The Linux Command Line Guide

Table of Contents

1. Introduction to Linux for Software Testers . . . 3
   1.1. Pervasive Use in Docker Environments . . . 3
   1.2. Dominance on Servers . . . 3
   1.3. Integration with Cloud Platforms . . . 3
   1.4. Empowering Test Automation . . . 4
   1.5. How "The Linux Command Line Guide" Can Help . . . 4
   1.6. Conclusion . . . 4
2. File Management Commands . . . 5
   2.1. `cd` - Change Directory . . . 5
   2.2. `mkdir` - Create Directories . . . 5
   2.3. `rmdir` - Remove Directories . . . 5
   2.4. `ls` - List Directory Contents . . . 6
   2.5. `cp` - Copy Files and Directories . . . 6
   2.6. `mv` - Move and Rename Files and Directories . . . 7
   2.7. `rm` - Remove Files and Directories . . . 7
   2.8. `find` - Search for Files and Directories . . . 8
   2.9. `du` - Estimate File Space Usage . . . 8
   2.10. `df` - Report File System Disk Space Usage . . . 9
   2.11. `ln` - Create Symbolic Links . . . 10
3. Text Processing Commands . . . 11
   3.1. `cat` - Concatenate and Display File Content . . . 11
   3.2. `grep` - Print Lines Matching a Pattern . . . 11
   3.3. `awk` - Pattern Scanning and Processing Language . . . 11
   3.4. `sed` - Stream Editor . . . 11
   3.5. `sort` - Sort Lines of Text Files . . . 11
   3.6. `uniq` - Report or Omit Repeated Lines . . . 12
   3.7. `wc` - Print Newline, Word, and Byte Counts . . . 12
4. Process Management Commands . . . 13
   4.1. `ps` - Report a Snapshot of Current Processes . . . 13
   4.2. `top` - Display Linux Tasks . . . 13
   4.3. `htop` - Interactive Process Viewer . . . 13
   4.4. `kill` - Send a Signal to a Process . . . 13
   4.5. `pkill` - Send a Signal to Processes by Name . . . 13
   4.6. `nice` and `renice` - Modify Scheduling Priority . . . 13
   4.7. `bg` and `fg` - Manage Jobs in the Background and Foreground . . . 14
5. Networking Commands . . . 15
   5.1. `ping` - Send ICMP ECHO_REQUEST to Network Hosts . . . 15

5.2. ifconfig - Configure a Network Interface . . . 15
5.3. ip - Show/Manipulate Routing, Devices, Policy Routing, and Tunnels . . . 15
5.4. netstat - Print Network Connections . . . 15
5.5. ss - Investigate Sockets . . . 15
5.6. curl - Transfer Data from or to a Server . . . 15
5.7. wget - Non-Interactive Network Downloader . . . 16
5.8. nc (Netcat) - Read and Write Network Connections . . . 16
6. System Monitoring and Performance Commands . . . 17
6.1. dmesg - Print or Control the Kernel Ring Buffer . . . 17
6.2. iostat - Report CPU and I/O Statistics . . . 17
6.3. vmstat - Report Virtual Memory Statistics . . . 17
6.4. sar - Collect, Report, or Save System Activity Information . . . 17
6.5. free - Display Amount of Free and Used Memory . . . 17
6.6. uptime - Tell How Long the System Has Been Running . . . 18
6.7. lsof - List Open Files . . . 18
7. Package Management Commands . . . 19
7.1. pacman - Package Manager Utility for Arch Linux and Manjaro . . . 19
7.2. yay - AUR Helper . . . 19
8. Version Control Commands . . . 20
8.1. git - Distributed Version Control System . . . 20
9. Managing Command Output in Linux . . . 21
9.1. Piping . . . 21
9.2. Redirecting Output to a File . . . 21
9.3. Appending Output to a File . . . 21
9.4. tee . . . 22
9.5. Redirecting Standard Error . . . 23
9.6. Redirecting Both Standard Output and Standard Error . . . 23
9.7. xargs . . . 23
9.8. Process Substitution . . . 23

Chapter 1. Introduction to Linux for Software Testers

In today's rapidly evolving tech landscape, having a solid understanding of Linux is invaluable for software testers. Linux's robustness, flexibility, and widespread adoption make it a cornerstone in various environments, including Docker containers, on-premises servers, and even cloud platforms such as Azure. Here's why gaining Linux proficiency can significantly enhance your effectiveness and versatility as a software tester:

1.1. Pervasive Use in Docker Environments

Docker has revolutionized the way applications are developed, tested, and deployed. At the heart of Docker containers lies Linux. Containers encapsulate applications along with their dependencies, ensuring consistency across different stages of development and production. As a software tester, knowing Linux allows you to:

- Navigate and manage Docker containers efficiently.
- Debug issues that arise within containers.
- Write and execute test scripts within the containerized environments.

1.2. Dominance on Servers

Linux is the operating system of choice for a majority of servers worldwide. Its stability, security, and performance make it ideal for server environments. Whether you're testing applications hosted on-premises or in the cloud, understanding Linux helps you:

- Set up and configure test environments.
- Monitor server performance and diagnose issues.
- Automate testing processes using shell scripts and other Linux tools.

1.3. Integration with Cloud Platforms

While cloud platforms like Azure offer support for various operating systems, Linux remains a popular choice due to its cost-effectiveness and compatibility with open-source tools. As cloud adoption grows, familiarity with Linux can benefit you by:

- Enhancing your ability to work with cloud-based development and testing tools.
- Leveraging Linux's built-in features for scalable and automated testing.
- Ensuring your skills are applicable across multiple cloud providers, not just Azure.

1.4. Empowering Test Automation

Linux's powerful command-line interface and scripting capabilities are a boon for test automation. As a software tester, you can use Linux to:

- Automate repetitive testing tasks using shell scripts.
- Schedule and manage automated test runs with cron jobs.
- Utilize Linux tools like `grep`, `awk`, and `sed` for log analysis and data manipulation.

1.5. How "The Linux Command Line Guide" Can Help

"The Linux Command Line Guide" is an invaluable resource for software testers, providing a comprehensive collection of essential commands and practical examples. This guide can help you in several ways:

- **Efficiency**: Quickly find the right commands to perform routine tasks such as file manipulation, directory navigation, and system monitoring.
- **Troubleshooting**: Learn how to use powerful Linux tools to diagnose and resolve issues in your testing environments.
- **Automation**: Discover commands and scripting techniques that can automate your testing workflows, saving time and reducing manual effort.
- **Skill Enhancement**: Deepen your understanding of the Linux command line, making you more proficient and confident in handling various testing scenarios.
- **Consistency**: Ensure consistency and reliability in your testing processes by leveraging the robust and well-documented Linux commands.

Incorporating "The Linux Command Line Guide" into your toolkit will empower you to work more effectively and efficiently, harnessing the full potential of Linux in your daily testing activities.

1.6. Conclusion

In summary, acquiring Linux skills opens up a myriad of opportunities for software testers. From working seamlessly with Docker containers to managing servers and automating tests, Linux proficiency equips you with the tools and knowledge to excel in diverse testing environments. As the industry continues to evolve, staying abreast of Linux and its applications ensures you remain a valuable and adaptable professional in the field of software testing.

Chapter 2. File Management Commands

2.1. `cd` - Change Directory

The `cd` command is used to change the current working directory.

Example: Change to the user's home directory

```
$ cd ~
```

Example: Change to a specific directory

```
$ cd /path/to/directory
```

Example: Change to the parent directory

```
$ cd ..
```

Example: Change to the previous directory (using the `cd` command with a hyphen)

```
$ cd -
```

2.2. `mkdir` - Create Directories

The `mkdir` command is used to create directories.

Example: Create a single directory

```
$ mkdir directory_name
```

Example: Create multiple directories

```
$ mkdir dir1 dir2 dir3
```

Example: Create a directory with specific permissions

```
$ mkdir -m 755 directory_name
```

2.3. `rmdir` - Remove Directories

The `rmdir` command is used to remove directories.

Example: Remove an empty directory

```
$ rmdir directory_name
```

Example: Remove multiple empty directories

```
$ rmdir dir1 dir2 dir3
```

2.4. `ls` - List Directory Contents

The `ls` command is used to list the contents of a directory.

Example: List all files and directories in the current directory

```
$ ls
```

Example: View detailed information with file sizes in a human-readable format

```
$ ls -lh
```

Example: Sort by modification time (most recent first)

```
$ ls -lt
```

Example: Include hidden files (those starting with a dot)

```
$ ls -a
```

Example: Create an alias for your favorite `ls` combination

```
$ alias lsl='ls -lh --color=auto'
```

2.5. `cp` - Copy Files and Directories

The `cp` command is used to copy files and directories.

Example: Copy a file to a destination directory

```
$ cp file.txt destination/
```

Example: Copy a directory and its contents recursively to a destination directory

```
$ cp -r directory/ destination/
```

Example: Preserve the original file attributes (permissions, timestamps, etc.) during copy

```
$ cp -a file.txt destination/
```

Example: Prompt before overwriting an existing file

```
$ cp -i source/file.txt destination/
```

Example: Display progress while copying files

```
$ cp -v file.txt destination/
```

2.6. `mv` - Move and Rename Files and Directories

The `mv` command is used to move and rename files and directories.

Example: Move a file to a destination directory

```
$ mv file.txt destination/
```

Example: Rename a file

```
$ mv oldfile.txt newfile.txt
```

Example: Move a directory and its contents to a new location

```
$ mv directory/ new_location/
```

Example: Move multiple files to a destination directory

```
$ mv file1.txt file2.txt destination/
```

Example: Move a file and preserve the original file attributes

```
$ mv -p file.txt destination/
```

2.7. `rm` - Remove Files and Directories

The `rm` command is used to remove files and directories.

Example: Remove a file

```
$ rm file.txt
```

Example: Remove a directory and its contents recursively

```
$ rm -r directory/
```

Example: Forcefully remove files without prompting for confirmation

```
$ rm -f file.txt
```

Example: Remove a directory only if it is empty

```
$ rmdir directory/
```

Example: Remove multiple files at once

```
$ rm file1.txt file2.txt
```

2.8. find - Search for Files and Directories

The find command is used to search for files and directories in a directory hierarchy.

Example: Find files with a specific name in the current directory

```
$ find . -name "filename.txt"
```

Example: Find files modified in the last 7 days

```
$ find . -mtime -7
```

Example: Find directories with a specific name

```
$ find . -type d -name "dirname"
```

Example: Find files of a specific type (e.g., text files)

```
$ find . -type f -name "*.txt"
```

Example: Execute a command on the found files (e.g., delete)

```
$ find . -type f -name "*.tmp" -exec rm {} +
```

2.9. du - Estimate File Space Usage

The du command is used to estimate file space usage.

Example: Display disk usage for the current directory

```
$ du
```

Example: Display disk usage in human-readable format

```
$ du -h
```

Example: Display disk usage for a specific directory

```
$ du -h /path/to/directory
```

Example: Display total disk usage for multiple directories

```
$ du -h /path/to/directory1 /path/to/directory2
```

Example: Display disk usage only for directories, not individual files

```
$ du -h --max-depth=1
```

2.10. df - Report File System Disk Space Usage

The `df` command is used to report file system disk space usage.

Example: Display disk space usage for all mounted file systems

```
$ df
```

Example: Display disk space usage in human-readable format

```
$ df -h
```

Example: Display disk space usage for a specific file system

```
$ df -h /dev/sda1
```

Example: Display disk space usage for specific file systems

```
$ df -h /dev/sda1 /dev/sdb1
```

Example: Display disk space usage with total and available space in 1K blocks

```
$ df -k
```

2.11. `ln` - Create Symbolic Links

The `ln` command is used to create symbolic links to files or directories.

Example: Create a symbolic link to a file

```
$ ln -s file1.txt link1.txt
```

Example: Create a symbolic link to a directory

```
$ ln -s directory1 directory2
```

Example: Create a symbolic link with an absolute path

```
$ ln -s /path/to/file1.txt link1.txt
```

Chapter 3. Text Processing Commands

3.1. `cat` - Concatenate and Display File Content

The `cat` command is used to concatenate and display file content.

Example: Display file content

```
$ cat file.txt
```

3.2. `grep` - Print Lines Matching a Pattern

The `grep` command is used to print lines matching a pattern.

Example: Search for a pattern in a file

```
$ grep 'pattern' file.txt
```

Example: Recursively search for a pattern

```
$ grep -r 'pattern' /path
```

3.3. `awk` - Pattern Scanning and Processing Language

The `awk` command is used for pattern scanning and processing.

Example: Print the first field of each line

```
$ awk '{print $1}' file.txt
```

3.4. `sed` - Stream Editor

The `sed` command is used for filtering and transforming text.

Example: Replace all occurrences of `old` with `new`

```
$ sed 's/old/new/g' file.txt
```

3.5. `sort` - Sort Lines of Text Files

The `sort` command is used to sort lines of text files.

Example: Sort lines in a file

```
$ sort file.txt
```

3.6. `uniq` - Report or Omit Repeated Lines

The `uniq` command is used to report or omit repeated lines.

Example: Remove duplicate lines

```
$ sort file.txt | uniq
```

3.7. `wc` - Print Newline, Word, and Byte Counts

The `wc` command is used to print newline, word, and byte counts for each file.

Example: Count the number of lines in a file

```
$ wc -l file.txt
```

Chapter 4. Process Management Commands

4.1. `ps` - Report a Snapshot of Current Processes

The `ps` command is used to report a snapshot of current processes.

Example: Display all running processes

```
$ ps aux
```

4.2. `top` - Display Linux Tasks

The `top` command provides an interactive, real-time view of running processes.

4.3. `htop` - Interactive Process Viewer

`htop` is an enhanced version of `top` for viewing processes interactively.

4.4. `kill` - Send a Signal to a Process

The `kill` command is used to send a signal to a process.

Example: Forcefully terminate a process with a specific PID

```
$ kill -9 PID
```

4.5. `pkill` - Send a Signal to Processes by Name

The `pkill` command is used to send a signal to processes by name.

Example: Kill processes by name

```
$ pkill -f processname
```

4.6. `nice` and `renice` - Modify Scheduling Priority

The `nice` and `renice` commands are used to run a program with modified scheduling priority.

Example: Run a command with a lower priority

```
$ nice -n 10 command
```

Example: Change the priority of an existing process

```
$ renice 10 -p PID
```

4.7. bg and fg - Manage Jobs in the Background and Foreground

The bg and fg commands are used to manage jobs in the background and foreground.

Example: Resume job 1 in the background

```
$ bg %1
```

Example: Bring job 1 to the foreground

```
$ fg %1
```

Chapter 5. Networking Commands

5.1. `ping` - Send ICMP ECHO_REQUEST to Network Hosts

The `ping` command is used to check connectivity to a host.

Example: Check connectivity to a host

```
$ ping google.com
```

5.2. `ifconfig` - Configure a Network Interface

The `ifconfig` command is deprecated. Use `ip` instead.

5.3. `ip` - Show/Manipulate Routing, Devices, Policy Routing, and Tunnels

The `ip` command is used to show or manipulate routing, devices, policy routing, and tunnels.

Example: Show all IP addresses

```
$ ip a
```

Example: Bring an interface up

```
$ ip link set eth0 up
```

5.4. `netstat` - Print Network Connections

The `netstat` command is deprecated. Use `ss` instead.

5.5. `ss` - Investigate Sockets

The `ss` command is used to investigate sockets.

Example: List all listening ports

```
$ ss -tuln
```

5.6. `curl` - Transfer Data from or to a Server

The `curl` command is used to transfer data from or to a server.

Example: Download a webpage

```
$ curl http://example.com
```

Example: Download a file

```
$ curl -O http://example.com/file.zip
```

5.7. wget - Non-Interactive Network Downloader

The wget command is used to download files non-interactively.

Example: Download a file

```
$ wget http://example.com/file.zip
```

5.8. nc (Netcat) - Read and Write Network Connections

The nc (Netcat) command is used for reading and writing network connections.

Example: Listen on port 1234

```
$ nc -l 1234
```

Example: Connect to port 1234 on a host

```
$ nc hostname 1234
```

Chapter 6. System Monitoring and Performance Commands

6.1. dmesg - Print or Control the Kernel Ring Buffer

The dmesg command is used to print or control the kernel ring buffer.

Example: Check for hardware errors

```
$ dmesg | grep error
```

6.2. iostat - Report CPU and I/O Statistics

The iostat command is used to report CPU and I/O statistics.

Example: Display extended I/O statistics

```
$ iostat -x
```

6.3. vmstat - Report Virtual Memory Statistics

The vmstat command is used to report virtual memory statistics.

Example: Display system performance continuously

```
$ vmstat 1
```

6.4. sar - Collect, Report, or Save System Activity Information

The sar command is used to collect, report, or save system activity information.

Example: CPU usage every second

```
$ sar -u 1
```

6.5. free - Display Amount of Free and Used Memory

The free command is used to display the amount of free and used memory in the system.

Example: Display memory in a human-readable format

```
$ free -h
```

6.6. `uptime` - Tell How Long the System Has Been Running

The `uptime` command is used to show how long the system has been running.

6.7. `lsof` - List Open Files

The `lsof` command is used to list open files.

Example: List processes using port 80

```
$ lsof -i TCP:80
```

Chapter 7. Package Management Commands

7.1. pacman - Package Manager Utility for Arch Linux and Manjaro

The pacman command is the package manager utility for Arch Linux and Manjaro.

Example: Update the system

```
$ sudo pacman -Syu
```

Example: Install a package

```
$ sudo pacman -S package_name
```

Example: Remove a package

```
$ sudo pacman -R package_name
```

7.2. yay - AUR Helper

The yay command is used to manage packages from the Arch User Repository (AUR).

Example: Install an AUR package

```
$ yay -S package_name
```

Example: Remove an AUR package

```
$ yay -R package_name
```

Example: Update all packages including AUR

```
$ yay -Syu
```

Chapter 8. Version Control Commands

8.1. `git` - Distributed Version Control System

The `git` command is used for distributed version control.

Example: Clone a repository

```
$ git clone repo_url
```

Example: Show the working tree status

```
$ git status
```

Example: Add file contents to the index

```
$ git add file.txt
```

Example: Record changes to the repository

```
$ git commit -m "message"
```

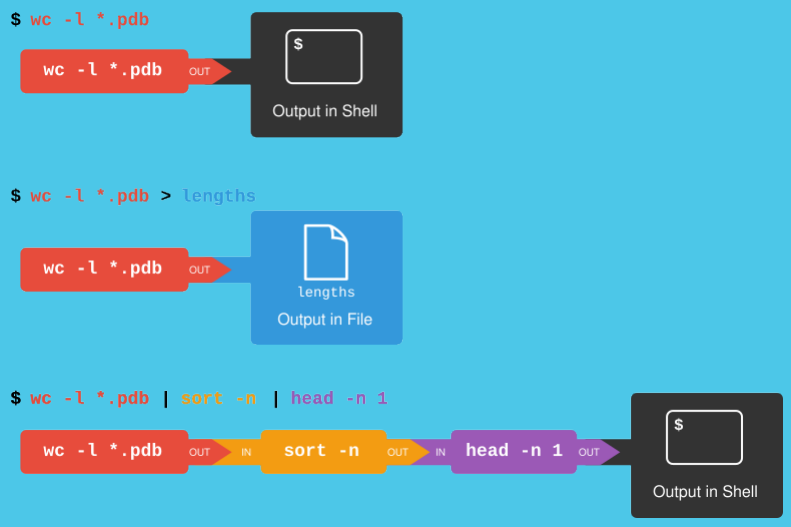
Example: Update remote refs along with associated objects

```
$ git push origin branch_name
```

Example: Fetch from and integrate with another repository or a local branch

```
$ git pull origin branch_name
```

Chapter 9. Managing Command Output in Linux



Linux provides powerful mechanisms for managing command output, including piping, redirecting, appending, and other advanced tools. These mechanisms enable users to perform complex tasks efficiently by combining simple commands. This section explains these mechanisms in detail.

9.1. Piping

Piping is a mechanism that allows the output of one command to be used as the input for another command. This is achieved using the pipe symbol (`|`). It enables chaining commands to perform complex tasks efficiently.

Example: Using `grep` and `wc` to count lines containing "ERROR"

```
$ grep "ERROR" application.log | wc -l
```

9.2. Redirecting Output to a File

Redirecting output to a file involves saving the output of a command to a file instead of displaying it on the terminal. This is done using the `>` symbol. If the file already exists, it will be overwritten.

Example: Saving the output of `ls` to a file

```
$ ls > output.txt
```

9.3. Appending Output to a File

Appending output to a file is similar to redirecting, but instead of overwriting the file, it adds the

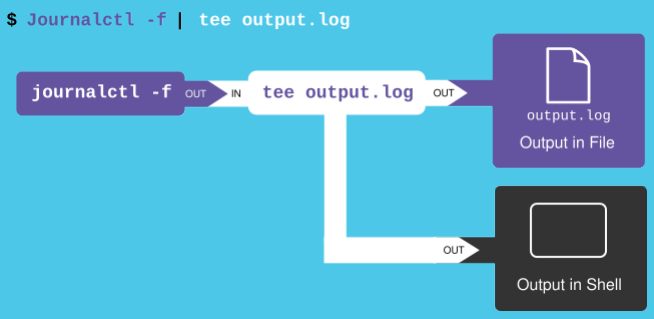
output to the end of the file. This is done using the >> symbol.

Example: Appending the output of `echo` *to a file*

```
$ echo "New log entry" >> output.txt
```

9.4. tee

The `tee` command reads from standard input and writes to both standard output and one or more files simultaneously. This is useful when you want to see the output on the screen and save it to a file at the same time.



Example: Using `tee` *to write journaling output to a file and display it added with date and time*

```
$ journalctl -f | grep --line-buffered "refused" | awk '{ print strftime("[%Y-%m-%d
%H:%M:%S]"), $0; fflush(); }' | tee journalFiltered.txt
```

Below is a detailed explanation of each part of the command:

1. `journalctl -f`
   - This command continuously fetches new log entries from the systemd journal.
   - The `-f` option makes `journalctl` follow the log, displaying new entries as they are added.
2. `grep --line-buffered "refused"`
   - The output from `journalctl` is piped to `grep` to filter for lines containing the word "refused".
   - The `--line-buffered` option ensures that `grep` outputs each line as soon as it is matched, providing real-time filtering.
3. `awk '{ print strftime("[%Y-%m-%d %H:%M:%S]"), $0; fflush(); }'`
   - The filtered lines are then piped to `awk`, which processes each line to prepend the current date and time.
   - `strftime("[%Y-%m-%d %H:%M:%S]")` formats the current date and time.
   - `$0` represents the current line of input.
   - `print` outputs the formatted date and time followed by the original line.
   - `fflush()` ensures that `awk` flushes its output buffer after each line, maintaining real-time processing.

4. `tee journalFiltered.txt`
    - Finally, the output from `awk` is piped to `tee`.
    - `tee` writes the output to both the console and the specified file (`journalFiltered.txt`).

Using this pipeline, you can monitor logs containing "refused" in real-time, with each line prefixed by a timestamp, and save the output to a file for further analysis.

9.5. Redirecting Standard Error

You can redirect standard error (stderr) separately from standard output (stdout). This is useful for capturing error messages.

Example: Redirecting stderr to a file

```
$ command 2> error.log
```

9.6. Redirecting Both Standard Output and Standard Error

You can redirect both stdout and stderr to the same file or different files.

Example: Redirecting both stdout and stderr to the same file

```
$ command > output.log 2>&1
```

Example: Redirecting stdout and stderr to different files

```
$ command > output.log 2> error.log
```

9.7. `xargs`

The `xargs` command builds and executes command lines from standard input. It is useful for processing a list of items and executing commands on each item.

Example: Using `xargs` to remove files listed by `find`

```
$ find . -name "*.tmp" | xargs rm
```

9.8. Process Substitution

Process substitution allows you to pass the output of a command as an input to another command that expects a file. This is done using `<(command)`.

Example: Comparing the output of two commands with `diff`

```
$ diff <(ls dir1) <(ls dir2)
```