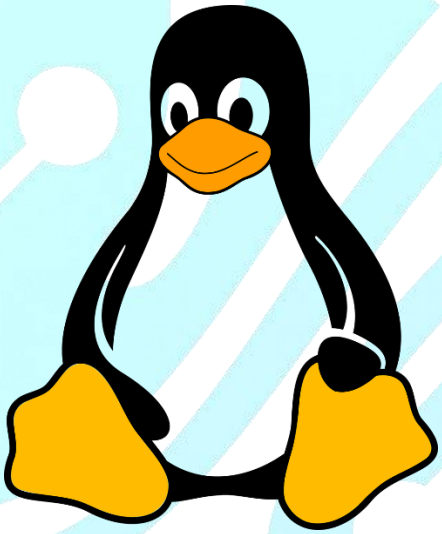
Version 1.0.0

AIR Hub
SMART SOLUTIONS

# Professional Reference Guide
# Linux Commands



# Table of Contents

# 1 File and Directory Operations Commands

## 1.1 Basic Navigation

***Print Working Directory***

This command displays the full path of the current directory you are working in. It shows your present location in the filesystem hierarchy, helping you orient yourself when navigating through complex directory structures.

```bash
pwd
```

Displays the current directory path

***List Directory Contents***

The ls command displays all files and directories within the specified location. When used with the -l option, it provides detailed information including permissions, ownership, size, and modification dates. The -a option reveals hidden files that start with a dot, while -h makes file sizes human-readable by displaying them in KB, MB, or GB.

```bash
ls
```

- `ls -l` - Detailed list with permissions
- `ls -a` - Show hidden files
- `ls -lh` - Human-readable file sizes
- `ls -t` - Sort by modification time

***Change Directory***

This fundamental command enables navigation between different directories. You can use absolute paths starting from root or relative paths from your current location. The tilde symbol represents your home directory, and the double dot notation moves you up one level in the directory tree.

```bash
cd
```

- `cd /path/to/dir` - Absolute path
- `cd ../parent` - Relative path
- `cd ~` - Home directory
- `cd -` - Previous directory

## 1.2 File Operations

### 1.2.1 Copy Files/Directories

```bash
cp
```

The copy command duplicates files or entire directory structures. The recursive option is essential for copying directories and their contents. The preserve option maintains original file attributes like timestamps and permissions, which is crucial for backup operations.

```bash
cp file1 file2          # Copy file
cp -r dir1 dir2         # Recursive copy
cp -p file1 dir/        # Preserve attributes
```

### 1.2.2 Move/Rename Files

```bash
mv
```

This versatile command serves dual purposes: relocating files to different directories or changing their names. When moving files across filesystems, it effectively performs a copy followed by delete operation. The command preserves file timestamps unless the destination is on a different filesystem.

```bash
mv oldname newname          # Rename
mv file1 /target/directory/ # Move file
mv *.txt /backup/           # Move multiple files
```

### 1.2.3 Remove Files/Directories

```bash
rm
```

The remove command deletes files and directories permanently. Caution is advised as deleted files cannot be easily recovered. The recursive option allows deletion of directories and their contents, while the force option suppresses confirmation prompts. The interactive mode provides safety by requesting confirmation for each deletion.

```bash
rm file.txt              # Remove file
rm -r directory/         # Recursive remove
rm -f file.txt           # Force remove
rm -i file.txt           # Interactive removal
```

### 1.2.4 Create Empty File/Update Timestamp

```bash
touch
```

Primarily used to create new empty files, touch also updates access and modification timestamps of existing files. This is particularly useful for triggering build processes or resetting file states in automated workflows.

```bash
touch newfile.txt              # Create new file
touch -t 202312251200 file.txt # Set specific timestamp
```

## 1.3 Directory Operations

### 1.3.1 Create Directory

**Bash**

```
mkdir
```

This command creates new directories in the filesystem. The parents option enables creating nested directory structures in a single operation, automatically generating any missing parent directories in the specified path

**Bash**

```
mkdir newdir                    # Create single directory
mkdir -p parent/child/grandchild    # Create nested directories
```

### 1.3.2 Remove Empty Directory

**Bash**

```
rmdir
```

The remove directory command deletes empty directories only. This safety feature prevents accidental deletion of directories containing files, ensuring you must explicitly remove all contents first or use the recursive remove command instead.

**Bash**

```
rmdir emptydir          # Remove empty directory
```

## 1.4 File Viewing and Editing

### 1.4.1 Concatenate and Display Files

**Bash**

```
cat
```

Originally designed to combine multiple files, cat is commonly used to display entire file contents in the terminal. It works well for small files but becomes impractical for large files due to terminal scroll limitations.

**Bash**

```
cat file.txt                    # Display entire file
cat file1 file2 > combined.txt  # Combine files
```

### 1.4.2 less / more - Page Through Files

**Bash**

```
less
more
```

These commands display file contents in scrollable, page-by-page views, making them ideal for examining large files or log files. Less provides more advanced features like backward scrolling and search capabilities compared to more.

**Bash**

```
less largefile.log      # Scrollable view
more largefile.log      # Page-by-page view
```

### 1.4.3 head / tail - View File Beginnings/Endings

```bash
head
tail
```

Head displays the beginning portions of files, typically the first 10 lines by default. Tail shows the ending portions and includes a follow mode that continuously displays new lines as they're added to growing files, extremely valuable for monitoring log files in real-time.

```bash
head -n 20 file.log          # First 20 lines
tail -n 15 file.log          # Last 15 lines
tail -f live.log             # Follow live updates
```

### 1.4.4 nano / vim / emacs - Text Editors

```bash
nano
vim
```

These represent the spectrum of text editors available in Linux environments. Nano offers simplicity for beginners, vim provides powerful modal editing for experts, and emacs delivers an extensible editing environment that can function as a complete development platform.

```bash
nano file.txt          # Simple editor
vim file.txt           # Advanced editor
```

# 2 File Permission Commands

## 2.1 Permission System Fundamentals

Linux employs a comprehensive permission system that controls file and directory access through three distinct entities: the file owner, the group associated with the file, and all other users. Each entity can be granted different levels of access through three permission types: read access for viewing contents, write access for modification, and execute access for running programs or scripts.

***Linux uses a 3-tier permission system:***

- User (u) - File owner
- Group (g) - Group members
- Others (o) - All other users

***Permission types:***

- Read (r) – 4 (100)
- Write (w) – 2 (010)
- Execute (x) – 1 (001)

## 2.2 Permission Modification Commands

**Bash**

```
chmod \ chown \ chgrp \ umask
```

### 2.2.1 chmod - Change File Mode

This command modifies file and directory permissions using either symbolic notation or octal numbers. Symbolic notation uses letters to represent who and what permissions to change, while octal notation uses three-digit numbers where each digit represents permissions for owner, group, and others respectively.

**Bash**

```
chmod 755 script.sh          # rwxr-xr-x
chmod u+x executable         # Add execute for user
chmod g-w file.txt           # Remove write for group
chmod o=r file.txt           # Set others to read-only
chmod -R 644 /path/to/dir/   # Recursive change
```

### 2.2.2 chown - Change File Ownership

The change owner command transfers file ownership between users and groups. System administrators frequently use this when reassigning project files or correcting ownership issues. The recursive option applies ownership changes to entire directory trees.

**Bash**

```
chown user:group file.tx          # Change owner and group
chown username file.txt           # Change owner only
chown :groupname file.txt         # Change group only
chown -R user:group /directory/   # Recursive ownership change
```

### 2.2.3 chgrp - Change Group Ownership

This specialized command modifies only the group association of files and directories without affecting user ownership. It's particularly useful in collaborative environments where multiple users need access to shared resources.

**Bash**

```
chgrp developers script.sh        # Change file group
```

### 2.2.4 umask - User File Creation Mask

Umask determines the default permissions for newly created files and directories by subtracting the mask value from the maximum possible permissions. This provides a security baseline ensuring files aren't accidentally created with excessive permissions.

**Bash**

```
umask                             # Show current umask
umask 0022                        # Set new umask
```

# 3 File Compression and Archiving Commands

## 3.1 Archive Creation

### 3.1.1 tar - Tape Archive

Originally designed for tape backups, tar now serves as the standard archiving tool for combining multiple files and directories into a single archive file. It preserves file attributes, directory structures, and permissions, making it ideal for backups and software distribution.

**Bash**

```
tar -cvf archive.tar /path/          # Create archive
tar -xvf archive.tar                 # Extract archive
tar -tzf archive.tar.gz              # List contents
tar -xzf archive.tar.gz -C /target/  # Extract to specific directory
```

## 3.2 Compression Tools

### 3.2.1 gzip / gunzip

Gzip provides fast compression using the DEFLATE algorithm, offering a good balance between compression speed and ratio. It's widely used for compressing individual files and works seamlessly with tar for creating compressed archives.

**Bash**

```
gzip file.txt          # Compress to file.txt.gz
gunzip file.txt.gz     # Decompress
gzip -9 file.txt       # Maximum compression
```

### 3.2.2 bzip2 / bunzip2

Bzip2 utilizes the Burrows-Wheeler algorithm to achieve better compression ratios than gzip, particularly for text files, at the cost of slower compression speeds. It's ideal for archival purposes where compression ratio outweighs speed considerations.

**Bash**

```
bzip2 file.txt         # Compress to file.txt.bz2
bunzip2 file.txt.bz2   # Decompress
```

### 3.2.3 xz / unxz

Xz implements the LZMA2 algorithm, providing superior compression ratios among commonly available tools, making it excellent for distributing large software packages. The trade-off is significantly higher memory usage during compression.

**Bash**

```
xz file.txt          # Compress to file.txt.xz
unxz file.txt.xz     # Decompress
```

## 3.3 Combined Operations

Modern workflows typically combine tar with compression utilities in a single operation. This approach first creates an archive of multiple files then applies compression, resulting in efficient storage and transmission of complex directory structures while maintaining file integrity and metadata.

**Bash**

```
tar -czf archive.tar.gz /path/      # Create compressed tar
tar -xzf archive.tar.gz             # Extract compressed tar
tar -cjf archive.tar.bz2 /path/     # Create bzip2 compressed tar
zip -r archive.zip /path/           # Create zip archive
unzip archive.zip                   # Extract zip archive
```

# 4 Process Management Commands

## 4.1 Process Monitoring

### 4.1.1 ps - Process Status

This command provides a snapshot of currently running processes. The aux options display comprehensive information including process owners, resource utilization, and execution states. Different option combinations cater to various display formats and information detail levels.

```bash
ps aux              # All processes detailed
ps -ef              # Full format listing
ps -u username      # User's processes
ps --forest         # Show process tree
```

### 4.1.2 top / htop - Real-time Process Monitoring

These interactive commands continuously update process information, showing system resource usage in real-time. Top provides standard process monitoring while htop offers enhanced features like color coding, vertical and horizontal scrolling, and mouse operation support.

```bash
top                 # Interactive process viewer
htop                # Enhanced top with colors
```

### 4.1.3 pstree - Process Tree

Pstree visualizes processes in a tree structure, clearly showing parent-child relationships between processes. This is invaluable for understanding process dependencies and identifying which processes spawned others.

```bash
pstree              # Visual process hierarchy
pstree -p           # Show PIDs
```

## 4.2 Process Control

### 4.2.1 kill - Terminate Processes

The kill command sends specific signals to processes, with the termination signal being the most common for graceful shutdowns. The force kill signal immediately stops unresponsive processes. Pkill extends this functionality by allowing process selection by name rather than process ID.

**Bash**

```
kill 1234              # Terminate PID 1234
kill -9 1234           # Force kill
kill -TERM 1234        # Send TERM signal
kill -HUP 1234         # Send HUP signal (reload)
```

### 4.2.2 pkill - Kill by Name

**Bash**

```
pkill processname      # Kill by process name
pkill -u username      # Kill user's processes
```

### 4.2.3 killall - Kill All Instances

This command terminates all processes matching a specified name, useful for stopping multiple instances of the same application. Care must be taken as it can affect more processes than intended if names are ambiguous.

**Bash**

```
killall processname    # Kill all matching processes
```

## 4.3 Background/Foreground

### 4.3.1 & - Run in Background

```bash
./script.sh &              # Run in background
```

### 4.3.2 jobs - List Background Jobs

These commands manage background and foreground processes within a shell session. Jobs lists currently running background processes, while fg and bg move processes between foreground and background execution states, enabling efficient multitasking.

```bash
jobs               # Show background jobs
jobs -l            # With PID information
```

### 4.3.3 fg / bg - Foreground/Background

```bash
fg %1              # Bring job 1 to foreground
bg %2              # Run job 2 in background
```

### 4.3.4 nohup - Run Process After Logout

Nohup allows processes to continue running after the user logs out by ignoring hangup signals. This is essential for long-running tasks that must complete regardless of user session status.

```bash
nohup ./long_running_script.sh & # Continue after logout
```

## 4.4 Process Priority

***nice and renice - Process Priority Adjustment***

These commands influence process scheduling priority by modifying the "niceness" value. Lower priority values give processes more CPU access, while higher values make them more "nice" by yielding to other processes. Renice adjusts priorities of already running processes.

### 4.4.1 nice - Set Process Priority

```Bash
nice -n 10 ./script.sh        # Lower priority (higher nice value)
```

### 4.4.2 renice - Change Running Process Priority

```Bash
renice 5 1234              # Change PID 1234 nice to 5
```

# 5 System Information Commands

## 5.1 System Overview

### 5.1.1 uname - System Information

This command reveals fundamental system characteristics including kernel version, hardware architecture, and operating system details. System administrators use this information for compatibility checking and system documentation.

**Bash**

```
uname -a          # All system info
uname -r          # Kernel release
uname -m          # Machine architecture
```

### 5.1.2 hostname - System Hostname

Hostname displays or sets the system's network name, which is crucial for network communications and system identification in multi-server environments. The option to show all IP addresses helps in network configuration troubleshooting.

**Bash**

```
hostname          # Show hostname
hostname -I       # Show all IP addresses
```

### 5.1.3 uptime - System Uptime

Uptime shows how long the system has been running since last boot, along with load averages that indicate system resource pressure over different time intervals.

**Bash**

```
uptime            # System uptime and load average
```

## 5.2 Hardware Information

### 5.2.1 lscpu - CPU Information

This detailed command displays processor information including architecture, number of cores, threading capabilities, cache sizes, and processor family characteristics essential for performance tuning and capacity planning.

```bash
lscpu                # Detailed CPU architecture
```

### 5.2.2 free - Memory Usage

Free shows system memory utilization including physical RAM, swap space, and kernel buffers. The human-readable option converts bytes into appropriate units for easier interpretation.

```bash
free -h              # Human-readable memory info
free -m              # Memory in MB
```

### 5.2.3 df - Disk Space

The disk free command reports filesystem disk space usage, showing available capacity, used space, and mount points. The human-readable option and inode information option help administrators manage storage resources effectively.

```bash
df -h                # Human-readable disk space
df -i                # Inode information
```

### 5.2.4 du - Directory Space Usage

Disk usage provides detailed analysis of directory and file space consumption. The summary option shows total sizes without overwhelming detail, while depth control enables focused analysis of specific directory levels.

```bash
du -sh /path/              # Summary of directory size
du -h --max-depth=1 /path/ # Sizes of first-level subdirectories
```

### 5.2.5 lsblk - Block Devices

This command lists all block storage devices, their partitions, and mount points, providing a comprehensive view of the system's storage architecture.

```bash
lsblk          # List block devices
lsblk -f       # With filesystem information
```

## 5.3 Performance Monitoring

### 5.3.1 vmstat - Virtual Memory Statistics

Vmstat reports information about processes, memory, paging, block IO, traps, and CPU activity. Regular interval reporting helps identify trends and performance bottlenecks.

```bash
vmstat 1       # Update every second
vmstat -s      # Summary statistics
```

### 5.3.2 iostat - I/O Statistics

This command monitors system input/output device loading by observing the time devices are active relative to their average transfer rates, helping identify storage performance issues.

```bash
iostat -x 1    # Extended I/O stats every second
```

### 5.3.3 sar - System Activity Reporter

Sar collects, reports, and saves system activity information, providing comprehensive historical data for performance analysis and capacity planning.

```bash
sar -u 1 3     # CPU usage every 1 sec, 3 times
sar -r 1 3     # Memory usage
```

# 6 Networking Commands

## 6.1 Connectivity Testing

### 6.1.1 ping - Test Network Connectivity

Ping tests network connectivity by sending ICMP echo requests to target hosts and measuring response times. It's the first troubleshooting tool for network issues, helping identify connectivity problems and measure latency.

```Bash
ping google.com             # Continuous ping
ping -c 4 google.com        # Ping 4 times
ping -M do -s 1500 google.com   # Test MTU
```

### 6.1.2 traceroute / tracepath - Network Path Tracing

These commands map the route packets take to reach a destination, identifying each hop along the path and measuring transit delays. This is invaluable for diagnosing network routing issues and identifying bottlenecks.

```Bash
traceroute google.com       # Trace route to host
tracepath google.com        # Alternative traceroute
```

## 6.2 Network Configuration

### 6.2.1 ip - Modern Network Configuration

The ip command replaces older networking tools with a unified interface for managing network interfaces, IP addresses, routing tables, and neighbor entries. It's the current standard for network configuration on modern Linux systems.

```Bash
ip addr show            # Show IP addresses
ip route show           # Show routing table
ip link show            # Show network interfaces
ip neigh show           # Show ARP table
```

### 6.2.2 ifconfig - Legacy Network Configuration

Though deprecated in favor of ip, ifconfig remains widely used for configuring network interfaces, setting IP addresses, and controlling interface states.

```Bash
ifconfig              # Show interface configuration
ifconfig eth0 up      # Bring interface up
```

### 6.2.3 netstat - Network Statistics

Netstat displays network connections, routing tables, interface statistics, masquerade connections, and multicast memberships. It's essential for monitoring network activity and troubleshooting connection issues.

```Bash
netstat -tuln         # Listening TCP/UDP ports
netstat -r            # Routing table
netstat -s            # Network statistics
```

## 6.3 DNS and Port Testing

### 6.3.1 nslookup / dig - DNS Queries

These commands query Domain Name System servers to resolve hostnames to IP addresses and retrieve DNS records. Dig provides more detailed information and is preferred for DNS troubleshooting.

```Bash
nslookup domain.com       # DNS lookup
dig domain.com            # Detailed DNS information
dig domain.com MX         # MX records only
```

### 6.3.2 ss - Socket Statistics

Ss investigates sockets, replacing the older netstat command with faster performance and more detailed information about network connections.

```Bash
ss -tuln              # Listening sockets
ss -t                 # Established TCP connections
```

### 6.3.3 nc / netcat - Network Swiss Army Knife

This versatile networking tool reads and writes data across network connections using TCP or UDP protocols, serving as a backend tool for network debugging, port scanning, and data transfer.

```bash
nc -zv host 22          # Test port 22 on host
nc -l 8080              # Listen on port 8080
```

## 6.4 Network Services

### 6.4.1 ssh - Secure Shell

SSH provides encrypted remote terminal connections to other systems, enabling secure system administration across networks. Key-based authentication offers stronger security than password-based access.

```bash
ssh user@hostname           # SSH connection
ssh -p 2222 user@hostname   # SSH on specific port
ssh -i key.pem user@hostname  # SSH with key file
```

### 6.4.2 scp - Secure Copy

This command securely transfers files between systems using SSH encryption, providing confidentiality and integrity protection during file transfers.

```bash
scp file.txt user@host:/path/   # Copy to remote
scp -r dir/ user@host:/path/    # Recursive copy
scp -P 2222 file.txt user@host:/path/ # Specific port
```

# 7 I/O Redirection Commands

## 7.1 Standard Streams

Linux processes use three standard streams for communication: standard input for receiving data, standard output for normal results, and standard error for diagnostic messages.

Redirection operators control how these streams interact with files and other processes.

- stdin (0) - Standard Input
- stdout (1) - Standard Output
- stderr (2) - Standard Error

## 7.2 Redirection Operators

### 7.2.1 > - Redirect Output (Overwrite)

The greater-than symbol redirects standard output to files, overwriting existing content. The double greater-than symbol appends output to files without overwriting, ideal for log files that accumulate data over time.

**Bash**

```
ls > file.txt              # Redirect output to file
```

### 7.2.2 >> - Redirect Output (Append)

**Bash**

```
echo "new line" >> file.txt    # Append to file
```

### 7.2.3 < - Redirect Input

The less-than symbol redirects file contents to standard input, allowing programs to read from files without explicit file handling code.

**Bash**

```
sort < input.txt           # Use file as input
```

### 7.2.4 2> - Redirect stderr

The two greater-than symbol preceded by the number two redirects only error messages to files, separating normal output from error messages for better log management.

**Bash**

```
command 2> error.log       # Redirect errors to file
```

### 7.2.5 &> - Redirect stdout and stderr

The ampersand greater-than symbol redirects both standard output and standard error to the same destination, simplifying output capture when both stream types need identical handling.

```Bash
command &> output.log        # Redirect both output and errors
```

## 7.3 Advanced Redirection

### 7.3.1 | - Pipe

The vertical bar connects the output of one command to the input of another, creating powerful command pipelines that combine simple utilities to perform complex processing tasks.

```Bash
ls -l | grep ".txt"          # Pipe output to grep
cat file.txt | sort | uniq   # Multiple pipes
```

### 7.3.2 tee - Split Output

The tee command splits output streams, sending data both to standard output and to files simultaneously. This allows real-time monitoring while simultaneously capturing output for later analysis.

```Bash
ls -l | tee listing.txt      # Output to screen and file
ls -l | tee -a listing.txt   # Append to file
```

### 7.3.3 xargs - Build Commands from Input

Xargs constructs argument lists from standard input and executes commands repeatedly with the generated arguments. This efficiently handles operations on large numbers of files that exceed command line length limitations.

```Bash
find . -name "*.txt" | xargs rm # Delete all txt files
echo "file1 file2" | xargs cp -t /backup/ # Copy files
```

## 7.4 Here Documents

This redirection method embeds input text directly within shell scripts, providing inline data to commands without requiring external files.

**Bash**

```
cat << EOF > script.sh
```

***File Content:***

**Bash**

```
#!/bin/bash
echo "Script content"
EOF
```

# 8 Environment Variable Commands

## 8.1 Variable Management

### 8.1.1 export - Set Environment Variables

The export command defines environment variables that persist across process boundaries, making them available to child processes. These variables configure application behavior, define paths, and set system preferences.

```Bash
export PATH=$PATH:/new/path     # Add to PATH
export VAR_NAME="value"         # Set variable
```

### 8.1.2 unset - Remove Variables

This command removes environment variables from the current shell environment, effectively undoing their definitions and freeing associated resources.

```Bash
unset VAR_NAME              # Remove variable
```

### 8.1.3 env - Display Environment

The env command lists all currently defined environment variables, providing a comprehensive view of the process execution environment. It can also run commands in modified environments.

```Bash
env                     # Show all environment variables
env | grep PATH             # Search for specific variable
```

## 8.2 Common Environment Variables

Common environment variables include HOME for user home directories, PATH for executable search paths, USER for current username identification, SHELL for default shell specification, PWD for current working directory, and LANG for system language and localization settings.

**Bash**

```
echo $HOME      # User's home directory
echo $PATH      # Command search path
echo $USER      # Current username
echo $SHELL     # Current shell
echo $PWD       # Current working directory
echo $LANG      # System language
```

## 8.3 Configuration Files

System-wide environment settings reside in etc profile, while user-specific configurations are stored in hidden files within home directories. The bashrc file contains shell-specific settings loaded for interactive shells, while profile files contain login session configurations.

- /etc/profile - System-wide environment
- /etc/bash.bashrc - System-wide bash configuration
- ~/.bashrc - User-specific bash configuration
- ~/.bash_profile - User-specific login configuration
- ~/.profile - User-specific profile

## 8.4 Variable Examples

Temporary variables exist only in current sessions, while permanent variables require adding definitions to configuration files. Sourcing configuration files applies changes to current sessions without requiring logout and login cycles.

**Bash**

```
# Set temporary variable
MY_VAR="temporary value"

# Set permanent variable (add to ~/.bashrc)
echo 'export MY_VAR="permanent value"' >> ~/.bashrc

# Source the changes
source ~/.bashrc

# Check variable
echo $MY_VAR
```

# 9 User Management Commands

## 9.1 User Account Management

### 9.1.1 useradd - Create User

This command creates new user accounts with configurable options for home directories, default shells, group memberships, and account expiration dates. Proper user setup ensures appropriate resource access and system security.

```bash
useradd username                    # Create user
useradd -m -s /bin/bash username    # Create with home directory and shell
useradd -g groupname username       # Create with primary group
```

### 9.1.2 usermod - Modify User

The user modification command changes existing account properties including group memberships, home directories, login shells, and account status. The append option for groups prevents replacing existing group memberships.

```bash
usermod -aG groupname username  # Add user to supplementary group
usermod -s /bin/sh username     # Change user shell
usermod -L username             # Lock user account
usermod -U username             # Unlock user account
```

### 9.1.3 userdel - Delete User

This command removes user accounts from the system. The remove option deletes associated home directories and mail spools, ensuring complete account removal.

```bash
userdel username            # Delete user
userdel -r username         # Delete user with home directory
```

## 9.2 Group Management

### 9.2.1 groupadd - Create Group

Groupadd creates new user groups with optional specific group ID assignments. Groups facilitate permission management for multiple users requiring similar resource access.

```Bash
groupadd groupname          # Create new group
groupadd -g 1005 groupname  # Create with specific GID
```

### 9.2.2 groupmod - Modify Group

This command changes group properties including group names and group ID numbers, though caution is advised as changing IDs can affect file ownership.

```Bash
groupmod -n newname oldname  # Rename group
```

### 9.2.3 groupdel - Delete Group

Group deletion removes group definitions from the system, provided no users have the group as their primary group assignment.

```Bash
groupdel groupname          # Delete group
```

## 9.3 Password Management

### 9.3.1 passwd - Change Password

The passwd command changes user passwords and manages password properties. System administrators can lock accounts to prevent login while preserving user data, or expire passwords to force changes.

```bash
passwd                  # Change current user's password
passwd username         # Change another user's password (root only)
passwd -l username      # Lock user password
```

### 9.3.2 chage - Password Aging

This command manages password expiration policies including maximum password ages, change deadlines, and expiration warnings, enforcing security policies through password lifecycle management.

```bash
chage -l username       # List password aging info
chage -M 90 username    # Set maximum password days
```

## 9.4 User Information

### 9.4.1 id - User Identity

The id command shows user and group identities with corresponding numeric IDs, helping verify effective permissions and group memberships.

```bash
id                      # Current user identity
id username             # Specific user identity
```

### 9.4.2 who / w - Logged-in Users

These commands display currently logged-in users and their activities. The w command provides additional detail including login time and current processes.

```Bash
who                 # Show logged-in users
w                   # Show users and their processes
```

### 9.4.3 last - Login History

Last shows user login history, helping track system access patterns and investigate security incidents through historical authentication data.

```Bash
last                # Recent logins
last username               # Login history for user
```

## 9.5 Sudo Management

### 9.5.1 visudo - Edit Sudoers File

Visudo safely edits the sudoers file with syntax checking and lock protection, preventing configuration errors that could lock administrators out of privilege escalation capabilities.

```Bash
visudo              # Safe edit of sudoers file
```

### 9.5.2 su - Switch User

The substitute user command changes effective user identity, with the hyphen option providing complete login environment simulation including home directory and user-specific settings.

```Bash
su - username               # Switch to user with environment
su username                 # Switch to user
sudo -i             # Switch to root
```