

Shell Cheatsheet

Essential commands for command-line navigation and automation

This cheatsheet provides a comprehensive reference to fundamental Shell/Bash commands, syntax, and scripting techniques, ideal for both beginners and experienced users for efficient system administration and automation.

File Operations Create, copy, move and manage files	Navigation Navigate directories and file systems	Text Processing Search, filter and manipulate text
Process Management Control and monitor system processes	System Information View system status and resources	

File & Directory Operations

List Files: `ls`

Display files and directories in the current location.

```
# List files in current directory
ls
# List with detailed information
ls -l
# Show hidden files
ls -a
# List with human-readable file sizes
ls -lh
# Sort by modification time
ls -lt
```

Create Files: `touch`

Create empty files or update timestamps.

```
# Create a new file
touch newfile.txt
# Create multiple files
touch file1.txt file2.txt file3.txt
# Update timestamp of existing file
touch existing_file.txt
```

Create Directories: `mkdir`

Create new directories.

```
# Create a directory
mkdir my_directory
# Create nested directories
mkdir -p parent/child/grandchild
# Create multiple directories
mkdir dir1 dir2 dir3
```

Copy Files: `cp`

Copy files and directories.

```
# Copy a file
cp source.txt destination.txt
# Copy directory recursively
cp -r source_dir dest_dir
# Copy with confirmation prompt
cp -i file1.txt file2.txt
# Preserve file attributes
cp -p original.txt copy.txt
```

Move/Rename: `mv`

Move or rename files and directories.

```
# Rename a file
mv oldname.txt newname.txt
# Move file to directory
mv file.txt /path/to/directory/
# Move multiple files
mv file1 file2 file3 target_directory/
```

Delete Files: `rm`

Remove files and directories.

```
# Delete a file
rm file.txt
# Delete directory and contents
rm -r directory/
# Force delete without confirmation
rm -f file.txt
# Interactive deletion (confirm each)
rm -i *.txt
```

Navigation & Path Management

Navigate through the file system efficiently.

01	02	03
Current Directory: `pwd` Print the current working directory path.	Change Directory: `cd` Change to a different directory.	Directory Tree: `tree` Display directory structure in tree format.

```
# Show current directory
pwd
# Example output:
/home/user/documents
```

```
# Go to home directory
cd ~
# Go to parent directory
cd ..
# Go to previous directory
cd -
# Go to specific directory
cd /path/to/directory
```

```
# Show directory tree
tree
# Limit depth to 2 levels
tree -L 2
# Show only directories
tree -d
```

Text Processing & Search

View Files: `cat` / `less` / `head` / `tail`

Display file contents in different ways.

```
# Display entire file
cat file.txt
# View file page by page
less file.txt
# Show first 10 lines
head file.txt
# Show last 10 lines
tail file.txt
# Show last 20 lines
tail -n 20 file.txt
# Follow file changes (useful for logs)
tail -f logfile.txt
```

Search in Files: `grep`

Search for patterns in text files.

```
# Search for pattern in file
grep "pattern" file.txt
# Case-insensitive search
grep -i "pattern" file.txt
# Search recursively in directories
grep -r "pattern" directory/
# Show line numbers
grep -n "pattern" file.txt
# Count matching lines
grep -c "pattern" file.txt
```

Find Files: `find`

Locate files and directories based on criteria.

```
# Find files by name
find . -name "*.txt"
# Find files by type
find . -type f -name "config*"
# Find directories
find . -type d -name "backup"
# Find files modified in last 7 days
find . -mtime -7
# Find and execute command
find . -name "*.log" -delete
```

Text Manipulation: `sed` / `awk` / `sort`

Process and manipulate text data.

```
# Replace text in file
sed 's/old/new/g' file.txt
# Extract specific columns
awk '{print $1, $3}' file.txt
# Sort file contents
sort file.txt
# Remove duplicate lines
sort file.txt | uniq
# Count word frequency
cat file.txt | tr ' ' '\n' | sort | uniq -c
```

File Permissions & Ownership

View Permissions: `ls -l`

Display detailed file permissions and ownership.

```
# Show detailed file information
ls -l
# Example output:
-rw-r--r-- 1 user group 1024 Jan 1 12:00 file.txt
# d = directory, r = read, w = write, x = execute
```

Change Permissions: `chmod`

Modify file and directory permissions.

```
# Give execute permission to owner
chmod +x script.sh
# Set specific permissions (755)
chmod 755 file.txt
# Remove write permission for group/others
chmod go-w file.txt
# Recursive permission change
chmod -R 644 directory/
```

Change Ownership: `chown` / `chgrp`

Change file owner and group.

```
# Change owner
chown newowner file.txt
# Change owner and group
chown newowner:newgroup file.txt
# Change group only
chgrp newgroup file.txt
# Recursive ownership change
chown -R user:group directory/
```

Permission Numbers

Understanding numeric permission notation.

```
# Permission calculation:
# 4 = read (r), 2 = write (w), 1 = execute (x)
# 755 = rwx-r-x-r-x (owner: rwx, group: r-x, others: r-x)
# 644 = rw-r-r-- (owner: rw-, group: r--, others: r--)
# 777 = rwxrwxrwx (full permissions for all)
# 600 = rw----- (owner: rw-, group: ---, others: ---)
```

Process Management

Monitor and control running processes.

View Processes: `ps` / `top` / `htop`

Display information about running processes.

```
# Show processes for current user
ps
# Show all processes with details
ps aux
# Show processes in tree format
ps -ef --forest
# Interactive process viewer
top
# Enhanced process viewer (if available)
htop
```

Background Jobs: `&` / `jobs` / `fg` / `bg`

Manage background and foreground processes.

```
# Run command in background
command &
# List active jobs
jobs
# Bring job to foreground
fg %1
# Send job to background
bg %1
# Suspend current process
Ctrl+Z
```

Kill Processes: `kill` / `killall`

Terminate processes by PID or name.

```
# Kill process by PID
kill 1234
# Force kill process
kill -9 1234
# Kill all processes with name
killall firefox
# Send specific signal
kill -TERM 1234
```

System Monitoring: `free` / `df` / `du`

Monitor system resources and disk usage.

```
# Show memory usage
free -h
# Show disk space
df -h
# Show directory size
du -sh directory/
# Show largest directories
du -h --max-depth=1 | sort -hr
```

Input/Output Redirection

Control command input, output, and data flow.

Redirection: `>` / `>>` / `<`

Redirect command output and input.

```
# Redirect output to file (overwrite)
command > output.txt
# Append output to file
command >> output.txt
# Redirect input from file
command < input.txt
# Redirect both output and errors
command > output.txt 2>&1
# Discard output
command > /dev/null
```

Pipes: `|`

Chain commands together using pipes.

```
# Basic pipe usage
command1 | command2
# Multiple pipes
cat file.txt | grep "pattern" | sort | uniq
# Count lines in output
ps aux | wc -l
# Page through long output
ls -la | less
```

Tee: `tee`

Write output to both file and stdout.

```
# Save output and display it
command | tee output.txt
# Append to file
command | tee -a output.txt
# Multiple outputs
command | tee file1.txt file2.txt
```

Here Documents: `<<`

Provide multi-line input to commands.

```
# Create file with here document
cat << EOF > file.txt
Line 1
Line 2
Line 3
EOF

# Send email with here document
mail user@example.com << EOF
Subject: Test
This is a test message.
EOF
```

Variables & Environment

Work with shell variables and environment settings.

Variables: Assignment & Usage

Create and use shell variables.

```
# Assign variables (no spaces around =)
name="John"
count=42

# Use variables
echo $name
echo "Hello, $name"
echo "Count: ${count}"

# Command substitution
current_dir=$(pwd)
date_today=$(date +%Y-%m-%d)
```

Environment Variables: `export` / `env`

Manage environment variables.

```
# Export variable to environment
export PATH="/new/path:$PATH"
export MY_VAR="value"

# View all environment variables
env
# View specific variable
echo $HOME
echo $PATH

# Unset variable
unset MY_VAR
```

Special Variables

Built-in shell variables with special meanings.

```
# Script arguments
$0 # Script name
$1, $2, $3... # First, second, third argument
 $# # Number of arguments
$@ # All arguments as separate words
$* # All arguments as single word
 $? # Exit status of last command

# Process information
$$ # Current shell PID
$! # PID of last background command
```

Parameter Expansion

Advanced variable manipulation techniques.

```
# Default values
${var:-default} # Use default if var is empty
${var:=default} # Set var to default if empty

# String manipulation
${var#pattern} # Remove shortest match from beginning
${var##pattern} # Remove longest match from beginning
${var%pattern} # Remove shortest match from end
${var%%pattern} # Remove longest match from end
```

Scripting Basics

Write and execute shell scripts for automation.

Script Structure

Basic script format and execution.

```
#!/bin/bash
# This is a comment

# Variables
greeting="Hello, World!"
user=$(whoami)

# Output
echo $greeting
echo "Current user: $user"

# Make script executable:
chmod +x script.sh
# Run script:
./script.sh
```

Conditional Statements: `if`

Control script flow with conditions.

```
#!/bin/bash
if [ -f "file.txt" ]; then
    echo "File exists"
elif [ -d "directory" ]; then
    echo "Directory exists"
else
    echo "Neither exists"
fi

# String comparison
if [ "$USER" = "root" ]; then
    echo "Running as root"
fi

# Numeric comparison
if [ $count -gt 10 ]; then
    echo "Count is greater than 10"
fi
```

Loops: `for` / `while`

Repeat commands using loops.

```
#!/bin/bash

# For loop with range
for i in {1..5}; do
    echo "Number: $i"
done

# For loop with files
for file in *.txt; do
    echo "Processing: $file"
done

# While loop
count=1
while [ $count -le 5 ]; do
    echo "Count: $count"
    count=$((count + 1))
done
```

Functions

Create reusable code blocks.

```
#!/bin/bash

# Define function
greet() {
    local name=$1
    echo "Hello, $name!"
}

# Function with return value
add_numbers() {
    local sum=$(( $1 + $2 ))
    echo $sum
}

# Call functions
greet "Alice"
result=$(add_numbers 5 3)
echo "Sum: $result"
```

Network & System Commands

Network diagnostics and system information commands.

Network Commands

Test connectivity and network configuration.

```
# Test network connectivity
ping google.com
ping -c 4 google.com # Send only 4 packets

# DNS lookup
nslookup google.com
dig google.com

# Network configuration
ip addr show # Show IP addresses
ip route show # Show routing table

# Download files
wget https://example.com/file.txt
curl -O https://example.com/file.txt
```

System Information: `uname` / `whoami` / `date`

Get system and user information.

```
# System information
uname -a # All system info
uname -r # Kernel version
hostname # Computer name
whoami # Current username
id # User ID and groups

# Date and time
date # Current date/time
date +%Y-%m-%d # Custom format
uptime # System uptime
```

Archive & Compression: `tar` / `zip`

Create and extract compressed archives.

```
# Create tar archive
tar -czf archive.tar.gz directory/
# Extract tar archive
tar -xzf archive.tar.gz

# Create zip archive
zip -r archive.zip directory/
# Extract zip archive
unzip archive.zip

# View archive contents
tar -tzf archive.tar.gz
unzip -l archive.zip
```

File Transfer: `scp` / `rsync`

Transfer files between systems.

```
# Copy file to remote server
scp file.txt user@server:/path/to/destination

# Copy from remote server
scp user@server:/path/to/file.txt .

# Sync directories (local to remote)
rsync -avz local_dir/ user@server:/remote_dir/

# Sync with delete (mirror)
rsync -avz --delete local_dir/ user@server:/remote_dir/
```

Command History & Shortcuts

Navigate command history and use keyboard shortcuts efficiently.

Command History: `history`

View and reuse previous commands.

```
# Show command history
history
# Show last 10 commands
history 10

# Execute previous command
!!
# Execute command by number
!123
# Execute last command starting with 'ls'
!!s

# Search history interactively
Ctrl+R
```

History Expansion

Reuse parts of previous commands.

```
# Last command's arguments
!# # Last argument of previous command
!^ # First argument of previous command
!* # All arguments of previous command

# Example usage:
ls /very/long/path/to/file.txt
cd !$ # Goes to /very/long/path/to/file.txt
```

Keyboard Shortcuts

Essential shortcuts for efficient command line usage.

```
# Navigation
Ctrl+A # Move to beginning of line
Ctrl+E # Move to end of line
Ctrl+F # Move forward one character
Ctrl+B # Move backward one character
Alt+F # Move forward one word
Alt+B # Move backward one word

# Editing
Ctrl+U # Clear line before cursor
Ctrl+K # Clear line after cursor
Ctrl+W # Delete word before cursor
Ctrl+Y # Paste last deleted text

# Process control
Ctrl+C # Interrupt current command
Ctrl+Z # Suspend current command
Ctrl+D # Exit shell or EOF
```

Reference: This cheatsheet covers essential Shell/Bash commands and modern practices for efficient command-line operations and system administration workflows.