Linux System Administration

Manage Files from the Command Line

Summer Training June 2024

Pushpendra Kumar Pateriya

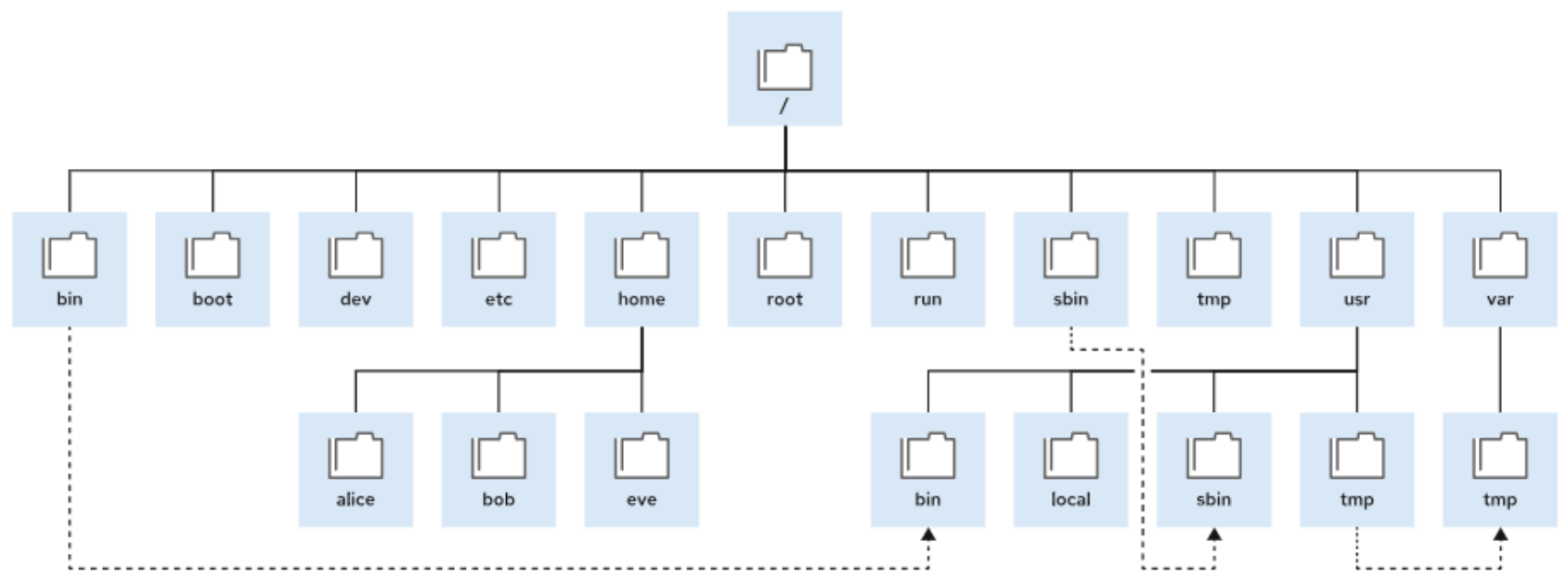
Lovely Professional University

pushpendra.mnnit@gmail.com pushpendrapateriya @ppsgsits

Objective(s)

- **Linux File Organization**: Explain how Linux arranges files and the roles of different directories in the file system hierarchy.
- **File Locations and Directories**: Identify the absolute and relative paths of files from the current working directory, and demonstrate how to find and change the working directory, as well as list directory contents.
- **File and Directory Management**: Show how to create, copy, move, and delete files and directories.
- **File Links**: Explain how to create multiple references to the same file using hard links and symbolic (soft) links.
- **Pattern Matching in Bash**: Demonstrate how to use pattern matching features in the Bash shell to efficiently execute commands that affect multiple files.

Significant file-system directories in Linux



Significant file-system directories in Linux

- **/boot**: Files to start the boot process.
- **/dev**: Special device files that the system uses to access hardware.
- **/etc**: System-specific configuration files.
- **/home**: Home directory for regular users' data and configuration files.
- **/root**: Home directory for the administrative superuser, root.
- **/run**: Runtime data for processes since the last boot, recreated on reboot.
- **/tmp**: Temporary files, deleted after 10 days of inactivity.
- **/usr**: Installed software, shared libraries, and read-only program data.
  - **/usr/bin**: User commands.
  - **/usr/sbin**: System administration commands.
  - **/usr/local**: Locally customized software.
- **/var**: Variable data such as logs, databases, and website content.

Question 1

Question

Which directory contains persistent, system-specific configuration data?

A. `/etc`
B. `/root`
C. `/run`
D. `/usr`

Answer

Answer

A. /etc

Question 2

Question

Which directory is the top of the system's file-system hierarchy?

A. `/etc`
B. `/`
C. `/home/root`
D. `/root`

Answer

Answer

B. /

Question 3

Question

Which directory contains user home directories?

A. `/`
B. `/home`
C. `/root`
D. `/user`

Answer

Answer

B. /home

Question 4

Question

Which directory contains files to boot the system?

A `/boot`

B `/home/root`

C `/bootable`

D `/etc`

Answer

Answer

A. /boot

Question 5

Question

Which directory contains system files to access hardware?

A. `/etc`
B. `/run`
C. `/dev`
D. `/usr`

Answer

Answer

C. /dev

Question 6

Question

Which directory is the administrative superuser's home directory?

A. `/etc`
B. `/`
C. `/home/root`
D. `/root`

Answer

Answer

D. /root

Question 7

Question

Which directory contains regular commands and utilities?

A. `/commands`
B. `/run`
C. `/usr/bin`
D. `/usr/sbin`

Answer

Answer

C. /usr/bin

Question 8

Question

Which directory contains non-persistent process runtime data?

A. `/tmp`
B. `/etc`
C. `/run`
D. `/var`

Answer

Answer

C. /run

Question 9

Question

Which directory contains installed software programs and libraries?

A. `/etc`
B. `/lib`
C. `/usr`
D. `/var`

Answer

Answer

C. /usr

Absolute Paths and Relative Paths

- A path in a Linux environment specifies the unique location of a file or directory in the file system.
- Paths can be absolute or relative:
  - An **absolute path** starts from the root directory, denoted by a forward slash (/), and specifies the full directory hierarchy to the file or directory.
  - A **relative path** starts from the current directory and specifies the path relative to the current location.
- Paths use a forward slash (/) as a delimiter to separate directories and subdirectories.
- Examples:
  - `/home/user/documents/file.txt` (absolute path)
  - `documents/file.txt` (relative path)

Special directory references

Special directory references include:

- A single dot (.) refers to the current directory.
- Two dots (..) refer to the parent directory.
- A tilde (~) represents the home directory of the current user.

Navigate Paths in the File System

- Navigating paths in the file system involves moving through directories to access files or subdirectories.
- Key commands and concepts include:
  - pwd (print working directory): Displays the current directory.
  - `cd` (change directory): Changes the current directory.
    - `cd /path/to/directory`: Moves to the specified absolute path.
    - `cd directory_name`: Moves to the specified relative path from the current directory.
    - `cd ..`: Moves up one level to the parent directory.
    - `cd ~`: Moves to the user's home directory.
  - `ls` (list): Lists the contents of the current directory.
    - `ls /path/to/directory`: Lists the contents of the specified directory.
- Examples:
  - `cd /home/user/documents`: Navigates to the `documents` directory located at `/home/user`.

Question 1

Question

Which command returns to your current home directory, assuming that the current working directory is `/tmp` and your home directory is `/home/user`?

A. `cd`
B. `cd ..`
C. `cd .`
D. `cd *`
E. `cd /home`

Answer

Answer

A. cd

Question 2

Question

Which command displays the absolute path name of the current location?

A. `cd`
B. `pwd`
C. `ls ~`
D. `ls -d`

Answer

Answer

B. pwd

Question 3

Question

Which command returns you to the working directory before the current working directory?

A. `cd -`
B. `cd -p`
C. `cd ~`
D. `cd ..`

Answer

Answer

A. cd -

Question 4

Question

Which command changes the working directory up two levels from the current location?

A. `cd ~/..`
B. `cd ../ ..`
C. `cd ../..`
D. `cd ~/`

Answer

Answer

C. cd ../..

Question 5

> ***Question***
>
> Which command lists files in the current location, with a long format, and including hidden files?

A. `llong ~`
B. `ls -a`
C. `ls -l`
D. `ls -al`

Answer

Answer

D. ls -al

Question 6

> ***Question***
>
> Which command creates an empty file called `helloworld.py` in the user home directory, assuming that your current directory is `/home`?

A. `touch ./helloworld.py`
B. `touch ~/helloworld.py`
C. `touch helloworld.py`
D. `touch ../helloworld.py`

Answer

Answer

B. `touch ~/helloworld.py`

Question 7

Question

Which command changes the working directory to the parent of the current location?

A. `cd ~`
B. `cd ..`
C. `cd ../..`
D. `cd -u1`

Answer

Answer

B. `cd ..`

Question 8

Question

Which command changes the working directory to `/tmp` if the current working directory is `/home/student`?

A. `cd tmp`
B. `cd ..`
C. `cd ../../tmp`
D. `cd ~tmp`

Answer

Answer

C. `cd ../../tmp`

Command-line File Management

- Command-line file management involves using a shell or terminal to navigate and manipulate files and directories.
- Key concepts and commands include:
  - Navigating directories
  - Listing files and directories
  - Creating and deleting files
  - Creating and deleting directories
  - Copying and moving files
  - Viewing file contents

Navigating Directories

- pwd - Print Working Directory: Displays the current directory path.
- `cd` - Change Directory: Changes the current directory.
- Examples:
  - `cd /path/to/directory` - Moves to the specified directory.
  - `cd ..` - Moves up one level to the parent directory.
  - `cd ~` - Moves to the home directory.

Listing Files and Directories

- `ls` - List: Displays the contents of a directory.
- Common options:
  - `ls -a` - Includes hidden files.
  - `ls -l` - Long format listing.
  - `ls -al` - Combines both options.

Creating and Deleting Files

- `touch` - Creates an empty file.
- `rm` - Removes (deletes) a file.
- Examples:
  - `touch filename.txt` - Creates an empty file named `filename.txt`.
  - `rm filename.txt` - Deletes `filename.txt`.

Creating and Deleting Directories

- `mkdir` - Makes a new directory.
- `rmdir` - Removes an empty directory.
- `rm -r` - Recursively removes a directory and its contents.
- Examples:
  - `mkdir newdir` - Creates a directory named `newdir`.
  - `rmdir newdir` - Deletes the empty directory `newdir`.
  - `rm -r newdir` - Deletes `newdir` and all its contents.

Copying and Moving Files

- `cp` - Copies files or directories.
- `mv` - Moves (or renames) files or directories.
- Examples:
  - `cp source.txt destination.txt` - Copies `source.txt` to `destination.txt`.
  - `mv oldname.txt newname.txt` - Renames `oldname.txt` to `newname.txt`.
  - `mv file.txt /path/to/destination/` - Moves `file.txt` to the specified directory.

Viewing File Contents

- `cat` - Concatenates and displays file contents.
- `less` - Views file contents one screen at a time.
- `head` - Displays the first part of a file.
- `tail` - Displays the last part of a file.
- Examples:
  - `cat filename.txt` - Displays the contents of `filename.txt`.
  - `less filename.txt` - Opens `filename.txt` for paginated viewing.
  - `head filename.txt` - Shows the first 10 lines of `filename.txt`.
  - `tail filename.txt` - Shows the last 10 lines of `filename.txt`.

Make Links Between Files

- Objectives:
  - Make multiple file names reference the same file with hard links and symbolic links.

Manage Links Between Files

Hard Links

- Every file starts with a single hard link, from its initial name to the data on the file system.
- Hard links point to the same data on the storage device.
- Use `ln` command to create a hard link.

Manage Links Between Files

Symbolic Links

- Also called "soft links."
- Symbolic links point to an existing file or directory.
- Created using `ln -s` command.
- Advantages over hard links include flexibility in linking files across file systems and the ability to link to directories or special files.

Comparison between Hard Links and Symbolic Links

Aspect	**Hard Links**	**Symbolic Links**
File Type	Regular files only	Any file type (regular, directory, special)
Location	Must be on the same filesystem	Can span different filesystems
File System Metadata	Shares same inode, permissions, owner, etc.	Points to file name, not metadata
Creation	Created using `ln` command	Created using `ln -s` command
Modification Impact	Modifications affect all links	Modifications affect original file

Match File Names with Shell Expansions

- **Objectives:**
  - Efficiently run commands that affect many files by using pattern matching features of the Bash shell.

Command-line Expansions

- When you type a command at the Bash shell prompt, the shell processes that command line through multiple expansions before running it.
- Key expansions include:
  - Brace expansion
  - Tilde expansion
  - Variable expansion
  - Command substitution
  - Pathname expansion (globbing)

Metacharacters and Matches

Pattern	Matches
*	Any string of zero or more characters
?	Any single character
[abc…]	Any one character in the enclosed class (between the square brackets)
[!*abc*…]	Any one character not in the enclosed class
[^*abc* . . .]	Any one character not in the enclosed class
[[: *alpha* :]]	Any alphabetic character
[[: *lower* :]]	Any lowercase character
[[: *upper* :]]	Any uppercase character
[[: *alnum* :]]	Any alphabetic character or digit
[[: *punct* :]]	Any printable character that is not a space or alphanumeric
[[: *digit* :]]	Any single digit from 0 to 9
[[: *space* :]]	Any single white space character, which might include tabs, newlines, carriage returns, form feeds, or spaces

Examples

The first two commands use an asterisk (*) to list files that start with "a" and files that have an "a" anywhere in their name. The third command lists files that start with either "a" or "c" using an asterisk and square brackets.

Command Line Prompt

```
[user@host glob]$ ls a*
able alfa
[user@host glob]$ ls *a*
able alfa baker bravo cast charlie delta easy
[user@host glob]$ ls [ac]*
able alfa cast charlie
```

Example

Command Line Prompt

```
[user@host glob]$ ls ????
able cast easy echo
[user@host glob]$ ls ?????
baker bravo delta
```

The first command lists files with four characters. The second command lists files with five characters.

Brace Expansion

- Brace expansion in Bash is a feature that allows you to generate multiple strings based on patterns defined within curly braces . It's a powerful tool for quickly creating lists of strings or filenames, simplifying repetitive tasks. Here's how it works:

Command Line Prompt

```
[user@host glob]$ echo {Sunday,Monday,Tuesday,Wednesday}.log
Sunday.log Monday.log Tuesday.log Wednesday.log
[user@host glob]$ echo file{1..3}.txt
file1.txt file2.txt file3.txt
[user@host glob]$ echo file{a..c}.txt
filea.txt fileb.txt filec.txt
[user@host glob]$ echo file{a,b}{1,2}.txt
filea1.txt filea2.txt fileb1.txt fileb2.txt
[user@host glob]$ echo file{a{1,2},b,c}.txt
filea1.txt filea2.txt fileb.txt filec.txt
```

A practical use of brace expansion is to create multiple files or directories.

Command Line Prompt

```
[user@host glob]$ mkdir ../RHEL{7,8,9}
[user@host glob]$ ls ../RHEL*
RHEL7 RHEL8 RHEL9
```

Tilde Expansion

- The tilde character (~) represents the current user's home directory.
- If it is followed by a string of characters that doesn't start with a slash (/), the shell treats the string before the slash as a username.
- If the username exists, the shell replaces the string with the absolute path to that user's home directory.
- If no matching username is found, the shell uses the actual tilde followed by the string of characters.

Example

In the example below, the echo command shows the value of the tilde character. You can also use the echo command to display the results of brace and variable expansions, among other things.

Command Line Prompt

```
[user@host glob]$ echo ~root
/root
[user@host glob]$ echo ~user
/home/user
[user@host glob]$ echo ~/glob
/home/user/glob
[user@host glob]$ echo ~nonexistinguser
~nonexistinguser
```

Variable Expansion

A variable acts like a named container that stores a value in memory. Variables simplify accessing and modifying the stored data either from the command line or within a shell script.

You can assign data as a value to a variable with the following syntax:

Command Line Prompt
`[user@host ~]`*VARIABLENAME* = *value*

Variable Expansion (continued)

You can use variable expansion to convert the variable name to its value on the command line. If a string starts with a dollar sign ($), then the shell tries to use the rest of that string as a variable name and to replace it with the variable value.

Command Line Prompt

```
[user@host ~]$ USERNAME=operator
[user@host ~]$ echo $USERNAME
operator
```

Variable Expansion (continued)

To prevent mistakes due to other shell expansions, you can put the name of the variable in curly braces, for example ${VARIABLENAME}.

Command Line Prompt

```
[user@host ~]$ USERNAME=operator
[user@host ~]$ echo ${USERNAME}
operator
```

Note

- Variable names can contain only letters (uppercase and lowercase), numbers, and underscores.
- Variable names are case-sensitive and cannot start with a number.

Command Substitution

- Command substitution enables the output of a command to replace the command itself on the command line.
- Command substitution occurs when you enclose a command in parentheses and precede it by a dollar sign ($).
- The $(command) form can nest multiple command expansions inside each other.

Command Line Prompt

```
[user@host glob]$ echo Today is $(date +%A).
Today is Wednesday.
[user@host glob]$ echo The time is $(date +%M)
minutes past $(date +%l%p).
The time is 26 minutes past 11AM.
```

Protecting Arguments from Expansion

Many characters have a special meaning in the Bash shell. To prevent shell expansions on parts of your command line, you can quote and escape characters and strings.

The backslash (is an escape character in the Bash shell. It protects the following character from expansion.

Command Line Prompt

```
[user@host glob]$ echo The value of $HOME is your home directory.

The value of /home/user is your home directory.

[user@host glob]$ echo The value of \$HOME is your home directory.

The value of $HOME is your home directory.
```

Protecting Arguments from Expansion (continued)

- To protect longer character strings, you can use single quotation marks (’) or double quotation marks (") to enclose strings. They have slightly different effects.
- Single quotation marks stop all shell expansion.
- Double quotation marks stop most shell expansion.
- Double quotation marks suppress special characters other than the dollar sign ($), backslash (\), backtick (‘), and exclamation point (!) from operating inside the quoted text.
- Double quotation marks block pathname expansion, but still allow command substitution and variable expansion to occur.

Command Line Prompt

```
[user@host glob]$ myhost=$(hostname -s); echo $myhost
host
[user@host glob]$ echo "***** hostname is ${myhost} *****"
***** hostname is host *****
```

Protecting Arguments from Expansion (continued)

Use single quotation marks to interpret all text between the quotes literally.

Command Line Prompt

```
[user@host glob]$ echo "Will variable $myhost evaluate to $(hostname -s)?"
Will variable host evaluate to host?
[user@host glob]$ echo 'Will variable $myhost evaluate to $(hostname -s)?'
Will variable $myhost evaluate to $(hostname -s)?
```

Question 1

Question

Which pattern matches only file names that end with "b"?

A. `b*`
B. `*b`
C. `*b*`
D. `[!b]*`

Answer

Answer

B. *b

Question 2

Question

Which pattern matches only file names that begin with "b"?

A. `b*`
B. `*b`
C. `*b*`
D. `[!b]*`

Answer

Answer

A. b*

Question 3

Question

Which pattern matches only file names where the first character is not "b"?

A. `b*`
B. `*b`
C. `*b*`
D. `[!b]*`

Answer

Answer

D. `[!b]*`

Question 4

Question

Which pattern matches all file names that contain a "b"?

A. `b*`
B. `*b`
C. `*b*`
D. `[!b]*`

Answer

Answer

C. *b*

Question 5

Question

Which pattern matches only file names that contain a number?

A. `*#*`
B. `*[[:digit:]]*`
C. `*[[digit]]*`
D. `[0-9]`

Answer

Answer

B. `*[[:digit:]]*`

Question 6

Question

Which pattern matches only file names that begin with an uppercase letter?

A. `^?*`
B. `^*`
C. `[upper]*`
D. `[[:upper:]]*`
E. `[[CAP]]*`

Answer

Answer

D. `[[:upper:]]*`

Question 7

Question

Which pattern matches only file names with at least three characters?

A. `???*`
B. `???`
C. `\3*`
D. `+++*`
E. `...*`

Answer

Answer

A. ???*

Thanks!

Thank you
for your attention.