Chapter 3
Manage Files from the Command Line

1. Manage Files from the Command Line

Understanding the Linux File System Hierarchy

Overview

This guide provides a comprehensive overview of:

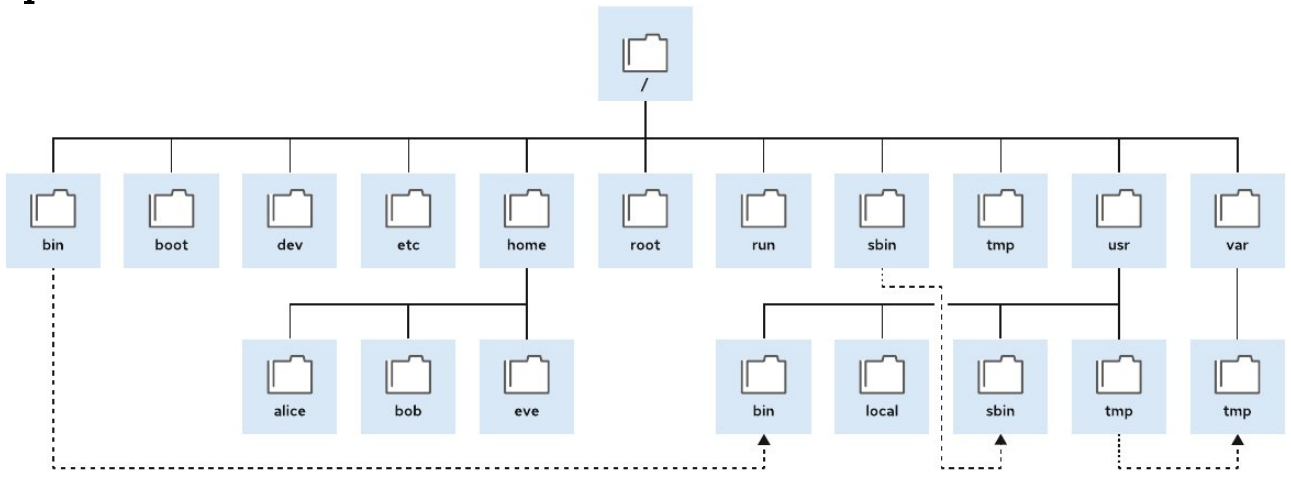
- Linux File System Structure: Understand key directories and their roles.
- Navigating the File System: Learn to specify file locations using absolute and relative paths.
- Directory Management: Efficiently navigate and list directory contents.
- File Operations: Create, copy, move, and delete files and directories using command-line tools.
- File Linking: Establish multiple references to the same file using hard and symbolic (soft) links, understanding their differences, benefits, and limitations.
- Pattern Matching: Execute commands affecting multiple files using Bash shell pattern-matching techniques.
- Task Scheduling:
  - Schedule a command to run at a later time.
  - Automate recurring tasks using a user's crontab file.
  - Set up system-wide scheduled tasks using crontab files and directories.

The Linux File System Structure

In Linux, all files are organized into a single unified structure known as the file system hierarchy. This structure follows an inverted tree model, where the root of the system is at the top, and directories and subdirectories extend downward.

The root directory (/) serves as the starting point for all files and directories. The / symbol not only represents the root but also functions as a separator in file paths. For instance, if etc is a subdirectory within /, it is referred to as /etc. Likewise, a file named issue inside /etc would be referenced as /etc/issue.

Each subdirectory within / is designated for a specific purpose, organizing system files efficiently. For example, /boot is responsible for storing essential files required for booting the system.

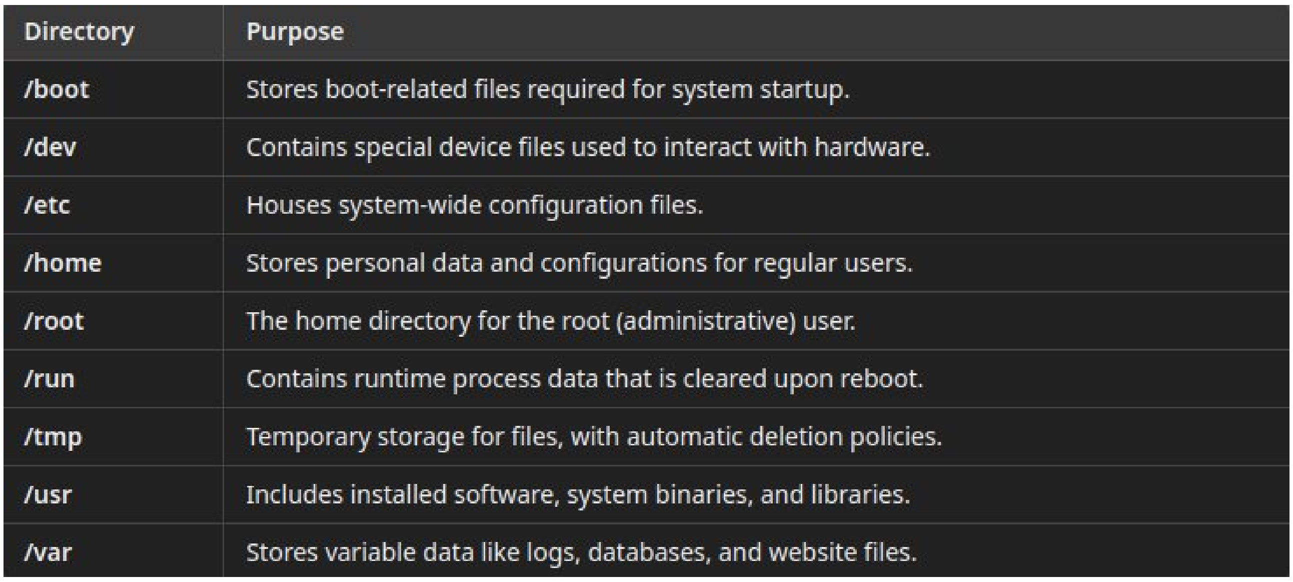


Types of File System Content

To better understand the organization of files, Linux classifies them into different types:

- Static files: Remain unchanged unless manually modified (e.g., configuration files).
- Dynamic or variable files: Can be altered or updated by running processes (e.g., log files).
- Persistent files: Retain their data even after a system reboot (e.g., user configurations).
- Runtime files: Exist only while the system is running and are deleted upon reboot (e.g., temporary process data).

Key Directories and Their Roles



Directory	Purpose
/boot	Stores boot-related files required for system startup.
/dev	Contains special device files used to interact with hardware.
/etc	Houses system-wide configuration files.
/home	Stores personal data and configurations for regular users.
/root	The home directory for the root (administrative) user.
/run	Contains runtime process data that is cleared upon reboot.
/tmp	Temporary storage for files, with automatic deletion policies.
/usr	Includes installed software, system binaries, and libraries.
/var	Stores variable data like logs, databases, and website files.

Important File System Changes in Modern Red Hat Linux

Starting with Red Hat Enterprise Linux 7, certain directories that were previously separate have been merged under /usr. These include:

- **/bin → now a symbolic link to /usr/bin**
- **/sbin → now a symbolic link to /usr/sbin**
- **/lib → now a symbolic link to /usr/lib**
- **/lib64 → now a symbolic link to /usr/lib64**

These changes were implemented to simplify system structure and enhance compatibility with other Linux distributions.

For further details, consult the hier(7) manual page using the command:

```
man 7 hier
```

Specifying Files by Name in Linux

Absolute vs. Relative Paths

A file or directory’s path defines its location within the Linux file system. Paths are composed of directories separated by forward slashes (/). Since Linux organizes files in a hierarchical structure, directories can contain both files and subdirectories.

Handling Spaces in File Names

File names in Linux can contain spaces, but since the shell interprets spaces as argument separators, special handling is required. If a file name includes spaces, wrap it in quotation marks to prevent misinterpretation. For simplicity and to avoid errors, it is generally recommended to avoid spaces in file names.

Absolute Paths

An absolute path specifies a file’s full location from the root directory (/). Since each file has a unique absolute path, the system can directly locate it. Absolute paths always begin with /. For example, the system log file has an absolute path:
/var/log/messages

Absolute paths can be long to type, so relative paths provide a more convenient alternative in many cases.

The Current Working Directory and Relative Paths

When a user logs in, their default working directory is their home directory. System processes also have default directories. The current working directory refers to the location from which commands are executed. Relative paths specify file locations based on the working directory.
A relative path does not begin with / but instead provides directions from the current directory. For example, from the /var directory, the log file can be accessed as:
log/messages

Case Sensitivity in Linux File Systems

Linux file systems (e.g., ext4, XFS, BtrFS) are case-sensitive, meaning FileCase.txt and filecase.txt are distinct files. However, some non-Linux file systems, like VFAT and NTFS, are case-preserving but not case-sensitive, meaning the two names would refer to the same file.

Navigating the File System

Displaying the Current Directory

Use the pwd command to print the current working directory:

```
[user@host ~]$ pwd
/home/user
```

Listing Directory Contents

To list files in a directory, use ls. If no argument is specified, it lists the contents of the current directory:

```
[user@host ~]$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  Templates
Videos
```

Changing Directories

The cd command changes the current directory. Without arguments, it returns to the home directory:

```
[user@host ~]$ cd Videos
[user@host Videos]$ pwd
/home/user/Videos
[user@host Videos]$ cd /home/user/Documents
[user@host Documents]$ pwd
/home/user/Documents
[user@host Documents]$ cd
[user@host ~]$ pwd
/home/user
```

The default shell prompt displays the last component of the absolute path. If the home directory is the working directory, ~ is shown instead.

Creating and Modifying Files3

The touch command updates a file's timestamp or creates an empty file:

```
[user@host ~]$ touch Videos/movie1.ogg
[user@host ~]$ touch Documents/report1.odf
```

Listing Files with Detailed Information

The ls command has options to display file details:

- -l shows detailed information.
- -a includes hidden files.
- -R lists subdirectory contents.

```
[user@host ~]$ ls -l
```

Special Directories and Hidden Files

- . represents the current directory.
- .. refers to the parent directory.
- Hidden files start with a dot (.) and are excluded from normal ls output. Use ls -a to view them.

Quick Directory Navigation

- cd - switches to the last used directory.
- cd .. moves to the parent directory.

```
[user@host ~]$ cd Videos
[user@host Videos]$ cd /home/user/Documents
[user@host Documents]$ cd -
[user@host Videos]$
[user@host Videos]$ cd ..
[user@host ~]$
```

References

man pages: bash(1), cd(1), ls(1), pwd(1), unicode(7), utf-8(7)

Managing Files with Command-Line Utilities

File Management via Command Line

System administrators frequently perform file and directory operations such as creation, copying, moving, and deletion. Some commands directly interact with files, while others manipulate directories when specific options are used.
Understanding command options is crucial, as their functionality can vary across different commands.

Creating Directories

The mkdir command allows users to generate directories. It accepts a list of paths as arguments to create one or multiple directories at once.
Example:

```
[user@host ~]$ cd Documents
[user@host Documents]$ mkdir ProjectA ProjectB ProjectC
[user@host Documents]$ ls
ProjectA  ProjectB  ProjectC
```

If a directory already exists or a required parent directory is missing, mkdir fails and outputs an error message.

To avoid this, use the -p flag to create any necessary parent directories:

```
[user@host Documents]$ mkdir -p Reports/Section1 Reports/Section2
Reports/Section3
[user@host Documents]$ ls -R Reports/
Reports/:
Section1  Section2  Section3
```

Be cautious when using -p, as mistyped directory names may create unintended folders without errors.

Copying Files and Directories

The cp command duplicates files, either within the same directory or into another specified location.
Example:

```
[user@host Videos]$ cp movie1.mp4 movie2.mp4
```

To copy multiple files into a directory, specify the target directory as the final argument:

```
[user@host Documents]$ cp file1.txt file2.txt Backup/
```

By default, cp does not copy directories unless the -r (recursive) flag is used:

```
[user@host Documents]$ cp -r Reports/ Archive/
```

Moving and Renaming Files and Directories

The mv command moves files between locations or renames them.
Example of renaming a file:

```
[user@host Documents]$ mv draft.txt final.txt
```

Moving a file into a different directory:

```
[user@host Documents]$ mv final.txt Archive/
```

Adding -v provides verbose output for confirmation:

```
[user@host Documents]$ mv -v report.pdf Reports/
```

Removing Files and Directories

The rm command deletes files. It does not remove directories unless the -r (recursive) flag is applied.
Example:

```
[user@host Documents]$ rm oldfile.txt
```

To remove a directory and its contents:

```
[user@host Documents]$ rm -r OldProjects/
```

For safer deletion, use -i to prompt confirmation:

```
[user@host Documents]$ rm -ri TempFiles/
```

The rmdir command can remove empty directories:

```
[user@host Documents]$ rmdir UnusedDir/
```

However, non-empty directories require rm -r.

References

Refer to the manual pages for further details:

- man cp
- man mkdir
- man mv
- man rm
- man rmdir

Creating Links Between Files

Managing File Links

In Linux, you can assign multiple file names to the same data. These alternate names, known as links, allow files to be accessed using different paths. There are two primary types of links:

- Hard Links – Directly associate a file name with its data on disk.
- Symbolic Links (Soft Links) – Create a reference that points to another file name.

Each method has its advantages and specific use cases.

Hard Links: How They Work

When a file is created, it starts with one hard link—its original file name mapped to its data. Creating a hard link generates an additional name for the same data, making the two indistinguishable.

Identifying Hard Links

To determine if a file has multiple hard links, use the ls -l command. The output includes a link count, showing how many hard links reference the file.
Example:

```
[user@host ~]$ ls -l newfile.txt
-rw-r--r--. 1 user user 0 Mar 11 19:19 newfile.txt
```

Here, the file newfile.txt has one hard link.

Creating a Hard Link

Use the ln command with two arguments:

1. The path of the existing file.
2. The new name for the hard link.

Example:

```
[user@host ~]$ ln newfile.txt /tmp/newfile-hlink2.txt
[user@host ~]$ ls -l newfile.txt /tmp/newfile-hlink2.txt
-rw-rw-r--. 2 user user 12 Mar 11 19:19 newfile.txt
-rw-rw-r--. 2 user user 12 Mar 11 19:19 /tmp/newfile-hlink2.txt
```

Both names now refer to the same data.

Verifying Hard Links with Inodes

Use ls -i to display inode numbers. If two files share the same inode, they are hard links to the same data.

```
[user@host ~]$ ls -il newfile.txt /tmp/newfile-hlink2.txt
8924107 -rw-rw-r--. 2 user user 12 Mar 11 19:19 newfile.txt
8924107 -rw-rw-r--. 2 user user 12 Mar 11 19:19 /tmp/newfile-
hlink2.txt
```

Since both files share inode 8924107, they point to the same data.

Key Characteristics of Hard Links

- All hard links to a file share permissions, ownership, timestamps, and content.
- Changes made to one hard link affect all others.
- The file's data remains accessible as long as at least one hard link exists, even if the original name is deleted.

Example:

```
[user@host ~]$ rm -f newfile.txt
[user@host ~]$ cat /tmp/newfile-hlink2.txt
Hello World
```

Even after deleting newfile.txt, the data is still accessible via newfile-hlink2.txt.

Limitations of Hard Links

- Cannot link to directories (for safety reasons).
- Only work within the same file system.

To determine file system boundaries, use df:

```
[user@host ~]$ df
Filesystem               1K-blocks    Used Available Use% Mounted
on
/dev/mapper/system-root  10258432 1630460  8627972  16% /
/dev/sda1                1038336  167128  871208  17% /boot
```

Here, you can create a hard link between /home/user/file and /var/tmp/link1 (same / filesystem), but not between /boot/test/badlink and /home/user/file (different partitions).

Symbolic Links (Soft Links)

Unlike hard links, symbolic links create a pointer to another file or directory, rather than directly linking to data.

Advantages of Symbolic Links

- Can link across different file systems.
- Can reference directories, not just files.

Creating a Symbolic Link

Use ln -s with two arguments:

- The target file (original).
- The symlink name.

Example:

```
[user@host ~]$ ln -s /home/user/newfile-link2.txt /tmp/newfile-symlink.txt
[user@host ~]$ ls -l newfile-link2.txt /tmp/newfile-symlink.txt
-rw-rw-r--. 1 user user 12 Mar 11 19:19 newfile-link2.txt
lrwxrwxrwx. 1 user user 11 Mar 11 20:59 /tmp/newfile-symlink.txt -> /home/user/newfile-link2.txt
```

The l at the start of the listing indicates a symbolic link.

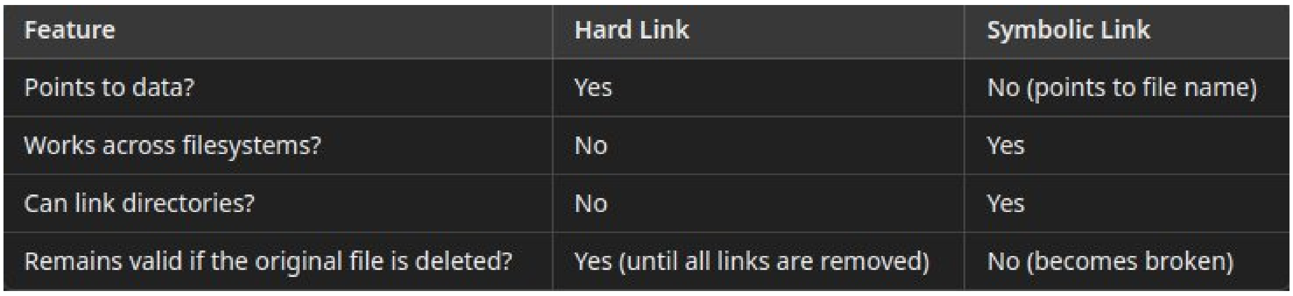
Handling Deleted Targets (Dangling Links)

If the original file is deleted, the symlink remains but becomes broken:

```
[user@host ~]$ rm -f newfile-link2.txt
[user@host ~]$ ls -l /tmp/newfile-symlink.txt
lrwxrwxrwx. 1 user user 11 Mar 11 20:59 /tmp/newfile-symlink.txt -> /home/user/newfile-link2.txt
[user@host ~]$ cat /tmp/newfile-symlink.txt
cat: /tmp/newfile-symlink.txt: No such file or directory
```

A broken symlink still exists but points to a non-existent file.

Key Differences Between Hard and Symbolic Links



Feature	Hard Link	Symbolic Link
Points to data?	Yes	No (points to file name)
Works across filesystems?	No	Yes
Can link directories?	No	Yes
Remains valid if the original file is deleted?	Yes (until all links are removed)	No (becomes broken)

Using Symlinks with Directories

Symlinks to directories act like the target directory itself:

```
[user@host ~]$ ln -s /etc /home/user/configfiles
[user@host ~]$ cd /home/user/configfiles
[user@host configfiles]$ pwd
/home/user/configfiles
```

The -P option in cd forces the system to resolve the actual directory path:

```
[user@host configfiles]$ cd -P /home/user/configfiles
[user@host etc]$ pwd
/etc
```

Summary

- Hard links create an additional name for a file’s data.
- Symbolic links create a reference to another file or directory.
- Hard links must be on the same file system, while symlinks can span across file systems.
- Deleting the original file breaks a symbolic link but does not affect hard links.

For more details, refer to:

- man ln
- info ln

Matching File Names with Shell Expansions

Understanding Shell Expansions

When you enter a command in the Bash shell, the system processes it through multiple expansion stages before execution. These expansions allow you to handle complex tasks with ease, enabling powerful automation techniques.

The primary types of expansions in Bash include:

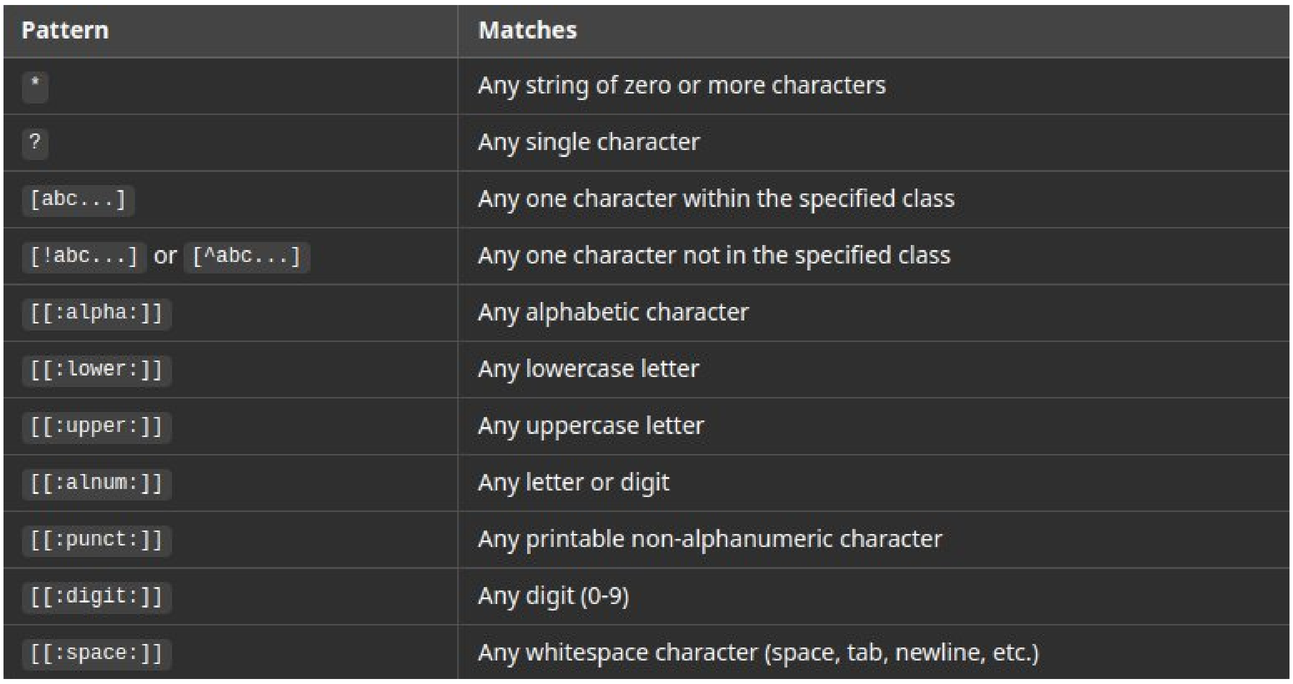
- Brace Expansion: Generates multiple string variations.
- Tilde Expansion: Resolves to a user's home directory path.
- Variable Expansion: Replaces variable names with their stored values.
- Command Substitution: Substitutes the output of a command into the command line.
- Pathname Expansion: Uses pattern matching to select multiple files dynamically.

Pathname expansion, historically referred to as globbing, is especially useful when managing multiple files simultaneously. Using specific metacharacters, you can match filenames based on patterns, streamlining batch operations.

Pathname Expansion and Pattern Matching

Pathname expansion transforms patterns containing special wildcard characters into a list of filenames that match the specified criteria. Before executing a command, the shell replaces the pattern with the matching filenames. If no match is found, the shell treats the pattern as a literal string.

Common Wildcard Patterns



Pattern	Matches
`*`	Any string of zero or more characters
`?`	Any single character
`[abc...]`	Any one character within the specified class
`[!abc...]` or `[^abc...]`	Any one character not in the specified class
`[[:alpha:]]`	Any alphabetic character
`[[:lower:]]`	Any lowercase letter
`[[:upper:]]`	Any uppercase letter
`[[:alnum:]]`	Any letter or digit
`[[:punct:]]`	Any printable non-alphanumeric character
`[[:digit:]]`	Any digit (0-9)
`[[:space:]]`	Any whitespace character (space, tab, newline, etc.)

Examples

Creating Sample Files

```
$ mkdir glob && cd glob
$ touch alfa bravo charlie delta echo able baker cast dog easy
$ ls
able alfa baker bravo cast charlie delta dog easy echo
```

Using Wildcards for File Matching

```
ls a*
# Matches files starting with 'a': able alfa

ls *a*
# Matches files containing 'a': able alfa baker bravo cast charlie delta easy

ls [ac]*
# Matches files starting with 'a' or 'c': able alfa cast charlie
```

Matching by Character Length

```
ls ????
# Matches four-character filenames: able cast easy echo

ls ?????
# Matches five-character filenames: baker bravo delta
```

Brace Expansion

Brace expansion generates sequences or multiple string variations within curly brackets.

```
echo {Sunday,Monday,Tuesday}.log
# Output: Sunday.log Monday.log Tuesday.log

echo file{1..3}.txt
# Output: file1.txt file2.txt file3.txt

echo file{a,b}{1,2}.txt
# Output: filea1.txt filea2.txt fileb1.txt fileb2.txt
```

Practical use includes batch file or directory creation:

```
mkdir ../RHEL{7,8,9}
ls ../RHEL*
# Output: RHEL7 RHEL8 RHEL9
```

Tilde Expansion

The tilde (~) expands to a user's home directory:

```
echo ~root  # Output: /root
echo ~user  # Output: /home/user
echo ~/glob # Output: /home/user/glob
```

If the username doesn't exist, the tilde remains unchanged:

```
echo ~nonexistentuser
# Output: ~nonexistentuser
```

Variable Expansion

Variables store data for easy retrieval and modification. Assign and retrieve values like this:

```
USERNAME=operator
echo $USERNAME
# Output: operator
```

For clarity, use curly braces:

```
echo ${USERNAME}
# Output: operator
```

Variable names must be alphanumeric and cannot begin with a number.

Command Substitution

Command substitution replaces a command with its output:

```
echo "Today is $(date +%A)."
# Output: Today is Wednesday.
```

Avoid using backticks (command) for substitution due to readability issues.

Protecting Arguments from Expansion

To prevent unintended expansion, escape special characters:

```
echo The value of \$HOME is your home directory.
# Output: The value of $HOME is your home directory.
```

Single quotes (' ') prevent all expansion, while double quotes (" ") allow variable and command substitution but block pathname expansion:

```
echo "***** hostname is $(hostname -s) *****"
# Output: ***** hostname is host *****

echo 'Will variable $myhost evaluate to $(hostname -s)?'
# Output: Will variable $myhost evaluate to $(hostname -s)?
```

Conclusion

Understanding shell expansions and pattern matching in Bash helps streamline file management and automate tasks efficiently. By leveraging these features, you can handle multiple files dynamically and execute complex operations with minimal effort.

- ## References
- man bash
- man cd
- man glob
- man ls
- man path_resolution
- man pwd