CAROTUL

Basic Linux Commands

ZeroToLinux

Arpit Pandit
CAROTUL

Contents

CAROTUL
GET CERTIFIED!

CAROTUL
GET CERTIFIED!

Command: pwd

Description: The `pwd` command stands for "print working directory" and is used to display the current working directory in a Linux or Unix-like operating system. It provides the absolute path to the directory where the user is currently located within the file system.

Syntax: `pwd [OPTION]`

Some Important Options:

- `-L, --logical`: Display the logical path of the current directory. This is the default behavior.
- `-P, --physical`: Display the physical path of the current directory, which may contain symbolic links.

Examples:

1. Display the logical current directory:

   ```
   pwd
   ```
2. Display the physical current directory:

   ```
   pwd -P
   ```

Command: cd

Description: The `cd` command, short for "change directory," is used to change the current working directory in a Linux or Unix-like operating system. It allows you to navigate the file system by moving from one directory to another.

Syntax: `cd [OPTIONS] [DIRECTORY]`

Some Important Options:

- `-P`: Use the physical directory structure for navigation, resolving symbolic links to their target paths. This is the default behavior.
- `-L`: Use the logical directory structure for navigation, not following symbolic links.
- `--`: Treat subsequent arguments as directories, even if they start with hyphens.

Examples:

1. Change to the user's home directory:

   ```
   cd
   ```
2. Change to a specific directory (e.g., `/var/www/html`):

   ```
   cd /var/www/html
   ```
3. Navigate one level up in the directory tree:

   ```
   cd ..
   ```
4. Change to the previous directory (equivalent to `cd -`):

   ```
   cd -
   ```
5. Use the logical directory structure for navigation:

   ```
   cd -L /path/to/symlinked-dir
   ```
6. Use the physical directory structure for navigation:

   ```
   cd -P /path/to/symlinked-dir
   ```

Command: mkdir

Description: The `mkdir` command, short for "make directory," is used to create one or more directories (folders) within a Linux or Unix-like file system. It allows users to specify the names of the directories they want to create.

Syntax: `mkdir [OPTIONS] DIRECTORY...`

Some Important Options:

- `-p, --parents`: Create parent directories as needed. If a directory's parent does not exist, it will be created as well.
- `-m, --mode=MODE`: Set the permissions (mode) for the created directories. You can specify the mode in octal format, such as 755.
- `-v, --verbose`: Display a message for each directory created, showing the name of the directory.

Examples:

1. Create a single directory named "my_directory":

   `mkdir my_directory`
2. Create multiple directories in one command:

   `mkdir dir1 dir2 dir3`
3. Create a directory and its parent directories as needed:

   `mkdir -p path/to/parent/child`
4. Create a directory with specific permissions (e.g., read-write-execute for the owner and read-execute for others):

   `mkdir -m 755 my_directory`
5. Create directories with verbose output (shows the names of directories as they are created):

   `mkdir -v dir1 dir2 dir3`

Command: rmdir

Description: The `rmdir` command is used to remove empty directories in a Linux or Unix-like operating system. It allows you to delete directories that contain no files or subdirectories. If a directory is not empty, `rmdir` will not remove it, and you need to use `rm -r` to delete non-empty directories.

Syntax: `rmdir [OPTIONS] DIRECTORY...`

Some Important Options:

- `-p, --parents`: Remove parent directories if they become empty after removing the specified directory.
- `--ignore-fail-on-non-empty`: Ignore the error if the specified directory is not empty and continue removing other directories.

Examples:

1. Remove an empty directory named "my_directory":

   ```
   rmdir my_directory
   ```

2. Remove multiple empty directories in one command:

   ```
   rmdir dir1 dir2 dir3
   ```

3. Remove a directory and its parent directories if they become empty:

   ```
   rmdir -p path/to/parent/child
   ```

Command: rm

Description: The `rm` command, short for "remove," is used to delete files and directories in a Linux or Unix-like operating system. It allows users to remove one or more files or directories from the file system.

Syntax: `rm [OPTIONS] FILE...`

Some Important Options:

- `-i, --interactive`: Prompt the user for confirmation before removing each file.
- `-r, -R, --recursive`: Remove directories and their contents recursively.
- `-f, --force`: Forcefully remove files and directories without prompting for confirmation, even if they are write-protected.
- `--no-preserve-root`: Do not treat the root directory ("/") specially when removing recursively. Use with caution.

Examples:

1. Remove a single file named "file.txt":

   ```
   rm file.txt
   ```

2. Remove multiple files in one command:

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

3. Remove a directory and its contents (recursively):

   ```
   rm -r directory_name
   ```

4. Remove a directory and its contents without being prompted for confirmation:

   ```
   rm -rf directory_name
   ```

5. Interactively remove a file and get a confirmation prompt:

   ```
   rm -i sensitive_file.txt
   ```

6. Remove a file with a space in its name:

   ```
   rm "file with spaces.txt"
   ```

Command: clear

Description: The `clear` command is used to clear the contents of the terminal screen, providing a clean slate for new commands and output.

Syntax: `clear`

Command: man

Description: The `man` command is used to display the manual pages (documentation) of commands, utilities, and other topics on a Linux or Unix-like operating system.

Syntax: `man [OPTIONS] [COMMAND/SECTION]`

Some Important Options:

- `-k, --apropos`: Search for keywords in the manual page names and descriptions, providing a list of matching pages.
- `-f, --whatis`: Display a concise description of the specified command or topic.

Examples:

1. Display the manual page for a specific command (e.g., "ls"):

- `man ls`

2. Search for manual pages related to a keyword (e.g., "file permissions"):

- `man -k "file permissions"`

3. Display a concise description of a command (e.g., "grep"):

- `man -f grep`

Command: which

Description: The `which` command is used to locate and display the absolute path of an executable file associated with a specified command or program in a Linux or Unix-like operating system. It helps you find the location of the binary that will be executed when you run a particular command.

Syntax: `which [OPTIONS] COMMAND`

Some Important Options:

- `-a, --all`: Display all instances of the specified command in the system's PATH, if there are multiple matches.

Examples:

1. Find the location of the `ls` command:

   ```
   which ls
   ```
2. Locate the absolute path of the `python` interpreter:

   ```
   which python
   ```
3. Check for the locations of multiple commands (e.g., `ls` and `gcc`):

   ```
   which ls gcc
   ```
4. Display all instances of a command in the PATH (e.g., if there are multiple matches for `gcc`):

   ```
   which -a gcc
   ```

Command: echo

Description: The `echo` command is used to print text or messages to the terminal or standard output (usually the screen).

Syntax: `echo [OPTIONS] [TEXT]`

Some Important Options:

- `-e`: Interpret escape sequences in the text. This allows you to use backslashes to represent special characters, such as newline ("\n") or tab ("\t").
- `-n`: Do not output the trailing newline character. By default, `echo` adds a newline after the text it prints.

Examples:

1. Print a simple message to the terminal:

   ```
   echo "Hello, World!"
   ```

2. Print the content of a variable (e.g., `variable_name`):

   ```
   echo $variable_name
   ```

3. Use escape sequences to print a new line and a tab:

   ```
   echo -e "This is the first line\nThis is the second line\tIndented"
   ```

4. Print text without a trailing newline character (useful for inline text):

   ```
   echo -n "This will not have a newline at the end."
   ```

Command: ls

Description: The `ls` command is used to list the files and directories in a specified directory.

Syntax: `ls [OPTIONS] [DIRECTORY]`

Some Important Options:

- `-l`: Use the long format to display detailed information about files, including permissions, owner, group, file size, modification time, and file name.
- `-a, --all`: List all files, including hidden files (those whose names start with a dot).
- `-h, --human-readable`: Print file sizes in a human-readable format (e.g., using "K" for kilobytes, "M" for megabytes).
- `-t`: Sort files by modification time, with the most recently modified files displayed first.
- `-r, --reverse`: Display the list of files in reverse order, such as from Z to A.

Examples:

1. List the files and directories in the current directory:

   ```
   ls
   ```
2. List files and directories in a specific directory (e.g., `/home/user/documents`):

   ```
   ls /home/user/documents
   ```
3. Display a detailed list of files with permissions, owner, and file sizes:

   ```
   ls -l
   ```
4. List all files, including hidden ones:

   ```
   ls -a
   ```
5. List files in reverse order of their names:

   ```
   ls -r
   ```
6. List files in the current directory sorted by modification time (newest first):

   ```
   ls -t
   ```
7. List files with human-readable file sizes:

   ```
   ls -h
   ```

Command: cat

Description: The `cat` command, short for "concatenate," is used to display the content of text files, concatenate files, or create new ones.

Syntax: `cat [OPTIONS] [FILE...]`

Some Important Options:

- `-n`: Number all the output lines, providing line numbers for each line of text.
- `-E`: Display a `$` character at the end of each line, indicating the line's end.
- `-T`: Display tabs as `^I` and end-of-line characters as `$`.
- `-A`: Equivalent to using both `-E` and `-T`, displaying tabs and end-of-line characters.

Examples:

1. Display the content of a text file (e.g., "file.txt"):

   ```
   cat file.txt
   ```

2. Display the content of multiple text files in sequence:

   ```
   cat file1.txt file2.txt
   ```

3. Number the lines of a text file while displaying the content:

   ```
   cat -n file.txt
   ```

4. Display the content of a file, showing end-of-line characters with `$`:

   ```
   cat -E file.txt
   ```

5. Join two files and store output in a new file (file3)

   ```
   cat file1 file2 > file3
   ```

6. Create a file:

   ```
   cat > filename
   ```

7. Append something to already existing file:

   ```
   cat >> filename
   ```

Command: mv

Description: The `mv` command, short for "move," is used to rename or move files and directories in a Linux or Unix-like operating system.

Syntax: `mv [OPTIONS] SOURCE DESTINATION`

Some Important Options:

- `-i, --interactive`: Prompt for confirmation if the destination file already exists or if you are overwriting an existing file.
- `-u, --update`: Move the source to the destination only if the source is newer or the destination does not exist.
- `-b, --backup`: Create backups of files that would otherwise be overwritten during the move.

Examples:

1. Rename a file, changing its name from "file.txt" to "newfile.txt":

   ```
   mv file.txt newfile.txt
   ```

2. Move a file to a different directory (e.g., from the current directory to `/home/user/documents/`):

   ```
   mv file.txt /home/user/documents/
   ```

3. Rename a directory, changing its name from "old_directory" to "new_directory":

   ```
   mv old_directory new_directory
   ```

4. Move a directory and its contents to a different location:

   ```
   mv directory_to_move /home/user/new_location/
   ```

5. Prompt for confirmation before overwriting an existing file during a move:

   ```
   mv -i file.txt /destination/
   ```

6. Update the destination file only if the source is newer:

   ```
   mv -u source.txt /destination/
   ```

7. Create backups of files when overwriting during a move:

   ```
   mv -b file.txt /destination/
   ```

Command: more

Description: The `more` command is a text file viewer in Unix and Unix-like operating systems, including Linux. It allows you to view the contents of a file one screen at a time.

Syntax:

```
more [options] [filename]
```

Important Options:

- `-d`: Display help screen.
- `-p`: Disable scrolling, instead use only Enter or Spacebar to advance a screen.
- `-c`: Clear the screen before displaying the next page.
- `-s`: Squeeze multiple blank lines into one.

Examples:

1. View a file using `more`:

   ```
   more filename.txt
   ```

2. View a file with line numbers displayed:

   ```
   more -d filename.txt
   ```

3. Disable scrolling and use only Enter key to navigate:

   ```
   more -p filename.txt
   ```

4. Clear the screen before displaying each page:
   r

   ```
   more -c filename.txt
   ```

5. Squeeze multiple blank lines into one:

   ```
   more -s filename.txt
   ```

Command: less

Description: The `less` command is a terminal-based text file viewer in Unix and Unix-like operating systems, including Linux. It allows you to view the contents of a file one screen at a time, similar to `more`, but it provides more advanced features for navigating and searching through the text.

Syntax:

```
less [options] [filename]
```

Important Options:

- `-N`: Display line numbers.
- `-i`: Ignore case when searching.
- `/pattern`: Search for a specific pattern within the file.
- `?pattern`: Reverse search for a specific pattern.
- `q`: Quit `less`.

Examples:

1. View a file using `less`:
   ```
   less filename.txt
   ```
2. View a file with line numbers displayed:
   ```
   less -N filename.txt
   ```
3. Ignore case when searching for a pattern:
   ```
   less -i filename.txt
   ```
4. Search for a specific pattern within the file (press `/` and type the pattern):
   ```
   less filename.txt /search_pattern
   ```
5. Reverse search for a specific pattern (press `?` and type the pattern):
   ```
   less filename.txt ?search_pattern
   ```
6. Quit `less` (press `q`):
   ```
   less filename.txt
   ```

Command: head

Description: The `head` command in Linux is used to display the beginning or the first few lines of a text file

Syntax:

```
head [options] [filename]
```

Important Options:

- `-n NUM` or `--lines=NUM`: Specify the number of lines to display from the beginning of the file. By default, it shows the first 10 lines.
- `-c NUM` or `--bytes=NUM`: Display the first NUM bytes of the file.
- `-q` or `--quiet`: Suppress the display of file headers when multiple files are provided.
- `-v` or `--verbose`: Always display file headers when multiple files are provided.

Examples:

1. Display the first 10 lines of a file:

   ```
   head filename.txt
   ```

2. Display the first 5 lines of a file:

   ```
   head -n 5 filename.txt
   ```

3. Display the first 100 bytes of a file:

   ```
   head -c 100 filename.txt
   ```

4. Display the first 10 lines of multiple files with file headers:

   ```
   head -v file1.txt file2.txt file3.txt
   ```

5. Display the first 15 lines of multiple files without file headers:

   ```
   head -q -n 15 file1.txt file2.txt file3.txt
   ```

Command: tail

Description: The `tail` command in Linux is used to display the end or the last few lines of a text file.

Syntax:

```
tail [options] [filename]
```

Important Options:

- `-n NUM` or `--lines=NUM`: Specify the number of lines to display from the end of the file. By default, it shows the last 10 lines.
- `-c NUM` or `--bytes=NUM`: Display the last NUM bytes of the file.
- `-f` or `--follow`: Continuously display new lines added to the file in real-time (useful for log files).
- `-q` or `--quiet`: Suppress the display of file headers when multiple files are provided.
- `-v` or `--verbose`: Always display file headers when multiple files are provided.

Examples:

1. Display the last 10 lines of a file:

   ```
   tail filename.txt
   ```

2. Display the last 5 lines of a file:

   ```
   tail -n 5 filename.txt
   ```

3. Display the last 100 bytes of a file:

   ```
   tail -c 100 filename.txt
   ```

4. Continuously monitor and display new lines added to a log file in real-time:

   ```
   tail -f logfile.txt
   ```

5. Display the last 10 lines of multiple files with file headers:

   ```
   tail -v file1.txt file2.txt file3.txt
   ```

6. Display the last 15 lines of multiple files without file headers:

   ```
   tail -q -n 15 file1.txt file2.txt file3.txt
   ```

Command: find

Description: The `find` command in Linux is a powerful utility for searching and locating files and directories within a specified directory hierarchy.

Syntax:

```
find [directory] [options] [expression]
```

Important Options:

- `-name PATTERN`: Search for files and directories with a specific name pattern.
- `-type TYPE`: Search for files of a specific type (e.g., `f` for regular files, `d` for directories).
- `-size SIZE`: Search for files of a specific size (e.g., `+10M` for files larger than 10 megabytes).
- `-mtime DAYS`: Search for files modified within a certain number of days.
- `-exec COMMAND`: Execute a command on each found file or directory.
- `-print`: Print the file paths of the found items (the default action).

Examples:

1. Search for a file named "example.txt" in the current directory and its subdirectories:
   ```
   find . -name "example.txt"
   ```
2. Search for all directories within the `/var` directory:
   ```
   find /var -type d
   ```
3. Search for files larger than 100MB in the home directory
   ```
   find ~/ -type f -size +100M
   ```
4. Find files modified in the last 7 days in the `/tmp` directory:
   ```
   find /tmp -type f -mtime -7
   ```
5. Delete all `.log` files in the `/var/logs` directory:
   ```
   find /var/logs -type f -name "*.log" -exec rm {} \;
   ```
6. Search for all files and directories in the current directory and its subdirectories and print their paths:
   ```
   find . -print
   ```

Command: diff

Description: The `diff` command in Linux is used to compare and display the differences between two text files. It provides a line-by-line comparison of the files and highlights the lines that differ.

Syntax:

```
diff [options] file1 file2
```

Important Options:

- `-u` or `--unified`: Generate unified diff output.
- `-q` or `--brief`: Display only whether the files differ or not.
- `-r` or `--recursive`: Recursively compare directories.
- `-c` or `--context`: Generate context diff output.
- `-i` or `--ignore-case`: Ignore case when comparing.

Examples:

1. Compare two files and display the differences:

   ```
   diff file1.txt file2.txt
   ```

2. Generate a unified diff between two files:

   ```
   diff -u file1.txt file2.txt
   ```

3. Check if two files are different using a brief output (returns exit status 1 if different):

   ```
   diff -q file1.txt file2.txt
   ```

4. Recursively compare two directories and their contents:

   ```
   diff -r dir1 dir2
   ```

5. Generate a context diff with 3 lines of context:

   ```
   diff -c3 file1.txt file2.txt
   ```

Command: sdiff

Description: The `sdiff` command is used to view and compare two text files side by side, highlighting the differences. It can be a helpful tool for visualizing and merging differences between two files.

Syntax:

```
sdiff [options] file1 file2
```

Important Options:

- `-o` or `--output=FILE`: Specify an output file for merging differences.
- `-s` or `--suppress-common-lines`: Suppress lines that are identical in both files.
- `-w` or `--width=NUM`: Set the width of the output columns.

Examples:

1. Compare two files side by side:

   ```
   sdiff file1.txt file2.txt
   ```

2. Suppress common lines and display only differences:

   ```
   sdiff -s file1.txt file2.txt
   ```

3. Merge differences between two files and save the result to a new file:

   ```
   sdiff -o merged.txt file1.txt file2.txt
   ```

4. Set the output column width to 80 characters:

   ```
   sdiff -w 80 file1.txt file2.txt
   ```

Command: vimdiff

Description: `vimdiff` is a command that launches the Vim text editor in a side-by-side mode, allowing you to compare and edit two or three files simultaneously. It is particularly useful for code comparisons, merging changes, and resolving conflicts.

Syntax:

```
vimdiff file1 file2
```

Important Options: No specific options are needed to use `vimdiff`. Vim will automatically open in diff mode.

Examples:

1. Compare and edit two files side by side using `vimdiff`:

   ```
   vimdiff file1.txt file2.txt
   ```

2. Resolve merge conflicts in Git using `vimdiff` as the merge tool:

   ```
   git mergetool -t vimdiff
   ```

3. Use `vimdiff` with three files (common ancestor, current, and incoming changes):

   ```
   vimdiff ancestor.txt current.txt incoming.txt
   ```

Command: sort

Description: The `sort` command in Linux is used to sort lines of text files or standard input. It arranges lines in ascending or descending order based on the specified criteria.

Syntax: `sort [options] [file]`

Important Options:

- `-r` or `--reverse`: Sort in reverse order (descending).
- `-n` or `--numeric-sort`: Sort numerically rather than lexicographically.
- `-k FIELD` or `--key=FIELD[.CHAR][OPTS]`: Specify a field to use for sorting (field starts at 1).
- `-t CHAR` or `--field-separator=CHAR`: Set the field separator character (default is whitespace).
- `-u` or `--unique`: Remove duplicate lines.
- `-o OUTPUT` or `--output=OUTPUT`: Write the sorted output to a file.

Examples:

1. Sort lines in a file in lexicographical (default) order:
   `sort file.txt`
2. Sort lines in reverse order (descending):
   `sort -r file.txt`
3. Sort lines as numbers rather than text:
   `sort -n numbers.txt`
4. Sort a CSV file based on the second field (comma-separated):
   `sort -t ',' -k 2,2 data.csv`
5. Sort a tab-separated file based on the third field in reverse order:
   `sort -t $'\t' -k 3,3r data.tsv`
6. Remove duplicate lines from a file:
   `sort -u file.txt`
7. Sort a file and save the sorted output to another file:
   `sort -o sorted.txt unsorted.txt`

Command: tar

Description: The `tar` command in Linux is used for creating and manipulating compressed archive files. It is a versatile utility that can bundle multiple files and directories into a single archive, often compressed with gzip, bzip2, or other compression formats. `tar` is commonly used for data backup, distributing files, and packaging software.

Syntax:

```
tar [options] [archive-filename] [files or directories to be archived]
```

Important Options:

- `-c` or `--create`: Create a new archive.
- `-x` or `--extract`: Extract files from an archive.
- `-t` or `--list`: List the contents of an archive.
- `-v` or `--verbose`: Display verbose output.
- `-f FILE` or `--file=FILE`: Specify the archive file name.
- `-z` or `--gzip`: Compress the archive using gzip.
- `-j` or `--bzip2`: Compress the archive using bzip2.
- `-r` or `--append`: Append files to an existing archive.
- `-u` or `--update`: Add files to an archive only if they are newer than the archive.
- `-A` or `--catenate`: Concatenate multiple archives.
- `-d` or `--diff`: Find differences between an archive and the file system.

Examples:

1. Create a compressed tar archive of a directory:

   ```
   tar -czvf archive.tar.gz directory/
   ```

2. Extract files from a compressed archive:

   ```
   tar -xzvf archive.tar.gz
   ```

3. List the contents of an archive without extracting:

   ```
   tar -tvf archive.tar
   ```

4. Append files to an existing archive:

   ```
   tar -rvf archive.tar newfile.txt
   ```

5. Update an archive with files only if they are newer:

```
tar -uvf archive.tar newerfile.txt
```

6. Concatenate two or more archives into one:

```
tar -A -f combined.tar archive1.tar archive2.tar
```

7. Find differences between an archive and the file system:

```
tar -df archive.tar
```

CAROTUL
GET CERTIFIED!

Command: grep

Description: The `grep` command in Linux is used to search text for patterns or regular expressions within files or standard input.

Syntax:

```
grep [options] pattern [file(s)]
```

Important Options:

- `-i` or `--ignore-case`: Ignore case when matching patterns.
- `-v` or `--invert-match`: Select non-matching lines.
- `-r` or `--recursive`: Recursively search in directories.
- `-n` or `--line-number`: Prefix lines with line numbers.
- `-w` or `--word-regexp`: Match whole words only.
- `-l` or `--files-with-matches`: List only the filenames with matches.

Examples:

1. Search for the word "example" in a file:
   ```
   grep "example" file.txt
   ```
2. Search for "pattern" in all text files in a directory:
   ```
   grep "pattern" *.txt
   ```
3. Perform a case-insensitive search for "key" in a file:
   ```
   grep -i "key" file.txt
   ```
4. List files containing the pattern "error" in a directory:
   ```
   grep -rl "error" /path/to/directory/
   ```
5. Display line numbers for matches in a file:
   ```
   grep -n "pattern" file.txt
   ```

Command: egrep

Description: The `egrep` command is essentially an extended version of `grep` (Global Regular Expression Print). It allows the use of extended regular expressions, which include features like the `+` (one or more), `?` (zero or one), and `|` (alternation). This can make complex pattern matching easier.

Syntax:

```
egrep [options] pattern [file(s)]
```

Important Options:

- `-i`, `-v`, `-r`, `-n`, `-w`, `-l`, and other options similar to `grep` can be used with `egrep` as well.

Examples:

1. Search for lines containing "word1" or "word2" in a file:
   ```
   egrep "word1|word2" file.txt
   ```
2. Search for lines containing "one or more" in a file using the `+` quantifier:
   ```
   egrep "one or more" file.txt
   ```
3. List files with lines that match either "pattern1" or "pattern2" in a directory:
   ```
   egrep -rl "pattern1|pattern2" /path/to/directory/
   ```

Command: fgrep (or grep -F)

Description: The `fgrep` command (or `grep -F`) is used to perform fixed string searches instead of regular expression searches. It's useful when you want to search for a literal string without any special character interpretation.

Syntax: `fgrep [options] pattern [file(s)]`

Important Options:

- `-i`, `-v`, `-r`, `-n`, `-w`, `-l`, and other options similar to `grep` can be used with `fgrep` as well.

Examples:

1. Search for the string "exact match" in a file without interpreting special characters:
   ```
   fgrep "exact match" file.txt
   ```
2. Search for a list of words stored in a file:
   ```
   fgrep -f words.txt textfile.txt
   ```

VAROTIL
GET CERTIFIED!

Command: gzip

Description: The `gzip` command in Linux is used for file compression. It reduces the size of files by compressing them using the DEFLATE algorithm. Compressed files have a `.gz` extension and can be decompressed using `gunzip`.

Syntax:

```
gzip [options] [file(s)]
```

Important Options:

- `-d` or `--decompress`: Decompress a compressed file.
- `-k` or `--keep`: Keep the original file (default is to remove it).
- `-c` or `--stdout`: Write compressed data to standard output.
- `-r` or `--recursive`: Recursively compress directories.

Examples:

1. Compress a file (creates a compressed file with the `.gz` extension):
   ```
   gzip file.txt
   ```
2. Compress multiple files and keep the original files:
   ```
   gzip -k file1.txt file2.txt
   ```
3. Decompress a compressed file:
   ```
   gzip -d compressed_file.gz
   ```
4. Compress a directory and its contents recursively:
   ```
   gzip -r directory/
   ```

Command: gunzip

Description: The **`gunzip`** command is used to decompress files compressed with **`gzip`**. It restores the original file to its uncompressed state.

Syntax:

```
gunzip [options] [file(s)]
```

Important Options:

- **`-k`** or **`--keep`**: Keep the original compressed file.
- **`-c`** or **`--stdout`**: Write decompressed data to standard output.

Examples:

1. Decompress a file compressed with **`gzip`**:
   ```
   gunzip compressed_file.gz
   ```
2. Decompress multiple files and keep the original compressed files:
   ```
   gunzip -k file1.gz file2.gz
   ```
3. Decompress a file and display the decompressed data in the terminal:
   ```
   gunzip -c compressed_file.gz
   ```

Command: zcat

Description: The `zcat` command is used to view the contents of compressed files without decompressing them. It is essentially a combination of `cat` and `gunzip` that allows you to read the content of compressed files directly in the terminal.

Syntax:

```
zcat [file(s)]
```

Important Options: No specific options are commonly used with `zcat`.

Examples:

1. View the contents of a compressed file without decompressing it:
   ```
   zcat compressed_file.gz
   ```
2. Concatenate and view multiple compressed files:
   ```
   zcat file1.gz file2.gz
   ```

Command: strings

Description: The `strings` command in Linux is used to extract human-readable text strings from binary files.

Syntax:

```
strings [options] [file(s)]
```

Important Options:

- `-n NUM` or `--bytes=NUM`: Set the minimum string length to extract (default is 4).
- `-a` or `--all`: Display all strings, including very short ones.
- `-t FORMAT` or `--radix=FORMAT`: Display the offsets as specified in the format (o for octal, x for hexadecimal, d for decimal).

Examples:

1. Extract and display printable strings from an executable file:
   ```
   strings /path/to/binaryfile
   ```
2. Display all strings from a binary file, including very short ones:
   ```
   strings -a binaryfile
   ```
3. Extract and display strings with a minimum length of 8 characters:
   ```
   strings -n 8 binaryfile
   ```
4. Display strings along with their offsets in hexadecimal format:
   ```
   strings -t x binaryfile
   ```

Command: file

Description: The `file` command in Linux is used to determine the file type and provide information about the format and content of a file.

Syntax:

```
file [options] [file(s)]
```

Important Options:

- `-b` or `--brief`: Display output in a brief format.
- `-i` or `--mime`: Display MIME type (e.g., text/plain, image/jpeg).
- `-z` or `--uncompress`: Attempt to uncompress files before determining the type.

Examples:

1. Determine the file type of a single file:
   ```
   file myfile.txt
   ```
2. Identify the MIME type of a file:
   ```
   file -i image.jpg
   ```
3. Display brief information about a file:
   ```
   file -b document.pdf
   ```
4. Attempt to uncompress a compressed file and determine its type:
   ```
   file -z archive.tar.gz
   ```

Command: cut

Description: The `cut` command in Linux is used to extract and display specific columns or fields from a line of text or data. It is commonly used to split text files into smaller parts based on a delimiter (usually a tab or space).

Syntax:

```
cut [options] [file(s)]
```

Important Options:

- `-d DELIMITER` or `--delimiter=DELIMITER`: Set the delimiter character (default is a tab).
- `-f FIELDS` or `--fields=FIELDS`: Select specific fields or columns (e.g., 1,2,3).
- `-c CHARACTERS` or `--characters=CHARACTERS`: Select specific character positions.

Examples:

1. Extract the first two columns from a tab-separated file:

   ```
   cut -f 1,2 data.txt
   ```

2. Split a comma-separated CSV file and display the first and third fields:

   ```
   cut -d ',' -f 1,3 file.csv
   ```

3. Select specific characters from each line:

   ```
   cut -c 1-5,10-15 data.txt
   ```

4. Extract usernames from a list of email addresses (assuming usernames are before '@'):

   ```
   cut -d '@' -f 1 emails.txt
   ```

Command: tr

Description: The `tr` (translate) command in Linux is used for character-level text manipulation. It can perform operations like character replacement, deletion, squeezing (compressing multiple characters into one), and character set translation.

Syntax:

```
tr [options] SET1 [SET2]
```

Important Options:

- `-d` or `--delete`: Delete characters in SET1.
- `-s` or `--squeeze-repeats`: Squeeze consecutive identical characters in SET1 into one.
- `-c` or `--complement`: Use the complement of SET1.
- `-t` or `--truncate-set1`: Truncate SET1.

Examples:

1. Translate all uppercase characters to lowercase in a file:
   ```
   tr 'A-Z' 'a-z' < input.txt > output.txt
   ```
2. Delete all spaces from a text file:
   ```
   tr -d ' ' < input.txt > output.txt
   ```
3. Squeeze consecutive whitespace characters into a single space:
   ```
   tr -s ' ' < input.txt > output.txt
   ```
4. Complement a character set (e.g., convert all non-digits to spaces):
   ```
   tr -c '0-9' ' ' < input.txt > output.txt
   ```

I/O (Input/Output) streams

Stdin, **Stdout**, and **Stderr** are fundamental I/O (Input/Output) streams in Linux and other Unix-like operating systems. These streams are crucial for communicating with processes and handling data in the terminal.

1. **Stdin** (Standard Input):
   - **Description**: Stdin represents the input stream, which is typically the keyboard by default. It allows you to provide data to a program by typing it in or by reading data from a file. In Unix-based systems, stdin is associated with the file descriptor `0`.
   - **Syntax**: Stdin is not a command or utility itself but rather a fundamental concept in command-line interaction. You can redirect stdin using the `<` operator or pipe data into a command.
   - **Examples**:
     - Redirect stdin from a file:

       ```
       cat < input.txt
       ```

     - Pipe the output of one command as stdin to another:

       ```
       cat file.txt | grep "pattern"
       ```

2. **Stdout** (Standard Output):
   - **Description**: Stdout is the output stream where a program writes its normal output. It is typically the terminal screen, but it can also be redirected to a file. In Unix-based systems, stdout is associated with the file descriptor `1`.
   - **Syntax**: Stdout is not a command, but you can redirect it using the `>` operator or append output to an existing file using `>>`.
   - **Examples**:
     - Redirect stdout to a file (creates or overwrites the file):

       ```
       ls > output.txt
       ```

     - Append output to an existing file:

       ```
       echo "New data" >> existing_file.txt
       ```

3. **Stderr** (Standard Error):
   - **Description**: Stderr is the output stream for error messages and diagnostics. It is also usually displayed on the terminal screen but can be redirected separately from stdout. In Unix-based systems, stderr is associated with the file descriptor `2`.

- **Syntax**: Stderr is not a command, but you can redirect it to a file using `2>` or append to an existing file with `2>>`.
- **Examples**:
  - Redirect stderr to a separate error log file:

    ```
    myprogram 2> error.log
    ```
  - Append error messages to an existing error log file:

    ```
    myprogram 2>> existing_error.log
    ```

CAROTUL
GET CERTIFIED!

Command: scp (Secure Copy)

Description: The `scp` command is used to securely copy files or directories between a local system and a remote system or between two remote systems. It operates over SSH (Secure Shell) and provides encryption and authentication, making it a secure method for transferring data.

Syntax:

```
scp [options] [source] [destination]
```

Important Options:

- `-P PORT` or `--port=PORT`: Specify the SSH port on the remote system.
- `-r` or `--recursive`: Copy directories and their contents recursively.
- `-i IDENTITY_FILE` or `--identity-file=IDENTITY_FILE`: Use a specific private key for authentication.
- `-q` or `--quiet`: Suppress progress information.

Examples:

1. Copy a local file to a remote server:
   ```
   scp localfile.txt user@remote:/path/to/destination/
   ```
2. Copy a remote file to the local system:
   ```
   scp user@remote:/path/to/remote/file.txt localfile.txt
   ```
3. Copy a local directory and its contents to a remote server:
   ```
   scp -r local_directory user@remote:/path/to/destination/
   ```
4. Copy a remote directory and its contents to the local system:
   ```
   scp -r user@remote:/path/to/remote/directory local_directory
   ```

Command: sftp (Secure File Transfer Protocol)

Description: The `sftp` command is an interactive file transfer program that provides a secure way to transfer files between a local system and a remote system over SSH. It is similar to using an FTP client but with the added benefit of encryption and security.

Syntax:

```
sftp [user@]host
```

Important Options: Once you're in the `sftp` interactive session, you don't need specific options. You can use a variety of commands for file and directory operations.

Examples:

1. Start an `sftp` session with a remote server:

   ```
   sftp user@remote_server
   ```

2. Upload a local file to the remote server within the `sftp` session:

   ```
   put localfile.txt
   ```

3. Download a remote file to the local system within the `sftp` session:

   ```
   get remote_file.txt
   ```

4. Change the current remote directory:

   ```
   cd remote_directory
   ```

5. List files and directories in the current remote directory:

   ```
   ls
   ```

Command: ftp (File Transfer Protocol)

Description: The `ftp` command in Linux is a client program used for transferring files to and from remote servers via the FTP protocol. It provides a simple command-line interface for interacting with FTP servers and performing file transfers.

Syntax: `ftp [options] [hostname]`

Important Options:

- `-n`: Suppress auto-login.
- `-v`: Enable verbose output.
- `-d`: Debug mode (show debugging messages).
- `-p`: Enable passive mode (for connections through firewalls).
- `-i`: Disable interactive prompting.
- `-s:FILE`: Read commands from a specified script file.

Examples:

1. Start an FTP session with a remote server:
   ```
   ftp example.com
   ```
2. Start an FTP session in passive mode (useful for connections through firewalls):
   ```
   ftp -p example.com
   ```
3. Disable interactive prompting for multiple file transfers:
   ```
   ftp -i example.com
   ```
4. Run FTP commands from a script file (e.g., `ftp_commands.txt`):
   ```
   ftp -s:ftp_commands.txt example.com
   ```
5. Upload a local file to the remote server within the FTP session:
   ```
   put localfile.txt
   ```
6. Download a remote file from the remote server within the FTP session:
   ```
   get remotefile.txt
   ```
7. Change the current remote directory within the FTP session:
   ```
   cd remote_directory
   ```
8. List files and directories on the remote server within the FTP session:
   ```
   ls
   ```

Command: nano (Text Editor)

Description: Nano is a user-friendly text editor for Unix and Linux systems. It is known for its simplicity and ease of use, making it a popular choice for users who may not be familiar with more complex text editors like Vim or Emacs. Nano provides a basic set of commands and features for editing text files from within the command line.

Syntax: Once you've opened a file with `nano`, you can use various commands to edit the text. Here are some common commands and their functions:

1. **Basic Navigation:**
   - **Arrow Keys:** Navigate the cursor.
   - `Ctrl` + `F`: Move forward one page.
   - `Ctrl` + `B`: Move backward one page.
   - `Ctrl` + `A`: Move to the beginning of the line.
   - `Ctrl` + `E`: Move to the end of the line.
2. **Editing Text:**
   - `Ctrl` + `K`: Cut (delete) the current line.
   - `Ctrl` + `U`: Uncut (paste) the previously cut text.
   - `Ctrl` + `O`: Save the current file (Write Out).
   - `Ctrl` + `X`: Exit the editor (if changes are made, it will prompt to save).
3. **Search and Replace:**
   - `Ctrl` + `W`: Search for text within the file.
   - `Ctrl` + `R`: Replace text within the file.
4. **Copy and Paste:**
   - `Alt` + `^` (Shift + 6): Mark text for copying.
   - `Ctrl` + `K`: Cut the marked text.
   - `Ctrl` + `U`: Uncut (paste) the previously cut text.
5. **Indentation:**
   - `Ctrl` + `]`: Indent the current line.
   - `Ctrl` + `[`: Unindent the current line.
6. **Other Commands:**
   - `Ctrl` + `G`: Show help menu with a list of commands.
   - `Ctrl` + `C`: Display the current cursor position.
   - `Ctrl` + `T`: Check spelling of the text (if enabled).
7. **Exiting Nano:**
   - `Ctrl` + `X`: If changes were made, it will prompt to save before exiting.
   - `Ctrl` + `C`: Abort the current operation.
   - `Ctrl` + `D`: Exit the editor without saving changes.

Command: vi (Visual Editor)

Description: Vi, often referred to as the "visual editor," is a powerful and highly efficient text editor that is available on Unix and Linux systems.

Syntax: When you open a file in `vi`, you enter different modes to navigate, edit, and manipulate text. Here are the primary modes and some common commands associated with each:

1. **Normal Mode:**
   - `Esc`: Return to normal mode from other modes.
   - `h`, `j`, `k`, `l`: Navigate left, down, up, and right, respectively.
   - `0` (zero), `$`: Move to the beginning and end of a line.
   - `G`: Go to the end of the file.
   - `:n` or `:n<m>`: Move to line number "n" or the middle of the screen.
   - `u`: Undo the last action.
   - `Ctrl` + `r`: Redo the last undone action.
   - `yy` or `Y`: Yank (copy) a line.
   - `dd`: Delete (cut) a line.
   - `p`: Paste the last yanked or deleted text.
   - `:q`: Quit vi.
   - `:q!`: Quit vi without saving changes.
   - `:w`: Write (save) the file.
   - `:wq` or `:x`: Write and quit vi.
2. **Insert Mode:**
   - `i`: Enter insert mode before the cursor position.
   - `I`: Enter insert mode at the beginning of the line.
   - `a`: Enter insert mode after the cursor position.
   - `A`: Enter insert mode at the end of the line.
   - `o`: Open a new line below the current line and enter insert mode.
   - `O`: Open a new line above the current line and enter insert mode.
3. **Visual Mode:**
   - `v`: Enter visual mode to select text.
   - `V`: Enter visual line mode to select entire lines.
   - `Ctrl` + `v`: Enter visual block mode to select rectangular areas.
4. **Command Mode:**
   - `:`: Enter command-line mode to issue advanced commands.
   - `:s/old/new/g`: Replace "old" with "new" throughout the file.

- `:%s/old/new/g`: Replace "old" with "new" throughout the file, with a confirmation for each occurrence.
- `:w filename`: Save the current file as "filename."
- `:e filename`: Open a different file while in vi.

5. **Search Mode:**
   - `/pattern`: Search for "pattern" forward in the file.
   - `?pattern`: Search for "pattern" backward in the file.
   - `n`: Find the next occurrence of the search pattern.
   - `N`: Find the previous occurrence of the search pattern.

CAROTUL
GET CERTIFIED!

Command: `emacs` (Extensible Text Editor)

Description: Emacs is a highly customizable and extensible text editor that offers an extensive set of features beyond basic text editing

Syntax: In Emacs, you use a combination of keys and key sequences to issue commands and navigate through the interface. Here are some common commands and key sequences:

1. **Basic Navigation:**
   - `Ctrl` + `N`, `Ctrl` + `P`: Move the cursor down and up, respectively.
   - `Ctrl` + `F`, `Ctrl` + `B`: Move the cursor forward and backward, respectively.
   - `Ctrl` + `A`, `Ctrl` + `E`: Move to the beginning and end of a line, respectively.
   - `Meta` + `<`, `Meta` + `>`: Move to the beginning and end of the document, respectively.
2. **Editing Text:**
   - `Ctrl` + `D`: Delete the character under the cursor.
   - `Ctrl` + `K`: Delete from the cursor position to the end of the line.
   - `Ctrl` + `Y`: Yank (paste) previously deleted or cut text.
   - `Ctrl` + `Space`: Set the mark to start selecting text.
   - `Ctrl` + `W`: Cut the selected text.
   - `Ctrl` + `X`, `Ctrl` + `S`: Save the current buffer to a file.
   - `Ctrl` + `X`, `Ctrl` + `C`: Quit Emacs.
3. **Search and Replace:**
   - `Ctrl` + `S`: Search forward for text.
   - `Ctrl` + `R`: Search backward for text.
   - `Meta` + `%`: Replace text in the buffer.
4. **Copy and Paste:**
   - `Meta` + `W`: Copy (kill) the selected text.
   - `Ctrl` + `Y`: Yank (paste) the copied or killed text.
   - `Meta` + `W` followed by `Ctrl` + `Y`: Copy and paste.
5. **Multiple Buffers:**
   - `Ctrl` + `X`, `Ctrl` + `B`: List and switch between open buffers.
6. **File Operations:**
   - `Ctrl` + `X`, `Ctrl` + `F`: Open a file.
   - `Ctrl` + `X`, `Ctrl` + `W`: Save the current buffer to a different file.
7. **Command Execution:**
   - `Meta` + `x`: Execute a command by entering its name.
8. **Customization:**

- `Meta` + `:`: Enter the Emacs command prompt.
- `Meta` + `x`, `customize`: Customize Emacs settings.

9. **Help:**
    - `Ctrl` + `H`, `Ctrl` + `?`: Access the built-in help system.
10. **Undo and Redo:**
    - `Ctrl` + `/`: Undo the last action.
    - `Ctrl` + `x`, `u` (Ctrl + x followed by u): Undo multiple actions.

CAROTUL
GET CERTIFIED!