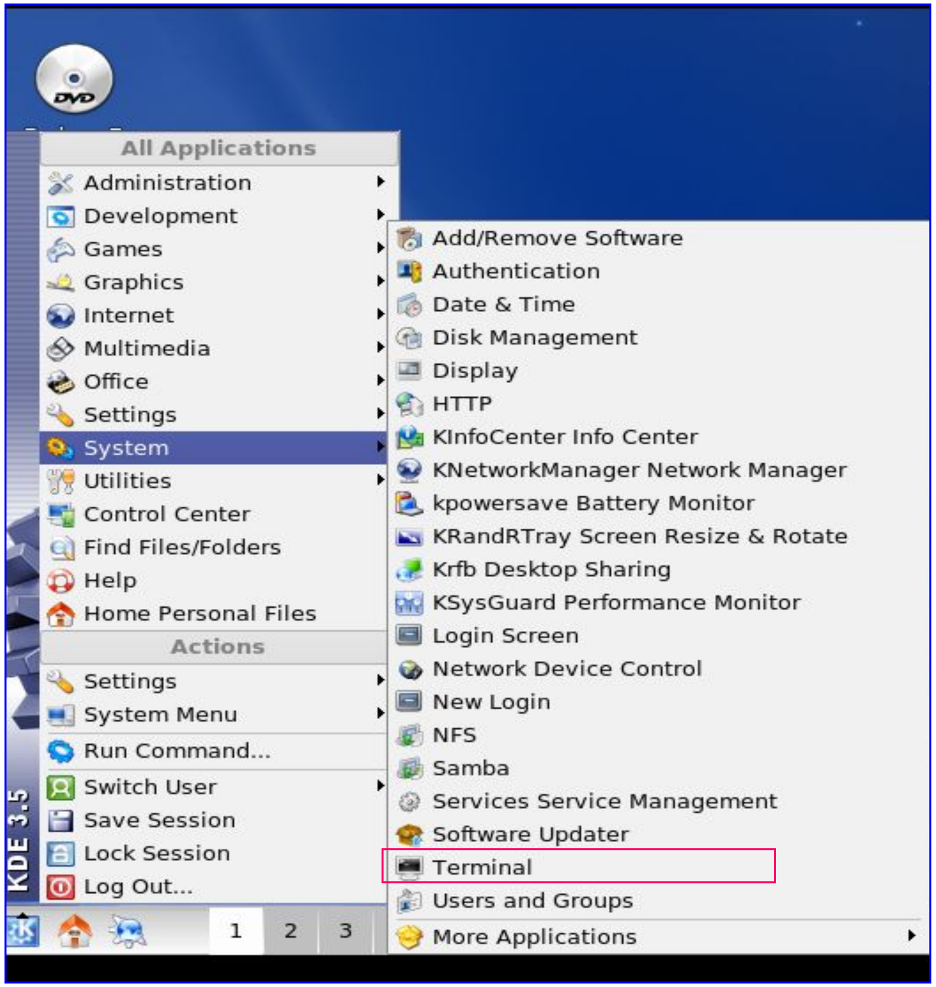
Kernel –

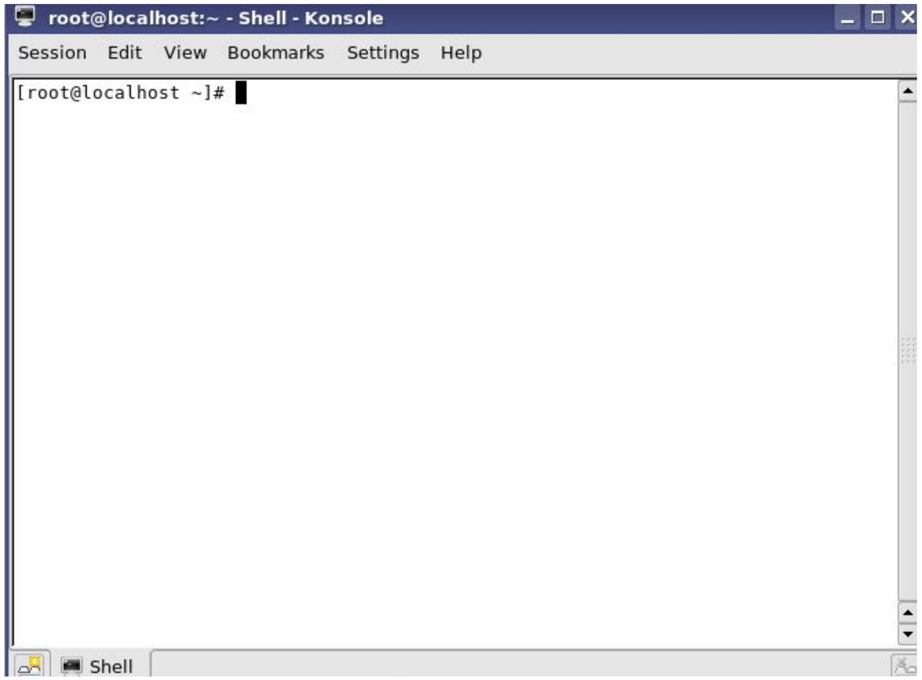
- The **kernel** is the **core component** of the Linux operating system. It **manages and allocates system resources**, including the **CPU, memory (RAM), and hardware devices**.
- Key Functions
  - Process scheduling – Controls which processes run and when
  - Memory management – Allocates and manages system memory
  - File system support – Provides access to files and directories
  - Process control – Creates and terminates processes
  - Device access – Manages communication with hardware devices
  - Network access – Handles data transmission over networks
  - System call interface – Provides APIs for applications to interact with the system

SHELL

- The shell is a **command-line interpreter**, acting as a special application that allows users to **interact with the operating system**.
- Key Features
  - Interprets and executes commands
  - Provides simple scripting capabilities
  - Bridges user input and system-level operations
- Common Shells
  - sh – Bourne Shell
  - csh – C Shell
  - ksh – Korn Shell
  - **bash** – Bourne Again Shell (most widely used)

SHELL





```
[root@localhost ~]#
```

Command Syntax

- General syntax: **command [flags] arg1 arg2 …**
  - Components are separated by spaces
  - Flags usually start with - (single-letter) or -- (multi-letter)
  - Examples

```
ls -a -l -F       # Separate flags
ls --color        # Long-format flag
ls -al            # Combined short flags (same as -a -l)
```

- Notes
  - Some commands may **not require a dash (-)** before flags
  - Use --help or man to view help for a command:

```
ls --help
man ls
```

- To check which shell you're using:

```
echo $SHELL
```

Wildcard Characters in Linux

- File or directory names used as command-line arguments don't always need to be explicit. You can use wildcards to match part or all of a name.
- Common wildcards

Symbol	Meaning
`*`	Matches **any sequence of characters**, including none
`?`	Matches **any single character**
`[abc]`	Matches **one character** from the set `a`, `b`, or `c`
`[!abc]`	Matches **any character except** `a`, `b`, or `c`
`\`	**Escapes** special meaning of wildcards (e.g., `\*`, `\?`)

- Examples

  `ls *.txt` → all files ending with `.txt`

  `ls file?.sh` → matches `file1.sh`, `fileA.sh`, etc.

  `ls [a-c]*` → files starting with `a`, `b`, or `c`

Command Auto-Completion

Press the <Tab> key to auto-complete commands, file names, or paths in the terminal. If multiple matches exist, pressing <Tab> twice shows suggestions.

```
$ cd /usr/lo<Tab> (/usr/local)

$ cp<Tab><Tab>
cp cpp cpio cproto

$ cd dir<Tab><Tab>
dir1 dir2 dir3
```

Commonly Used Linux Commands

Command	Description
`pwd`	Show current working directory
`cd`	Change directory
`ls`	List directory contents
`cp`	Copy files and directories
`mv`	Move or rename files
`rm`	Remove files and directories
`find`	Search for files and directories
`more`	View file content one page at a time
`grep`	Print lines that match a given pattern
`file`	Determine the type of a file

pwd and cd

- How to determine the current working directory?
  - Print working directory: pwd
- Change directory: cd
  - Examples:

```
cd /etc
cd ~ ( ~: macro to indicate the user home directory)
cd /home/sv
cd ..
cd ../../data
```

echo

- Print a string to the screen

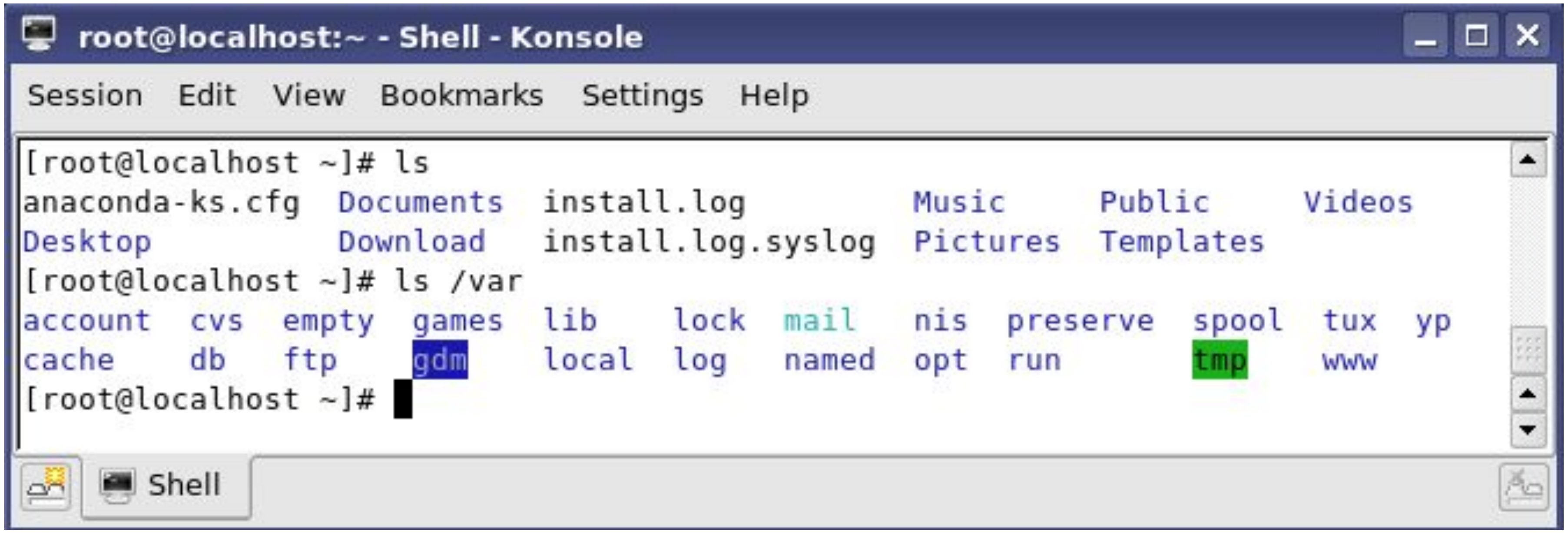
```
echo "Hello World"
```

- Print a string without a newline

```
echo –n "Enter your name:"
```

ls

❑ Listing the directory contents


root@localhost:~ - Shell - Konsole
Session Edit View Bookmarks Settings Help
Shell

```
[root@localhost ~]# ls
anaconda-ks.cfg  Documents  install.log         Music     Public     Videos
Desktop          Download   install.log.syslog  Pictures  Templates
[root@localhost ~]# ls /var
account  cvs  empty  games  lib    lock  mail   nis  preserve  spool  tux  yp
cache    db   ftp    gdm    local  log   named  opt  run       tmp    www
[root@localhost ~]# 
```

mkdir, rmdir, touch

- mkdir – create a new directory

```
$ mkdir –p dir3/dir4
```

(flag -p: create the parent directory if it does not exists)

- rmdir – delete an empty directory
- touch – create a new empty file

```
$ touch file.txt
```

cp, mv, rm, ln

- cp – copy file

```
$ cp file1 file2

$ cp file1 dir1
```

-f : force overwrite without asking

```
cp -f source.txt destination.txt
```

-i : prompt before overwriting

```
cp -i source.txt destination.txt
# Output: overwrite 'destination.txt'? (y/n)
```

-R,-r : copy the whole directory

```
$ cp –r dir1 dir2
```

cp, mv, rm, ln

- ❑ mv – move/rename

  ```
  $ mv   file1 file2
  $ mv   dir1 dir2
  ```

- ❑ rm – delete file/directory

  ```
  $ rm file1 file2
  $ rm –r dir3
  ```

  (flag -r: delete children files and directories)

- ❑ ln – is used to create links to files or directories (shortcut)

  ```
  $ ln -s /path/to/dir1 firstdir
  ```

Create a symbolic (soft) link named `firstdir` that points to the directory `dir1`. After that, `firstdir` behaves like `dir1`.

```
$ ln –f /tmp/test.txt
```

Create a hard link to the file `/tmp/test.txt` in the current directory with the same name (`test.txt`).

The `-f` option **forces the link creation** by removing any existing file with the same name.

Wildcard Characters in cp, mv Commands

***** matches any string, including the empty string

```
cp *.txt /backup/
```

? matches exactly one arbitrary character

```
mv file?.log /logs/
```

[...] matches any one of the characters inside the brackets

```
cp report[123].pdf /archive/
```

#copies `report1.pdf`, `report2.pdf`, or `report3.pdf`

[!...] or **[^...]** matches any character not inside the brackets

```
mv data[!0-9].csv /data/
```

\ removes the special meaning of the next character (escape character)

```
cp file?.txt /backup/
```

Redirection

- ❑ Redirection: Redirecting Data Streams Elsewhere
- ❑ **Types of Redirection:**
- ❑ `<` : input redirection (read input from a file)
- ❑ `>` : output redirection (write output to a file, overwriting it)
- ❑ `>>` : output redirection (append output to the end of a file)
- ❑ Examples:
  - ls -l / > /root/list.txt

Lists the contents of the `/` directory. The output is **not shown on the screen**, but saved to the file `/root/list.txt`. If the file already exists, it will be **overwritten**.

  - ls –l / >> /root/list.txt

Same as above, but instead of overwriting (`>`), the output will be **appended** to the end of `/root/list.txt`.

Standard Data Streams

Stream	Number
`stdin`	0
`stdout`	1
`stderr`	2

Example: Run the `ls` command, redirecting error messages to a file called `error.txt`:

```
ls -R / 2>/root/error.txt
```

This runs `ls -R /` (recursive listing of `/`) and sends all error messages (`stderr`) to `/root/error.txt`.

Pipe

- Pipe: The output of one command is passed as the input to the next command, using the | character
- Example: ls –R / | less
  - The ls -R / command lists files recursively from /, and its output is sent to less for paginated viewing.
- Paging with **more** and **less**:
  - **more** lets you view output page by page.
  - **less** provides more flexible navigation:
  - Enter: to move down one line
  - Spacebar: to move down one page
  - b: to move back one page
  - q: to quit

tee

- Output results both to the screen and to a file
- Example:

```
ls –l /etc | tee /root/list.txt
```

String functions

- cat & tac
- head & tail
- nl & wc,
- od & hexdump
- join, sort, tr
- grep

cat & tac

- cat: View file contents
- Example: View the content of the file /etc/passwd

  ```
  cat /etc/passwd
  ```
- Options:
  - -n : number all output lines
  - -b : number non-blank output lines only
  - -A : display all characters, including line endings
- tac is the reverse of cat (displays file contents from the end to the beginning)

head & tail

- head: View the first lines of data
- Examples:
  - View the 4 first lines of the file /etc/passwd

    ```
    head -4 /etc/passwd
    cat /etc/passwd | head -4
    ```
  - View the 4 first files or directories of the directory /

    ```
    ls –l / | head -4
    ```
- tail: View the last lines of data
- Examples:
  - View the 5 last lines of the file /etc/passwd

    ```
    tail -5 /etc/passwd
    cat /etc/passwd | tail -5
    ```
  - View the contents of the file /etc/passwd **starting from line 4 until the end:**

    ```
    tail –lines=+4 /etc/passwd
    cat /etc/passwd | tail --lines=+4
    ```
- **Note:** The -f option allows tail to follow and continuously display appended data, useful for monitoring dynamic log files.

wc: Lines, Words, Bytes count

- Syntax: wc [option] [files]

  -l : count lines

  -c or –m : count characters

  -w : count words

- Examples:
  - $ wc -l   file1   - Counts the number of lines in file1
  - $ wc file[123]   - Counts lines, words, and bytes for the three files file1, file2, and file3.
  - $ wc -c file1   - Counts the number of characters (bytes) in file1.

nl : Number Lines

- nl
- Example:

```
ls –l / | nl
```

View the list of files with line numbers.

join

- Syntax: join [*options*] *file1 file2*
  - *Options*: -j field
- Example:

File1:	1 one	File2:	1 11
	2 two		2 22
	3 three		4 44

```
$ join file1 file2
$ join –j 1 file1 file2
```

tr – translate text

- Syntax: tr [options] [[string1 [string2]]
  - Options: –d delete characters, -s : replace repeated characters with a single instance (squeeze)

`$ cat file1 | tr a-z A-Z`	Convert lowercase letters to uppercase.
`$ cat file1 | tr -d a`	Delete all occurrences of the character a
`$ tr '[A-B]' '[a-b]'< file.txt`	Convert uppercase A and B to lowercase a and b
`$ tr ':' ' ' < /etc/passwd`	Replace all characters : with spaces
`$ cat file1 | tr -d abc`	Delete all occurrences of characters a, b, and c

[:lower:] lowercase letters

[:upper:] uppercase letters

[:alnum:] alphanumeric characters (letters and digits)

```
cat file.txt | tr '[:lower:]' '[:upper:]'
cat file.txt | tr '[:upper:]' '[:lower:]'
cat file.txt | tr -d '[:alnum:]'
cat file.txt | tr -cd '[:alnum:]'
```

- **Note:** The tr command only accepts two arguments (string1 and string2).

cut

- Syntax:

  cut -d<delimiter> -f<field_number>

- Example: given a string

  1;2;3;4;5;6

To extract the number 5 (the 5th field):

```
echo “1;2;3;4;5;6” | cut -d”;” -f5
```

Cutting Strings with awk

- Syntax: Print the n^th^ field

```
awk -F<delimiter> '{ print $n }'
```

- By default, the delimiter is whitespace.
- Example: given the input string:

```
1;2;3;4;5;6
```

To extract the number 5 (the 5th field):

```
echo "1;2;3;4;5;6" | awk -F";" '{ print $5 }'
```

grep

- Searching Content with grep. Syntax:

grep [OPTION] PATTERN [FILE]

Options:

-i : case-insensitive search

-n : show line numbers with output

-r : recursively search in subdirectories

-v : invert match (show lines that do not match)

-w : match whole words only

- Examples:

grep root /etc/passwd : search for lines containing the word **root** in the file /etc/passwd.

ls –l /etc/ | grep conf : find files containing the string **conf** in the /etc directory listing.

Regular Expressions for grep

- ❑ `[abc]` : matches character **a**, **b**, or **c**
- ❑ `[a-h]` : matches any one character from **a** to **h**
- ❑ `[^abc]` : matches any character **except** a, b, or c
- ❑ `(ab|bc|cd)` : matches **ab** or **bc** or **cd**
- ❑ `^` : matches the **start** of a line
- ❑ `$` : matches the **end** of a line
- ❑ `.` : matches **any single character**

Number of occurrences:

- `*` : matches the preceding element **0 or more times**
- `+` : matches the preceding element **1 or more times**

find – file search command

find [path] [expression]

- $ find / -name “*.txt” : find file with the .txt extension in the directory /
- $ find /usr/local -type f –print : find files only in /usr/local
- $ find /usr/X11R6 -type d : find directory only in /usr/X11R6
- $ find . -perm 755 -a -type f : find files with permission 755 in the current directory

Restart and Shutdown

- Shutdown:

```
init 0
```

or

```
shutdown –h  now
```

- Restart:

```
init 6
```

or

```
shutdown –r  now
```

init modes

- Syntax: init <number>

Runlevel	Name	Description
0	**Shutdown**	Halts the system (power off). Safe to use when shutting down.
1	**Single-user mode**	For maintenance. Only root can log in. No networking or multi-user support.
2	Multi-user (no NFS)	Rarely used. Multi-user without network file system.
3	**Multi-user mode**	Full multi-user with networking, but **no graphical interface**.
4	Undefined/Custom	Not used by default. You can customize it for special needs.
5	**Graphical mode**	Multi-user mode **with GUI login (X11/Display Manager)**.
6	**Reboot**	Reboots the system.

history

The list of executed commands is stored in:

“~/.bash_history”

- ^P, <Up> previous command
- ^N, <Down> next command
- history: display the list of previously executed commands

```
$ history
1 clear
2 cd /
3 ls
4 mkdir /tmp/dir1
```

- !n: re-execute command number n
- !string: re-execute the most recent command that **starts with "string"**

Q&A