Linux Basic Commands

This document covers the foundational Linux commands for navigating and managing files and directories: `ls`, `cd`, `mv`, `cp`, `rm`, `touch`, and `mkdir`. Each command includes a description, common options, and practical examples.

1. ls (List Directory Contents)

- **Description**: Displays files and directories in the current directory.
- **Common Options**:
  - `-l`: Long format (shows permissions, owner, size, etc.).
  - `-a`: Includes hidden files (those starting with `.`).
  - `-h`: Human-readable file sizes (used with `-l`).
  - `-t`: Sort by modification time (newest first).
  - `-r`: Reverse sort order.

Examples:

```
ls              # List files and directories
ls -l           # List in long format
ls -la          # List all files, including hidden, in long format
ls -lh          # List with human-readable sizes
ls -ltr         # List in long format, sorted by time, reversed
```

2. cd (Change Directory)

- **Description**: Navigates to a specified directory.
- **Common Options**: None typically used, but special arguments exist:
  - `..`: Move up one directory.
  - `~`: Go to the user's home directory.
  - `-`: Return to the previous directory.

Examples:

```
cd /home/user/docs  # Navigate to the docs directory
cd ..           # Move up one directory
cd ~            # Go to home directory
cd -            # Return to the previous directory
cd              # Go to home directory (no argument)
```

3. mv (Move/Rename Files or Directories)

- **Description**: Moves or renames files and directories.
- **Common Options**:
  - `-i`: Prompt before overwriting.
  - `-f`: Force overwrite without prompting.
  - `-v`: Verbose, show what is being done.

Examples:
```
mv file.txt /home/user/docs/       # Move file.txt to docs directory
mv file.txt newname.txt            # Rename file.txt to newname.txt
mv -i file.txt /home/user/docs/    # Move with overwrite prompt
mv -v myfolder /home/user/backup/  # Move directory with verbose output
```

4. cp (Copy Files or Directories)

- **Description**: Copies files or directories.
- **Common Options**:
  - `-r`: Recursive, copy directories and their contents.
  - `-i`: Prompt before overwriting.
  - `-v`: Verbose, show what is being done.

Examples:
```
cp file.txt /home/user/backup/         # Copy file.txt to backup directory
cp -r myfolder /home/user/backup/      # Copy myfolder and its contents
cp -i file.txt /home/user/backup/      # Copy with overwrite prompt
cp -v file.txt file_copy.txt           # Copy with verbose output
```

5. rm (Remove Files or Directories)

- **Description**: Deletes files or directories.
- **Common Options**:
  - `-r`: Recursive, delete directories and their contents.
  - `-f`: Force deletion without prompting.
  - `-i`: Prompt before each deletion.
  - `-v`: Verbose, show what is being deleted.

Examples:

```
rm file.txt              # Delete file.txt
rm -r myfolder           # Delete myfolder and its contents
rm -rf myfolder          # Force delete myfolder without prompting
rm -i file.txt           # Delete with prompt
rm -v *.txt              # Delete all .txt files with verbose output
```

- **Caution**: Use `rm -rf` carefully, as it deletes files permanently without recovery.

6. touch (Create Empty Files or Update Timestamps)

- **Description**: Creates empty files or updates the timestamp of existing files.
- **Common Options**:
  - `-m`: Update modification time only.
  - `-a`: Update access time only.
  - `-c`: Do not create the file if it doesn't exist.

Examples:

```
touch newfile.txt          # Create empty file newfile.txt
touch -m existing.txt      # Update modification time of existing.txt
touch -c nonexistent.txt   # Do nothing if file doesn't exist
touch file1.txt file2.txt  # Create multiple files
```

7. mkdir (Make Directories)

- **Description**: Creates new directories.
- **Common Options**:
  - `-p`: Create parent directories as needed (no error if they exist).
  - `-v`: Verbose, show each directory created.

Examples:

```
mkdir myfolder                 # Create a directory named myfolder
mkdir -p /home/user/docs/new   # Create nested directories
mkdir -v folder1 folder2       # Create multiple directories with verbose output
```

Practical Tips

- **Tab Completion**: Press `Tab` to auto-complete file or directory names.
- **Command History**: Use `up` and `down` arrows to navigate previous commands.
- **Combine Commands**: Use `&&` to run commands sequentially (e.g., `mkdir myfolder && cd myfolder`).
- **Safety with `rm`**: Double-check paths with `rm -rf` to avoid accidental deletion.
- **Practice Environment**: Use a virtual machine or non-critical directory for testing.

Practice Tasks

1. Create a directory `test`, navigate into it, and create an empty file `example.txt`.
2. Copy `example.txt` to `backup.txt` in the same directory.
3. Rename `backup.txt` to `backup_copy.txt`.
4. List all files in long format, including hidden ones.
5. Delete `example.txt` and the `test` directory.

Solution:

```
mkdir test
cd test
touch example.txt
cp example.txt backup.txt
mv backup.txt backup_copy.txt
ls -la
rm example.txt
cd ..
rm -r test
```

Linux File Permissions

This document covers Linux file permission commands: `chmod`, `chown`, `umask`, and `stat`. Each command includes a description, common options, and practical examples to help you understand and manage file permissions effectively.

1. chmod (Change File Mode)

- **Description**: Modifies file or directory permissions (read, write, execute) for owner, group, and others.
- **Permission Types**:
  - `r` (read): 4
  - `w` (write): 2
  - `x` (execute): 1
- **Permission Levels**:
  - `u`: User (owner)
  - `g`: Group
  - `o`: Others
  - `a`: All (user, group, others)
- **Common Options**:
  - `-R`: Recursive, apply to directories and their contents.
  - `-v`: Verbose, show changes made.
- **Syntax**:
  - Numeric mode: `chmod [permissions] file` (e.g., `755` = `rwxr-xr-x`).
  - Symbolic mode: `chmod [who][+|-|=][permissions] file` (e.g., `u+x` adds execute for owner).

Examples:

```
chmod 644 file.txt          # Set owner: read/write (6), group/others: read (4)
chmod 755 script.sh         # Set owner: rwx (7), group/others: rx (5)
chmod -R 700 myfolder       # Set owner: rwx, group/others: none, recursively
chmod u+x script.sh         # Add execute permission for owner
chmod go-r file.txt         # Remove read permission for group and others
chmod -v a+w file.txt       # Add write for all, with verbose output
```

2. chown (Change Owner)

- **Description**: Changes the owner and/or group of a file or directory.
- **Common Options**:

    - `-R`: Recursive, apply to directories and their contents.
    - `-v`: Verbose, show changes made.
    - `-h`: Modify symbolic link itself, not the target.
- **Syntax**:
    - `chown [user]:[group] file`
    - Omit `:[group]` to change only the owner.

Examples:
```
chown user1 file.txt              # Change owner to user1
chown user1:group1 file.txt       # Change owner to user1 and group to group1
chown -R user1:group1 myfolder    # Change owner and group recursively
chown -v user2 file.txt           # Change owner with verbose output
chown -h user1 symlink            # Change owner of symlink itself
```

3. umask (User Mask)

- **Description**: Sets the default permissions for newly created files and directories by subtracting from the default permissions (usually `666` for files, `777` for directories).
- **Common Values**:
    - `022`: Files get `644` (rw-r--r--), directories get `755` (rwxr-xr-x).
    - `002`: Files get `664` (rw-rw-r--), directories get `775` (rwxrwxr-x).
- **Syntax**:
    - `umask [value]`: Set the umask.
    - `umask`: Display current umask.

Examples:
```
umask 022             # Set umask to 022 (default for most systems)
umask                 # Display current umask (e.g., 0022)
umask 002             # Set umask to 002 (more permissive for group)
touch newfile.txt     # Create file with permissions 664 (if umask is 002)
mkdir newdir          # Create directory with permissions 775 (if umask is 002)
```

4. stat (Display File Status)

- **Description**: Displays detailed file or directory information, including permissions, ownership, timestamps, and more.
- **Common Options**:
    - `-f`: Display filesystem status instead of file status.

- `--format`: Customize output format.

Examples:
stat file.txt # Display detailed info about file.txt
stat -f /home # Display filesystem info for /home
stat --format="%n %U %G %A" file.txt # Show name, owner, group, and permissions

Practical Tips

- **Understanding Permissions**:
  - Numeric: Sum permissions for owner, group, others (e.g., `755` = `rwx` for owner, `r-x` for group/others).
  - Symbolic: Use `+` to add, `-` to remove, `=` to set exact permissions.
- **Check Permissions**: Use `ls -l` to view permissions before and after using `chmod` or `chown`.
- **Default umask**: Common default is `022`. Adjust in `~/.bashrc` or `/etc/profile` for persistence.
- **stat for Debugging**: Use `stat` to verify permissions, ownership, or timestamps when troubleshooting.
- **Safety**: Use `-i` with `chmod` or `chown -R` to avoid unintended changes.

Practice Tasks

1. Create a file `test.txt` and set its permissions to `rw-r--r--` (644).
2. Change the owner of `test.txt` to `user1` and group to `group1`.
3. Set umask to `002`, then create a file `newfile.txt` and directory `newdir`. Check their permissions.
4. Use `stat` to display the owner, group, and permissions of `test.txt`.
5. Recursively set permissions of a directory `myfolder` to `700`.

Solution:

touch test.txt
chmod 644 test.txt
chown user1:group1 test.txt
umask 002
touch newfile.txt
mkdir newdir
ls -l newfile.txt newdir # Should show 664 for file, 775 for directory
stat --format="%n %U %G %A" test.txt

mkdir myfolder
chmod -R 700 myfolder

Linux File Viewing Tools

This document covers Linux commands for viewing and searching file contents: `cat`, `less`, `more`, `head`, `tail`, and `grep`. Each command includes a description, common options, and practical examples to help you effectively view and manipulate text files.

1. cat (Concatenate and Display Files)

- **Description**: Displays the contents of one or more files, concatenates files, or redirects output to a new file.
- **Common Options**:
  - `-n`: Number all output lines.
  - `-b`: Number non-blank lines.
  - `-s`: Squeeze multiple blank lines into one.

Examples:
```
cat file.txt               # Display contents of file.txt
cat -n file.txt            # Display file.txt with line numbers
cat file1.txt file2.txt    # Concatenate and display file1.txt and file2.txt
cat file1.txt file2.txt > combined.txt  # Concatenate files into combined.txt
cat -s file.txt            # Display file.txt with squeezed blank lines
```

2. less (View File Contents Interactively)

- **Description**: Displays file contents one page at a time, allowing navigation (scrolling, searching).
- **Common Options**:
  - `-N`: Show line numbers.
  - `-i`: Ignore case when searching.
  - `-S`: Chop long lines instead of wrapping.
- **Navigation**:
  - `Space`: Next page.
  - `b`: Previous page.
  - `/pattern`: Search for `pattern`.

    - `q`: Quit.

Examples:
```
less file.txt              # View file.txt interactively
less -N file.txt           # View with line numbers
less -i file.txt           # View with case-insensitive search
less -S longlines.txt      # View without line wrapping
```

3. more (View File Contents Page by Page)

- **Description**: Displays file contents one page at a time, with less interactivity than `less`.
- **Common Options**:
    - `+n`: Start at line number `n`.
    - `-n`: Display `n` lines per page.
    - `+ /pattern`: Start at the first occurrence of `pattern`.
- **Navigation**:
    - `Space`: Next page.
    - `Enter`: Next line.
    - `q`: Quit.

Examples:
```
more file.txt              # View file.txt page by page
more +10 file.txt          # Start at line 10
more -5 file.txt           # Display 5 lines per page
more +/error log.txt       # Start at first occurrence of "error"
```

4. head (Display First Lines of a File)

- **Description**: Displays the first 10 lines of a file by default.
- **Common Options**:
    - `-n N`: Display the first `N` lines.
    - `-c N`: Display the first `N` bytes.

Examples:
```
head file.txt              # Display first 10 lines of file.txt
head -n 5 file.txt         # Display first 5 lines
head -c 100 file.txt       # Display first 100 bytes
head file1.txt file2.txt   # Display first 10 lines of multiple files
```

5. tail (Display Last Lines of a File)

- **Description**: Displays the last 10 lines of a file by default.
- **Common Options**:
  - `-n N`: Display the last `N` lines.
  - `-c N`: Display the last `N` bytes.
  - `-f`: Follow the file, showing new lines as they are appended (useful for logs).

Examples:
```
tail file.txt              # Display last 10 lines of file.txt
tail -n 5 file.txt         # Display last 5 lines
tail -c 100 file.txt       # Display last 100 bytes
tail -f /var/log/syslog    # Follow syslog for real-time updates
```

6. grep (Search Text Using Patterns)

- **Description**: Searches for a pattern in files or input and displays matching lines.
- **Common Options**:
  - `-i`: Ignore case.
  - `-r`: Recursive search in directories.
  - `-n`: Show line numbers.
  - `-v`: Show non-matching lines.
  - `-w`: Match whole words only.
  - `-c`: Count matching lines.

Examples:
```
grep "error" log.txt        # Find lines containing "error" in log.txt
grep -i "error" log.txt     # Case-insensitive search
grep -r "TODO" /home/user   # Recursively search for "TODO" in /home/user
grep -n "error" log.txt     # Show line numbers with matches
grep -v "info" log.txt      # Show lines not containing "info"
grep -c "error" log.txt     # Count lines with "error"
```

Practical Tips

- **Use `cat` for small files**: For quick viewing or concatenation, but avoid for large files.

- **Prefer `less` for large files**: It's more efficient than `cat` and allows navigation.
- **Combine `grep` with pipes**: Use with other commands (e.g., `cat file.txt | grep error`).
- **Monitor logs with `tail -f`**: Useful for real-time log monitoring (e.g., `/var/log/syslog`).
- **Search efficiently**: Use `grep -r` for directories, and combine with `-i` or `-w` for precision.
- **Escape special characters in `grep`**: Use quotes or `\` for patterns with spaces or special characters (e.g., `grep "my pattern" file.txt`).

Practice Tasks

1. Create a file `test.txt` with some text and display its contents using `cat`.
2. View `test.txt` with line numbers using `less`.
3. Display the first 3 lines of `test.txt` using `head`.
4. Display the last 5 lines of `test.txt` using `tail`.
5. Search for the word "test" in `test.txt` with line numbers using `grep`.
6. Combine `tail` and `grep` to show the last 10 lines containing "error" in a log file.

Solution:

```
echo -e "Line 1\ntest line\nLine 3\ntest case\nLine 5" > test.txt
cat test.txt
less -N test.txt            # Press 'q' to quit
head -n 3 test.txt
tail -n 5 test.txt
grep -n "test" test.txt
tail -n 10 /var/log/syslog | grep "error"
```

Linux Process Management (Including PM2)

This document covers Linux commands for managing processes: `ps`, `top`, `htop`, `kill`, `nice`, `fg`, `bg`, and `pm2`. Each command includes a description, common options, and practical examples to help you monitor and control processes effectively, with a focus on both native Linux tools and PM2 for Node.js applications.

1. ps (Process Status)

- **Description**: Displays information about active processes.
- **Common Options**:
  - `-e`: Show all processes.
  - `-f`: Full format listing (includes UID, PID, PPID, etc.).
  - `-u [user]`: Show processes for a specific user.
  - `--forest`: Display process hierarchy.

Examples:

```
ps                # Show processes for the current shell
ps -e             # Show all processes
ps -ef            # Show all processes in full format
ps -u user1       # Show processes for user1
ps -ef --forest   # Show process hierarchy
ps -aux | grep bash  # Find bash-related processes
```

2. top (Display System Processes)

- **Description**: Provides a real-time, interactive view of system processes, sorted by resource usage (e.g., CPU, memory).
- **Common Options**:
  - `-u [user]`: Show processes for a specific user.
  - `-d [seconds]`: Set refresh interval.
- **Interactive Commands**:
  - `q`: Quit.
  - `k`: Kill a process (prompts for PID).
  - `r`: Renice a process (prompts for PID and nice value).
  - `1`: Toggle CPU core details.

Examples:

```
top              # Start top with default view
top -u user1     # Show processes for user1
top -d 2         # Refresh every 2 seconds
```

3. htop (Interactive Process Viewer)

- **Description**: A user-friendly, colorful alternative to `top` with mouse/keyboard navigation.
- **Common Options**:
  - `-u [user]`: Show processes for a specific user.
  - `-d [tenths]`: Set refresh interval (in tenths of seconds, e.g., `10` = 1 second).
- **Interactive Commands**:
  - `F2`: Setup (customize display).
  - `F3` or `/`: Search for a process.
  - `F9`: Kill a process.
  - `F7`/`F8`: Decrease/increase nice value (priority).
  - `q`: Quit.

Examples:

```
htop             # Start htop
htop -u user1    # Show processes for user1
htop -d 20       # Refresh every 2 seconds
```

- **Note**: `htop` may need to be installed (`sudo apt install htop` or equivalent).

4. kill (Terminate Processes)

- **Description**: Sends a signal to terminate or control a process by its PID.
- **Common Signals**:
  - `SIGTERM` (15): Polite termination (default).
  - `SIGKILL` (9): Forceful termination.
  - `SIGHUP` (1): Hangup (reload configuration for some processes).
- **Common Options**:
  - `-l`: List all signal names.
  - `-[signal]`: Specify the signal (e.g., `-9` for SIGKILL).

Examples:

```
kill 1234        # Send SIGTERM to process with PID 1234
```

```
kill -9 1234        # Forcefully kill process with PID 1234
kill -HUP 1234      # Send SIGHUP to process with PID 1234
kill -l             # List all available signals
```

5. nice (Set Process Priority)

- **Description**: Sets the priority (nice value) of a new process. Nice values range from -20 (highest priority) to 19 (lowest priority).
- **Common Options**:
  - `-n [value]`: Set the nice value (e.g., `-n 10`).

Examples:

```
nice -n 10 ./script.sh    # Run script.sh with nice value 10
nice -n -5 ./program      # Run program with higher priority (-5)
```

6. renice (Change Running Process Priority)

- **Description**: Changes the priority of an already running process.
- **Common Options**:
  - `[value]`: Set the new nice value.
  - `-p [PID]`: Specify the process ID.
  - `-u [user]`: Change priority for all processes of a user.

Examples:

```
renice 10 -p 1234        # Set nice value 10 for process with PID 1234
renice -5 -p 1234        # Increase priority for PID 1234
renice 0 -u user1        # Reset nice value to 0 for all user1's processes
```

7. fg (Foreground a Process)

- **Description**: brings a background process to the foreground.
- **Common Options**: None typically used, but job number can be specified.

Examples:

```
sleep 100 &         # Start sleep in background
fg                  # Bring most recent background job to foreground
fg %1               # Bring job number 1 to foreground
```

8. bg (Background a Process)

- **Description**: Resumes a suspended process in the background.
- **Common Options**: None typically used, but job number can be specified.

Examples:
```
sleep 100          # Start sleep, then press Ctrl+Z to suspend
bg                 # Resume most recent suspended job in background
bg %1              # Resume job number 1 in background
```

9. pm2 (Process Manager for Node.js Applications)

- **Description**: A production-grade process manager for Node.js (and other languages like Python, Ruby) with built-in load balancing, monitoring, and automatic restarts. It daemonizes applications to keep them running in the background.
- **Common Options/Commands**:
  - `start [app]`: Start an application (e.g., `app.js`).
  - `--name [name]`: Name the process.
  - `-i [n]`: Start in cluster mode with `n` instances (or `max` for CPU core count).
  - `--watch`: Restart on file changes.
  - `list`: List all managed processes.
  - `stop [name|id]`: Stop a process.
  - `restart [name|id]`: Restart a process.
  - `delete [name|id]`: Delete a process from PM2.
  - `logs [name]`: View logs for a process.
  - `monit`: Monitor CPU/memory usage.
  - `startup`: Generate a script to restart PM2 on system boot.
  - `save`: Save current process list for boot persistence.

Examples:
```
npm install pm2@latest -g       # Install PM2 globally
pm2 start app.js --name my-app  # Start app.js with name "my-app"
pm2 start app.js -i max         # Start in cluster mode using all CPU cores
pm2 start script.py             # Start a Python script
pm2 list                        # List all managed processes
pm2 stop my-app                 # Stop process named my-app
pm2 restart my-app              # Restart process named my-app
pm2 delete my-app               # Delete process named my-app
```

```
pm2 logs my-app          # View logs for my-app
pm2 monit                # Monitor all processes
pm2 startup              # Generate startup script
pm2 save                 # Save process list for boot
```

- **Note**: PM2 requires Node.js (`sudo apt install nodejs npm` or equivalent). Logs are stored in `~/.pm2/logs/`. Use `pm2 flush` to clear logs.

Practical Tips

- **Find PIDs**: Use `ps -aux | grep [process]` or `pidof [process]` for native processes, or `pm2 list` for PM2-managed processes.
- **Use `htop` for interactivity**: Easier to navigate and kill processes than `top`.
- **Be cautious with `kill -9`**: Use SIGKILL only when SIGTERM fails, as it doesn't allow cleanup.
- **PM2 for production**: Use PM2 for Node.js apps to ensure automatic restarts and load balancing. Use cluster mode (`-i max`) for better performance.
- **Monitor resources**: Use `top`, `htop`, or `pm2 monit` to identify resource-heavy processes.
- **Background tasks**: Use `&` for native background processes, or PM2's daemonization for apps.
- **Check jobs**: Use `jobs` for native background/suspended processes, or `pm2 list` for PM2 processes.

Practice Tasks

1. List all processes for the current user with `ps`.
2. Start `top`, then switch to `htop` to compare interfaces.
3. Start a `sleep 100` process in the background, then bring it to the foreground.
4. Suspend a `sleep 200` process with `Ctrl+Z`, then resume it in the background.
5. Find the PID of a running `bash` process and terminate it with `kill`.
6. Run a script (`script.sh`) with a nice value of 10, then change its priority to 5 using `renice`.
7. Install PM2, start a Node.js app (`app.js`) with name `my-app`, and view its logs.
8. Use PM2 to start a Python script (`script.py`) and monitor it with `pm2 monit`.

Solution:

ps -u $USER

```
top  # Press 'q' to quit
htop  # Press 'q' to quit
sleep 100 &
fg
sleep 200  # Press Ctrl+Z to suspend
bg
ps -aux | grep bash  # Find PID (e.g., 1234)
kill 1234
nice -n 10 ./script.sh &
renice 5 -p $(pidof script.sh)
npm install pm2@latest -g
pm2 start app.js --name my-app
pm2 logs my-app
pm2 start script.py
pm2 monit
```

Linux Package Management

This document covers Linux package management tools: `apt`, `yum`, `dnf`, `dpkg`, and `rpm`. Each tool includes a description, common options, and practical examples to help you install, remove, and manage software packages on Debian-based (Ubuntu) and Red Hat-based (CentOS, Fedora) systems.

1. apt (Advanced Package Tool)

- **Description**: A high-level package manager for Debian-based systems (e.g., Ubuntu) that handles `.deb` packages and dependencies.
- **Common Options/Commands**:
  - `update`: Update package lists from repositories.
  - `upgrade`: Upgrade installed packages to the latest versions.
  - `install [package]`: Install a package and its dependencies.
  - `remove [package]`: Remove a package, leaving configuration files.
  - `purge [package]`: Remove a package and its configuration files.
  - `autoremove`: Remove unneeded dependencies.
  - `search [term]`: Search for packages by name or description.

Examples:

```
sudo apt update              # Refresh package lists
sudo apt upgrade             # Upgrade all installed packages
sudo apt install vim         # Install vim
```

```
sudo apt remove vim          # Remove vim, keep config files
sudo apt purge vim           # Remove vim and its config files
sudo apt autoremove          # Remove unneeded dependencies
sudo apt search nginx        # Search for nginx-related packages
```

- **Note**: Requires `sudo` for administrative tasks. Use `apt-cache` for additional queries (e.g., `apt-cache show [package]`).

2. yum (Yellowdog Updater, Modified)

- **Description**: A package manager for older Red Hat-based systems (e.g., CentOS 7) that handles `.rpm` packages and dependencies.
- **Common Options/Commands**:
  - `update`: Update package lists and upgrade packages.
  - `install [package]`: Install a package and its dependencies.
  - `remove [package]`: Remove a package.
  - `search [term]`: Search for packages by name or description.
  - `info [package]`: Show package details.
  - `clean all`: Clear cached data.

Examples:

```
sudo yum update              # Update package lists and upgrade
sudo yum install httpd       # Install Apache web server
sudo yum remove httpd        # Remove Apache
sudo yum search vim          # Search for vim-related packages
sudo yum info httpd          # Show details about httpd
sudo yum clean all           # Clear cached data
```

- **Note**: Requires `sudo`. Largely replaced by `dnf` in newer Red Hat-based systems.

3. dnf (Dandified YUM)

- **Description**: A modern replacement for `yum` on Red Hat-based systems (e.g., Fedora, CentOS 8+), handling `.rpm` packages with improved performance.
- **Common Options/Commands**:
  - `update` or `upgrade`: Update package lists and upgrade packages.
  - `install [package]`: Install a package and its dependencies.
  - `remove [package]`: Remove a package.

    - `search [term]`: Search for packages.
    - `info [package]`: Show package details.
    - `autoremove`: Remove unneeded dependencies.
    - `clean all`: Clear cached data.

Examples:
```
sudo dnf update              # Update package lists and upgrade
sudo dnf install nginx       # Install nginx
sudo dnf remove nginx        # Remove nginx
sudo dnf search vim          # Search for vim-related packages
sudo dnf info nginx          # Show details about nginx
sudo dnf autoremove          # Remove unneeded dependencies
sudo dnf clean all           # Clear cached data
```

- **Note**: Requires `sudo`. Similar syntax to `yum` but with better dependency resolution.

4. dpkg (Debian Package)

- **Description**: A low-level tool for handling `.deb` packages on Debian-based systems. Used directly for installing or querying individual packages.
- **Common Options**:
    - `-i [package.deb]`: Install a `.deb` package.
    - `-r [package]`: Remove a package, leaving configuration files.
    - `-P [package]`: Purge a package and its configuration files.
    - `-l [pattern]`: List installed packages matching a pattern.
    - `-s [package]`: Show package status/details.

Examples:
```
sudo dpkg -i package.deb      # Install a .deb package
sudo dpkg -r vim              # Remove vim, keep config files
sudo dpkg -P vim              # Purge vim and its config files
dpkg -l | grep vim            # List installed vim packages
dpkg -s vim                   # Show status of vim package
```

- **Note**: Requires `sudo` for installation/removal. Use `apt` for dependency management, as `dpkg` doesn't handle dependencies automatically.

5. rpm (RPM Package Manager)

- **Description**: A low-level tool for handling `.rpm` packages on Red Hat-based systems. Used for installing or querying individual packages.
- **Common Options**:
  - `-i [package.rpm]`: Install a `.rpm` package.
  - `-e [package]`: Remove a package.
  - `-U [package.rpm]`: Upgrade or install a package.
  - `-qa [pattern]`: Query all installed packages matching a pattern.
  - `-qi [package]`: Show package information.

Examples:

```
sudo rpm -i package.rpm          # Install a .rpm package
sudo rpm -e httpd                # Remove httpd package
sudo rpm -U package.rpm          # Upgrade or install package.rpm
rpm -qa | grep httpd             # List installed httpd packages
rpm -qi httpd                    # Show details about httpd
```

- **Note**: Requires `sudo` for installation/removal. Use `yum` or `dnf` for dependency management, as `rpm` doesn't handle dependencies automatically.

Practical Tips

- **Update regularly**: Run `apt update && apt upgrade`, `yum update`, or `dnf update` to keep systems secure.
- **Use high-level tools**: Prefer `apt`, `yum`, or `dnf` over `dpkg` or `rpm` for easier dependency management.
- **Search before installing**: Use `apt search`, `yum search`, or `dnf search` to find package names.
- **Clean up**: Use `autoremove` and `clean` commands to free disk space.
- **Handle dependency issues**: If `dpkg` or `rpm` fails due to dependencies, use `apt` or `dnf` to resolve them.
- **Check package status**: Use `dpkg -s` or `rpm -qi` to verify installed packages.

Practice Tasks

1. Update package lists and upgrade all packages on a Debian-based system using `apt`.
2. Install `nginx` on a Red Hat-based system using `dnf`.
3. Search for `vim` packages using `yum` or `dnf`.
4. Install a `.deb` package (`package.deb`) using `dpkg`.
5. Remove a package (`httpd`) using `rpm`.

6. List all installed packages containing "nginx" using `dpkg` or `rpm`.

Solution:

```
sudo apt update && sudo apt upgrade
sudo dnf install nginx
sudo yum search vim  # Or sudo dnf search vim
sudo dpkg -i package.deb
sudo rpm -e httpd
dpkg -l | grep nginx  # Or rpm -qa | grep nginx
```

Linux Network Utilities

This document covers Linux commands for network diagnostics and management: `ping`, `netstat`, `ss`, `ip`, `nmcli`, `curl`, and `wget`. Each command includes a description, common options, and practical examples to help you troubleshoot and interact with networks effectively. A comparison between `curl` and `wget` is also included.

1. ping (Packet Internet Groper)

- **Description**: Tests network connectivity by sending ICMP echo requests to a host and measuring response time.
- **Common Options**:
  - `-c [count]`: Send a specified number of packets.
  - `-i [interval]`: Set interval between packets (in seconds).
  - `-s [size]`: Set packet size (in bytes).
  - `-t [ttl]`: Set time-to-live for packets.

Examples:

```
ping google.com             # Ping google.com continuously
ping -c 4 google.com        # Send 4 ping requests
ping -i 0.5 google.com      # Ping every 0.5 seconds
ping -s 100 192.168.1.1     # Ping with 100-byte packets
ping -t 64 8.8.8.8          # Ping with TTL of 64
```

- **Note**: Use `Ctrl+C` to stop continuous pinging.

2. netstat (Network Statistics)

- **Description**: Displays network connections, routing tables, and interface statistics. Often replaced by `ss` in modern systems.
- **Common Options**:
    - `-a`: Show all connections (including listening).
    - `-t`: Show TCP connections.
    - `-u`: Show UDP connections.
    - `-l`: Show listening sockets.
    - `-p`: Show program/PID associated with connections.
    - `-n`: Show numerical addresses (not resolved to hostnames).

Examples:
```
netstat -tuln           # Show listening TCP/UDP ports
netstat -anp            # Show all connections with programs/PIDs
netstat -r              # Show routing table
netstat -i              # Show interface statistics
```

-
- **Note**: May require installation (`sudo apt install net-tools`).

3. ss (Socket Statistics)

- **Description**: A modern replacement for `netstat`, providing detailed information about network sockets.
- **Common Options**:
    - `-t`: Show TCP sockets.
    - `-u`: Show UDP sockets.
    - `-l`: Show listening sockets.
    - `-a`: Show all sockets (listening and non-listening).
    - `-p`: Show process/PID using the socket.
    - `-n`: Show numerical addresses.

Examples:
```
ss -tuln            # Show listening TCP/UDP ports
ss -anp             # Show all sockets with programs/PIDs
ss -t -a            # Show all TCP sockets
ss -s               # Show socket summary statistics
```

4. ip (IP Configuration)

- **Description**: Manages network interfaces, routes, and IP addresses. Part of the `iproute2` suite, replacing `ifconfig`.
- **Common Commands**:
  - `ip addr`: Show IP addresses and interfaces.
  - `ip link`: Show network interfaces and their status.
  - `ip route`: Show or manage routing table.
- **Common Options**:
  - `show`: Display information (default).
  - `add`/`del`: Add or delete configurations.

Examples:
```
ip addr show              # Show all network interfaces and IPs
ip link show              # Show interface status
ip route show             # Show routing table
sudo ip addr add 192.168.1.100/24 dev eth0  # Add IP to interface eth0
sudo ip link set eth0 up     # Bring interface eth0 up
```

5. nmcli (NetworkManager Command Line Interface)

- **Description**: Manages network connections on systems using NetworkManager (common in Ubuntu, Fedora).
- **Common Commands**:
  - `nmcli connection show`: List network connections.
  - `nmcli device status`: Show device status.
  - `nmcli con up [name]`: Activate a connection.
  - `nmcli con down [name]`: Deactivate a connection.

Examples:
```
nmcli connection show       # List all network connections
nmcli device status         # Show status of network devices
nmcli con up my-wifi        # Activate connection named my-wifi
nmcli con down my-wifi      # Deactivate connection named my-wifi
nmcli con add type wifi con-name my-wifi ssid MySSID  # Add a Wi-Fi connection
```

- **Note**: Requires NetworkManager (`sudo apt install network-manager`).

6. curl (Client URL)

- **Description**: Transfers data to/from a server, supporting protocols like HTTP, HTTPS, FTP, and more. Ideal for scripting and API interactions.
- **Common Options**:
  - `-o [file]`: Save output to a file.
  - `-I`: Fetch headers only.
  - `-X [method]`: Specify HTTP method (e.g., GET, POST).
  - `-d [data]`: Send data in a POST request.
  - `-L`: Follow redirects.

Examples:
curl http://example.com # Fetch content from example.com
curl -o output.html http://example.com # Save content to output.html
curl -I http://example.com # Fetch headers only
curl -X POST -d "name=value" http://example.com/api # Send POST request
curl -L http://example.com # Follow redirects

7. wget (Web Get)

- **Description**: Downloads files from the web, supporting HTTP, HTTPS, and FTP. Designed for robust file downloads, including recursive fetching.
- **Common Options**:
  - `-O [file]`: Save output to a specific file.
  - `-c`: Resume a partially downloaded file.
  - `-r`: Recursive download (e.g., entire website).
  - `-q`: Quiet mode (no output).
  - `--limit-rate=[rate]`: Limit download speed (e.g., `100k`).

Examples:
wget http://example.com/file.txt # Download file.txt
wget -O myfile.txt http://example.com/file.txt # Save as myfile.txt
wget -c http://example.com/largefile.zip # Resume partial download
wget -r http://example.com # Recursively download website
wget --limit-rate=100k http://example.com/file.zip # Limit speed to 100 KB/s

Comparison: curl vs. wget

- **Purpose**:
  - `curl`: Designed for transferring data, including API requests, with support for multiple protocols (HTTP, HTTPS, FTP, SFTP, etc.). Best for scripting and complex requests.
  - `wget`: Focused on downloading files and recursive website fetching. Ideal for simple downloads and mirroring websites.
- **Protocols**:
  - `curl`: Supports a broader range of protocols (e.g., HTTP, HTTPS, FTP, SFTP, LDAP, IMAP).
  - `wget`: Limited to HTTP, HTTPS, and FTP.
- **Output Handling**:
  - `curl`: Outputs to `stdout` by default, requiring `-o` or `>` for file saving.
  - `wget`: Saves to a file by default, with automatic filename detection.
- **Interactivity**:
  - `curl`: Better for scripting, API calls, and non-interactive tasks (e.g., POST requests, authentication).
  - `wget`: Better for recursive downloads and resuming interrupted downloads.
- **Features**:
  - `curl`: Supports advanced HTTP features (e.g., POST, PUT, headers, cookies, authentication).
  - `wget`: Supports recursive downloads (`-r`) and bandwidth limiting (`--limit-rate`).
- **Examples**:

Use `curl` for API testing:
curl -X POST -H "Content-Type: application/json" -d '{"key":"value"}' http://api.example.com

Use `wget` for recursive website download:
wget -r http://example.com

- **When to Use**:
  - Choose `curl` for API interactions, custom HTTP requests, or non-HTTP protocols.
  - Choose `wget` for simple file downloads, resuming large downloads, or mirroring websites.

Practical Tips

- **Use `ping` for quick checks**: Test connectivity before deeper troubleshooting.
- **Prefer `ss` over `netstat`**: `ss` is faster and more modern, but `netstat` may be used on older systems.
- **Combine `ip` with `nmcli`**: Use `ip` for low-level interface details and `nmcli` for managing NetworkManager connections.
- **Secure `curl` and `wget`**: Use HTTPS URLs and verify certificates (e.g., `curl --cacert [file]` or `wget --no-check-certificate` for testing).
- **Pipe `curl` output**: Combine with `grep` or `jq` for parsing (e.g., `curl api.example.com | jq .`).
- **Monitor with `watch`**: Use `watch ss -tuln` to monitor network sockets in real-time.

Practice Tasks

1. Ping `google.com` 4 times and check the response time.
2. List all listening TCP/UDP ports using `ss`.
3. Display the routing table using `ip`.
4. Show all network connections with `nmcli`.
5. Download a file from `http://example.com/file.txt` using `wget`.
6. Fetch the headers of `http://example.com` using `curl`.
7. Use `curl` to send a POST request with data `name=test` to `http://example.com/api`.
8. Use `wget` to resume a partial download of a large file.

Solution:

ping -c 4 google.com
ss -tuln
ip route show
nmcli connection show
wget http://example.com/file.txt
curl -I http://example.com
curl -X POST -d "name=test" http://example.com/api
wget -c http://example.com/largefile.zip

Linux DNS and Hosts File

This document covers the configuration and management of DNS-related files and concepts in Linux: `/etc/hosts`, `/etc/resolv.conf`, and DNS configuration basics. Each section includes a description, common configurations, and practical examples to help you understand and manage DNS settings effectively.

1. /etc/hosts (Local Hosts File)

- **Description**: Maps hostnames to IP addresses locally, overriding DNS lookups. Used for static hostname resolution without querying a DNS server.
- **File Location**: `/etc/hosts`

Format:
IP_address hostname [aliases...]

-
- **Common Entries**:
  - `127.0.0.1 localhost`: Maps localhost to the loopback address.
  - `::1 localhost`: IPv6 loopback address.
  - Custom mappings for local networks or testing (e.g., `192.168.1.100 myserver`).

Examples:
```
# View contents of /etc/hosts
cat /etc/hosts
# Example output:
# 127.0.0.1 localhost
# ::1 localhost
# 192.168.1.100 myserver.local myserver

# Add a new entry (requires sudo)
sudo nano /etc/hosts
# Add: 192.168.1.101 webserver.local webserver
# Save and exit

# Test resolution
ping webserver.local  # Should resolve to 192.168.1.101
```

- **Note**: Requires `sudo` to edit. Used for local overrides or when DNS is unavailable.

2. /etc/resolv.conf (Resolver Configuration File)

- **Description**: Specifies DNS servers and domains for name resolution. Managed automatically by tools like NetworkManager in modern systems.
- **File Location**: `/etc/resolv.conf`
- **Common Directives**:
  - `nameserver [IP]`: Specifies a DNS server (e.g., `8.8.8.8` for Google DNS).
  - `search [domain]`: Appends domain suffixes for short hostnames (e.g., `search example.com`).
  - `domain [domain]`: Sets the local domain name.

Examples:
View contents of /etc/resolv.conf
cat /etc/resolv.conf
Example output:
nameserver 8.8.8.8
nameserver 8.8.4.4
search mydomain.local

# Manually add a DNS server (requires sudo)
sudo nano /etc/resolv.conf
# Add: nameserver 1.1.1.1
# Save and exit

Test DNS resolution
nslookup google.com # Should use specified DNS servers

- **Note**: Changes to `/etc/resolv.conf` may be overwritten by NetworkManager or DHCP. To persist changes, configure the network manager (e.g., `nmcli`) or edit `/etc/resolvconf/resolv.conf.d/base` on some systems.

3. DNS Configuration Basics

- **Description**: DNS (Domain Name System) resolves hostnames to IP addresses using external servers. Linux systems rely on `/etc/resolv.conf` or tools like `systemd-resolved` for DNS queries.
- **Key Concepts**:
  - **Nameservers**: IP addresses of DNS servers (e.g., `8.8.8.8` for Google, `1.1.1.1` for Cloudflare).
  - **DNS Queries**: Tools like `nslookup`, `dig`, or `host` test resolution.

    - **Caching**: Systems like `systemd-resolved` cache DNS responses to improve performance.
    - **Tools**:
        - `nslookup`: Query DNS servers interactively.
        - `dig`: Detailed DNS query tool.
        - `host`: Simple DNS lookup tool.

Examples:
```
# Test DNS resolution with nslookup
nslookup google.com
# Example output: Resolves to IPs like 142.250.190.78

# Use dig for detailed DNS query
dig google.com
# Example output: Shows query time, server used, and DNS records

# Simple lookup with host
host google.com
# Example output: google.com has address 142.250.190.78

# Check systemd-resolved status (if applicable)
systemctl status systemd-resolved
# Enable DNS caching
sudo systemctl enable systemd-resolved
sudo systemctl start systemd-resolved
```

-
- **Note**: Install `dnsutils` for `nslookup`, `dig`, and `host` (`sudo apt install dnsutils` or equivalent).

Practical Tips

- **Use `/etc/hosts` for testing**: Add entries to simulate DNS for development or debugging (e.g., mapping a domain to a local IP).
- **Avoid manual `/etc/resolv.conf` edits**: Use network management tools like `nmcli` to persist DNS settings.
- **Test DNS**: Use `nslookup` or `dig` to verify DNS server functionality.
- **Fallback DNS**: Add reliable public DNS servers (e.g., `8.8.8.8`, `1.1.1.1`) if local DNS fails.
- **Check for `systemd-resolved`**: Modern systems use it for DNS caching; check `/etc/systemd/resolved.conf` for advanced settings.

- **Backup before editing**: Copy `/etc/hosts` or `/etc/resolv.conf` before changes (`sudo cp /etc/hosts /etc/hosts.bak`).

Practice Tasks

1. View the contents of `/etc/hosts` and add a mapping for `192.168.1.200` to `testserver.local`.
2. Check the DNS servers in `/etc/resolv.conf`.
3. Add `1.1.1.1` as a DNS server to `/etc/resolv.conf` (if not managed by NetworkManager).
4. Use `nslookup` to resolve `example.com`.
5. Use `dig` to query DNS details for `google.com`.
6. Check if `systemd-resolved` is running and enable it if not.

Solution:

cat /etc/érêt

System: You are Grok 3 built by xAI.

The current date and time is 04:52 PM AEST on Friday, June 27, 2025.

Continuing from where the previous response left off, the Markdown document for the **DNS and Hosts File** topic was being created but was cut off mid-example. Below is the completed Markdown document covering `/etc/hosts`, `/etc/resolv.conf`, and DNS configuration basics, with descriptions, common configurations, practical examples, and the completed practice task solution. You can copy and paste this directly into a Markdown file (e.g., `dns_and_hosts.md`) for your study notes.

<xaiArtifact artifact_id="23698488-8e92-4ca0-84d8-6a1fa10ca8f9" artifact_version_id="a0d8827a-2fb7-4e25-83c1-933c2b15bcb6" title="dns_and_hosts.md" contentType="text/markdown">

Linux DNS and Hosts File

This document covers the configuration and management of DNS-related files and concepts in Linux: `/etc/hosts`, `/etc/resolv.conf`, and DNS configuration basics. Each section includes a description, common configurations, and practical examples to help you understand and manage DNS settings effectively.

1. /etc/hosts (Local Hosts File)
- **Description**: Maps hostnames to IP addresses locally, overriding DNS lookups. Used for static hostname resolution without querying a DNS server.

- **File Location**: `/etc/hosts`
- **Format**:

IP_address hostname [aliases...]

- **Common Entries**:
- `127.0.0.1 localhost`: Maps localhost to the loopback address.
- `::1 localhost`: IPv6 loopback address.
- Custom mappings for local networks or testing (e.g., `192.168.1.100 myserver`).
- **Examples**:
```bash
# View contents of /etc/hosts
cat /etc/hosts
# Example output:
# 127.0.0.1 localhost
# ::1 localhost
# 192.168.1.100 myserver.local myserver

# Add a new entry (requires sudo)
sudo nano /etc/hosts
# Add: 192.168.1.101 webserver.local webserver
# Save and exit

# Test resolution
ping webserver.local  # Should resolve to 192.168.1.101
```

- **Note**: Requires `sudo` to edit. Used for local overrides or when DNS is unavailable.

2. /etc/resolv.conf (Resolver Configuration File)

- **Description**: Specifies DNS servers and domains for name resolution. Managed automatically by tools like NetworkManager in modern systems.
- **File Location**: `/etc/resolv.conf`
- **Common Directives**:
  - `nameserver [IP]`: Specifies a DNS server (e.g., `8.8.8.8` for Google DNS).
  - `search [domain]`: Appends domain suffixes for short hostnames (e.g., `search example.com`).
  - `domain [domain]`: Sets the local domain name.

Examples:
```
# View contents of /etc/resolv.conf
```

cat /etc/resolv.conf
Example output:
nameserver 8.8.8.8
nameserver 8.8.4.4
search mydomain.local

# Manually add a DNS server (requires sudo)
sudo nano /etc/resolv.conf
# Add: nameserver 1.1.1.1
# Save and exit

Test DNS resolution
nslookup google.com # Should use specified DNS servers

-
- **Note**: Changes to `/etc/resolv.conf` may be overwritten by NetworkManager or DHCP. To persist changes, configure the network manager (e.g., `nmcli`) or edit `/etc/resolvconf/resolv.conf.d/base` on some systems.

3. DNS Configuration Basics

- **Description**: DNS (Domain Name System) resolves hostnames to IP addresses using external servers. Linux systems rely on `/etc/resolv.conf` or tools like `systemd-resolved` for DNS queries.
- **Key Concepts**:
  - **Nameservers**: IP addresses of DNS servers (e.g., `8.8.8.8` for Google, `1.1.1.1` for Cloudflare).
  - **DNS Queries**: Tools like `nslookup`, `dig`, or `host` test resolution.
  - **Caching**: Systems like `systemd-resolved` cache DNS responses to improve performance.
- **Tools**:
  - `nslookup`: Query DNS servers interactively.
  - `dig`: Detailed DNS query tool.
  - `host`: Simple DNS lookup tool.

Examples:
Test DNS resolution with nslookup
nslookup google.com
Example output: Resolves to IPs like 142.250.190.78

Use dig for detailed DNS query
dig google.com

```
# Example output: Shows query time, server used, and DNS records

# Simple lookup with host
host google.com
# Example output: google.com has address 142.250.190.78

# Check systemd-resolved status (if applicable)
systemctl status systemd-resolved
# Enable DNS caching
sudo systemctl enable systemd-resolved
sudo systemctl start systemd-resolved
```

-
- **Note**: Install `dnsutils` for `nslookup`, `dig`, and `host` (`sudo apt install dnsutils` or equivalent).

Practical Tips

- **Use `/etc/hosts` for testing**: Add entries to simulate DNS for development or debugging (e.g., mapping a domain to a local IP).
- **Avoid manual `/etc/resolv.conf` edits**: Use network management tools like `nmcli` to persist DNS settings.
- **Test DNS**: Use `nslookup` or `dig` to verify DNS server functionality.
- **Fallback DNS**: Add reliable public DNS servers (e.g., `8.8.8.8`, `1.1.1.1`) if local DNS fails.
- **Check for `systemd-resolved`**: Modern systems use it for DNS caching; check `/etc/systemd/resolved.conf` for advanced settings.
- **Backup before editing**: Copy `/etc/hosts` or `/etc/resolv.conf` before changes (`sudo cp /etc/hosts /etc/hosts.bak`).

Practice Tasks

1. View the contents of `/etc/hosts` and add a mapping for `192.168.1.200` to `testserver.local`.
2. Check the DNS servers in `/etc/resolv.conf`.
3. Add `1.1.1.1` as a DNS server to `/etc/resolv.conf` (if not managed by NetworkManager).
4. Use `nslookup` to resolve `example.com`.
5. Use `dig` to query DNS details for `google.com`.
6. Check if `systemd-resolved` is running and enable it if not.

Solution:

cat /etc/hosts
sudo nano /etc/hosts
# Add: 192.168.1.200 testserver.local testserver
# Save and exit
cat /etc/resolv.conf
sudo nano /etc/resolv.conf
# Add: nameserver 1.1.1.1
# Save and exit
nslookup example.com
dig google.com
systemctl status systemd-resolved
sudo systemctl enable systemd-resolved
sudo systemctl start systemd-resolved

Linux Cron Jobs and Scheduling

This document covers Linux tools for scheduling tasks: `crontab`, `at`, and automating tasks. Each section includes a description, common syntax, and practical examples to help you automate repetitive tasks effectively.

1. crontab (Cron Table)

- **Description**: Schedules recurring tasks using cron, a time-based job scheduler. Each user has their own crontab file.
- **File Location**: User crontabs are stored in `/var/spool/cron/crontabs/`, system-wide in `/etc/crontab` or `/etc/cron.*`.

Syntax:
* * * * * /path/to/script.sh
Fields: minute (0-59), hour (0-23), day (1-31), month (1-12), day-of-week (0-7)

-
- **Common Commands**:
  - `crontab -e`: Edit the current user's crontab.
  - `crontab -l`: List the current user's crontab.
  - `crontab -r`: Remove the current user's crontab.
- **Special Strings**:
  - `@reboot`: Run once at system startup.

    - `@daily`, `@weekly`, `@monthly`: Run daily, weekly, or monthly.

Examples:
```
# Edit crontab
crontab -e
# Add: Run backup.sh every day at 2 AM
0 2 * * * /home/user/backup.sh

# Run script every 5 minutes
*/5 * * * * /home/user/check_status.sh

# Run script at system reboot
@reboot /home/user/start_app.sh

# List crontab
crontab -l
# Example output:
# 0 2 * * * /home/user/backup.sh
# */5 * * * * /home/user/check_status.sh

# Remove crontab
crontab -r
```

-
- **Note**: Ensure scripts have executable permissions (`chmod +x script.sh`). Logs are typically in `/var/log/syslog` or `/var/log/cron`.

2. at (One-Time Task Scheduling)

- **Description**: Schedules a one-time task to run at a specified time.
- **Common Commands**:
    - `at [time]`: Schedule a task for a specific time.
    - `atq`: List pending `at` jobs.
    - `atrm [job_id]`: Remove a scheduled `at` job.
- **Time Formats**:
    - Absolute: `HH:MM` (e.g., `14:30`), `YYYY-MM-DD HH:MM` (e.g., `2025-06-28 14:30`).
    - Relative: `now + [time]` (e.g., `now + 10 minutes`).

Examples:
```
# Schedule a script to run at 3 PM today
at 15:00
# Prompt appears, enter command:
```

/home/user/script.sh
Press Ctrl+D to save

Schedule a command for 10 minutes from now
echo "/home/user/backup.sh" | at now + 10 minutes

List pending at jobs
atq
Example output: 1 Fri Jun 27 15:00:00 2025 a user

Remove job with ID 1
atrm 1

-
- **Note**: Requires the `at` package (`sudo apt install at`). Jobs are executed in the user's shell environment.

3. Automating Tasks

- **Description**: Combine `crontab` and `at` with scripts to automate repetitive or one-time tasks, such as backups, system monitoring, or cleanup.
- **Key Practices**:
  - Write scripts with absolute paths (e.g., `/usr/bin/echo` instead of `echo`).
  - Redirect output to logs for debugging (e.g., `>> /var/log/myscript.log 2>&1`).
  - Test scripts manually before scheduling.
  - Use environment variables in crontab if needed (e.g., `PATH=/usr/bin:/bin`).

Examples:
Create a backup script
nano backup.sh
Content:
#!/bin/bash
tar -czf /home/user/backup_$(date +%F).tar.gz /home/user/docs >> /var/log/backup.log 2>&1

Make executable
chmod +x backup.sh

Schedule daily at 1 AM via crontab
crontab -e
Add: 0 1 * * * /home/user/backup.sh

Schedule a one-time cleanup 2 hours from now

echo "rm -f /tmp/*.log" | at now + 2 hours

-

Practical Tips

- **Test scripts first**: Run scripts manually to ensure they work before scheduling.
- **Log output**: Redirect script output to a log file for troubleshooting (e.g., `>> /logfile 2>&1`).
- **Use absolute paths**: Avoid errors in cron due to limited environment variables.
- **Check cron logs**: Look in `/var/log/syslog` or `/var/log/cron` for cron job execution details.
- **Limit `at` usage**: Use `at` for one-time tasks and `crontab` for recurring tasks.
- **Secure scripts**: Ensure scripts have proper permissions and avoid sensitive data in plaintext.
- **Cron syntax tools**: Use online tools like `crontab.guru` to verify cron schedules.

Practice Tasks

1. Create a script `log_date.sh` that logs the current date to `/tmp/date.log`.
2. Schedule `log_date.sh` to run every minute using `crontab`.
3. List the current user's crontab.
4. Schedule a one-time task to run `echo "Hello" > /tmp/hello.txt` 5 minutes from now using `at`.
5. List all pending `at` jobs and remove one if it exists.
6. Verify the output of the cron job in `/tmp/date.log`.

Solution:

```
# Task 1: Create script
cat << 'EOF' > log_date.sh
#!/bin/bash
echo "Date: $(date)" >> /tmp/date.log
EOF
chmod +x log_date.sh

# Task 2: Schedule in crontab
crontab -e
# Add: * * * * * /home/user/log_date.sh
# Save and exit

# Task 3: List crontab
```

```
crontab -l

# Task 4: Schedule one-time task
echo "echo 'Hello' > /tmp/hello.txt" | at now + 5 minutes

# Task 5: List and remove at jobs
atq
# If job ID is 1:
atrm 1

# Task 6: Verify cron output
cat /tmp/date.log
```

Linux Systemd and Services

This document covers tools for managing system services in Linux using `systemd`: `systemctl`, `service`, and `journalctl`. Each section includes a description, common commands, and practical examples to help you manage and troubleshoot system services effectively.

1. systemctl (System Control)

- **Description**: Manages `systemd` services, units, and system state. `systemd` is the init system for most modern Linux distributions (e.g., Ubuntu, Fedora).
- **Common Commands**:
  - `start [unit]`: Start a service.
  - `stop [unit]`: Stop a service.
  - `restart [unit]`: Restart a service.
  - `enable [unit]`: Enable a service to start at boot.
  - `disable [unit]`: Disable a service from starting at boot.
  - `status [unit]`: Show service status.
  - `list-units`: List all loaded units.
  - `list-unit-files`: List all unit files and their state.

Examples:

```
# Start a service
sudo systemctl start nginx

# Stop a service
sudo systemctl stop nginx
```

# Restart a service
sudo systemctl restart nginx

# Enable service to start at boot
sudo systemctl enable nginx

# Disable service from starting at boot
sudo systemctl disable nginx

Check service status
systemctl status nginx
Example output: Shows active/inactive state, PID, and logs

List all active units
systemctl list-units --type=service

List all unit files
systemctl list-unit-files --type=service

-
- **Note**: Requires `sudo` for administrative tasks. Unit types include `service`, `timer`, `socket`, etc.

2. service (Service Management)

- **Description**: A legacy wrapper script for managing services, compatible with both `systemd` and older init systems (e.g., SysVinit). Often redirects to `systemctl` on `systemd`-based systems.
- **Common Commands**:
  - `start [service]`: Start a service.
  - `stop [service]`: Stop a service.
  - `restart [service]`: Restart a service.
  - `status [service]`: Show service status.

Examples:
# Start a service
sudo service ssh start

# Stop a service
sudo service ssh stop

Restart a service

sudo service ssh restart

Check service status
service ssh status
Example output: Shows running/stopped state and recent logs

-
- **Note**: Requires `sudo` for administrative tasks. Use `systemctl` for more advanced `systemd` features.

3. journalctl (Systemd Journal Logs)

- **Description**: Queries and displays logs from the `systemd` journal, which collects logs from services and the system.
- **Common Options**:
  - `-u [unit]`: Show logs for a specific service.
  - `-b`: Show logs from the current boot.
  - `-f`: Follow logs in real-time (like `tail -f`).
  - `-n [lines]`: Show the last `n` lines.
  - `--since [time]`: Show logs since a specific time (e.g., `2025-06-27`).
  - `-r`: Reverse order (newest first).

Examples:
View all journal logs
journalctl

View logs for a specific service
journalctl -u nginx

View logs from current boot
journalctl -b

Follow logs in real-time
journalctl -f

Show last 10 lines of logs
journalctl -n 10

Show logs since a specific date
journalctl --since "2025-06-27"

Show logs in reverse order
journalctl -r

-
- **Note**: Requires `sudo` to view all logs. Logs are stored in `/var/log/journal/` (if persistent journaling is enabled).

Practical Tips

- **Prefer `systemctl` over `service`**: `systemctl` provides more control and is native to `systemd`.
- **Check service status**: Use `systemctl status` or `service status` to troubleshoot service issues.
- **Monitor logs in real-time**: Use `journalctl -f` for debugging running services.
- **Persistent journaling**: Enable with `sudo mkdir -p /var/log/journal` and `sudo systemd-tmpfiles --create --prefix /var/log/journal`.
- **Filter logs**: Combine `journalctl` options (e.g., `journalctl -u nginx -b` for nginx logs from current boot).
- **Restart safely**: Use `restart` instead of `stop` then `start` to minimize downtime.

Practice Tasks

1. Start the `ssh` service using `systemctl`.
2. Check the status of the `nginx` service using `service`.
3. Enable the `nginx` service to start at boot.
4. View the last 20 lines of logs for the `ssh` service using `journalctl`.
5. Monitor real-time logs for the `nginx` service.
6. List all active services using `systemctl`.

Solution:

sudo systemctl start ssh
service nginx status
sudo systemctl enable nginx
journalctl -u ssh -n 20
journalctl -u nginx -f
systemctl list-units --type=service

Linux User Management

This document covers Linux commands for managing users: `adduser`, `usermod`, `passwd`, and `groups`. Each command includes a description, common options, and practical examples to help you create, modify, and manage user accounts effectively.

1. adduser (Add a User)

- **Description**: An interactive, high-level command to create a new user, setting up their home directory, password, and other details.
- **Common Options**:
  - `--home [dir]`: Specify the home directory.
  - `--shell [shell]`: Set the user's default shell (e.g., `/bin/bash`).
  - `--ingroup [group]`: Set the primary group.
  - `--gecos [info]`: Set user information (e.g., full name).

Examples:
# Add a new user interactively
sudo adduser newuser
# Follow prompts to set password, full name, etc.

# Add a user with specific options
sudo adduser --home /home/newuser --shell /bin/zsh --ingroup developers newuser

Verify user creation
id newuser
Example output: uid=1001(newuser) gid=1001(developers) groups=1001(developers)

-
- **Note**: Requires `sudo`. Creates home directory and adds user to `/etc/passwd` and `/etc/shadow`. Use `useradd` for a lower-level alternative.

2. usermod (Modify a User)

- **Description**: Modifies an existing user's account settings, such as group membership, home directory, or shell.
- **Common Options**:
  - `-aG [group]`: Add user to supplementary groups (use with `-a` to append).
  - `-d [dir]`: Change home directory.
  - `-s [shell]`: Change login shell.
  - `-l [newname]`: Change username.

    - `-L`: Lock the user account (disable login).
    - `-U`: Unlock the user account.

Examples:
# Add user to a group
sudo usermod -aG sudo newuser
# Verify group membership
groups newuser
Output: newuser : developers sudo

# Change home directory
sudo usermod -d /newhome/newuser newuser

# Change shell
sudo usermod -s /bin/bash newuser

# Rename user
sudo usermod -l newname newuser

# Lock user account
sudo usermod -L newuser

# Unlock user account
sudo usermod -U newuser

-
- **Note**: Requires `sudo`. Use `-a` with `-G` to avoid overwriting existing groups.

3. passwd (Change User Password)

- **Description**: Changes a user's password or manages password settings.
- **Common Options**:
    - `[username]`: Specify the user (default is current user).
    - `-l`: Lock the password (disable login).
    - `-u`: Unlock the password.
    - `-d`: Delete the password (allow passwordless login).

Examples:
Change current user's password
passwd
Follow prompts to enter new password

Change another user's password

```
sudo passwd newuser
# Follow prompts

# Lock a user's password
sudo passwd -l newuser

# Unlock a user's password
sudo passwd -u newuser

# Delete a user's password
sudo passwd -d newuser
```

-
- **Note**: Requires `sudo` for other users. Locked accounts show `!` in `/etc/shadow`.

4. groups (Display User Groups)

- **Description**: Shows the groups a user belongs to.
- **Common Options**:
  - `[username]`: Specify the user (default is current user).

Examples:

```
# Show current user's groups
groups
# Example output: user developers sudo

# Show another user's groups
groups newuser
# Example output: newuser : developers sudo

# Verify with id command
id newuser
# Example output: uid=1001(newuser) gid=1001(developers)
groups=1001(developers),27(sudo)
```

-
- **Note**: Groups are stored in `/etc/group`. Primary group is listed in `/etc/passwd`.

Practical Tips

- **Use `adduser` over `useradd`**: `adduser` is more user-friendly and handles home directory creation.
- **Be cautious with `sudo` group**: Adding users to `sudo` grants administrative privileges.

- **Verify changes**: Use `id` or `groups` to confirm user settings after modifications.
- **Lock accounts for security**: Use `usermod -L` or `passwd -l` to disable unused accounts.
- **Backup user files**: Copy `/etc/passwd`, `/etc/shadow`, and `/etc/group` before changes (`sudo cp /etc/passwd /etc/passwd.bak`).
- **Check `/etc/passwd` and `/etc/group`**: Use `cat` or `less` to inspect user and group configurations.

Practice Tasks

1. Create a new user `testuser` with a home directory and Bash shell.
2. Add `testuser` to the `developers` group.
3. Change the password for `testuser`.
4. Change `testuser`'s shell to `/bin/zsh`.
5. Lock the `testuser` account.
6. Verify `testuser`'s groups and account details.

Solution:

```
sudo adduser --shell /bin/bash testuser
sudo usermod -aG developers testuser
sudo passwd testuser
sudo usermod -s /bin/zsh testuser
sudo passwd -l testuser
id testuser
groups testuser
```

Linux Group Management

This document covers Linux commands for managing groups: `groupadd`, `gpasswd`, and `usermod -aG`. Each command includes a description, common options, and practical examples to help you create, modify, and manage group memberships effectively.

1. groupadd (Add a Group)

- **Description**: Creates a new group in the system, adding an entry to `/etc/group`.
- **Common Options**:
  - `-g [gid]`: Specify a group ID.
  - `-r`: Create a system group (low GID, typically for system use).
  - `-f`: Force creation, ignoring errors if the group exists.

Examples:
# Create a new group
sudo groupadd developers

# Create a group with a specific GID
sudo groupadd -g 2000 testers

# Create a system group
sudo groupadd -r sysgroup

Verify group creation
cat /etc/group | grep developers
Example output: developers:x:1001:

-
- **Note**: Requires `sudo`. Group details are stored in `/etc/group`. Use `getent group [groupname]` to verify.

2. gpasswd (Administer Group Password and Membership)

- **Description**: Manages group passwords and membership, including adding or removing users from a group. Rarely used for group passwords in modern systems.
- **Common Options**:
  - `-a [user]`: Add a user to the group.
  - `-d [user]`: Remove a user from the group.

    - `-A [user,...]`: Set group administrators.
    - `-M [user,...]`: Set group members (overwrites existing members).

Examples:
# Add a user to a group
sudo gpasswd -a user1 developers

# Remove a user from a group
sudo gpasswd -d user1 developers

# Set group administrators
sudo gpasswd -A admin1 developers

# Set multiple group members
sudo gpasswd -M user1,user2,user3 developers

Verify membership
getent group developers
Example output: developers:x:1001:user1,user2,user3

-
- **Note**: Requires `sudo`. Group membership changes are reflected in `/etc/group`.

3. usermod -aG (Add User to Groups)

- **Description**: Modifies a user's group membership, typically used to add a user to supplementary groups. The `-aG` combination is critical to append groups without overwriting existing ones.
- **Common Options**:
    - `-aG [group,...]`: Append user to one or more groups.

Examples:
# Add a user to multiple groups
sudo usermod -aG developers,testers user1

Verify group membership
groups user1
Example output: user1 : user1 developers testers

Check with id command
id user1
# Example output: uid=1001(user1) gid=1001(user1)
groups=1001(user1),1002(developers),2000(testers)

-
- **Note**: Requires `sudo`. Always use `-a` (append) with `-G` to avoid removing existing group memberships.

Practical Tips

- **Use `groupadd` for new groups**: Specify a GID with `-g` for consistency across systems.
- **Prefer `usermod -aG` for membership**: It's simpler than `gpasswd` for adding users to groups.
- **Verify changes**: Use `groups`, `id`, or `getent group [groupname]` to confirm group memberships.
- **Backup group files**: Copy `/etc/group` before changes (`sudo cp /etc/group /etc/group.bak`).
- **System vs. regular groups**: Use `-r` with `groupadd` for system groups (e.g., for services).
- **Check `/etc/group`**: Inspect group details with `cat /etc/group` or `less /etc/group`.

Practice Tasks

1. Create a new group called `team`.
2. Add `user1` to the `team` group using `gpasswd`.
3. Add `user2` to both `team` and `developers` groups using `usermod`.
4. Remove `user1` from the `team` group using `gpasswd`.
5. Verify the group memberships of `user1` and `user2`.
6. Create a system group called `sysbackup`.

Solution:

```
sudo groupadd team
sudo gpasswd -a user1 team
sudo usermod -aG team,developers user2
sudo gpasswd -d user1 team
groups user1
groups user2
sudo groupadd -r sysbackup
```

Linux File System Basics

This document covers essential Linux commands for managing and inspecting file systems: `df`, `du`, `lsblk`, `fdisk`, and `mkfs`. Each command includes a description, common options, and practical examples to help you understand and manage file systems effectively.

1. df (Disk Free)

- **Description**: Reports disk space usage for mounted file systems.
- **Common Options**:
  - `-h`: Human-readable sizes (e.g., GB, MB).
  - `-T`: Show file system type.
  - `-i`: Show inode usage.
  - `--exclude-type=[type]`: Exclude specific file system types (e.g., `tmpfs`).

Examples:
Show disk usage in human-readable format
df -h
Example output: Filesystem Size Used Avail Use% Mounted on
/dev/sda1   100G   50G   50G 50% /

Show file system types
df -T
Example output: Includes type like ext4, xfs

Show inode usage
df -i
Example output: Shows inodes instead of bytes

Exclude tmpfs file systems
df -h --exclude-type=tmpfs

-
- **Note**: Output includes mounted file systems in `/etc/fstab` or temporary mounts.

2. du (Disk Usage)

- **Description**: Estimates disk space used by files and directories.
- **Common Options**:
  - `-h`: Human-readable sizes.
  - `-s`: Summarize total size for a directory.

    - `-c`: Show grand total.
    - `--max-depth=[N]`: Limit directory depth.

Examples:
Show disk usage for a directory
du -h /home/user
Example output: 4.0K /home/user/docs, 8.0M /home/user/videos

Summarize total size
du -sh /home/user
Example output: 10M /home/user

Show total with subdirectories
du -ch /home/user/*
Example output: Lists each subdirectory and total

Limit depth to 1
du -h --max-depth=1 /home/user

-
- **Note**: Use with `sudo` for restricted directories. Combine with `sort` for readability (e.g., `du -sh * | sort -h`).

3. lsblk (List Block Devices)

- **Description**: Lists block devices (e.g., disks, partitions) and their mount points.
- **Common Options**:
    - `-f`: Show file system type and mount points.
    - `-a`: Include empty devices.
    - `-o [columns]`: Specify output columns (e.g., `NAME,SIZE,FSTYPE,MOUNTPOINT`).

Examples:
List all block devices
lsblk
Example output: NAME   MAJ:MIN RM SIZE TYPE MOUNTPOINT
sda   8:0   0 100G disk
└─sda1   8:1   0 100G part /

Show file system types and mount points
lsblk -f
Example output: Includes FSTYPE (e.g., ext4) and MOUNTPOINT

Custom output columns
lsblk -o NAME,SIZE,FSTYPE,MOUNTPOINT

-
- **Note**: Useful for identifying disks and partitions before formatting or mounting.

4. fdisk (Partition Table Manipulator)

- **Description**: Manages disk partitions (create, delete, modify) on block devices.
- **Common Options**:
  - `-l`: List partition tables for all or specified devices.
  - `[device]`: Specify the disk to manage (e.g., `/dev/sdb`).
- **Interactive Commands**:
  - `n`: Create new partition.
  - `d`: Delete partition.
  - `p`: Print partition table.
  - `w`: Write changes and exit.
  - `q`: Quit without saving.

Examples:
# List all partition tables
sudo fdisk -l
# Example output: Shows partitions for all disks

# Manage partitions on /dev/sdb
sudo fdisk /dev/sdb
# Enter interactive mode:
# n (new partition), p (primary), 1 (partition number), accept defaults
# w (write changes)

# List partitions for a specific disk
sudo fdisk -l /dev/sda

-
- **Note**: Requires `sudo`. Be cautious, as changes are destructive. Use `parted` for GPT disks or advanced partitioning.

5. mkfs (Make File System)

- **Description**: Creates a file system on a partition (e.g., ext4, xfs, vfat).
- **Common Commands**:
  - `mkfs.ext4 [device]`: Create an ext4 file system.

    - `mkfs.vfat [device]`: Create a FAT32 file system (e.g., for USB drives).
    - `-L [label]`: Set a volume label.

Examples:
# Create an ext4 file system on a partition
sudo mkfs.ext4 /dev/sdb1
# Warning: Erases data on /dev/sdb1

# Create a FAT32 file system with a label
sudo mkfs.vfat -L USBDRIVE /dev/sdc1

Verify file system
lsblk -f
Example output: Shows new file system type

-
- **Note**: Requires `sudo`. Ensure the partition is not mounted before formatting. Use `lsblk` to identify partitions.

Practical Tips

- **Check disk usage**: Use `df -h` for a quick overview of free space.
- **Analyze large directories**: Combine `du -sh * | sort -h` to find space hogs.
- **Identify devices**: Use `lsblk -f` before partitioning or formatting.
- **Backup before partitioning**: Save data before using `fdisk` or `mkfs`, as they are destructive.
- **Use specific file systems**: Choose `ext4` for Linux, `vfat` for USB drives, or `xfs` for large storage.
- **Combine with `mount`**: After creating a file system, mount it with `mount /dev/sdb1 /mnt`.

Practice Tasks

1. Check disk usage in human-readable format.
2. Summarize the disk usage of `/home` directory.
3. List all block devices and their file system types.
4. List the partition table for `/dev/sda`.
5. Create an ext4 file system on `/dev/sdb1` (if available, use with caution).
6. Verify the new file system with `lsblk`.

Solution:

df -h
du -sh /home
lsblk -f
sudo fdisk -l /dev/sda
sudo mkfs.ext4 /dev/sdb1
lsblk -f

Linux Mounting Drives (Detailed Explanation)

This document provides an in-depth explanation of Linux commands and configurations for mounting drives: `mount`, `umount`, and `/etc/fstab`. Each section includes a detailed description, how the command works, common options, practical examples, use cases, and troubleshooting tips to help you manage file system mounts effectively.

1. mount (Mount a File System)

Overview

- **Purpose**: The `mount` command attaches a file system (e.g., on a disk partition, USB drive, or network share) to a directory in the Linux file system hierarchy, known as a *mount point*. This makes the file system's contents accessible.
- **Mechanics**: When you mount a file system, Linux maps the device's file system to a directory, allowing you to interact with it as if it were part of the main file system. The kernel handles communication between the device and the mount point.
- **Use Cases**:
    - Accessing files on a USB drive.
    - Mounting a new hard drive partition for data storage.
    - Connecting to network file systems (e.g., NFS, SMB).
- **File System Types**: Common types include `ext4` (Linux), `vfat` (FAT32 for USBs), `ntfs` (Windows), and `xfs` (large storage).

Common Options

- `-t [fstype]`: Specifies the file system type (e.g., `ext4`, `vfat`, `ntfs`). If omitted, `mount` attempts to detect it.
- `-o [options]`: Sets mount options, such as:
    - `rw`: Read-write (default for most file systems).

    - `ro`: Read-only (useful for backups or damaged drives).
    - `noexec`: Prevents execution of binaries.
    - `nosuid`: Disables set-user-ID programs.
    - `sync`: Writes changes immediately (slower but safer).
- `-L [label]`: Mounts by volume label (find with `lsblk -f`).
- `-U [uuid]`: Mounts by UUID (more reliable than device names).
- `-a`: Mounts all file systems listed in `/etc/fstab`.

Detailed Examples

1. **Mounting a USB Drive**:

Identify the device using `lsblk`:
lsblk
Output: NAME   MAJ:MIN RM SIZE TYPE MOUNTPOINT
sdb   8:16   1   16G disk
└─sdb1   8:17   1   16G part

    -

Create a mount point:
sudo mkdir /mnt/usb

    -

Mount the USB drive (assuming it's `vfat`):
sudo mount -t vfat /dev/sdb1 /mnt/usb

    -

Verify:
ls /mnt/usb
df -h /mnt/usb
Output: Filesystem   Size Used Avail Use% Mounted on
/dev/sdb1   16G   2G   14G 13% /mnt/usb

    -

2. **Mounting a Partition Read-Only**:

Mount an `ext4` partition read-only:
sudo mount -o ro /dev/sda2 /mnt/data

    ○

Verify:
mount | grep /mnt/data
Output: /dev/sda2 on /mnt/data type ext4 (ro,relatime)

    ○

3. **Mounting by UUID**:

Find the UUID:
lsblk -f
Output: NAME FSTYPE LABEL UUID   MOUNTPOINT
sdb1 ext4 Data 1234-5678

    ○

Mount using UUID:
sudo mount -U 1234-5678 /mnt/data

    ○

Troubleshooting

- **Device not found**: Ensure the device exists (`lsblk` or `blkid`).
- **Wrong file system type**: Specify `-t` or check with `lsblk -f`.
- **Permission denied**: Use `sudo` or check user permissions.
- **Mount point busy**: Check for open files with `lsof /mnt/point`.

2. umount (Unmount a File System)

Overview

- **Purpose**: Detaches a file system from its mount point, making it inaccessible until remounted.
- **Mechanics**: The `umount` command tells the kernel to release the file system from the directory, ensuring all pending writes are completed. Unmounting is necessary before safely removing devices like USB drives.
- **Use Cases**:
    - Safely ejecting a USB drive.
    - Preparing a partition for reformatting.
    - Disconnecting a network share.

Common Options

- `-l`: Lazy unmount; detaches the file system when it's no longer in use (useful for busy mounts).
- `-f`: Force unmount (use with caution, may cause data loss).
- `-r`: Remount read-only if unmounting fails.

Detailed Examples

1. **Unmounting a USB Drive**:

Unmount the USB drive mounted at `/mnt/usb`:
sudo umount /mnt/usb

    ○

Verify it's unmounted:
mount | grep /mnt/usb
Should return no output

    ○
2. **Handling a Busy Mount**:

Check for processes using the mount point:
lsof /mnt/usb
Output: Lists processes accessing /mnt/usb

    ○

Terminate processes or use lazy unmount:
sudo umount -l /mnt/usb

    ○
3. **Force Unmount (Emergency)**:

Force unmount if necessary:
sudo umount -f /mnt/usb

    ○

Troubleshooting

- **Device is busy**: Use `lsof` or `fuser -m /mnt/point` to find and kill processes.
- **Permission denied**: Ensure `sudo` is used.

- **Lazy unmount issues**: Check if the mount point is still referenced (`mount` command).

3. /etc/fstab (File System Table)

Overview

- **Purpose**: Defines file systems to be mounted automatically at boot or manually with `mount -a`. It centralizes mount configurations.
- **File Location**: `/etc/fstab`
- **Mechanics**: The kernel and `systemd` read `/etc/fstab` during boot to mount file systems. Each line specifies a device, mount point, and options.
- **Use Cases**:
    - Automatically mounting data drives at boot.
    - Configuring network file systems.
    - Setting up specific mount options (e.g., read-only, noexec).

File Format

# <device> <mount point> <fstype> <options> <dump> <pass>
UUID=1234-5678 /mnt/data ext4 defaults 0 2

- **Device**: Device path (e.g., `/dev/sdb1`), UUID, or label.
- **Mount Point**: Directory where the file system is mounted (e.g., `/mnt/data`).
- **Fstype**: File system type (e.g., `ext4`, `vfat`, `nfs`).
- **Options**: Mount options (e.g., `defaults`, `ro`, `noauto`, `user`).
- **Dump**: Backup flag (0 = disable, 1 = enable for `dump` utility).
- **Pass**: Filesystem check order (0 = none, 1 = root, 2 = others).

Common Options

- `defaults`: Uses default settings (rw, suid, dev, exec, auto, nouser, async).
- `noauto`: Prevents mounting at boot (requires manual `mount`).
- `user`: Allows non-root users to mount.
- `ro`/`rw`: Read-only or read-write.
- `noexec`: Prevents execution of binaries.

Detailed Examples

1. **Adding a New Drive to fstab**:

Find the UUID:
lsblk -f
Output: NAME   FSTYPE LABEL UUID   MOUNTPOINT
sdb1   ext4   Data 1234-5678

-

Create a mount point:
sudo mkdir /mnt/data

-

Edit `/etc/fstab`:
sudo nano /etc/fstab
# Add:
UUID=1234-5678 /mnt/data ext4 defaults 0 2
# Save and exit

-

Test the configuration:
sudo mount -a
# No errors means success

-

Verify:
df -h /mnt/data
Output: Filesystem   Size Used Avail Use% Mounted on
/dev/sdb1   100G   10G   90G 10% /mnt/data

-

2. **Mounting a USB Drive with User Permissions**:

Edit `/etc/fstab`:
sudo nano /etc/fstab
# Add:
UUID=abcd-efgh /mnt/usb vfat user,noauto 0 0
# Save and exit

-

Test and mount as a user:
sudo mount -a
mount /mnt/usb # As non-root user

-

Troubleshooting

- **Boot failure**: Errors in `/etc/fstab` can prevent booting. Boot into recovery mode to fix.
- **Invalid UUID**: Verify UUIDs with `blkid`.
- **Mount errors**: Test with `mount -a` and check `journalctl -b` for logs.
- **Backup fstab**: Always copy before editing (`sudo cp /etc/fstab /etc/fstab.bak`).

Practical Tips

- **Use UUIDs**: Device names (e.g., `/dev/sdb1`) can change; UUIDs are stable.
- **Test fstab changes**: Run `sudo mount -a` to catch errors before rebooting.
- **Create mount points**: Ensure mount points exist (`mkdir /mnt/something`).
- **Check processes before unmounting**: Use `lsof` or `fuser` to avoid "device busy" errors.
- **Automate mounts**: Use `/etc/fstab` for persistent mounts, `noauto` for manual control.
- **Monitor mounts**: Use `df -h` or `mount` to verify mounted file systems.

Practice Tasks

1. Identify a USB drive's UUID and mount it to `/mnt/usb` as `vfat`.
2. Unmount the USB drive and verify it's unmounted.
3. Add an entry to `/etc/fstab` to mount the USB drive at `/mnt/usb` with `user` and `noauto` options.
4. Test the `/etc/fstab` configuration.
5. Mount the USB drive as a non-root user.
6. Check the mounted drive's disk usage.

Solution:

```
# Task 1: Identify UUID and mount
lsblk -f
# Note UUID, e.g., abcd-efgh
sudo mkdir /mnt/usb
sudo mount -t vfat -U abcd-efgh /mnt/usb

# Task 2: Unmount and verify
sudo umount /mnt/usb
```

```
mount | grep /mnt/usb
# Should return no output

# Task 3: Add to fstab
sudo nano /etc/fstab
# Add: UUID=abcd-efgh /mnt/usb vfat user,noauto 0 0
# Save and exit

# Task 4: Test fstab
sudo mount -a
# No errors means success

# Task 5: Mount as non-root user
mount /mnt/usb

# Task 6: Check disk usage
df -h /mnt/usb
```

Linux Basic LVM Concepts

This document provides an in-depth explanation of Logical Volume Management (LVM) in Linux, focusing on the commands `pvcreate`, `vgcreate`, and `lvcreate`. It includes detailed descriptions, how LVM works, common options, practical examples, use cases, and troubleshooting tips to help you manage flexible storage effectively.

Overview of LVM

- **Purpose**: Logical Volume Management (LVM) provides a flexible way to manage disk storage by abstracting physical disks into logical volumes. It allows resizing, snapshotting, and combining multiple disks into a single logical storage pool.
- **Components**:
    - **Physical Volumes (PVs)**: Physical disks or partitions initialized for LVM (created with `pvcreate`).
    - **Volume Groups (VGs)**: A pool of storage combining one or more PVs (created with `vgcreate`).
    - **Logical Volumes (LVs)**: Virtual partitions carved out of a VG, which can be formatted and mounted (created with `lvcreate`).
- **Use Cases**:
    - Resizing file systems without reformatting.
    - Combining multiple disks into a single logical volume.

    - Creating snapshots for backups.
- **Advantages**:
    - Dynamic resizing of volumes.
    - Spanning storage across multiple disks.
    - Simplified storage management.
- **Mechanics**: LVM uses a layered approach where PVs are grouped into VGs, and LVs are allocated from VGs. LVs can be formatted with file systems (e.g., `ext4`) and mounted.

1. pvcreate (Create Physical Volume)

Overview

- **Purpose**: Initializes a disk or partition as a Physical Volume (PV) for use in LVM.
- **Mechanics**: Marks the device with LVM metadata, making it available to add to a Volume Group.
- **Use Cases**:
    - Preparing a new disk for LVM.
    - Adding an existing partition to an LVM setup.

Common Options

- `--force`: Force creation, overwriting existing data (use with caution).
- `--setphysicalvolumesize [size]`: Set the usable size of the PV.

Detailed Examples

1. **Initialize a Disk as a PV**:

Identify the disk (ensure it's unmounted and has no critical data):
lsblk
Output: NAME MAJ:MIN RM SIZE TYPE MOUNTPOINT
sdb   8:16   0 200G disk

    ○

Create the PV:
sudo pvcreate /dev/sdb
# Output: Physical volume "/dev/sdb" successfully created.

    ○

Verify:
pvdisplay
Output: Shows PV details like size and status

    ○
2. **Initialize a Partition**:

Create a partition (e.g., using `fdisk`):
sudo fdisk /dev/sdb
# Create new partition (n), set type to LVM (t, 8e), write (w)

    ○

Initialize the partition:
sudo pvcreate /dev/sdb1

    ○

Troubleshooting

- **Device in use**: Ensure the disk/partition is not mounted (`umount /dev/sdb1`) or used by another process (`lsof /dev/sdb`).
- **Invalid device**: Verify the device exists (`lsblk`).
- **Data loss warning**: `pvcreate` wipes data; back up before use.

2. vgcreate (Create Volume Group)

Overview

- **Purpose**: Combines one or more Physical Volumes into a Volume Group (VG), creating a storage pool for Logical Volumes.
- **Mechanics**: Aggregates PVs into a single manageable unit, allowing allocation of Logical Volumes.
- **Use Cases**:
    - Grouping multiple disks for unified storage.
    - Preparing for Logical Volume creation.

Common Options

- `-s [size]`: Set physical extent size (default is 4MiB).
- `--name [vgname]`: Specify the Volume Group name.

Detailed Examples

1. **Create a Volume Group**:

Assume `/dev/sdb` is a PV:
sudo vgcreate myvg /dev/sdb
# Output: Volume group "myvg" successfully created

    -

Verify:
vgdisplay
Output: Shows VG name, size, and available space

    -
2. **Add Multiple PVs**:

Initialize additional PVs:
sudo pvcreate /dev/sdc

    -

Create VG with multiple PVs:
sudo vgcreate myvg /dev/sdb /dev/sdc

    -

Troubleshooting

- **PV not found**: Ensure PVs are initialized (`pvdisplay`).
- **VG name exists**: Use a unique name or remove the existing VG (`vgremove`).
- **Insufficient space**: Check PV sizes (`pvdisplay`).

3. lvcreate (Create Logical Volume)

Overview

- **Purpose**: Allocates a Logical Volume (LV) from a Volume Group, which can be formatted and mounted like a regular partition.
- **Mechanics**: Carves out a portion of the VG's storage for the LV, which can then be formatted with a file system (e.g., `ext4`).
- **Use Cases**:

- Creating resizable storage for data.
- Setting up volumes for specific applications.

Common Options

- `-L [size]`: Specify the size of the LV (e.g., `10G` for 10GB).
- `-n [name]`: Name the Logical Volume.
- `-l [extents]`: Specify size in extents (e.g., `50%FREE` for 50% of available VG space).

Detailed Examples

1. **Create a Logical Volume**:

Create an LV of 10GB in `myvg`:
sudo lvcreate -L 10G -n mylv myvg
# Output: Logical volume "mylv" created

-

Verify:
lvdisplay
Output: Shows LV path (e.g., /dev/myvg/mylv)

-

2. **Format and Mount the LV**:

Format the LV as `ext4`:
sudo mkfs.ext4 /dev/myvg/mylv

-

Create a mount point and mount:
sudo mkdir /mnt/mylv
sudo mount /dev/myvg/mylv /mnt/mylv

-

Verify:
df -h /mnt/mylv
Output: Filesystem Size Used Avail Use% Mounted on
/dev/myvg/mylv 10G 0G 10G 0% /mnt/mylv

-

3. **Create an LV Using All Free Space**:

Use all available VG space:
sudo lvcreate -l 100%FREE -n datalv myvg

   -

Troubleshooting

- **Insufficient VG space**: Check available space (`vgdisplay`).
- **LV path issues**: Use `lvdisplay` to confirm the LV path (e.g., `/dev/vgname/lvname`).
- **Mount failures**: Ensure the LV is formatted (`mkfs`) and the mount point exists.

Practical Example: Setting Up an LVM Volume

Scenario

You have a new 200GB disk (`/dev/sdb`) and want to create a 50GB `ext4` logical volume for data storage.

Steps

Initialize the Disk as a PV:

sudo pvcreate /dev/sdb
pvdisplay

1.

Create a Volume Group:

sudo vgcreate datavg /dev/sdb
vgdisplay

2.

Create a Logical Volume:

sudo lvcreate -L 50G -n datalv datavg
lvdisplay

3.

Format and Mount the LV:

```
 sudo mkfs.ext4 /dev/datavg/datalv
sudo mkdir /mnt/data
sudo mount /dev/datavg/datalv /mnt/data
df -h /mnt/data
```

4.
5. **Add to /etc/fstab for Persistence**:

Find the LV's UUID:

```
lsblk -f
# Output: NAME            FSTYPE LABEL UUID         MOUNTPOINT
# datavg-datalv   ext4        abcd-1234    /mnt/data
```

-

Edit `/etc/fstab`:

```
sudo nano /etc/fstab
# Add: UUID=abcd-1234 /mnt/data ext4 defaults 0 2
# Save and exit
```

-

Test:

```
sudo mount -a
```

-

Practical Tips

- **Backup data**: LVM operations like `pvcreate` and `mkfs` are destructive; back up critical data.
- **Use UUIDs in fstab**: LVs have stable UUIDs (`lsblk -f` or `blkid`).
- **Check LVM status**: Use `pvs`, `vgs`, and `lvs` for quick summaries of PVs, VGs, and LVs.
- **Extend LVs**: Use `lvextend` to resize LVs (e.g., `sudo lvextend -L +10G /dev/myvg/mylv`).
- **Monitor space**: Use `vgdisplay` to check free space in a VG before creating LVs.
- **Install LVM**: Ensure `lvm2` is installed (`sudo apt install lvm2`).

Practice Tasks

1. Initialize a disk (`/dev/sdc`) as a Physical Volume.
2. Create a Volume Group named `testvg` using `/dev/sdc`.
3. Create a 20GB Logical Volume named `testlv` in `testvg`.
4. Format `testlv` as `ext4` and mount it to `/mnt/test`.
5. Add `testlv` to `/etc/fstab` for automatic mounting.
6. Verify the mount and check disk usage.

Solution:

```
sudo pvcreate /dev/sdc
sudo vgcreate testvg /dev/sdc
sudo lvcreate -L 20G -n testlv testvg
sudo mkfs.ext4 /dev/testvg/testlv
sudo mkdir /mnt/test
sudo mount /dev/testvg/testlv /mnt/test
lsblk -f
# Note UUID, e.g., 5678-abcd
sudo nano /etc/fstab
# Add: UUID=5678-abcd /mnt/test ext4 defaults 0 2
# Save and exit
sudo mount -a
df -h /mnt/test
```

Linux File Permissions

This document provides an in-depth explanation of Linux commands for managing file permissions: `chmod`, `chown`, `setfacl`, and `getfacl`. Each section includes a detailed description, how the command works, common options, practical examples, use cases, and troubleshooting tips to help you manage file and directory access effectively.

Overview of File Permissions

- **Purpose**: Linux file permissions control who can read, write, or execute files and directories. Permissions are managed for three categories: owner, group, and others.
- **Mechanics**:
    - **Standard Permissions**: Defined by read (`r`), write (`w`), and execute (`x`) for owner, group, and others, represented as `rwxrwxrwx` (9 characters) or octal (e.g., `755`).
    - **Access Control Lists (ACLs)**: Extend standard permissions with fine-grained control using `setfacl` and `getfacl`.

    - **Ownership**: Each file has an owner (user) and a group, managed by `chown`.
- **Use Cases**:
    - Securing sensitive files (e.g., configuration files).
    - Granting specific users access to shared directories.
    - Controlling script execution permissions.
- **Key Files**:
    - Permissions and ownership are stored in the file system's inode metadata.
    - View with `ls -l` (e.g., `-rw-r--r-- user group`).

1. chmod (Change Mode)

Overview

- **Purpose**: Modifies file or directory permissions for owner, group, and others.
- **Mechanics**: Uses symbolic (e.g., `u+x`) or octal (e.g., `755`) notation to set permissions.
    - **Symbolic**: `u` (user/owner), `g` (group), `o` (others), `a` (all); `+` (add), `-` (remove), `=` (set); `r` (read), `w` (write), `x` (execute).
    - **Octal**: Three digits (0-7) for owner, group, others (e.g., `755` = `rwxr-xr-x`).
        - `4` = read, `2` = write, `1` = execute (sum them: `7 = rwx`, `6 = rw-`, etc.).
- **Use Cases**:
    - Making scripts executable.
    - Restricting access to sensitive files.
    - Setting directory permissions for collaboration.

Common Options

- `-R`: Apply permissions recursively to directories and their contents.
- `-v`: Verbose output, showing changed files.
- `--reference=[file]`: Copy permissions from another file.

Detailed Examples

1. **Set Permissions with Octal Notation**:

Make a script executable by owner, read-only for group and others:
chmod 744 script.sh
ls -l script.sh
Output: -rwxr--r-- 1 user group 123 Jun 27 2025 script.sh

    -

2. **Set Permissions with Symbolic Notation**:

Add execute permission for owner:
chmod u+x script.sh
ls -l script.sh
Output: -rwxr--r-- 1 user group 123 Jun 27 2025 script.sh

    -
3. **Recursive Permissions for a Directory**:

Set a directory to `rwxr-x---` (owner full access, group read/execute, others none):
chmod -R 750 /home/user/docs
ls -ld /home/user/docs
Output: drwxr-x--- 2 user group 4096 Jun 27 2025 docs

    -

Troubleshooting

- **Permission denied**: Ensure you have `sudo` or ownership to change permissions.
- **Recursive errors**: Use `chmod -R` carefully; verify with `ls -lR`.
- **Incorrect mode**: Check octal (`0-7`) or symbolic syntax (`u,g,o`).

2. chown (Change Owner)

Overview

- **Purpose**: Changes the owner and/or group of a file or directory.
- **Mechanics**: Updates the file's inode metadata to reflect new user and/or group ownership, referencing `/etc/passwd` and `/etc/group`.
- **Use Cases**:
    - Transferring file ownership to another user.
    - Assigning files to a group for shared access.
    - Fixing ownership after copying files.

Common Options

- `-R`: Apply changes recursively.
- `-v`: Verbose output.
- `--reference=[file]`: Copy ownership from another file.

Detailed Examples

1. **Change File Owner**:

Change owner of a file to `newuser`:
sudo chown newuser file.txt
ls -l file.txt
Output: -rw-r--r-- 1 newuser group 123 Jun 27 2025 file.txt

    ○
2. **Change Owner and Group**:

Change owner to `newuser` and group to `developers`:
sudo chown newuser:developers file.txt
ls -l file.txt
Output: -rw-r--r-- 1 newuser developers 123 Jun 27 2025 file.txt

    ○
3. **Recursive Ownership Change**:

Change ownership of a directory and its contents:
sudo chown -R user1:users /home/user1/docs
ls -lR /home/user1/docs

    ○

Troubleshooting

- **User/group not found**: Verify users (`id user`) and groups (`getent group groupname`).
- **Permission denied**: Use `sudo` or ensure you have sufficient privileges.
- **Recursive issues**: Confirm the directory path and use `ls -lR` to verify changes.

3. setfacl (Set File Access Control Lists)

Overview

- **Purpose**: Sets Access Control Lists (ACLs) to provide fine-grained permissions beyond standard owner/group/others.
- **Mechanics**: ACLs extend permissions by defining specific access for users or groups, stored in the file system's extended attributes.

- **Use Cases**:
    - Granting specific users access to a file without changing group ownership.
    - Allowing multiple groups different permissions on a directory.
    - Managing shared directories with complex access rules.

Common Options

- `-m [acl]`: Modify or add ACL entries (e.g., `u:user:rw`, `g:group:rx`).
- `-x [acl]`: Remove an ACL entry.
- `-R`: Apply recursively.
- `-b`: Remove all ACLs, reverting to standard permissions.
- `--set`: Set new ACLs, replacing existing ones.

Detailed Examples

1. **Grant a User Specific Permissions**:

Allow `user2` read and write access to a file:

```
setfacl -m u:user2:rw file.txt
getfacl file.txt
# Output: user::rw-
#       group::r--
#       other::r--
#       user:user2:rw-
```

    -

2. **Grant a Group Permissions**:

Allow `developers` group read and execute access:

```
setfacl -m g:developers:rx /home/user/docs
getfacl /home/user/docs
# Output: Includes group:developers:r-x
```

    -

3. **Recursive ACLs**:

Apply ACLs to a directory and its contents:

```
setfacl -R -m u:user2:rwx /home/user/docs
```

    -

Troubleshooting

- **ACL not supported**: Ensure the file system supports ACLs (e.g., `ext4`, `xfs`; enable with `mount -o acl`).
- **Command not found**: Install `acl` package (`sudo apt install acl`).
- **Unexpected access**: Verify ACLs with `getfacl` and check standard permissions (`ls -l`).

4. getfacl (Get File Access Control Lists)

Overview

- **Purpose**: Displays the ACLs of a file or directory, including standard permissions and extended ACL entries.
- **Mechanics**: Reads extended attributes to show detailed access rules.
- **Use Cases**:
    - Verifying ACLs after modification.
    - Debugging permission issues.
    - Documenting access rules for shared resources.

Common Options

- `-R`: Display ACLs recursively for directories.
- `--omit-header`: Skip header information for cleaner output.

Detailed Examples

View ACLs for a File:

```
 getfacl file.txt
# Output: # file: file.txt
# owner: user
# group: group
user::rw-
user:user2:rw-
group::r--
other::r--
```

1.

View ACLs Recursively:

```
 getfacl -R /home/user/docs
```

Output: Shows ACLs for all files and subdirectories

2.

Clean Output:

```
 getfacl --omit-header file.txt
# Output: user::rw-
#       user:user2:rw-
#       group::r--
#       other::r--
```

3.

Troubleshooting

- **No ACLs displayed**: File system may not support ACLs or none are set.
- **Permission denied**: Use `sudo` for restricted files.
- **Complex output**: Use `ls -l` alongside `getfacl` to compare standard permissions.

Practical Tips

- **Check permissions first**: Use `ls -l` to view standard permissions before modifying.
- **Use octal for `chmod`**: Octal notation (e.g., `755`) is faster for setting all permissions at once.
- **Secure sensitive files**: Use `600` for private files and `700` for directories.
- **Combine `chown` and `chmod`**: Set ownership and permissions together for new files.
- **Use ACLs sparingly**: Reserve ACLs for complex permission scenarios; standard permissions are simpler.
- **Backup before changes**: Copy critical files or directories before modifying permissions.

Practice Tasks

1. Create a file `test.txt` and set permissions to `rw-r-----` using `chmod`.
2. Change the owner of `test.txt` to `user1` and group to `developers`.
3. Grant `user2` read and execute permissions on `test.txt` using `setfacl`.
4. Verify the ACLs on `test.txt` with `getfacl`.
5. Recursively set permissions on `/home/user/docs` to `rwxr-x---` and add `user2` with full access via ACLs.
6. Check ownership and permissions of `/home/user/docs`.

Solution:

touch test.txt
chmod 640 test.txt
sudo chown user1:developers test.txt
setfacl -m u:user2:rx test.txt
getfacl test.txt
chmod -R 750 /home/user/docs
setfacl -R -m u:user2:rwx /home/user/docs
ls -ld /home/user/docs
getfacl /home/user/docs

Linux Logs and Log Management

This document provides an in-depth explanation of Linux tools and configurations for managing system logs: `tail`, `grep`, `/var/log`, and `logrotate`. Each section includes a detailed description, how the tool or configuration works, common options, practical examples, use cases, and troubleshooting tips to help you monitor and manage logs effectively.

Overview of Logs

- **Purpose**: Logs record system and application events, such as errors, warnings, authentication attempts, and service activities, aiding in debugging, monitoring, and auditing.
- **Mechanics**: Logs are typically stored as text files in `/var/log`, managed by the system logger (e.g., `rsyslog` or `syslog`). Tools like `tail` and `grep` help analyze logs, while `logrotate` manages log file growth.
- **Use Cases**:
    - Troubleshooting service failures (e.g., checking `nginx` logs).
    - Monitoring security events (e.g., failed login attempts).
    - Managing disk space by rotating old logs.
- **Key Directory**: `/var/log` contains most system and service logs (e.g., `/var/log/syslog`, `/var/log/auth.log`).

1. tail (View End of File)

Overview

- **Purpose**: Displays the last few lines of a file, useful for monitoring log files in real-time or checking recent events.

- **Mechanics**: Reads the end of a file (default 10 lines) and can follow changes as they occur.
- **Use Cases**:
    - Monitoring live logs (e.g., web server errors).
    - Checking the most recent entries in a log file.

Common Options

- `-n [lines]`: Show the last `n` lines.
- `-f`: Follow the file, displaying new lines as they are written.
- `-q`: Quiet mode (suppress headers for multiple files).
- `--pid=[pid]`: Stop following when a specific process ends.

Detailed Examples

View Last 10 Lines of a Log:

```
 tail /var/log/syslog
# Output: Shows the last 10 lines of system logs
```

1.

View Last 20 Lines:

```
 tail -n 20 /var/log/auth.log
# Output: Shows the last 20 lines of authentication logs
```

2.

Monitor Logs in Real-Time:

```
 tail -f /var/log/nginx/error.log
# Output: Displays new error log entries as they occur
# Press Ctrl+C to stop
```

3.

Follow Until Process Ends:

```
 # Find nginx PID
pidof nginx
# Output: e.g., 1234
tail -f --pid=1234 /var/log/nginx/access.log
# Stops when nginx process ends
```

4.

Troubleshooting

- **Permission denied**: Use `sudo` for restricted logs (e.g., `sudo tail /var/log/auth.log`).
- **No output**: Verify the log file exists (`ls /var/log`) and has recent entries.
- **Too much output**: Use `-n` to limit lines or combine with `grep`.

2. grep (Search Text in Files)

Overview

- **Purpose**: Searches for patterns in log files or other text, filtering relevant lines for analysis.
- **Mechanics**: Uses regular expressions to match text, outputting matching lines or counts.
- **Use Cases**:
    - Finding error messages in logs.
    - Filtering logs by user or IP address.
    - Counting occurrences of specific events.

Common Options

- `-i`: Case-insensitive search.
- `-r`: Recursive search in directories.
- `-n`: Show line numbers.
- `-c`: Count matching lines.
- `-v`: Show non-matching lines (inverse search).
- `-w`: Match whole words only.

Detailed Examples

Search for Errors in a Log:

```
 grep "error" /var/log/syslog
# Output: Lines containing "error" (case-sensitive)
```

1.

Case-Insensitive Search with Line Numbers:

```
 grep -in "fail" /var/log/auth.log
# Output: e.g., 123:Failed password for user1
```

2.

Count Failed Logins:

```
 grep -c "Failed password" /var/log/auth.log
# Output: e.g., 5
```

3.

Recursive Search in Log Directory:

```
 grep -r "crash" /var/log
# Output: Matches in all files under /var/log
```

4.

Troubleshooting

- **No matches**: Check case sensitivity (`-i`) or pattern accuracy.
- **Permission denied**: Use `sudo` for restricted files.
- **Too many matches**: Narrow the search with `-w` or combine with `tail`.

3. /var/log (System Log Directory)

Overview

- **Purpose**: Stores log files for the system and services, managed by the system logger (e.g., `rsyslog` or `systemd-journald`).
- **Mechanics**: Log files are written by services, daemons, and the kernel. Common files include:
    - `/var/log/syslog` or `/var/log/messages`: General system logs.
    - `/var/log/auth.log`: Authentication-related logs.
    - `/var/log/kern.log`: Kernel logs.
    - Service-specific logs (e.g., `/var/log/nginx/access.log`).
- **Use Cases**:
    - Debugging system or application issues.
    - Auditing security events.
    - Monitoring service activity.

Common Log Files

- `/var/log/syslog` (Ubuntu) or `/var/log/messages` (CentOS): System-wide logs.
- `/var/log/auth.log` (Ubuntu) or `/var/log/secure` (CentOS): Authentication logs.

- `/var/log/dmesg`: Kernel ring buffer (boot and hardware messages).
- `/var/log/nginx/*.log`: Web server logs (if nginx is installed).

Detailed Examples

View System Logs:

```
 sudo less /var/log/syslog
# Navigate with arrow keys, q to quit
```

1.

Check Recent Authentication Events:

```
 sudo tail -n 10 /var/log/auth.log
# Output: Last 10 authentication events
```

2.

Search for Kernel Issues:

```
 sudo grep "error" /var/log/kern.log
```

3.

Combine with journalctl:

```
 journalctl --since "2025-06-27" -u nginx
# Output: nginx logs from systemd journal
```

4.

Troubleshooting

- **File not found**: Ensure the log file exists (`ls /var/log`) and the service is running.
- **Empty logs**: Check if the service is configured to log or if `rsyslog`/`systemd-journald` is active.
- **Permission issues**: Use `sudo` to access restricted logs.

4. logrotate (Rotate Log Files)

Overview

- **Purpose**: Manages log file growth by rotating, compressing, and deleting old logs to save disk space.

- **Mechanics**: Controlled by configuration files in `/etc/logrotate.conf` and `/etc/logrotate.d/`. Runs periodically via cron (`/etc/cron.daily/logrotate`).
- **Use Cases**:
    - Preventing logs from filling disk space.
    - Archiving old logs for auditing.
    - Compressing logs to save space.

Configuration File

- **Location**: `/etc/logrotate.conf` (global settings), `/etc/logrotate.d/` (service-specific configs).

Format:

```
/var/log/example.log {
    daily
    rotate 7
    compress
    missingok
    notifempty
}
```

-
    - **Options**:
        - `daily`/`weekly`/`monthly`: Rotation frequency.
        - `rotate [n]`: Keep `n` old logs.
        - `compress`: Compress old logs (e.g., `.gz`).
        - `missingok`: Ignore missing log files.
        - `notifempty`: Skip rotation if the log is empty.

Detailed Examples

View Global Logrotate Config:

```
 cat /etc/logrotate.conf
# Output: Shows default rotation settings
```

1.

Check Service-Specific Config:

```
 cat /etc/logrotate.d/nginx
# Example output:
# /var/log/nginx/*.log {
#     daily
```

```
#     rotate 14
#     compress
#     missingok
# }
```

2.

Manually Run Logrotate:

```
 sudo logrotate -f /etc/logrotate.conf
# Forces rotation of all logs
```

3.

Create a Custom Logrotate Config:

```
 sudo nano /etc/logrotate.d/myapp
# Add:
/var/log/myapp.log {
    weekly
    rotate 4
    compress
    missingok
}
# Save and exit
# Test:
sudo logrotate -d /etc/logrotate.d/myapp
# Debug mode, shows what would happen
```

4.

Troubleshooting

- **Logs not rotating**: Check cron job (`/etc/cron.daily/logrotate`) and permissions.
- **Config errors**: Test with `logrotate -d` to debug.
- **Disk space issues**: Verify old logs are compressed or deleted (`ls /var/log`).

Practical Tips

- **Monitor logs in real-time**: Use `tail -f` for live debugging.
- **Filter efficiently**: Combine `grep` with `tail` (e.g., `tail -f /var/log/syslog | grep error`).

- **Secure log files**: Ensure logs like `/var/log/auth.log` have restricted permissions (`ls -l /var/log`).
- **Check disk usage**: Use `du -sh /var/log/*` to monitor log sizes.
- **Automate log analysis**: Script `grep` searches in cron jobs for alerts.
- **Backup configs**: Copy `/etc/logrotate.conf` and `/etc/logrotate.d/` before editing.

Practice Tasks

1. View the last 15 lines of `/var/log/syslog`.
2. Search for "error" in `/var/log/nginx/error.log`.
3. Monitor `/var/log/auth.log` in real-time for failed logins.
4. Create a `logrotate` config for `/var/log/myapp.log` to rotate weekly, keeping 5 logs.
5. Test the `logrotate` config in debug mode.
6. Check the size of all logs in `/var/log`.

Solution:

```
sudo tail -n 15 /var/log/syslog
sudo grep "error" /var/log/nginx/error.log
sudo tail -f /var/log/auth.log | grep "Failed password"
sudo nano /etc/logrotate.d/myapp
# Add:
# /var/log/myapp.log {
#     weekly
#     rotate 5
#     compress
#     missingok
# }
# Save and exit
sudo logrotate -d /etc/logrotate.d/myapp
du -sh /var/log/*
```

Linux Process Management

This document provides an in-depth explanation of Linux commands for managing processes: `ps`, `top`, `htop`, `kill`, and `nice`. Each section includes a detailed description, how the command works, common options, practical examples, use cases, and troubleshooting tips to help you monitor and control processes effectively.

Overview of Process Management

- **Purpose**: Process management involves monitoring, controlling, and prioritizing running processes (programs in execution) on a Linux system.
- **Mechanics**: Each process has a unique Process ID (PID), user, priority, and resource usage. The kernel manages processes, and tools like `ps`, `top`, and `htop` provide visibility, while `kill` and `nice` control execution.
- **Use Cases**:
    - Identifying resource-heavy processes.
    - Terminating unresponsive applications.
    - Adjusting process priorities for performance.
- **Key Concepts**:
    - **PID**: Unique identifier for a process.
    - **PPID**: Parent Process ID (the process that started it).
    - **Priority**: Determines CPU scheduling (lower `nice` value = higher priority).

1. ps (Process Status)

Overview

- **Purpose**: Displays a snapshot of current processes, including their PID, user, and command.
- **Mechanics**: Queries the `/proc` file system to show process details.
- **Use Cases**:
    - Listing all processes for a user.
    - Finding the PID of a specific process.
    - Debugging process ownership or status.

Common Options

- `-e` or `aux`: Show all processes (BSD style: `aux`).
- `-u [user]`: Show processes for a specific user.
- `-f`: Full format, including PPID and command arguments.
- `--forest`: Show process hierarchy (tree view).

- `-p [pid]`: Show specific PID(s).

Detailed Examples

List All Processes:

```
 ps aux
# Output: USER  PID %CPU %MEM    VSZ   RSS TTY STAT START TIME COMMAND
#         user1 1234  0.1  0.5 123456 7890 ?   S    10:00 0:01 /usr/bin/nginx
```

1.

Show Processes for a User:

```
 ps -u user1
# Output: Lists processes owned by user1
```

2.

Display Process Hierarchy:

```
 ps --forest
# Output: Shows parent-child process relationships
```

3.

Find a Specific Process:

```
 ps aux | grep nginx
# Output: Lists processes matching "nginx"
```

4.

Troubleshooting

- **No output**: Ensure the process exists or use `ps -e` for all processes.
- **Too much output**: Pipe to `grep` (e.g., `ps aux | grep [pattern]`).
- **Missing details**: Use `-f` for full output or specify columns with `-o`.

2. top (Table of Processes)

Overview

- **Purpose**: Provides a real-time, interactive view of system processes, including CPU, memory usage, and process details.

- **Mechanics**: Continuously updates process information, allowing sorting and filtering.
- **Use Cases**:
    - Monitoring system resource usage.
    - Identifying high-CPU or high-memory processes.
    - Interactively killing processes.

Common Keys (Interactive)

- `q`: Quit.
- `k`: Kill a process (prompts for PID).
- `r`: Renice a process (change priority).
- `f`: Manage fields (add/remove columns).
- `P`: Sort by CPU usage, `M`: Sort by memory.

Detailed Examples

Start top:

```
 top
# Output: Real-time display with CPU, memory, and process list
```

1.

Kill a Process in top:

```
 top
# Press k, enter PID (e.g., 1234), confirm with Enter
```

2.

Sort by Memory Usage:

```
 top
# Press M to sort by %MEM
```

3.

Troubleshooting

- **High CPU load**: Check `%CPU` column to identify culprits.
- **Unresponsive**: Use `q` to exit or `Ctrl+C` if frozen.
- **Missing processes**: Ensure you have permissions (`sudo top` for all processes).

3. htop (Enhanced top)

Overview

- **Purpose**: A user-friendly, interactive process viewer with a colorful interface, mouse support, and scrolling.
- **Mechanics**: Similar to `top` but with improved visuals and usability, built on `ncurses`.
- **Use Cases**:
    - Easier process monitoring with intuitive navigation.
    - Filtering or searching processes by name.
    - Adjusting priorities or killing processes interactively.

Common Keys (Interactive)

- `q` or `F10`: Quit.
- `F9`: Kill a process.
- `F7`/`F8`: Decrease/increase nice value (priority).
- `F3`: Search for a process.
- `F5`: Show process tree.

Detailed Examples

Start htop:

```
 htop
# Output: Colorful interface with process list, CPU/memory bars
```

1.

Search for a Process:

```
 htop
# Press F3, type "nginx", Enter
# Highlights matching processes
```

2.

Kill a Process:

```
 htop
# Select process with arrow keys, press F9, choose signal (e.g., SIGTERM), Enter
```

3.

Troubleshooting

- **Command not found**: Install `htop` (`sudo apt install htop`).

- **Permission issues**: Use `sudo htop` for full process visibility.
- **Navigation issues**: Use arrow keys or mouse for selection.

4. kill (Send Signal to Process)

Overview

- **Purpose**: Sends signals to processes to terminate or control them (e.g., SIGTERM for graceful stop, SIGKILL for forced termination).
- **Mechanics**: Uses the kernel's signal system to communicate with processes, identified by PID.
- **Use Cases**:
    - Stopping unresponsive applications.
    - Sending custom signals to daemons (e.g., reload configuration).
- **Common Signals**:
    - `1` (SIGHUP): Reload configuration.
    - `9` (SIGKILL): Force kill (non-graceful).
    - `15` (SIGTERM): Graceful termination (default).

Common Options

- `-s [signal]`: Specify signal by name or number.
- `-l`: List available signals.

Detailed Examples

Gracefully Terminate a Process:

```
 # Find PID
ps aux | grep nginx
# Output: user1 1234 ... /usr/bin/nginx
sudo kill 1234
# Sends SIGTERM
```

1.

Force Kill a Process:

```
 sudo kill -9 1234
# Sends SIGKILL
```

2.

List Signals:

```
kill -l
# Output: Lists signals like 1) SIGHUP, 9) SIGKILL, 15) SIGTERM
```

3.

Troubleshooting

- **Process not terminated**: Use `SIGKILL` (`-9`) if `SIGTERM` fails.
- **Permission denied**: Use `sudo` for processes owned by other users.
- **Wrong PID**: Verify PID with `ps` or `pidof`.

5. nice (Set Process Priority)

Overview

- **Purpose**: Sets the priority (niceness) of a new process or adjusts existing ones with `renice`. Lower nice values = higher priority.
- **Mechanics**: Niceness ranges from -20 (highest priority) to 19 (lowest). Default is 0.
- **Use Cases**:
  - Prioritizing critical tasks (e.g., database backups).
  - Reducing priority for background tasks (e.g., file indexing).

Common Options

- `-n [value]`: Set nice value (e.g., `-n 10` for lower priority).
- For `renice`: `[value] -p [pid]` to adjust an existing process.

Detailed Examples

Run a Command with Low Priority:

```
nice -n 10 tar -czf backup.tar.gz /home/user
# Runs tar with lower priority
```

1.

Adjust Priority of a Running Process:

```
# Find PID
ps aux | grep nginx
# Output: user1 1234 ... /usr/bin/nginx
sudo renice 5 -p 1234
```

Sets nice value to 5

2.

Run with High Priority:

```
 sudo nice -n -10 /usr/bin/mysql
# Requires sudo for negative values
```

3.

Troubleshooting

- **Permission denied**: Use `sudo` for negative nice values or other users' processes.
- **No effect**: Check current priority (`ps -l` or `top`) and system load.
- **Invalid value**: Nice values must be between -20 and 19.

Practical Tips

- **Use `htop` for ease**: It's more intuitive than `top` for monitoring and managing processes.
- **Filter processes**: Combine `ps` with `grep` (e.g., `ps aux | grep [pattern]`).
- **Be cautious with SIGKILL**: Use `kill -9` only as a last resort, as it prevents graceful shutdown.
- **Monitor resources**: Use `top` or `htop` to identify resource-heavy processes before adjusting priorities.
- **Check process status**: Use `ps` or `top` to verify a process is running after starting.
- **Automate monitoring**: Script process checks with `ps` and `grep` in cron jobs.

Practice Tasks

1. List all processes owned by `user1` using `ps`.
2. Monitor system processes in real-time with `top`.
3. Use `htop` to find and kill a process named `sleep`.
4. Find the PID of `nginx` and terminate it gracefully.
5. Run a `tar` command with low priority (nice value 15).
6. Adjust the priority of an existing process (PID 1234) to 10.

Solution:

```
ps -u user1
top
htop
```

```
# In htop: F3, type "sleep", F9, select SIGTERM, Enter
ps aux | grep nginx
# Note PID, e.g., 1234
sudo kill 1234
nice -n 15 tar -czf backup.tar.gz /home/user
sudo renice 10 -p 1234
```

Linux Package Management

This document provides an in-depth explanation of Linux commands for managing packages: `apt`, `yum/dnf`, `dpkg`, and `rpm`. Each section includes a detailed description, how the tool works, common options, practical examples, use cases, and troubleshooting tips to help you install, update, and manage software packages effectively.

Overview of Package Management

- **Purpose**: Package management tools handle the installation, updating, and removal of software packages, including dependencies, on Linux systems.
- **Mechanics**: Packages are precompiled software bundles (e.g., `.deb` for Debian-based systems, `.rpm` for Red Hat-based systems) stored in repositories. High-level tools like `apt` and `yum/dnf` manage repositories, while low-level tools like `dpkg` and `rpm` handle individual package files.
- **Use Cases**:
  - Installing software (e.g., web servers, editors).
  - Updating system packages for security and performance.
  - Removing unused software to free disk space.
- **Key Distributions**:
  - **Debian-based** (e.g., Ubuntu): Use `apt` and `dpkg`.
  - **Red Hat-based** (e.g., CentOS, Fedora): Use `yum` (older) or `dnf` (newer) and `rpm`.

1. apt (Advanced Package Tool)

Overview

- **Purpose**: A high-level package manager for Debian-based systems (e.g., Ubuntu) that handles package installation, updates, and dependency resolution.
- **Mechanics**: Communicates with repositories defined in `/etc/apt/sources.list` or `/etc/apt/sources.list.d/` to download and install `.deb` packages.

- **Use Cases**:
    - Installing tools like `nginx` or `vim`.
    - Updating all system packages.
    - Managing repository configurations.

Common Options

- `update`: Refresh repository metadata.
- `upgrade`: Update installed packages.
- `install [package]`: Install a package and its dependencies.
- `remove [package]`: Remove a package (leaves config files).
- `purge [package]`: Remove a package and its config files.
- `autoremove`: Remove unneeded dependencies.
- `-y`: Assume "yes" to prompts.

Detailed Examples

Update Package Lists:

```
 sudo apt update
# Output: Fetches latest package metadata from repositories
```

1.

Upgrade Installed Packages:

```
 sudo apt upgrade
# Output: Updates all installed packages to latest versions
```

2.

Install a Package:

```
 sudo apt install nginx
# Output: Installs nginx and its dependencies
```

3.

Remove a Package:

```
 sudo apt remove nginx
# Removes nginx, keeps config files
sudo apt purge nginx
# Removes nginx and config files
```

4.

Clean Up Unused Dependencies:

```
 sudo apt autoremove
# Removes unneeded dependencies
```

5.

Troubleshooting

- **Broken dependencies**: Run `sudo apt update && sudo apt -f install` to fix.
- **Repository errors**: Check `/etc/apt/sources.list` for invalid URLs.
- **Permission denied**: Ensure `sudo` is used.
- **Package not found**: Verify package name or add a repository (`add-apt-repository`).

2. yum/dnf (Yellowdog Updater Modified/Dandified YUM)

Overview

- **Purpose**: High-level package managers for Red Hat-based systems (`yum` for older systems like CentOS 7, `dnf` for newer systems like Fedora or CentOS 8+). They manage `.rpm` packages and repositories.
- **Mechanics**: Queries repositories defined in `/etc/yum.repos.d/` (or `/etc/dnf/dnf.conf` for `dnf`) to handle package installation and updates.
- **Use Cases**:
    - Installing software on CentOS or Fedora.
    - Managing repository priorities.
    - Updating system packages.

Common Options (yum/dnf)

- `update` or `upgrade`: Update packages.
- `install [package]`: Install a package.
- `remove [package]`: Remove a package.
- `clean all`: Clear cached data.
- `--enablerepo=[repo]`: Enable a specific repository.
- `-y`: Assume "yes" to prompts.

Detailed Examples

Update Package Lists (dnf):

```
 sudo dnf update
# Output: Refreshes repository metadata and updates packages
```

1.

Install a Package (yum):

```
 sudo yum install httpd
# Output: Installs Apache web server and dependencies
```

2.

Remove a Package (dnf):

```
 sudo dnf remove httpd
# Output: Removes Apache and dependencies
```

3.

Enable a Specific Repository:

```
 sudo dnf --enablerepo=epel install mypackage
# Output: Installs mypackage from EPEL repository
```

4.

Troubleshooting

- **Dependency conflicts**: Use `dnf check` or `yum check` to diagnose.
- **Repository errors**: Verify `/etc/yum.repos.d/` files for correct URLs.
- **Slow performance**: Clear cache (`sudo dnf clean all` or `sudo yum clean all`).
- **Package not found**: Enable additional repositories (e.g., EPEL with `sudo yum install epel-release`).

3. dpkg (Debian Package Manager)

Overview

- **Purpose**: A low-level tool for installing, removing, and querying `.deb` package files on Debian-based systems.
- **Mechanics**: Directly manipulates individual `.deb` files, without handling dependencies or repositories (unlike `apt`).

- **Use Cases**:
    - Installing a downloaded `.deb` file.
    - Querying installed package details.
    - Fixing broken package installations.

Common Options

- `-i [file]`: Install a `.deb` file.
- `-r [package]`: Remove a package.
- `-P [package]`: Purge a package and its config files.
- `-l [pattern]`: List installed packages matching a pattern.
- `-s [package]`: Show package status.

Detailed Examples

Install a .deb File:

```
 sudo dpkg -i package.deb
# Output: Installs package.deb
```

1.

List Installed Packages:

```
 dpkg -l | grep nginx
# Output: Lists installed nginx packages
```

2.

Remove a Package:

```
 sudo dpkg -r nginx
# Removes nginx, keeps config files
sudo dpkg -P nginx
# Purges nginx and config files
```

3.

Fix Dependency Issues:

```
 sudo dpkg -i package.deb
# If errors occur due to dependencies:
sudo apt -f install
# Resolves missing dependencies
```

4.

Troubleshooting

- **Dependency errors**: Use `apt -f install` to fix after `dpkg -i`.
- **Package conflicts**: Remove conflicting packages (`dpkg -r`).
- **Broken packages**: Use `sudo dpkg --configure -a` to reconfigure.

4. rpm (RPM Package Manager)

Overview

- **Purpose**: A low-level tool for installing, removing, and querying `.rpm` package files on Red Hat-based systems.
- **Mechanics**: Directly handles `.rpm` files, without dependency resolution or repository management (unlike `yum/dnf`).
- **Use Cases**:
    - Installing a downloaded `.rpm` file.
    - Querying package metadata.
    - Verifying installed packages.

Common Options

- `-i [file]`: Install an `.rpm` file.
- `-e [package]`: Erase (remove) a package.
- `-qa`: Query all installed packages.
- `-qi [package]`: Show package information.
- `-qf [file]`: Find which package owns a file.

Detailed Examples

Install an .rpm File:

```
 sudo rpm -i package.rpm
# Output: Installs package.rpm
```

1.

List All Installed Packages:

```
 rpm -qa | grep httpd
# Output: Lists installed httpd packages
```

2.

Remove a Package:

```
 sudo rpm -e httpd
# Removes httpd
```

3.

Find Package Owning a File:

```
 rpm -qf /usr/sbin/httpd
# Output: e.g., httpd-2.4.37-1.el8
```

4.

Troubleshooting

- **Dependency errors**: Use `yum install` or `dnf install` to resolve dependencies after `rpm -i`.
- **Package already installed**: Use `--replacepkgs` with `rpm -i`.
- **Corrupted database**: Rebuild RPM database (`sudo rpm --rebuilddb`).

Practical Tips

- **Use high-level tools**: Prefer `apt` or `yum/dnf` over `dpkg`/`rpm` for dependency management.
- **Update regularly**: Run `apt update && apt upgrade` or `dnf update` to keep systems secure.
- **Check repositories**: Ensure `/etc/apt/sources.list` or `/etc/yum.repos.d/` are correctly configured.
- **Clean caches**: Use `apt autoclean` or `dnf clean all` to free disk space.
- **Verify packages**: Use `dpkg -l` or `rpm -qa` to list installed software.
- **Backup configs**: Copy `/etc/apt` or `/etc/yum.repos.d/` before changes.

Practice Tasks

1. Update package lists using `apt` or `dnf`.
2. Install `htop` using `apt` or `yum/dnf`.
3. List installed packages matching "nginx" using `dpkg` or `rpm`.
4. Remove `htop` using `apt` or `yum/dnf`.
5. Install a downloaded `.deb` or `.rpm` file using `dpkg` or `rpm`.

6. Check which package owns `/usr/bin/vim`.

Solution (Debian-based):

```
sudo apt update
sudo apt install htop
dpkg -l | grep nginx
sudo apt remove htop
sudo dpkg -i package.deb
sudo apt -f install  # Fix dependencies
dpkg -S /usr/bin/vim
```

Solution (Red Hat-based):

```
sudo dnf update
sudo dnf install htop
rpm -qa | grep nginx
sudo dnf remove htop
sudo rpm -i package.rpm
sudo dnf install -y  # Fix dependencies
rpm -qf /usr/bin/vim
```

Linux Firewall Management

This document provides an in-depth explanation of Linux commands and tools for managing firewalls: `ufw`, `iptables`, and `firewalld`. Each section includes a detailed description, how the tool works, common options, practical examples, use cases, and troubleshooting tips to help you configure and manage firewall rules effectively.

Overview of Firewall Management

- **Purpose**: Firewalls control incoming and outgoing network traffic based on predefined rules, enhancing system security by blocking unauthorized access and allowing legitimate traffic.
- **Mechanics**: Firewalls operate at the kernel level using `netfilter`, with tools like `ufw`, `iptables`, and `firewalld` providing user interfaces to configure rules.
- **Use Cases**:
    - Securing a web server by allowing only HTTP/HTTPS traffic.
    - Blocking specific IP addresses or ports.
    - Managing dynamic firewall rules for complex environments.
- **Key Concepts**:
    - **Rules**: Define actions (e.g., accept, drop) based on criteria (e.g., port, IP, protocol).
    - **Chains**: Categories for rules (e.g., INPUT, OUTPUT, FORWARD).
    - **Policies**: Default actions for unmatched traffic (e.g., DROP, ACCEPT).

1. ufw (Uncomplicated Firewall)

Overview

- **Purpose**: A user-friendly frontend for `iptables`, simplifying firewall configuration on Debian-based systems (e.g., Ubuntu).
- **Mechanics**: Translates high-level commands into `iptables` rules, managing the `netfilter` framework.
- **Use Cases**:
    - Quickly enabling/disabling firewall rules.
    - Allowing specific services (e.g., SSH, HTTP).
    - Blocking unwanted traffic for basic server security.

Common Options

- `enable`: Enable the firewall.
- `disable`: Disable the firewall.

- `status`: Show current rules and status.
- `allow [port/protocol]`: Allow traffic on a port or service.
- `deny [port/protocol]`: Block traffic on a port or service.
- `delete [rule]`: Remove a rule.
- `reset`: Reset to default (disables firewall, clears rules).

Detailed Examples

Enable ufw and Allow SSH:

```
 sudo ufw enable
sudo ufw allow ssh
# Allows port 22/tcp
sudo ufw status
# Output: Status: active
#       To                         Action      From
#       22/tcp                     ALLOW       Anywhere
```

1.

Allow HTTP and HTTPS:

```
 sudo ufw allow http
sudo ufw allow https
# Allows ports 80/tcp and 443/tcp
```

2.

Deny Specific Port:

```
 sudo ufw deny 23/tcp
# Blocks telnet
```

3.

Allow Specific IP:

```
 sudo ufw allow from 192.168.1.100
# Allows all traffic from 192.168.1.100
```

4.

Delete a Rule:

```
 sudo ufw status numbered
```

# Output: [1] 22/tcp ALLOW Anywhere
sudo ufw delete 1

5.

Troubleshooting

- **Locked out (e.g., SSH)**: Ensure `ufw allow ssh` is set before enabling.
- **Rules not applying**: Check status (`sudo ufw status`) and reload (`sudo ufw reload`).
- **Permission denied**: Use `sudo` for all `ufw` commands.
- **Conflicts with iptables**: Avoid mixing `ufw` and direct `iptables` rules.

2. iptables (IP Tables)

Overview

- **Purpose**: A low-level tool to configure `netfilter` firewall rules, offering fine-grained control over network traffic.
- **Mechanics**: Defines rules in chains (INPUT, OUTPUT, FORWARD) to accept, drop, or reject packets based on criteria like port, IP, or protocol.
- **Use Cases**:
    - Creating complex firewall rules.
    - Logging specific traffic.
    - Managing NAT (Network Address Translation).

Common Options

- `-A [chain]`: Append a rule to a chain.
- `-D [chain] [rule]`: Delete a rule.
- `-L [chain]`: List rules in a chain.
- `-F [chain]`: Flush (clear) rules in a chain.
- `-P [chain] [policy]`: Set default policy (e.g., DROP, ACCEPT).
- `-p [protocol]`: Specify protocol (e.g., tcp, udp).
- `--dport [port]`: Destination port.
- `-s [source]`: Source IP.
- `-j [target]`: Action (e.g., ACCEPT, DROP, REJECT).

Detailed Examples

List All Rules:

```
 sudo iptables -L -v -n
# Output: Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
#         pkts bytes target prot opt in out source destination
```

1.

Allow SSH Traffic:

```
 sudo iptables -A INPUT -p tcp --dport 22 -j ACCEPT
# Allows incoming SSH
```

2.

Block Specific IP:

```
 sudo iptables -A INPUT -s 192.168.1.100 -j DROP
# Drops all traffic from 192.168.1.100
```

3.

Set Default Policy to DROP:

```
 sudo iptables -P INPUT DROP
sudo iptables -A INPUT -p tcp --dport 80 -j ACCEPT
# Drops all incoming traffic except HTTP
```

4.

Save Rules:

```
 sudo iptables-save > /etc/iptables/rules.v4
# Persist rules (Debian-based, requires iptables-persistent package)
```

5.

Troubleshooting

- **Rules not persistent**: Use `iptables-save` and a package like `iptables-persistent`.
- **Locked out**: Ensure critical ports (e.g., SSH) are allowed before setting DROP policy.
- **Complex rules**: Test rules incrementally and use `-L -v -n` to verify.
- **Conflicts with ufw/firewalld**: Use only one firewall tool to avoid conflicts.

3. firewalld (Firewall Daemon)

Overview

- **Purpose**: A dynamic firewall management tool for Red Hat-based systems (e.g., CentOS, Fedora), providing a high-level interface for `netfilter`.
- **Mechanics**: Uses zones (e.g., public, trusted) and services to manage rules, with dynamic updates via D-Bus.
- **Use Cases**:
    - Managing firewall rules on servers with changing network conditions.
    - Supporting NetworkManager integration.
    - Configuring complex firewall policies with zones.

Common Options

- `--get-zones`: List available zones.
- `--get-active-zones`: Show currently active zones.
- `--add-service=[service]`: Allow a service (e.g., `http`, `ssh`).
- `--add-port=[port/protocol]`: Allow a specific port.
- `--remove-service=[service]`: Remove a service rule.
- `--permanent`: Make changes persistent across reboots.
- `--reload`: Apply changes without interrupting connections.

Detailed Examples

Check Active Zones:

```
 sudo firewall-cmd --get-active-zones
# Output: public
#         interfaces: eth0
```

1.

Allow HTTP Service:

```
 sudo firewall-cmd --add-service=http
sudo firewall-cmd --permanent --add-service=http
sudo firewall-cmd --reload
# Output: success
```

2.

Allow Specific Port:

```
 sudo firewall-cmd --add-port=8080/tcp --permanent
sudo firewall-cmd --reload
```

3.

List Rules in a Zone:

```
 sudo firewall-cmd --list-all
# Output: public (active)
#         services: ssh http
#         ports: 8080/tcp
```

4.

Block an IP:

```
 sudo firewall-cmd --add-source=192.168.1.100 --zone=drop --permanent
sudo firewall-cmd --reload
```

5.

Troubleshooting

- **Rules not applied**: Use `--permanent` and `--reload` for persistence.
- **Service not found**: Check available services (`firewall-cmd --get-services`).
- **Locked out**: Ensure `ssh` or management ports are allowed in the active zone.
- **Conflicts with iptables/ufw**: Disable other firewall tools (`systemctl stop ufw`).

Practical Tips

- **Choose one tool**: Use `ufw` (Ubuntu), `firewalld` (Red Hat), or `iptables` exclusively to avoid conflicts.
- **Test rules**: Apply rules temporarily before making them permanent.
- **Backup rules**: Save `iptables` rules (`iptables

Linux SSH and Remote Access

This document provides an in-depth explanation of Linux tools and configurations for secure remote access: `ssh`, `scp`, `sshd_config`, and key-based authentication. Each section includes a detailed description, how the tool or configuration works, common options, practical examples, use cases, and troubleshooting tips to help you manage remote connections securely.

Overview of SSH and Remote Access

- **Purpose**: Secure Shell (SSH) enables secure remote access to systems, allowing command execution, file transfers, and server management over encrypted connections.
- **Mechanics**: SSH uses client-server architecture with public-key cryptography for authentication and encryption. The SSH daemon (`sshd`) runs on the server, and clients use `ssh` or `scp` to connect.
- **Use Cases**:
  - Managing remote servers (e.g., web or database servers).
  - Securely transferring files between systems.
  - Setting up passwordless authentication for automation.
- **Key Files**:
  - `/etc/ssh/sshd_config`: SSH server configuration.
  - `~/.ssh/`: User directory for SSH keys and configs.
  - `/var/log/auth.log` or `/var/log/secure`: Logs for SSH authentication events.

1. ssh (Secure Shell Client)

Overview

- **Purpose**: Connects to a remote system to execute commands or start an interactive shell securely.
- **Mechanics**: Establishes an encrypted connection to the SSH daemon on the remote host, authenticating via password or key.
- **Use Cases**:
  - Running commands on a remote server.
  - Accessing a remote shell for administration.
  - Tunneling network traffic.

Common Options

- `-p [port]`: Specify SSH port (default: 22).
- `-i [keyfile]`: Use a specific private key for authentication.

- `-X`: Enable X11 forwarding for GUI applications.
- `-v`: Verbose output for debugging.
- `[user@host]`: Specify remote user and hostname/IP.

Detailed Examples

Connect to a Remote Server:

```
 ssh user1@192.168.1.100
# Prompts for password, opens remote shell
```

1.

Connect with Custom Port:

```
 ssh -p 2222 user1@192.168.1.100
# Connects to SSH server on port 2222
```

2.

Run a Single Command:

```
 ssh user1@192.168.1.100 "uptime"
# Output: 22:43:45 up 5 days, 3:12, 2 users, load average: 0.10, 0.15, 0.20
```

3.

Enable X11 Forwarding:

```
 ssh -X user1@192.168.1.100
# Allows running GUI apps (e.g., xterm) remotely
```

4.

Troubleshooting

- **Connection refused**: Ensure `sshd` is running (`sudo systemctl status sshd`) and the port is open (`ss -tuln`).
- **Permission denied**: Verify username, password, or key; check `/var/log/auth.log`.
- **Timeout**: Check firewall rules (`ufw status` or `firewall-cmd --list-all`) and network connectivity (`ping`).

2. scp (Secure Copy)

Overview

- **Purpose**: Securely transfers files between local and remote systems or between two remote systems using SSH.
- **Mechanics**: Uses SSH for encryption, copying files over a secure channel.
- **Use Cases**:
    - Uploading configuration files to a server.
    - Downloading logs from a remote system.
    - Transferring files between servers.

Common Options

- `-P [port]`: Specify SSH port.
- `-r`: Recursively copy directories.
- `-i [keyfile]`: Use a specific private key.
- `-v`: Verbose output for debugging.

Detailed Examples

Copy a File to a Remote Server:

```
 scp localfile.txt user1@192.168.1.100:/home/user1/
# Copies localfile.txt to remote /home/user1/
```

1.

Copy a Directory Recursively:

```
 scp -r /local/dir user1@192.168.1.100:/home/user1/
# Copies entire directory
```

2.

Copy from Remote to Local:

```
 scp user1@192.168.1.100:/home/user1/remotefile.txt .
# Copies remotefile.txt to current directory
```

3.

Copy with Custom Port:

```
 scp -P 2222 localfile.txt user1@192.168.1.100:/home/user1/
```

4.

Troubleshooting

- **Permission denied**: Check file permissions and SSH authentication.
- **Connection issues**: Verify SSH connectivity (`ssh user1@192.168.1.100`) and port.
- **Slow transfers**: Use `-v` to debug or check network bandwidth.

3. sshd_config (SSH Daemon Configuration)

Overview

- **Purpose**: Configures the SSH server (`sshd`) behavior, including port, authentication methods, and access controls.
- **Mechanics**: Located at `/etc/ssh/sshd_config`, defines settings like port, allowed users, and key-based authentication.
- **Use Cases**:
    - Changing the default SSH port for security.
    - Disabling password authentication.
    - Restricting access to specific users or groups.

Common Directives

- `Port [number]`: Set SSH port (default: 22).
- `PermitRootLogin [yes|no]`: Allow/disallow root login.
- `PasswordAuthentication [yes|no]`: Enable/disable password login.
- `AllowUsers [user1 user2]`: Restrict SSH to specific users.
- `PubkeyAuthentication [yes|no]`: Enable/disable key-based authentication.

Detailed Examples

Change SSH Port:

```
 sudo nano /etc/ssh/sshd_config
# Change: Port 22
# To:     Port 2222
# Save and exit
sudo systemctl restart sshd
```

1.

Disable Root Login:

```
 sudo nano /etc/ssh/sshd_config
# Set: PermitRootLogin no
```

sudo systemctl restart sshd

2.

Restrict to Specific Users:

```
 sudo nano /etc/ssh/sshd_config
# Add: AllowUsers user1 user2
sudo systemctl restart sshd
```

3.

Disable Password Authentication:

```
 sudo nano /etc/ssh/sshd_config
# Set: PasswordAuthentication no
sudo systemctl restart sshd
```

4.

Troubleshooting

- **Syntax errors**: Test config with `sudo sshd -t` before restarting.
- **Locked out**: Always test changes with an active session; ensure `AllowUsers` includes your user.
- **Service not restarting**: Check logs (`journalctl -u sshd`) for errors.
- **Firewall issues**: Update firewall rules for new port (`ufw allow 2222` or `firewall-cmd --add-port=2222/tcp`).

4. Key-Based Authentication

Overview

- **Purpose**: Enables passwordless SSH login using public-private key pairs, improving security and automation.
- **Mechanics**: A private key is stored on the client, and the corresponding public key is added to `~/.ssh/authorized_keys` on the server.
- **Use Cases**:
    - Automating scripts requiring SSH access.
    - Enhancing security by disabling passwords.
    - Simplifying logins for frequent access.

Steps to Set Up

Generate Key Pair on Client:

```
 ssh-keygen -t rsa -b 4096
# Press Enter for defaults (~/.ssh/id_rsa)
```

1.

Copy Public Key to Server:

```
 ssh-copy-id user1@192.168.1.100
# Adds public key to ~user1/.ssh/authorized_keys on server
```

2.

Test Key-Based Login:

```
 ssh user1@192.168.1.100
# Should log in without password
```

3.

Disable Password Authentication (Optional):

```
 sudo nano /etc/ssh/sshd_config
# Set: PasswordAuthentication no
sudo systemctl restart sshd
```

4.

Detailed Example

Full Setup:

```
# Generate key pair
ssh-keygen -t rsa -b 4096
# Copy key to server
ssh-copy-id -i ~/.ssh/id_rsa.pub user1@192.168.1.100
# Test login
ssh user1@192.168.1.100
# Verify authorized_keys
ssh user1@192.168.1.100 "cat ~/.ssh/authorized_keys"
```

-

Troubleshooting

- **Permission denied**: Ensure `~/.ssh` is 700 and `authorized_keys` is 600 (`chmod 700 ~/.ssh; chmod 600 ~/.ssh/authorized_keys`).
- **Key rejected**: Check server logs (`sudo journalctl -u sshd`) and verify key in `authorized_keys`.
- **SELinux issues**: On Red Hat-based systems, restore context (`sudo restorecon -R -v ~/.ssh`).
- **Firewall blocks**: Ensure SSH port is open (`ufw status` or `firewall-cmd --list-all`).

Practical Tips

- **Secure SSH**: Change default port, disable root login, and use key-based authentication.
- **Backup keys**: Store `~/.ssh/id_rsa` securely; losing it prevents access.
- **Monitor logs**: Check `/var/log/auth.log` or `/var/log/secure` for SSH issues.
- **Use `scp` for automation**: Combine with key-based authentication for scripts.
- **Test config changes**: Use `sshd -t` and keep an active session to avoid lockouts.
- **Combine with `tmux`**: Use `tmux` for persistent remote sessions.

Practice Tasks

1. Connect to a remote server (`user1@192.168.1.100`) using `ssh`.
2. Copy a local file (`data.txt`) to the remote server using `scp`.
3. Change the SSH port to 2222 in `sshd_config` and restart `sshd`.
4. Set up key-based authentication for `user1@192.168.1.100`.
5. Disable password authentication on the server.
6. Verify SSH login without a password.

Solution:

```
ssh user1@192.168.1.100
scp data.txt user1@192.168.1.100:/home/user1/
sudo nano /etc/ssh/sshd_config
# Set: Port 2222
sudo systemctl restart sshd
ssh-keygen -t rsa -b 4096
ssh-copy-id -i ~/.ssh/id_rsa.pub user1@192.168.1.100
sudo nano /etc/ssh/sshd_config
# Set: PasswordAuthentication no
sudo systemctl restart sshd
ssh -p 2222 user1@192.168.1.100
```

Linux System Monitoring and Performance

This document provides an in-depth explanation of Linux commands for system monitoring and performance analysis: `sar`, `vmstat`, `iostat`, and `free`. Each section includes a detailed description, how the tool works, common options, practical examples, use cases, and troubleshooting tips to help you monitor system resources effectively.

Overview of System Monitoring

- **Purpose**: System monitoring tools track resource usage (CPU, memory, disk, I/O) to diagnose performance issues, optimize system performance, and ensure stability.
- **Mechanics**: These tools query kernel data (e.g., `/proc`, `/sys`) to provide real-time or historical metrics on system performance.
- **Use Cases**:
    - Identifying CPU or memory bottlenecks.
    - Monitoring disk I/O performance.
    - Analyzing system load for capacity planning.
- **Key Files**:
    - `/proc/stat`: CPU and system statistics.
    - `/proc/meminfo`: Memory usage details.
    - `/proc/diskstats`: Disk I/O statistics.
    - `/var/log/sysstat/`: Historical data for `sar`.

1. sar (System Activity Reporter)

Overview

- **Purpose**: Collects, reports, and saves system activity statistics (CPU, memory, I/O, etc.) over time, part of the `sysstat` package.
- **Mechanics**: Gathers data via cron (`/etc/cron.d/sysstat`) and stores it in `/var/log/sysstat/saXX` for historical analysis.
- **Use Cases**:
    - Analyzing historical CPU usage trends.
    - Troubleshooting intermittent performance issues.
    - Generating performance reports.

Common Options

- `-u`: CPU usage (default).
- `-r`: Memory usage.
- `-b`: I/O and transfer rate statistics.
- `-d`: Disk activity (use `-p` for device names).
- `-A`: All statistics.
- `-f [file]`: Read from a specific `sa` file.
- `-s [start] -e [end]`: Specify time range (e.g., `10:00:00`).

Detailed Examples

View Current CPU Usage:

```
 sar -u 1 5
# Output: CPU usage every 1 second, 5 times
#        22:45:01    CPU  %user  %nice  %system  %iowait  %steal  %idle
#        22:45:02    all   5.00   0.00    2.00     0.50    0.00   92.50
```

1.

Check Historical Memory Usage:

```
 sar -r -f /var/log/sysstat/sa27
# Output: Memory stats for June 27
#        10:00:01    kbmemfree  kbmemused  %memused  kbbuffers  kbcached
#        10:00:02    500000     1500000    75.00     100000     200000
```

2.

Disk I/O for a Specific Device:

```
 sar -d -p 1 5
# Output: Disk stats every 1 second, 5 times
#        22:45:01    DEV       tps  rkB/s  wkB/s
#        22:45:02    sda      10.0  50.0   20.0
```

3.

All Stats for a Time Range:

```
 sar -A -s 10:00:00 -e 12:00:00
# Output: All metrics for specified time range
```

4.

Troubleshooting

- **Command not found**: Install `sysstat` (`sudo apt install sysstat` or `sudo dnf install sysstat`).
- **No data in files**: Enable data collection in `/etc/sysstat/sysstat` (`ENABLED="true"`) and restart (`sudo systemctl restart sysstat`).
- **Missing historical data**: Check `/var/log/sysstat/` and cron job (`/etc/cron.d/sysstat`).
- **Permission denied**: Use `sudo` for restricted files.

2. vmstat (Virtual Memory Statistics)

Overview

- **Purpose**: Reports system performance metrics, including CPU, memory, swap, I/O, and system activity.
- **Mechanics**: Reads `/proc/stat`, `/proc/meminfo`, and `/proc/vmstat` for real-time data.
- **Use Cases**:
  - Monitoring memory and swap usage.
  - Detecting I/O bottlenecks.
  - Checking system load during high activity.

Common Options

- `-s`: Summary of memory and CPU stats.
- `-d`: Disk statistics.
- `-a`: Show active/inactive memory.
- `[delay [count]]`: Update every `delay` seconds, `count` times.

Detailed Examples

Monitor System Activity:

```
 vmstat 2 5
# Output: Updates every 2 seconds, 5 times
#        procs  memory      swap      io     system  cpu
#        r  b   swpd  free  buff cache  si so  bi bo  in cs us sy id wa
#        1  0   0    500M  100M 1G    0  0   10 20 100 200 5  2  90 3
```

1.

Memory Summary:

```
 vmstat -s
```

```
# Output:       2000000 K total memory
#              500000 K used memory
#              300000 K active memory
```

2.

Disk Statistics:

```
 vmstat -d
# Output: disk  reads  merged  sectors  ms  writes  merged  sectors  ms
#         sda   1000   500    8000    200  2000   1000   16000   400
```

3.

Troubleshooting

- **No output**: Ensure `procps` is installed (`sudo apt install procps`).
- **High swap usage (`si/so`)**: Check memory pressure (`free -m`) and add RAM if needed.
- **Unclear metrics**: Refer to `man vmstat` for column explanations (e.g., `si` = swap in).

3. iostat (Input/Output Statistics)

Overview

- **Purpose**: Reports CPU and disk I/O statistics, part of the `sysstat` package.
- **Mechanics**: Reads `/proc/diskstats` and `/proc/stat` for disk and CPU performance data.
- **Use Cases**:
    - Identifying disk performance bottlenecks.
    - Monitoring CPU utilization alongside I/O.
    - Analyzing storage device performance.

Common Options

- `-c`: CPU statistics only.
- `-d`: Device (disk) statistics only.
- `-x`: Extended statistics (e.g., %util).
- `-p [device]`: Show stats for a specific device.
- `[interval [count]]`: Update every `interval` seconds, `count` times.

Detailed Examples

Monitor CPU and Disk I/O:

```
 iostat 2 5
# Output: CPU and device stats every 2 seconds, 5 times
#        avg-cpu:  %user  %nice %system %iowait %steal %idle
#                  5.00   0.00  2.00   0.50   0.00   92.50
#        Device  tps  kB_read/s  kB_wrtn/s
#        sda     10.0 50.0       20.0
```

1.

Extended Disk Stats:

```
 iostat -x
# Output: Device  tps  kB_read/s  kB_wrtn/s  %util
#        sda     10.0 50.0       20.0       5.0
```

2.

Specific Device:

```
 iostat -p sda
# Output: Stats for sda and its partitions
```

3.

Troubleshooting

- **Command not found**: Install `sysstat` (`sudo apt install sysstat`).
- **High %iowait**: Indicates disk bottleneck; check device utilization (`-x`).
- **No device stats**: Ensure disks are active (`lsblk`).

4. free (Memory Usage)

Overview

- **Purpose**: Displays memory and swap usage, including total, used, and free amounts.
- **Mechanics**: Reads `/proc/meminfo` for memory statistics.
- **Use Cases**:
    - Checking available RAM for applications.
    - Monitoring swap usage.
    - Diagnosing memory shortages.

Common Options

- `-h`: Human-readable format (e.g., GB, MB).
- `-m`: Display in megabytes.
- `-s [seconds]`: Continuous updates every `seconds`.
- `-t`: Show total for memory and swap.

Detailed Examples

View Memory Usage:

```
 free -h
# Output:            total  used  free  shared  buff/cache  available
#         Mem:       2.0G  500M  1.0G  100M    400M        1.2G
#         Swap:      1.0G    0B  1.0G
```

1.

Continuous Monitoring:

```
 free -h -s 2
# Updates every 2 seconds
```

2.

Show Totals:

```
 free -h -t
# Output: Includes a total line for Mem + Swap
```

3.

Troubleshooting

- **High swap usage**: Indicates low RAM; consider adding memory or closing apps.
- **Low available memory**: Check `buff/cache` (can be reclaimed) vs. `used`.
- **Inaccurate stats**: Cross-check with `vmstat -s` or `top`.

Practical Tips

- **Use `sar` for historical data**: Ideal for long-term analysis.
- **Combine tools**: Use `vmstat` for quick snapshots, `iostat` for disk details, and `free` for memory.
- **Monitor regularly**: Set up `sar` cron jobs for continuous data collection.
- **Check logs**: Use `/var/log/syslog` or `journalctl` for related errors.

- **Automate alerts**: Script checks for high CPU/memory usage with `sar` or `vmstat`.
- **Install sysstat**: Ensure `sysstat` is installed for `sar` and `iostat`.

Practice Tasks

1. Check current CPU usage with `sar` for 3 updates every 2 seconds.
2. Display memory statistics with `vmstat`.
3. Monitor disk I/O for `sda` with `iostat`.
4. View memory and swap usage in human-readable format with `free`.
5. Check historical CPU usage for today with `sar`.
6. Combine `vmstat` and `free` to monitor memory usage every 5 seconds.

Solution:

```
sar -u 2 3
vmstat -s
iostat -p sda
free -h
sar -u -f /var/log/sysstat/sa27
vmstat 5 & free -h -s 5
# Press Ctrl+C to stop
```