Linux Commands Course - Practice Exercises

Hands-on exercises for each section

Welcome to Practice Exercises

This file contains step-by-step practice exercises for each section of the Linux Commands Course.

Each exercise is designed to reinforce the concepts learned in that specific section, using only knowledge from that section and previous ones.

Section 0: Shell & Getting Help (Core)

Objective: Get comfortable with basic shell commands and help systems.

Tasks:

1. Check which shell you're using
2. Print your name using echo
3. Find a one-line description for the `ls` command
4. Use `type` and `which` to identify the `echo` command
5. Open the manual page for `echo`, search for "escape", then quit
6. Clear your terminal screen
7. Show your command history

Section 0: Solution

```bash
# 1. Check which shell you're using
echo $SHELL

# 2. Print your name using echo
echo "Your Name Here"

# 3. Find a one-line description for ls
whatis ls

# 4. Use type and which to identify echo
type echo
which echo

# 5. Open manual page for echo, search for "escape", then quit
man echo
# Inside man: type /escape, then press Enter, then press q to quit

# 6. Clear your terminal screen
clear

# 7. Show your command history
history
```

Section 1: Navigation & Filesystem Concepts (Core)

Objective: Master filesystem navigation and file listing.

Tasks:

1. Show your current directory
2. List all files including hidden ones
3. Change to `/etc` directory and list files sorted by modification time
4. Create three folders at once using brace expansion: `test/{a,b,c}`
5. Print your `$PATH` environment variable
6. Return to your home directory
7. Use `tree` to show directory structure (if available)

Section 1: Solution

```bash
# 1. Show your current directory
pwd

# 2. List all files including hidden ones
ls -a

# 3. Change to /etc and list files sorted by modification time
cd /etc
ls -lt

# 4. Create three folders at once using brace expansion
mkdir test/{a,b,c}

# 5. Print your PATH environment variable
echo $PATH

# 6. Return to your home directory
cd ~
# or just: cd

# 7. Use tree to show directory structure (if available)
tree -L 2
```

Section 2: Files & Directories (Core)

Objective: Create, read, and manage files and directories.

Tasks:

1. Create a directory named `lab`
2. Inside it, create an empty file called `report.txt`
3. View the file with `cat`, then with `less`
4. Copy the file to a new location and rename it
5. Create a symbolic link to the file called `latest_report`
6. Delete the original file and observe what happens to the symlink
7. Inspect the file type using `file`
8. Check file details with `stat`

Section 2: Solution

```
# 1. Create a directory named lab
mkdir lab

# 2. Inside it, create an empty file called report.txt
cd lab
touch report.txt

# 3. View the file with cat, then with less
cat report.txt
less report.txt
# Press q to quit less

# 4. Copy the file to a new location and rename it
cp report.txt ../backup_report.txt

# 5. Create a symbolic link to the file called latest_report
ln -s report.txt latest_report

# 6. Delete the original file and observe what happens to the symlink
rm report.txt
ls -l latest_report  # Shows broken symlink

# 7. Inspect the file type using file
file latest_report

# 8. Check file details with stat
stat latest_report
```

Section 3: Permissions & Ownership (Core)

Objective: Understand and manage file permissions and ownership.

Tasks:

1. Create a script file `hello.sh` and make it executable
2. Change its group to `users` (or your default group)
3. Remove read access for others
4. Create a `/tmp/shared` directory and give group write access
5. Apply setgid bit to the shared directory
6. Create a `/tmp/public` directory with sticky bit
7. Use `ls -l` to verify all your permission changes

Section 3: Solution

```
# 1. Create a script file hello.sh and make it executable
echo '#!/bin/bash' > hello.sh
echo 'echo "Hello World"' >> hello.sh
chmod +x hello.sh

# 2. Change its group to users (or your default group)
chgrp users hello.sh

# 3. Remove read access for others
chmod o-r hello.sh

# 4. Create a /tmp/shared directory and give group write access
sudo mkdir /tmp/shared
sudo chmod g+w /tmp/shared

# 5. Apply setgid bit to the shared directory
sudo chmod g+s /tmp/shared

# 6. Create a /tmp/public directory with sticky bit
sudo mkdir /tmp/public
sudo chmod +t /tmp/public

# 7. Use ls -l to verify all your permission changes
ls -l hello.sh
ls -ld /tmp/shared
ls -ld /tmp/public
```

Section 4: Finding Things (Core)

Objective: Master file and command searching techniques.

Tasks:

1. Use `which` to find the path to `bash`
2. Run `whereis` on `ls` and identify its man page location
3. Search your home directory for files larger than 1MB
4. Exclude `.cache` directories from a recursive search
5. Use `locate` to find any "shadow" file, then refresh the database
6. Find all `.txt` files in your home directory
7. Search for files modified in the last 7 days

Section 4: Solution

```
# 1. Use which to find the path to bash
which bash

# 2. Run whereis on ls and identify its man page location
whereis ls

# 3. Search your home directory for files larger than 1MB
find ~ -type f -size +1M

# 4. Exclude .cache directories from a recursive search
find ~ -type f -not -path "*/.cache/*"

# 5. Use locate to find any "shadow" file, then refresh the database
locate shadow
sudo updatedb

# 6. Find all .txt files in your home directory
find ~ -name "*.txt"

# 7. Search for files modified in the last 7 days
find ~ -type f -mtime -7
```

Section 5: Text Viewing & Pipelines (Core)

Objective: Master text processing and pipeline operations.

Tasks:

1. Count how many users have `/bin/bash` in `/etc/passwd`
2. Redirect all output of `ls -lh /etc` into a file called `etc_list.txt`
3. Append the current date to that same file using `>>`
4. Create two text files with names and merge them using `paste`
5. Compare two text files for common lines using `comm`
6. Use a pipeline with `tee` to save and count results simultaneously
7. Count lines, words, and characters in `/etc/passwd`

Section 5: Solution

```
# 1. Count how many users have /bin/bash in /etc/passwd
grep '/bin/bash' /etc/passwd | wc -l

# 2. Redirect all output of ls -lh /etc into a file
ls -lh /etc > etc_list.txt

# 3. Append the current date to that same file
date >> etc_list.txt

# 4. Create two text files with names and merge them using paste
echo -e "Alice\nBob\nCharlie" > names1.txt
echo -e "David\nEve\nFrank" > names2.txt
paste names1.txt names2.txt

# 5. Compare two text files for common lines using comm
# First, sort both files
sort names1.txt > names1_sorted.txt
sort names2.txt > names2_sorted.txt
comm names1_sorted.txt names2_sorted.txt

# 6. Use a pipeline with tee to save and count results simultaneously
ls /etc | tee file_list.txt | wc -l

# 7. Count lines, words, and characters in /etc/passwd
wc /etc/passwd
```

Section 6: Text Processing (Core → Plus)

Objective: Master advanced text processing with grep, sed, awk, and other tools.

Tasks:

1. Print only usernames from `/etc/passwd` using `cut`
2. Find all lines containing "error" in `/var/log/syslog` (if accessible)
3. Replace "failed" with "FAILED" in a test file using `sed -i`
4. Print fields 1 and 7 of `/etc/passwd` with `awk`
5. Convert a text file from one encoding to another using `iconv`
6. Parse JSON output from an API using `jq` (if available)
7. Combine `grep`, `tr`, and `tee` into one pipeline

Section 6: Solution

```
# 1. Print only usernames from /etc/passwd using cut
cut -d: -f1 /etc/passwd

# 2. Find all lines containing "error" in /var/log/syslog
sudo grep "error" /var/log/syslog

# 3. Replace "failed" with "FAILED" in a test file using sed -i
echo "The operation failed" > test.txt
sed -i 's/failed/FAILED/g' test.txt
cat test.txt

# 4. Print fields 1 and 7 of /etc/passwd with awk
awk -F: '{print $1, $7}' /etc/passwd

# 5. Convert a text file from one encoding to another using iconv
echo "Test file" > test_utf8.txt
iconv -f UTF-8 -t ISO-8859-1 test_utf8.txt > test_latin1.txt

# 6. Parse JSON output from an API using jq (if available)
curl -s https://api.github.com/users/torvalds | jq '.name, .public_repos'

# 7. Combine grep, tr, and tee into one pipeline
echo -e "info: This is information\nerror: This is an error\ninfo: More information" > logs.txt
grep "info" logs.txt | tr '[:lower:]' '[:upper:]' | tee filtered_logs.txt
```

Section 7: Archiving & Compression (Core)

Objective: Master file archiving and compression techniques.

Tasks:

1. Create a tar archive of your home directory
2. Compress it using gzip, bzip2, and xz — compare sizes
3. Extract each version and verify integrity
4. Create a `.zip` archive of your project folder
5. List contents without extracting
6. Try using `zstd` for fast modern compression (if available)
7. Create a compressed archive with specific files only

Section 7: Solution

```
# 1. Create a tar archive of your home directory
tar -cf home_backup.tar ~

# 2. Compress it using gzip, bzip2, and xz — compare sizes
gzip -c home_backup.tar > home_backup.tar.gz
bzip2 -c home_backup.tar > home_backup.tar.bz2
xz -c home_backup.tar > home_backup.tar.xz
ls -lh home_backup.tar*

# 3. Extract each version and verify integrity
mkdir test_extract
cd test_extract
tar -xf ../home_backup.tar.gz
tar -xf ../home_backup.tar.bz2
tar -xf ../home_backup.tar.xz

# 4. Create a .zip archive of your project folder
zip -r project.zip ~/lab

# 5. List contents without extracting
tar -tf home_backup.tar | head -10
unzip -l project.zip

# 6. Try using zstd for fast modern compression (if available)
zstd home_backup.tar -o home_backup.tar.zst

# 7. Create a compressed archive with specific files only
tar -czf specific_files.tar.gz ~/lab/*.txt
```

Section 8: Essential Linux Directories (Core)

Objective: Understand the Linux filesystem structure and key directories.

Tasks:

1. List files in `/etc` and identify one configuration file you recognize
2. View `/proc/cpuinfo` and `/proc/meminfo`
3. Check your user entry in `/etc/passwd`
4. Find your DNS nameservers in `/etc/resolv.conf`
5. Open `~/.bashrc` and add a custom alias (then reload with `source ~/.bashrc`)
6. Explore `/var/log` directory and identify different log files
7. Check what's in `/usr/bin` and `/usr/local/bin`

Section 8: Solution

```
# 1. List files in /etc and identify one configuration file you recognize
ls /etc
cat /etc/hostname  # Example configuration file

# 2. View /proc/cpuinfo and /proc/meminfo
cat /proc/cpuinfo | head -10
cat /proc/meminfo | head -5

# 3. Check your user entry in /etc/passwd
grep $USER /etc/passwd

# 4. Find your DNS nameservers in /etc/resolv.conf
cat /etc/resolv.conf

# 5. Open ~/.bashrc and add a custom alias (then reload with source ~/.bashrc)
echo 'alias ll="ls -la"' >> ~/.bashrc
source ~/.bashrc
ll  # Test the new alias

# 6. Explore /var/log directory and identify different log files
ls /var/log
cat /var/log/syslog | head -5

# 7. Check what's in /usr/bin and /usr/local/bin
ls /usr/bin | head -10
ls /usr/local/bin
```

Section 9: Users, Groups & sudo (Core)

Objective: Master user and group management, and understand sudo privileges.

Tasks:

1. Create a new user `labuser` with a home directory and bash shell
2. Set a password for the new user
3. Create a group called `developers` and add `labuser` to it
4. Check the user's groups with `id`
5. View sudo rules and understand the structure
6. Test sudo access (if you have sudo privileges)
7. Examine user account details

Section 9: Solution

```
# 1. Create a new user labuser with a home directory and bash shell
sudo useradd -m -s /bin/bash labuser

# 2. Set a password for the new user
sudo passwd labuser

# 3. Create a group called developers and add labuser to it
sudo groupadd developers
sudo usermod -a -G developers labuser

# 4. Check the user's groups with id
id labuser

# 5. View sudo rules and understand the structure
sudo cat /etc/sudoers | grep -v "^#"

# 6. Test sudo access (if you have sudo privileges)
sudo whoami

# 7. Examine user account details
getent passwd labuser
getent group developers
```

Section 10: Processes & Jobs (Core)

Objective: Master process management and job control.

Tasks:

1. Run `sleep 120` in background and list it with `jobs`
2. Bring it to foreground, then stop it with **Ctrl+Z**
3. Resume in background with `bg`
4. Start a command with low priority using `nice`
5. Use `top` to monitor processes and identify the one with low priority
6. Use `kill` to stop the process gracefully
7. List all processes and find specific ones

Section 10: Solution

```
# 1. Run sleep 120 in background and list it with jobs
sleep 120 &
jobs

# 2. Bring it to foreground, then stop it with Ctrl+Z
fg
# Press Ctrl+Z to stop it

# 3. Resume in background with bg
bg

# 4. Start a command with low priority using nice
nice -n 10 sleep 60 &

# 5. Use top to monitor processes and identify the one with low priority
top
# Press q to quit top

# 6. Use kill to stop the process gracefully
ps aux | grep sleep
kill <PID>  # Replace <PID> with actual process ID

# 7. List all processes and find specific ones
ps aux | grep sleep
pgrep sleep
```

Section 11: Services & Logs (Core)

Objective: Master systemd service management and log analysis.

Tasks:

1. Check which target your system boots into
2. Restart the SSH or networking service
3. Enable automatic NTP time sync with `timedatectl`
4. View all logs since last boot
5. Display only authentication errors from the system journal
6. Examine `/var/log/syslog` for today's entries
7. Check service status and dependencies

Section 11: Solution

```
# 1. Check which target your system boots into
systemctl get-default

# 2. Restart the SSH or networking service
sudo systemctl restart ssh
# or
sudo systemctl restart NetworkManager

# 3. Enable automatic NTP time sync with timedatectl
sudo timedatectl set-ntp true
timedatectl status

# 4. View all logs since last boot
journalctl -b

# 5. Display only authentication errors from the system journal
journalctl -p err | grep -i auth

# 6. Examine /var/log/syslog for today's entries
sudo tail -n 50 /var/log/syslog

# 7. Check service status and dependencies
systemctl status ssh
systemctl list-dependencies ssh
```

Section 12: Networking (Core)

Objective: Master network configuration and connectivity tools.

Tasks:

1. List your current IP address and default route
2. Show which services are listening on ports
3. Test connectivity to google.com and view the route it takes
4. Query DNS for the MX records of example.com
5. Download a web page with curl and wget
6. Copy a local file to a remote system using scp (if possible)
7. Bring your network connection down and up again with nmcli

---

Section 12: Solution

```
# 1. List your current IP address and default route
ip addr show
ip route show

# 2. Show which services are listening on ports
ss -tuln
netstat -tuln

# 3. Test connectivity to google.com and view the route it takes
ping -c 4 google.com
traceroute google.com

# 4. Query DNS for the MX records of example.com
dig MX example.com
nslookup -type=MX example.com

# 5. Download a web page with curl and wget
curl -o google.html https://www.google.com
wget -O google_wget.html https://www.google.com

# 6. Copy a local file to a remote system using scp (if possible)
# scp file.txt user@remote-server:/path/to/destination

# 7. Bring your network connection down and up again with nmcli
nmcli connection show
sudo nmcli connection down "Your Connection Name"
sudo nmcli connection up "Your Connection Name"
```

Section 13: Packages & Software Management (Core)

Objective: Master package management and software installation.

Tasks:

1. Run `sudo apt update && sudo apt upgrade`
2. Install and remove `curl` using APT
3. Install a `.deb` package manually and fix dependencies
4. List all installed packages containing "python"
5. Query info for an installed package with `apt show`
6. Try installing and launching a Snap or Flatpak application
7. Search for available packages and check their descriptions

---

Section 13: Solution

```
# 1. Run sudo apt update && sudo apt upgrade
sudo apt update && sudo apt upgrade

# 2. Install and remove curl using APT
sudo apt install curl
curl --version
sudo apt remove curl

# 3. Install a .deb package manually and fix dependencies
# Download a .deb file first, then:
# sudo dpkg -i package.deb
# sudo apt install -f  # Fix dependencies

# 4. List all installed packages containing "python"
apt list --installed | grep python

# 5. Query info for an installed package with apt show
apt show python3

# 6. Try installing and launching a Snap or Flatpak application
sudo snap install hello-world
hello-world

# 7. Search for available packages and check their descriptions
apt search vim
apt show vim
```

Section 14: Disks & Filesystems (Core)

Objective: Master disk and filesystem management.

Tasks:

1. List all disks and their filesystems with `lsblk -f`
2. Check total disk usage using `df -h`
3. Find which directory takes the most space using `du -sh *`
4. Mount a USB drive to `/mnt` and then unmount it (if available)
5. Inspect `/etc/fstab` and identify all entries
6. Enable or disable swap space with `swapon` and `swapoff`
7. Check filesystem health and repair if needed

Section 14: Solution

```
# 1. List all disks and their filesystems with lsblk -f
lsblk -f

# 2. Check total disk usage using df -h
df -h

# 3. Find which directory takes the most space using du -sh *
du -sh * | sort -hr

# 4. Mount a USB drive to /mnt and then unmount it (if available)
# sudo mount /dev/sdb1 /mnt
# ls /mnt
# sudo umount /mnt

# 5. Inspect /etc/fstab and identify all entries
cat /etc/fstab

# 6. Enable or disable swap space with swapon and swapoff
swapon --show
# sudo swapoff /dev/sda2  # Replace with actual swap device
# sudo swapon /dev/sda2

# 7. Check filesystem health and repair if needed
sudo fsck -n /dev/sda1  # Check without repair
```

Section 15: Scheduling Tasks (Core)

Objective: Master task scheduling with cron and system timers.

Tasks:

1. Schedule a command to run every minute (for testing)
2. Add a daily cleanup job at midnight with `crontab -e`
3. View your current crontab with `crontab -l`
4. Schedule a one-time notification with `at now + 2 minutes`
5. Check logs to confirm that your jobs ran successfully
6. Create a systemd timer for a custom service
7. List all scheduled tasks and their status

Section 15: Solution

```bash
# 1. Schedule a command to run every minute (for testing)
crontab -e
# Add this line: * * * * * echo "Test job" >> /tmp/cron_test.log

# 2. Add a daily cleanup job at midnight with crontab -e
# Add this line: 0 0 * * * find /tmp -type f -mtime +7 -delete

# 3. View your current crontab with crontab -l
crontab -l

# 4. Schedule a one-time notification with at now + 2 minutes
echo "echo 'One-time job completed' >> /tmp/at_test.log" | at now + 2 minutes

# 5. Check logs to confirm that your jobs ran successfully
tail -f /tmp/cron_test.log
journalctl -u cron

# 6. Create a systemd timer for a custom service
# Create a simple script
echo '#!/bin/bash' > /home/$USER/backup.sh
echo 'echo "Backup completed at $(date)" >> /tmp/backup.log' >> /home/$USER/backup.sh
chmod +x /home/$USER/backup.sh

# 7. List all scheduled tasks and their status
crontab -l
atq  # List pending at jobs
```

Section 16: Bash Scripting (Core → Plus)

Objective: Master Bash scripting fundamentals and advanced techniques.

Tasks:

1. Write a script that greets a named user and logs to a file
2. Parse `-i` and `-o` flags, transform input, and save output
3. Use a temporary workspace and ensure it's cleaned with `trap`
4. Compare two files using process substitution
5. Run `shellcheck` and `shfmt` on your script
6. Create a function that handles errors gracefully
7. Use arrays to store and process multiple values

Section 16.1: Solution

```
# 1. Write a script that greets a named user and logs to a file
cat > greet.sh << 'EOF'
#!/bin/bash
NAME=${1:-"World"}
echo "Hello, $NAME!" >> /tmp/greetings.log
echo "Hello, $NAME!"
EOF
chmod +x greet.sh
./greet.sh "Alice"

# 2. Parse -i and -o flags, transform input, and save output
cat > transform.sh << 'EOF'
#!/bin/bash
INPUT=""
OUTPUT=""

while [[ $# -gt 0 ]]; do
    case $1 in
        -i) INPUT="$2"; shift 2 ;;
        -o) OUTPUT="$2"; shift 2 ;;
        *) echo "Unknown option $1"; exit 1 ;;
    esac
done

if [[ -n "$INPUT" && -n "$OUTPUT" ]]; then
    tr '[:lower:]' '[:upper:]' < "$INPUT" > "$OUTPUT"
    echo "Transformed $INPUT to $OUTPUT"
fi
EOF
chmod +x transform.sh
echo "hello world" > input.txt
./transform.sh -i input.txt -o output.txt
```

Section 16.2: Solution

```
# 3. Use a temporary workspace and ensure it's cleaned with trap
cat > temp_work.sh << 'EOF'
#!/bin/bash
TEMP_DIR=$(mktemp -d)
trap "rm -rf $TEMP_DIR" EXIT

echo "Working in: $TEMP_DIR"
echo "test data" > "$TEMP_DIR/test.txt"
cat "$TEMP_DIR/test.txt"
EOF
chmod +x temp_work.sh
./temp_work.sh

# 4. Compare two files using process substitution
echo "file1 content" > file1.txt
echo "file2 content" > file2.txt
diff <(cat file1.txt) <(cat file2.txt)

# 5. Run shellcheck and shfmt on your script
# Install tools first: sudo apt install shellcheck shfmt
shellcheck greet.sh
shfmt greet.sh
```

Section 16.3: Solution

```
# 6. Create a function that handles errors gracefully
cat > error_handler.sh << 'EOF'
#!/bin/bash
handle_error() {
    echo "Error occurred in line $1"
    exit 1
}
trap 'handle_error $LINENO' ERR

# This will trigger the error handler
ls /nonexistent/directory
EOF
chmod +x error_handler.sh
./error_handler.sh

# 7. Use arrays to store and process multiple values
cat > array_example.sh << 'EOF'
#!/bin/bash
FRUITS=("apple" "banana" "cherry")
echo "First fruit: ${FRUITS[0]}"
echo "All fruits: ${FRUITS[@]}"
echo "Number of fruits: ${#FRUITS[@]}"

for fruit in "${FRUITS[@]}"; do
    echo "Processing: $fruit"
done
EOF
chmod +x array_example.sh
./array_example.sh
```

Section 17: Environment Customization (Core)

Objective: Master shell environment customization and configuration.

Tasks:

1. Add a custom alias `update` to run system updates
2. Add `$HOME/scripts` to your PATH
3. Enable timestamps in your history
4. Install and test `bash-completion` for `git`
5. Create a `/etc/profile.d/myenv.sh` that defines a global variable
6. Customize your prompt to show current directory and git branch
7. Set up environment variables for a development project

Section 17: Solution

```
# 1. Add a custom alias update to run system updates
echo 'alias update="sudo apt update && sudo apt upgrade"' >> ~/.bashrc
source ~/.bashrc
update

# 2. Add $HOME/scripts to your PATH
echo 'export PATH="$HOME/scripts:$PATH"' >> ~/.bashrc
mkdir -p ~/scripts
source ~/.bashrc

# 3. Enable timestamps in your history
echo 'export HISTTIMEFORMAT="%Y-%m-%d %H:%M:%S "' >> ~/.bashrc
source ~/.bashrc
history | head -5

# 4. Install and test bash-completion for git
sudo apt install bash-completion
echo 'source /usr/share/bash-completion/bash_completion' >> ~/.bashrc
source ~/.bashrc

# 5. Create a /etc/profile.d/myenv.sh that defines a global variable
sudo tee /etc/profile.d/myenv.sh << 'EOF'
export PROJECT_ROOT="/opt/myproject"
export EDITOR="nano"
EOF
source /etc/profile.d/myenv.sh

# 6. Customize your prompt to show current directory and git branch
echo 'export PS1="\[\033[01;32m\]\u@\h\[\033[00m\]:\[\033[01;34m\]\w\[\033[00m\]\$ "' >> ~/.bashrc
source ~/.bashrc

# 7. Set up environment variables for a development project
cat > ~/.env_project << 'EOF'
export PROJECT_NAME="MyProject"
export PROJECT_DIR="$HOME/projects/$PROJECT_NAME"
export PYTHONPATH="$PROJECT_DIR/src:$PYTHONPATH"
EOF
echo 'source ~/.env_project' >> ~/.bashrc
source ~/.bashrc
```

Section 18: System Information & Troubleshooting (Plus)

Objective: Master system monitoring and troubleshooting techniques.

Tasks:

1. Find your kernel version and CPU model
2. Check uptime and system load averages
3. View available memory and swap usage
4. List all PCI and USB devices
5. Get BIOS info using `dmidecode -t bios`
6. Run a quick system report combining the above tools
7. Monitor system performance in real-time

Section 18: Solution

```
# 1. Find your kernel version and CPU model
uname -r
cat /proc/cpuinfo | grep "model name" | head -1

# 2. Check uptime and system load averages
uptime
cat /proc/loadavg

# 3. View available memory and swap usage
free -h
cat /proc/meminfo | grep -E "(MemTotal|MemAvailable|SwapTotal)"

# 4. List all PCI and USB devices
lspci
lsusb

# 5. Get BIOS info using dmidecode -t bios
sudo dmidecode -t bios

# 6. Run a quick system report combining the above tools
cat > system_report.sh << 'EOF'
#!/bin/bash
echo "=== System Report ==="
echo "Kernel: $(uname -r)"
echo "Uptime: $(uptime)"
echo "Memory: $(free -h | grep Mem)"
echo "CPU: $(cat /proc/cpuinfo | grep "model name" | head -1 | cut -d: -f2)"
echo "Load: $(cat /proc/loadavg)"
EOF
chmod +x system_report.sh
./system_report.sh

# 7. Monitor system performance in real-time
htop
# or
top
```

Section 19: Security & Firewall (Core)

Objective: Master Linux security controls and firewall management.

Tasks:

1. View capabilities of `/bin/ping`
2. Enable `ufw` and allow SSH while denying Telnet
3. Check current firewall rules
4. Check whether your system uses SELinux or AppArmor
5. List loaded AppArmor profiles or SELinux mode
6. View firewall rules using `nft list ruleset`
7. Test security controls and access restrictions

Section 19: Solution

```
# 1. View capabilities of /bin/ping
getcap /bin/ping

# 2. Enable ufw and allow SSH while denying Telnet
sudo ufw enable
sudo ufw allow ssh
sudo ufw deny telnet
sudo ufw status

# 3. Check current firewall rules
sudo ufw status verbose
sudo iptables -L

# 4. Check whether your system uses SELinux or AppArmor
which getenforce 2>/dev/null && getenforce || echo "SELinux not found"
which aa-status 2>/dev/null && aa-status || echo "AppArmor not found"

# 5. List loaded AppArmor profiles or SELinux mode
aa-status 2>/dev/null || echo "AppArmor not available"
getenforce 2>/dev/null || echo "SELinux not available"

# 6. View firewall rules using nft list ruleset
sudo nft list ruleset

# 7. Test security controls and access restrictions
sudo ufw status numbered
sudo ufw delete 1  # Delete first rule (be careful!)
```

Section 20: Quality of Life (Plus)

Objective: Master modern terminal tools for improved productivity.

Tasks:

1. Use `tldr tar` to review archive usage
2. Search for a keyword in your home directory using `rg`
3. Replace `find` commands with `fd`
4. Try viewing scripts with `bat`
5. Monitor network traffic with `nload` or `iftop`
6. Explore system performance interactively using `btop`
7. Set up and configure these tools for daily use

Section 20: Solution

```
# 1. Use tldr tar to review archive usage
tldr tar

# 2. Search for a keyword in your home directory using rg
# Install ripgrep first: sudo apt install ripgrep
rg "TODO" ~

# 3. Replace find commands with fd
# Install fd first: sudo apt install fd-find
fd "*.txt" ~

# 4. Try viewing scripts with bat
# Install bat first: sudo apt install bat
bat ~/.bashrc

# 5. Monitor network traffic with nload or iftop
# Install tools first: sudo apt install nload iftop
sudo nload
# or
sudo iftop

# 6. Explore system performance interactively using btop
# Install btop first: sudo apt install btop
btop

# 7. Set up and configure these tools for daily use
echo 'alias cat="batcat"' >> ~/.bashrc
echo 'alias find="fdfind"' >> ~/.bashrc
source ~/.bashrc
```

Section 21: Text Editors (Core)

Objective: Master Nano and Vi/Vim text editors for Linux administration.

Tasks:

1. Open `/etc/hosts` in Nano and add a comment line, then save and exit
2. Open a file in Vi, enter Insert mode, write text, and save with `:wq`
3. Practice deleting, copying, and pasting lines in Vi
4. Enable syntax highlighting in Nano and Vim
5. Compare the two editors — which feels more comfortable for you?
6. Create a simple script using both editors
7. Practice advanced Vi commands and navigation

Section 21.1: Solution

```
# 1. Open /etc/hosts in Nano and add a comment line, then save and exit
sudo nano /etc/hosts
# Add a comment line like: # My custom comment
# Press Ctrl+X, then Y, then Enter to save and exit

# 2. Open a file in Vi, enter Insert mode, write text, and save with :wq
vi test_file.txt
# Press 'i' to enter Insert mode
# Type some text
# Press Esc to exit Insert mode
# Type :wq and press Enter to save and quit

# 3. Practice deleting, copying, and pasting lines in Vi
vi practice.txt
# Press 'i' to enter Insert mode, type a few lines, press Esc
# Use 'dd' to delete a line
# Use 'yy' to copy a line
# Use 'p' to paste
# Use 'u' to undo
```

Section 21.2: Solution

```
# 4. Enable syntax highlighting in Nano and Vim
# For Nano: Press Ctrl+X, then Ctrl+R, then Ctrl+W for syntax highlighting
# For Vim: Type :syntax on

# 5. Compare the two editors — which feels more comfortable for you?
# Try editing the same file with both editors

# 6. Create a simple script using both editors
nano simple_script.sh
# Add: #!/bin/bash
# Add: echo "Hello from Nano"
# Save and exit

vi simple_script2.sh
# Press 'i', add: #!/bin/bash
# Add: echo "Hello from Vi"
# Press Esc, type :wq

# 7. Practice advanced Vi commands and navigation
vi advanced_practice.txt
# Use: /search_term to search
# Use: n to find next occurrence
# Use: :set number to show line numbers
# Use: G to go to end of file
# Use: gg to go to beginning of file
# Use: :q! to quit without saving
```

Congratulations!

You've completed all the practice exercises for the Linux Commands Course!

These exercises are designed to reinforce your learning and provide hands-on experience with each topic.

Next Steps

- Practice these commands regularly
- Try variations and combinations of the exercises
- Explore additional options and flags for each command
- Apply these skills to real-world scenarios
- Continue learning and experimenting with Linux!

Remember: The best way to master Linux commands is through consistent practice and real-world application.