Linux Commands

Here's a more thorough list of commands found in the file, briefly describing what each one does:

- ls: Lists files and directories.
- ls -l: Lists files with details in long format.
- ls -r: Lists files in reverse order.
- ls -R: Recursively lists files in directories.
- ls -lt: Lists files sorted by modification time, in long format.
- ls -la: Lists all files, including hidden ones, in long format.
- cd: Changes the current directory.
- cd ././: Changes to the current directory (effectively does nothing).
- pwd: Prints the current working directory.
- cat <file>: Displays the content of a file.
- cat > <file>: Creates a file and allows writing to it.
- cat >> <file>: Appends content to a file.
- touch <file>: Creates an empty file.
- mkdir <dir>: Creates a new directory.
- mkdir <dir> && cd <dir>: Creates and enters a new directory.
- ls *.js: Lists all .js files in the directory.
- ls -R *.js: Recursively lists all .js files in subdirectories.
- grep: Searches for patterns in files or outputs.
- grep <pattern> <file>: Searches for a pattern in a specific file.
- ls -lR | grep <pattern>: Lists files recursively and filters the output based on the given pattern using grep.

- mv <file1> <file2>: Renames or moves a file or directory.
- touch <file>: Creates a new empty file.
- cp <source> <destination>: Copies or renames a file. If copying within the same directory with a new name, it acts as a renaming command.
- cp -r <source> <destination>: Recursively copies a directory.
- rm <file>: Removes a file.
- rm -r <dir>: Recursively removes a directory.
- chmod <mode> <file>: Changes the file permissions.
- echo <string>: Prints a string to the terminal.
- cat <file>: Displays the contents of a file.
- cat >> <file>: Appends content to a file.
- head <file>: Displays the first few lines of a file.
- tail <file>: Displays the last few lines of a file.
- head -n <number> <file>: Displays the first n lines of a file.
- tail -n +<number> <file>: Displays lines starting from the n-th line of a file.
- wc <file>: Displays word, line, and character counts of a file.
- grep <pattern> <file>: Searches for a pattern in a file.
- grep <pattern> <file> | wc: Combines the output of grep with wc to count lines, words, and characters matching the pattern.
- wc -l: Displays the line count.
- **CHMOD** -

  Here's how the numbering system works:

  4 represents read permission (r)

2 represents write permission (w)

1 represents execute permission (x)

0 represents no permissions (-)

These numbers are then added together to create a single digit for each category (owner, group, others). Here's a breakdown:

7 = 4 + 2 + 1 (rwx: read, write, and execute)

6 = 4 + 2 (rw-: read and write)

5 = 4 + 1 (r-x: read and execute)

4 = 4 (r--: read only)

3 = 2 + 1 (-wx: write and execute)

2 = 2 (-w-: write only)

1 = 1 (--x: execute only)

0 = 0 (---: no permissions)

So, when you see a permission like chmod 755, it means:

Owner: 7 (4+2+1) = rwx (read, write, execute)

Group: 5 (4+1) = r-x (read, execute)

Others: 5 (4+1) = r-x (read, execute)

- **`grep "s" -c a2.txt`**:
Counts the number of lines in `a2.txt` that contain the letter "s" and displays the count (`2` in this case).

- **`grep "s" -h a2.txt`**:
Searches for lines containing "s" in `a2.txt` but omits the file name in the output, showing only the matching lines.

- **`grep "one" -h a2.txt`**:
Searches for lines containing "one" in `a2.txt` and displays them without the file name in the output. Matches exact case.

- **`grep "one" -hi a2.txt`**:
Searches for lines containing "one" in `a2.txt` case-insensitively (`-i`), and displays the matching lines without the file name.

- **`grep "one" -hir ./`**:
Recursively searches for the word "one" (case-insensitively) in all files and directories starting from the current directory (`./`), without showing the file names.

- **grep "one" -n a2.txt**:

  grep searches for the string "one" in a2.txt.
  -n shows line numbers where matches are found.

- **grep "one" -w a2.txt**:

  grep searches for the string "one" in a2.txt.
  -w restricts the search to whole words only.

- **history** command displays a list of previously executed commands in the terminal, allowing you to view and reuse them. You can use **!number** to re-run a specific command from the history list.

- **`bash a3.sh`**:

  Executes the script `a3.sh`. It produced an error due to unexpected characters.

- **`grep -v "one" a2.txt`**:

  Displays lines from `a2.txt` that do not contain the string "one".

- **`grep -A 5 "one" a2.txt`**:

Shows lines containing "one" and the 5 lines following each match.

- **`grep -B 5 "one" a2.txt`**:

  Shows lines containing "one" and the 5 lines preceding each match.

- **`grep -C 5 "one" a2.txt`**:

  Displays lines containing "one" along with 5 lines before and after each match.

- **`sed -i 's/one/two/' a2.txt`**:

  Replaces the first occurrence of "one" with "two" on each line in the file `a2.txt` and saves the changes directly to the file.

- **`sed -ibackup 's/one/two/' a2.txt`**:

  Replaces the first occurrence of "one" with "two" on each line in `a2.txt` and saves the changes to the file. Additionally, it creates a backup of the original file named `a2.txtbackup`.

- **`sed '15,20 s/two/one/' a2.txt`**:

  Attempts to replace "two" with "one" only in lines 15 through 20 of `a2.txt` but was missing the correct command format. The correct format should be `sed '15,20 s/two/one/' a2.txt`.

- **`awk '/two/{print $0}' a2.txt`**:

  Prints lines from `a2.txt` that contain the string "two".

- **`awk '{gsub(/two/, "three")}{print}' a2.txt`**:

  Replaces all occurrences of "two" with "three" in each line of `a2.txt` and prints the modified lines.

- **`awk 'BEGIN {print "START OF FILE\n------"} {print} END {print "------\nEND OF FILE"}' a2.txt`**:

  Prints "START OF FILE" at the beginning, followed by the contents of `a2.txt`, and then prints "END OF FILE" at the end.

- **`awk '{print $1, $2}' a2.txt`**:

  Prints the first and second fields of each line from `a2.txt`. If there are fewer than two fields, only the first field is printed.

- **`awk -F "," '{print $1, $2}' a2.txt`**:

  Sets the field separator to a comma and prints the first and second fields of each line from `a2.txt`. This command didn't change the output since `a2.txt` does not use commas as separators.

- **`awk '{count[$1]++} END {print count["two"]}' a2.txt`**:

  Counts occurrences of each unique value in the first field of `a2.txt` and prints the count of the value "two".