In-Depth Cheat Sheet

📂 Core Linux Commands – Detailed Explanations

`awk` – Pattern scanning and processing language

Purpose: Used to process and analyze text line by line, split into fields.

Syntax:

```
awk 'pattern {action}' filename
```

Explanation:

- `awk` processes a file one line at a time.
- By default, it splits each line into fields based on whitespace (or a delimiter using `-F`).
- `$1`, `$2`, etc. represent fields.

Examples:

```
awk '{print $1}' file.txt          # Print the first word from each line
awk -F: '{print $1, $3}' /etc/passwd    # Show username and UID
awk '/bash/ {print $1}' /etc/passwd     # Print lines containing 'bash'
```

`cat` – Concatenate and print file contents

Purpose: View file contents or combine files.

Syntax:

```
cat filename
```

Examples:

```
cat file.txt                       # View file
cat file1.txt file2.txt > all.txt # Combine two files into one
```

`cp` – Copy files and directories

Purpose: Make a copy of a file or directory.

Syntax:

```
cp [options] source target
```

Important Flags:

- `-r`: Recursively copy directories
- `-p`: Preserve file attributes

Examples:

```
cp file.txt backup/                # Copy file to directory
cp -r project/ /tmp/project_copy/ # Recursively copy a folder
```

`cut` – Remove sections from each line of input

Purpose: Extract specific fields or characters.

Syntax:

```
cut -d'[delimiter]' -f[field_number] filename
```

Examples:

```
cut -d':' -f1 /etc/passwd          # Show usernames
cut -c1-4 file.txt                 # Show first 4 characters of each line
```

`grep` – Search text using patterns

Purpose: Filter lines from input that match a search pattern.

Syntax:

```
grep [options] pattern filename
```

Common Flags:

- `-i`: Case-insensitive
- `-v`: Invert match (exclude)
- `-r`: Recursively search directories

Examples:

```
grep root /etc/passwd            # Show lines with 'root'
grep -i error syslog.log         # Case-insensitive match
grep -v bash /etc/passwd         # Show lines without 'bash'
```

`head` and `tail` – Show file beginnings and ends

Purpose: View first or last lines of a file.

Examples:

```
head -n 10 log.txt               # First 10 lines
tail -n 5 log.txt                # Last 5 lines
tail -f log.txt                  # Live stream of new lines
```

`ls` – List directory contents

Purpose: See files and directories.

Useful Flags:

- `-l`: Long list format
- `-a`: Show hidden files
- `-h`: Human-readable file sizes

Examples:

```
ls                               # Basic list
ls -lah                          # Detailed with hidden files
```

`man` – Manual pages for commands

Purpose: Look up documentation.

Usage:

```
man ls                           # Open manual for 'ls'
man -k "search"                  # Search by keyword
```

- Use `/term` to search inside the man page.

`mkdir` – Create directories

Purpose: Make new folders.

Examples:

```
mkdir newdir                    # Simple folder
mkdir -p a/b/c                  # Create nested folders
```

`mv` – Move or rename files/directories

Purpose: Rename or relocate files.

Examples:

```
mv file.txt old_file.txt        # Rename
mv file.txt ~/Documents/        # Move file
```

`tac` – Reverse version of `cat`

Purpose: Print file lines in reverse order.

Examples:

```
tac log.txt                     # Reverse log view
```

`touch` – Create new files or update timestamps

Purpose: Quickly make empty files or update last-modified times.

Examples:

```
touch test.txt                  # New file
touch file{1..3}.log            # Multiple files
```

`tr` – Translate or delete characters

Purpose: Convert or clean up text.

Examples:

```
echo "hello" | tr 'a-z' 'A-Z'    # Convert to uppercase
cat notes.txt | tr -d '0-9'      # Remove all digits
```

`tree` – Visual tree of directory

Purpose: Show directory structure.

Examples:

```
tree                             # Full structure
tree -L 2                        # Limit depth
```

📤 Redirection and Pipes

Redirection

- `>`: Overwrite file with output
- `>>`: Append output
- `<`: Use a file as input

Examples:

```
echo "Start" > file.txt          # New file or overwrite
echo "Another" >> file.txt       # Append
cat < file.txt                   # Read from file
```

Pipes

Purpose: Send output of one command as input to another.

Examples:

```
ls -l | less                     # Scroll output
ps aux | grep ssh                # Search running processes
```

🧮 Wildcards and Brace Expansion

Wildcards

- `*`: Match any characters

- ?: Match a single character

Examples:

```
ls *.txt                        # All text files
rm file?.txt                    # file1.txt, file2.txt
```

Brace Expansion

Examples:

```
mkdir {logs,backup,temp}        # Create multiple folders
touch file{A,B,C}.txt           # fileA.txt, fileB.txt...
```

🖋 Shell Scripting

Basic Script

```
#!/bin/bash
echo "This is a script"
```

Variables

```
name="Rodrigo"
echo "Welcome $name"
```

Substitution

```
now=$(date)
echo "It is now $now"
```

Execution

```
chmod +x script.sh
./script.sh
```

🧪 Mandatory Practice Scripts

Script 1

```
#!/bin/bash
echo -e "\n User: $USER"
echo -e "\tHome: $HOME"
echo -e "\tCurrent Dir: $PWD"
cd ~
echo "At home: $PWD"
```

Script 4

```
#!/bin/bash
echo "┌─────────┐"
echo "│ Linux   │"
echo "└─────────┘"
```

Script 5

```
#!/bin/bash
dir="exam"
file="ready.txt"
mkdir "$dir"
touch "$dir/$file"
tree "$dir"
```

🔁 How to Create and Clone a GitHub Repository in Linux

🛠 1. Create the Repository on GitHub (in browser)

- Go to https://github.com
- Click the **+** icon → **New repository**
- Enter repository name (e.g., `linux-final`)
- Optional: add description
- Choose **Public** or **Private**
- ⚠ Do NOT initialize with README for CLI setup
- Click **Create repository**

🔄 2. Clone the Repository in Linux

```
git clone https://github.com/your-username/linux-final.git
cd linux-final
```

📝 3. Create or Edit Files

```
touch notes.md
nano notes.md
```

💾 4. Add, Commit, and Push

```
git add .
git commit -m "Initial commit"
git push origin main
```

If first push fails:

```
git push -u origin main
```

> 💡 Use a GitHub Personal Access Token if Git asks for your password.

🧭 Filesystem Concepts

Absolute vs Relative Paths

- Absolute: Full path from root `/`
- Relative: Path from current working directory

Examples:

```
cd /etc                          # Absolute
cd ../Documents                  # Relative
```
