Problem Solver (DI03032011) - Problem Solver

Antigravity AI

June 4, 2026

Contents

CHAPTER 1

Introduction to Linux System Administration

Linux system administration relies heavily on understanding the file system hierarchy and executing commands efficiently. This chapter focuses on the fundamental command-line procedures for navigating the system, managing files, viewing data, and utilizing built-in documentation.

1.1 Navigating the Linux File System

Linux organizes files in a hierarchical tree structure defined by the Filesystem Hierarchy Standard (FHS).

Methodology:

Navigating the Linux File System To navigate and interact with the Linux file system, use the following standard command-line procedures:

1. **Print Working Directory:** Execute `pwd` to display the absolute path of the current directory.
2. **Change Directory:** Execute `cd <path>` to move to a different location.
   - `cd /` navigates to the root directory.
   - `cd ~` navigates to the current user's home directory.
   - `cd ..` navigates up one directory level.
3. **List Directory Contents:** Execute `ls` to view files and subdirectories.
   - `ls -l` displays long format (shows permissions, ownership, size, and modification date).
   - `ls -a` displays all files including hidden files (starting with a dot `.`).
   - `ls -la` combines long format and hidden file display.

Worked Example:

Navigating and Listing Files A system administrator needs to verify their current location, move to the system configuration directory, and list all hidden files in long format to inspect permissions.
Solution:

1. **Step 1: Check current directory.** Execute the print working directory command:
   `pwd`
2. **Step 2: Change to the system configuration directory.** The standard configuration directory in the FHS is `/etc`. Execute:
   `cd /etc`
3. **Step 3: List all files including hidden ones in long format.** Execute the list command with `-l` and `-a` flags:
   `ls -la`
4. **Result:** The output will display detailed file metadata for all configuration files.

1.2 Basic File and Directory Operations

Methodology:

Creating and Managing Files and Directories The following algorithms are used for standard file and directory management:

1. **Create a Directory:** Use `mkdir <directory_name>`.
2. **Create an Empty File:** Use `touch <file_name>`.
3. **Copy Files or Directories:** Use `cp <source> <destination>`.
   - Add the `-r` flag (`cp -r`) to copy directories recursively.
4. **Move or Rename Files:** Use `mv <source> <destination>`.
5. **Remove Files or Directories:** Use `rm <file_name>`.
   - Add the `-r` flag (`rm -r`) to remove directories and their contents.
   - Use `rmdir <directory_name>` to remove only empty directories.

Worked Example:

File Organization Task You must create a project directory named `webapp`, create an `index.html` file inside it, copy the `nginx.conf` file from `/etc/` to the new directory, and then rename the directory to `production_app`.
Solution:

1. **Step 1: Create the directory.** `mkdir webapp`
2. **Step 2: Navigate into the directory and create the empty file.** `cd webapp`
   `touch index.html`
3. **Step 3: Copy the configuration file to the current location.** The dot (`.`) represents the current directory.
   `cp /etc/nginx.conf .`
4. **Step 4: Move up one level and rename the directory.** `cd ..`
   `mv webapp production_app`

1.3 Viewing File Contents

Methodology:

Tools for Viewing File Contents To read the contents of files in the terminal without opening an editor, apply the following commands based on the file size and reading requirements:

1. **Concatenate and Display:** Best for short files that fit on one screen.
   `cat <file_name>`
2. **Paging Through Files:** Best for large files. Allows vertical scrolling.
   `less <file_name>`
3. **Viewing the Beginning:** Extracts the first 10 lines by default.
   `head -n <lines> <file_name>`
4. **Viewing the End:** Extracts the last 10 lines by default.
   `tail -n <lines> <file_name>`

5. **Following Live Logs:** Continuously updates terminal output as new lines are appended to a file. `tail -f <log_file>`

Worked Example:

Log File Analysis A system error recently occurred. You need to view the last 15 lines of the system message log located at `/var/log/syslog`, and then monitor it continuously for any new incoming error messages.

Solution:

1. **Step 1: View the last 15 lines of the log.** Use the `tail` command with the `-n` option: `tail -n 15 /var/log/syslog`
2. **Step 2: Monitor the log file in real-time.** Use the `tail` command with the `-f` (follow) option: `tail -f /var/log/syslog`
3. **Step 3: Terminate the real-time monitoring.** Press `Ctrl + C` in the terminal to interrupt the command and return to the prompt.

1.4 Understanding File Permissions

Methodology:

Reading File Permissions Executing `ls -l` outputs a 10-character string (e.g., `-rwxr-xr-`) representing file type and access rights. Read it using the following method:

1. **Identify File Type:** Analyze the first character.
   - `-` indicates a regular file.
   - `d` indicates a directory.
   - `l` indicates a symbolic link.
2. **Identify Access Blocks:** The remaining 9 characters form three sets of 3 characters:
   - **Owner (User):** Characters 2-4
   - **Group:** Characters 5-7
   - **Others (World):** Characters 8-10
3. **Interpret Permissions:** Each triplet specifies Read (`r`), Write (`w`), and Execute (`x`) rights. A hyphen (`-`) means the permission is denied.
   - `r`: Allows reading file contents or listing directory contents.
   - `w`: Allows modifying the file or adding/removing files in a directory.
   - `x`: Allows executing a program or entering a directory via `cd`.

Worked Example:

Interpreting Permission Strings An administrator runs `ls -l` and observes the following output for a file named `backup.sh`:

```
-rwxrw-r- 1 root sysadmin 2048 Mar 15 10:00 backup.sh
```

Determine the file type and the exact permissions for the owner, group, and others.

Solution:

1. **Step 1: Analyze the first character.** The string starts with `-`, indicating that `backup.sh` is a regular file.

2. **Step 2: Analyze Owner permissions (characters 2-4).** The block is `rwx`. The owner (`root`) has read, write, and execute permissions.

3. **Step 3: Analyze Group permissions (characters 5-7).** The block is `rw-`. Members of the group (`sysadmin`) can read and write, but cannot execute the file.

4. **Step 4: Analyze Others permissions (characters 8-10).** The block is `r-`. Any other user on the system has strictly read-only access.

1.5 Accessing System Documentation

Methodology:

Using Man Pages and Help Systems To quickly find command syntax, flags, and usage examples, administrators rely on built-in documentation systems:

1. **Manual Pages (`man`):** To open the detailed manual for any command, use:
   `man <command_name>`
   - Press `Space` to page down.
   - Press `/` followed by a search term to find specific keywords.
   - Press `q` to exit the manual.
2. **Short Help Option:** To print a brief summary of flags directly in the terminal, use:
   `<command_name> -help`
3. **Keyword Search (`apropos`):** To find a command based on its functionality, search the manual descriptions using:
   `apropos <keyword>`

Worked Example:

Finding Command Information An administrator needs to list directory contents and sort them by modification time (newest first), but does not remember the exact flag for the `ls` command. Determine the flag using the manual pages.
Solution:

1. **Step 1: Open the manual page.** Execute: `man ls`
2. **Step 2: Search for the sorting keyword.** Type `/sort by time` or `/modification time` and press Enter to jump to relevant matches.
3. **Step 3: Locate the correct flag.** The manual will reveal that the `-t` flag is used to sort by modification time (newest first).
4. **Step 4: Exit and execute.** Press `q` to quit the manual page.
   Execute: `ls -t`

CHAPTER 2

User and Group Management

In Linux system administration, managing users, groups, and file permissions is crucial for system security and organization. This chapter focuses on the command-line techniques and problem-solving methods required to perform these administrative tasks efficiently.

2.1 Managing Users

Managing users involves creating new user accounts, modifying existing account properties, and defining password aging policies to maintain security.

Methodology:

User Creation and Modification To manage user accounts, use the following commands and their common options:

1. **Create a User (`useradd`):**
   - `useradd -m <username>` (Creates user and home directory)
   - `useradd -d <path> <username>` (Specifies custom home directory)
   - `useradd -s <shell> <username>` (Specifies default shell e.g. `/bin/bash`)
   - `useradd -e YYYY-MM-DD <username>` (Sets account expiration date)
2. **Modify a User (`usermod`):**
   - `usermod -l <newname> <oldname>` (Changes username)
   - `usermod -L <username>` (Locks the user account)
   - `usermod -U <username>` (Unlocks the user account)
3. **Delete a User (`userdel`):**
   - `userdel <username>` (Deletes user account only)
   - `userdel -r <username>` (Deletes user account and their home directory)
4. **Verify User Identity (`id`):**
   - `id <username>` (Displays UID GID and group memberships)

Worked Example:

Creating and Modifying a User Account **Problem:** Perform the following administrative tasks:

1. Create a user named `alice` with a custom home directory `/home/alicedev` and default shell `/bin/bash`.
2. Set the account to expire on December 31 2025.

3. Lock the account temporarily to restrict access.
4. Verify the user details to ensure correctness.

Solution:

1. Create the user with specified attributes:

```
sudo useradd -m -d /home/alicedev -s /bin/bash -e 2025-12-31 alice
```

2. Lock the user account:

```
sudo usermod -L alice
```

3. Verify the user creation and basic attributes:

```
id alice
```

To verify the expiration date and lock status you can check:

```
sudo chage -l alice
```

Methodology:

Password Management and Policies Administrators use `passwd` to set passwords and `chage` to enforce password aging policies.

1. **Set or Change Password:**
   - `passwd <username>` (Prompts to enter a new password)
2. **Configure Password Policies (`chage`):**
   - `chage -M <days> <username>` (Maximum days before password change is required)
   - `chage -m <days> <username>` (Minimum days between password changes)
   - `chage -W <days> <username>` (Warning days before password expires)
   - `chage -I <days> <username>` (Inactive days before account is locked after expiration)
   - `chage -l <username>` (Lists current password aging policies)

Worked Example:

Enforcing Password Aging Policies **Problem:** Configure password policies for the user `bob` such that:

1. He must change his password at least every 90 days.
2. He cannot change his password more than once every 7 days.
3. He receives a warning 14 days before expiration.

Solution:

1. Apply the password aging policies using `chage`:

```
sudo chage -M 90 -m 7 -W 14 bob
```

2. Verify the applied policies:

```
sudo chage -l bob
```

2.2 Managing Groups

Groups simplify the administration of multiple users by allowing collective permission assignments. Rather than setting permissions for each individual user they can be assigned to a group.

Methodology:

Group Creation and Membership Management

1. **Create a Group (`groupadd`):**
   - `groupadd <groupname>`
   - `groupadd -g <GID> <groupname>` (Creates a group with a specific GID)
2. **Modify and Delete Groups:**
   - `groupmod -n <newname> <oldname>` (Renames a group)
   - `groupdel <groupname>` (Deletes a group)
3. **Manage Group Memberships:**
   - `usermod -aG <groupname> <username>` (Appends user to a supplementary group without removing them from other groups)
   - `gpasswd -a <username> <groupname>` (Adds a single user to a group)
   - `gpasswd -d <username> <groupname>` (Removes a user from a group)
4. **View Group Memberships:**
   - `groups <username>`

Worked Example:

Configuring Developer Groups **Problem:** Assume you are the administrator managing access for a development team.

1. Create a new group named `developers`.
2. Add existing users `alice` and `bob` to the `developers` group.
3. Remove `bob` from the group.
4. Rename the group to `devteam`.

Solution:

1. Create the group:

```
sudo groupadd developers
```

2. Add users to the group (using the append parameter to preserve other groups):

```
sudo usermod -aG developers alice
sudo usermod -aG developers bob
```

3. Remove `bob` from the group using `gpasswd`:

```
sudo gpasswd -d bob developers
```

4. Rename the group using `groupmod`:

```
sudo groupmod -n devteam developers
```

2.3 File Permissions and Ownership

Linux uses a robust permission model to control access to files and directories based on three entity types: User (Owner) Group and Others. Understanding numerical values and symbolic representations is key to solving access control problems.

Methodology:

Changing File Permissions Using chmod Permissions are Read (r=4) Write (w=2) and Execute (x=1). They can be configured using two modes:

1. **Absolute (Octal) Mode:** Permissions are represented by a three-digit octal number (User-Group-Others).
   - `chmod 755 <file>` (Owner: rwx Group: r-x Others: r-x)
   - `chmod 644 <file>` (Owner: rw- Group: r– Others: r–)
   - `chmod -R 770 <directory>` (Recursively applies to directory and contents)
2. **Symbolic Mode:** Uses symbols to add (+) remove (-) or set (=) permissions. Entities are `u` (user) `g` (group) `o` (others) `a` (all).
   - `chmod u+x g-w <file>` (Adds execute for owner removes write for group)
   - `chmod a=r <file>` (Sets read-only for everyone)

Worked Example:

Calculating and Applying Octal Permissions **Problem:** A directory `/projectdata` requires strict access controls:

- The owner should have full access (read write execute).
- The group members should be able to read and execute but not write.
- Others should have absolutely no access.

Calculate the exact octal value and provide the command to apply it recursively.
Solution:

1. **Calculate User (Owner) permission:** Read(4) + Write(2) + Execute(1) = 7.
2. **Calculate Group permission:** Read(4) + Execute(1) = 5.

3. **Calculate Others permission:** No access = 0.
4. **Form the Octal value:** Combine the three digits to get 750.
5. **Write the Command:** Use the `-R` flag for recursive application.

```
sudo chmod -R 750 /projectdata
```

Methodology:

Changing File Ownership Using chown and chgrp Administrators can transfer file ownership to different users or groups depending on organizational shifts.

1. **Change Owner:**
   - `chown <newowner> <file>`
2. **Change Owner and Group simultaneously:**
   - `chown <newowner>:<newgroup> <file>`
3. **Change Group only (`chgrp`):**
   - `chgrp <newgroup> <file>`
4. **Recursive Ownership Change:**
   - `chown -R <newowner>:<newgroup> <directory>`

Worked Example:

Modifying Ownership for a Collaborative Directory **Problem:** You have a directory `/var/www/html/webapp`. You need to set the user ownership to `www-data` and the group ownership to `devteam`. This change must apply to all existing files and subdirectories inside it.
Solution:

1. Apply the recursive `chown` command specifying both user and group separated by a colon:

   ```
   sudo chown -R www-data:devteam /var/www/html/webapp
   ```

2. Verify the changes by listing the directory contents with the long format option:

   ```
   ls -l /var/www/html/webapp
   ```

Methodology:

Special Permissions Management Special permissions provide advanced access control features which are highly critical in multi-user environments:

1. **SUID (Set User ID - Value: 4000):** When applied to an executable file the file runs with the privileges of the file owner rather than the user executing it. (Symbolic: `u+s`)
2. **SGID (Set Group ID - Value: 2000):**
   - On a file: It runs with the privileges of the file's group.

- On a directory: New files created within inherit the directory's group ownership. (Symbolic: `g+s`)

3. **Sticky Bit (Value: 1000):** When applied to a directory only the file owner the directory owner or the root user can delete or rename files within it. This is widely used for `/tmp`. (Symbolic: `+t`)

Worked Example:

Implementing a Shared Workspace with Special Permissions **Problem:** Design a shared directory `/shareddocs` for the `staff` group. Enforce the following rules:

1. The directory is owned by `root` and the group `staff`.
2. Any new files created inside automatically belong to the `staff` group.
3. Users can only delete files they themselves created not files created by other users.
4. The baseline permissions should be full access for owner group and others (`rwxrwxrwx`) before applying special permissions.

Solution:

1. Create the directory:

   ```
   sudo mkdir /shareddocs
   ```

2. Set the ownership to root user and staff group:

   ```
   sudo chown root:staff /shareddocs
   ```

3. The requirement asks for SGID (ensures new files belong to staff group) and Sticky Bit (restricts deletion to file owners).

   - SGID value = 2000
   - Sticky Bit value = 1000
   - Total special permission = 3000
   - Baseline permissions = 777

   Combine these to get the four-digit octal value `3777` and apply it:

   ```
   sudo chmod 3777 /shareddocs
   ```

4. Verify the permissions. The output should display `drwxrwsrwt`:

   ```
   ls -ld /shareddocs
   ```

CHAPTER 3

Process and Service Management

3.1 Process Management

In a Linux environment, processes are executing instances of programs. Effective management involves viewing, monitoring, and controlling these processes to ensure system stability.

3.1.1 Viewing and Controlling Processes

Methodology:

Process Identification and Termination To identify and terminate specific processes, follow these steps:

1. **List Processes**: Use the `ps` command to display active processes.
   - `ps aux`: Displays all running processes for all users.
   - `ps -ef`: Displays a full-format listing of all processes.
2. **Search for a Specific Process**: Combine `ps` with `grep` to find a specific program's Process ID (PID).
3. **Terminate the Process**: Use the `kill` command followed by the PID.
   - `kill [PID]`: Sends the default TERM (15) signal for graceful termination.
   - `kill -9 [PID]`: Sends the KILL (9) signal for immediate and forced termination.

Worked Example:

Terminating an Unresponsive Application **Problem:** An application named "firefox" has become unresponsive. Locate its Process ID and force terminate it.
Solution:

1. **Find the PID**: Run the following command to search for the Firefox process:

   ```
   ps aux | grep firefox
   ```

   Output:

   ```
   user     14562  5.2  3.1 123456 54321 ?    Sl   10:00   0:15 /usr/lib/firefox
   user     14589  0.0  0.0   9876  1234 ?    S    10:01   0:00 grep firefox
   ```

   The PID for the main Firefox process is `14562`.
2. **Terminate the process gracefully**:

```
kill 14562
```

3. **Force terminate (if the process does not respond)**:

```
kill -9 14562
```

4. **Verify Termination**: Run the search command again to ensure the process is no longer listed.

3.1.2 Background and Foreground Job Control

Methodology:

Managing Jobs in the Shell Job control allows you to pause, resume, and move processes between the background and foreground.

1. **Start a Background Job**: Append an ampersand (`&`) to the command (e.g., `command &`).
2. **Suspend a Foreground Job**: Press `Ctrl + Z` to pause the currently running foreground process.
3. **List Active Jobs**: Execute the `jobs` command to view all suspended and background jobs along with their job IDs.
4. **Move to Background**: Use the `bg %[JobID]` command to resume a suspended job in the background.
5. **Bring to Foreground**: Use the `fg %[JobID]` command to bring a background or suspended job to the foreground.

Worked Example:

Moving a Task to the Background **Problem:** You started a script `backup.sh` in the foreground which is tying up your terminal. You need to move it to the background so you can continue using the terminal. **Solution:**

1. **Suspend the running process**: Press `Ctrl + Z`. The terminal will output:

```
[1]+  Stopped                 ./backup.sh
```

2. **Verify the job ID**: Run the `jobs` command.

```
jobs
```

Output:

```
[1]+  Stopped                 ./backup.sh
```

The job ID is `1`.

3. **Resume the job in the background**: Execute the `bg` command with the job ID.

```
bg %1
```

Output:

```
[1]+ ./backup.sh &
```

4. **Bring it back to the foreground (Optional)**: If you need to interact with it again, use:

```
fg %1
```

3.2 System Services Management

Services (or daemons) are processes that run continuously in the background to provide essential system functionalities.

Methodology:

Service Management using systemctl Modern Linux distributions use `systemd` to manage services. The primary tool is `systemctl`.

1. **Check Service Status**: `systemctl status [service_name]` displays whether the service is active, running, or failed.
2. **Start a Service**: `systemctl start [service_name]` initiates the service immediately.
3. **Stop a Service**: `systemctl stop [service_name]` halts the running service.
4. **Restart a Service**: `systemctl restart [service_name]` stops and then starts the service. Useful after configuration changes.
5. **Enable at Boot**: `systemctl enable [service_name]` configures the service to start automatically when the system boots.
6. **Disable at Boot**: `systemctl disable [service_name]` prevents the service from starting automatically.

Note: Legacy systems may use `service [name] start/stop` or `chkconfig [name] on/off`.

Worked Example:

Configuring the SSH Service **Problem:** You need to ensure the OpenSSH server (`sshd`) is currently running and will start automatically upon system reboots.
Solution:

1. **Check the current status**:

```
sudo systemctl status sshd
```

Output might indicate it is "inactive (dead)".

2. **Start the service**:

```
sudo systemctl start sshd
```

3. **Enable the service for automatic startup at boot**:

```
sudo systemctl enable sshd
```

Output:

```
Created symlink /etc/systemd/system/multi-user.target.wants/sshd.service ->
/usr/lib/systemd/system/sshd.service.
```

4. **Verify the final configuration**: Check the status again to ensure it is both active and enabled.

```
sudo systemctl status sshd
```

Output snippet:

```
Loaded: loaded (/usr/lib/systemd/system/sshd.service; enabled; vendor preset: enabled)
Active: active (running)
```

3.3 Monitoring System Performance

System administrators must constantly monitor resources like CPU, Memory, and Disk I/O to maintain optimal performance.

Methodology:

Resource Monitoring Tools Several command-line tools provide insights into system resource utilization:

1. **`top`**: Provides a dynamic and real-time view of running system processes, CPU usage, and memory consumption.
2. **`htop`**: An interactive and visually improved alternative to `top` that allows scrolling and easier process management.
3. **`free`**: Displays the total, used, and available amount of physical and swap memory in the system. Use the `-h` flag for human-readable output.
4. **`vmstat`**: Reports information about processes, memory, paging, block IO, traps, and CPU activity. Use `vmstat [delay] [count]` for continuous monitoring.

Worked Example:

Diagnosing System Slowness **Problem:** The system is responding sluggishly. Use built-in performance monitoring tools to determine if the issue is a lack of RAM or a CPU-intensive process.
Solution:

1. **Check Memory Usage**: Execute the `free` command.

```
free -h
```

Output:

```
               total        used        free      shared  buff/cache   available
Mem:            15Gi        14Gi       500Mi       200Mi       500Mi       800Mi
```

```
Swap:          4.0Gi       2.0Gi       2.0Gi
```

Observation: The "available" memory is very low (800Mi), and swap space is actively being used (2.0Gi). This indicates a potential memory bottleneck.

2. **Identify High Resource Consumers**: Run the `top` command.

```
top
```

Action within `top`: Press `Shift + M` to sort processes by Memory usage to find the culprit.

3. **Analyze the CPU Load**: While still in `top`, look at the `load average` (e.g., `load average: 4.50, 4.20, 3.90`). If this number is consistently higher than the number of CPU cores, the CPU is overloaded. Press `Shift + P` to sort by CPU usage.

4. **Continuous Monitoring (Optional)**: To watch memory and CPU trends over time, use `vmstat` with a 2-second delay for 5 iterations.

```
vmstat 2 5
```

Methodology:

Log File Analysis Log files are critical for diagnosing system errors and auditing security. Most logs are stored in the `/var/log/` directory.

1. `cat`: Outputs the entire content of a log file. Best for short logs.
2. `less`: Allows paging through large log files interactively. Search backward/forward using `/pattern`.
3. `tail`: Outputs the last 10 lines of a file by default.
4. `tail -f`: Continuously monitors a file and outputs new lines as they are written (useful for real-time debugging).
5. **Common Logs**:
   - `/var/log/syslog` or `/var/log/messages`: Global system activity logs.
   - `/var/log/auth.log` or `/var/log/secure`: Authentication and authorization events.

Worked Example:

Troubleshooting a Failed Web Server via Logs **Problem:** The Apache web server service failed to start. Analyze the system logs to identify the root cause.
Solution:

1. **Check the most recent system messages**: Use `tail` to view the last 20 lines of the system log.

```
sudo tail -n 20 /var/log/syslog
```

2. **Search for specific service errors**: Combine `cat` or `less` with `grep` to filter for "apache2".

```
sudo cat /var/log/syslog | grep apache2
```

Output snippet:

```
Aug 14 10:15:22 server apache2[4512]: Address already in use: make_sock:
could not bind to address 0.0.0.0:80
Aug 14 10:15:22 server systemd[1]: apache2.service: Control process exited,
code=exited, status=1
```

3. **Interpret the Error**: The log clearly states `Address already in use... could not bind to address 0.0.0.0:80`. This indicates another process is currently running on port 80.
4. **Monitor Logs in Real-time**: If you attempt to restart the service and want to see the error happen live, use:

   ```
   sudo tail -f /var/log/syslog
   ```

   Then, in a separate terminal window, try to start the service again.

CHAPTER 4

Shell Scripting Basics

Shell scripting is a powerful way to automate repetitive system administration tasks in Linux. A shell script is essentially a text file containing a sequence of commands that the shell interprets and executes. This chapter covers the foundational computational techniques for writing shell scripts, controlling execution flow, and scheduling automated jobs.

4.1 Writing and Executing a Simple Bash Script

Methodology:

Writing and Executing a Script Follow these steps to create and run a basic shell script:

1. **Create a file:** Open a text editor and create a new file with a `.sh` extension (e.g., `script.sh`).
2. **Add the Shebang:** The first line must be `#!/bin/bash`. This tells the operating system to use the Bash shell to execute the script.
3. **Write commands:** Add the Linux commands you want to execute sequentially.
4. **Make it executable:** Run `chmod +x script.sh` to grant execution permissions.
5. **Execute the script:** Run the script using `./script.sh` from the current directory.

Worked Example:

Basic Execution **Problem:** Write a shell script that clears the terminal screen and displays the current system date and time.
Solution:

1. Create the file `show_date.sh`.
2. Add the following code:

```
#!/bin/bash
clear
echo "The current system date and time is:"
date
```

3. Make the script executable:

```
chmod +x show_date.sh
```

4. Run the script:

```
./show_date.sh
```