Linux commands

1. **/ :** root directory, **.**: current directory directory , **~** home directory
2. **pwd** (print working directory)
3. **Cd** (change directory)
4. **Cd ..** (back)
5. / (from home directory)
6. **mkdir (make directory)**: Use `mkdir` to create a new directory. For example, `mkdir -p /home/animal/dog` creates nested directories, ensuring that both the `animal` and `dog` directories are created together.
7. **ls (list)**: The `ls` command lists the contents of a directory. Use `ls -l` to get more details for the list. Use `ls -ltr` to get a detailed list ordered by modification time in reverse order.
8. **mkdir -p dir1/dir2/dir3**: The `-p` option creates parent directories as needed, so `dir2` is inside `dir1` and `dir3` is inside `dir2`.
9. **mkdir -v folder1**: The `-v` option tells that it created `folder1`. This option works for every command.
10. **ls > output.txt**: This command captures the output of the `ls` command and saves it as a text file named `output.txt`.
11. **cat output.txt**: This command displays the contents of the `output.txt` file without having to open the file for editing.
12. **cat -n output.txt**: This command prints the contents of the `output.txt` file with line numbers included.
13. ***cat t* > combine.txt***: This command concatenates all the files whose names start with 't' and saves the combined content to a file named `combine.txt`. If the file already exists, the command overwrites it.
14. ***cat t* >> combined.txt***: This command appends the data of all files whose names start with 't' to the existing file named `combined.txt`. If the combined file already exists, the new data will be added to the end of the existing content.
15. **vi text.txt** or **vim text.txt**: Opens the file `text.txt` for editing in Vi or Vim.
16. **head text.txt** or **tail text.txt**: Use the `head` command to display the first part of the `text.txt` file, or use the `tail` command to display the last part
17. **head -n 100 text.txt**: This command displays the first 100 lines of the `text.txt` file using the `head` command.
18. **echo "This is a test" > test_1.txt**: The `echo` command is a Unix/Linux tool used for displaying lines of text. This command writes the text "This is a test" to a file named `test_1.txt`.
19. **date > text.txt**: This command inserts the current date and time into a text file named `text.txt`. Unlike using `echo`, which would literally insert the word "date", using `date` command will insert the actual date and time.

20. **>**: The greater-than symbol `>` is used for overwriting a file with new content. If the file already exists, it will be replaced with the new content.
21. **>>**: The double greater-than symbol `>>` is used for appending data to the end of a file. If the file already exists, the new data will be added to the existing content without overwriting it.
22. **>**: The greater-than symbol `>` is used to redirect output to a file. It will create the file if it does not exist, and overwrite the file if it already exists.
23. **>>**: The double greater-than symbol `>>` is used to append output to a file. Like `>`, it will create the file if it does not exist, but unlike `>`, it will append to the file if it already exists instead of overwriting it.
24. **?** matches exactly one occurrence of any character.
25. `*` matches zero or more occurrences of the preceding character or expression.
26. **Touch {a,b,c}.txt**: The `touch` command with curly braces creates multiple files at once. For example, `touch {a,b,c}.txt` creates `a.txt`, `b.txt`, and `c.txt`.
27. **touch app.{js,app,pdf,py}**: Using curly braces with the `touch` command creates multiple files at once. For example, `touch app.{js,app,pdf,py}` generates files named `app.app`, `app.js`, `app.pdf`, and `app.py`.
28. **{1..99}**: Curly braces with a range specified generates a sequence of numbers. For instance, `{1..99}` expands to numbers from 1 to 99.
29. **echo "I've appended a line!" >> combined.txt**: The `echo` command appends the specified text to a file named `combined.txt`. For example, `echo "I've appended a line!" >> combined.txt` adds the text "I've appended a line!" to the end of the file `combined.txt`.
30. **Less combined.txt**: The `less` command is a Linux utility used to read the contents of a text file one page (one screen) at a time. For example, `less combined.txt` opens the file `combined.txt` in the `less` pager, allowing you to navigate through its contents one page at a time.
31. Unix systems are **case-sensitive**, meaning they treat "A.txt" and "a.txt" as two distinct files.
32. **mv combined.txt dir1**: Moves the file `combined.txt` to the `dir1` directory.
33. **mv named.txt renamed.txt**: Renames the file `named.txt` to `renamed.txt`.
34. ***mv combined.txt test_* dir3 dir2***: Moves `combined.txt`, any file matching `test_*`, and the contents of `dir3` to `dir2`.
35. **dir1/***: Refers to any files in the `dir1` directory.
36. **cp source target**: Copies the file named `source` to a new file named `target`.
37. **cp dir4/dir5/dir6/combined.txt .**: Copies the file `combined.txt` from `dir4/dir5/dir6` to the current directory.
38. **cp -r**: Use `-r` to recursively copy directories and their contents.
39. **rm folder_***: Removes files and folders matching the pattern `folder_*`.

40. **rmdir folder_***: Removes directories only if they are empty and match the pattern `folder_*`.
41. **rm -r folder_6**: Removes the directory `folder_6` and its contents recursively, even if it's not empty.
42. **rm -ri folder**: The `-ri` option prompts for confirmation before recursively removing the directory and its contents.
43. **wc combined.txt**: The `wc` command counts the number of lines, words, and bytes in the file `combined.txt`.
44. **wc -l combined.txt**: The `-l` option tells `wc` to count only the number of lines in `combined.txt`.
45. **wc -w text.txt**: The `-w` option tells `wc` to count only the number of words in `text.txt`, while `-m` counts characters, and `-c` counts bytes.
46. **| (pipe)**: The pipe symbol `|` is used to redirect the output of one command as the input to another command. For example, `ls /etc | wc -l` counts the number of items in the `/etc` directory by first listing its contents and then counting the lines of the output.
47. **uniq file.txt**: Lists unique lines from `file.txt`, but if lines are combined, you won't get fully unique values.
48. **sort -u file.txt**: Sorts `file.txt` and outputs only the fully unique values.
49. **sort text.txt | uniq -d**: Lists only the duplicated lines from `text.txt`.
50. **sort text.txt | uniq -u**: Lists only the non-duplicated lines from `text.txt`.
51. **sort text.txt | uniq -c**: Displays the count of each line in `text.txt`.
52. **sort text.txt | uniq -c | sort -n**: Sorts the count values of each unique line in `text.txt` numerically.
53. **cat combined.txt | uniq | wc -l**: Uses `uniq` to get unique values from `combined.txt`, then counts the number of unique lines.
54. **sort text.txt**: Sorts the contents of `text.txt` and displays the sorted result in the terminal.
55. **sort text.txt > sorted.txt**: Sorts the contents of `text.txt` and saves the sorted result to a new file named `sorted.txt`.
56. **sort -n text.txt**: Sorts the contents of `text.txt` numerically.
57. **sort -r text.txt**: Sorts the contents of `text.txt` in reverse order.
58. **sort -nr text.txt**: Sorts the numerical values in `text.txt` in reverse order.
59. **sort -nu text.txt**: Sorts and outputs only the unique numerical values from `text.txt`.
60. **diff file1.txt file2.txt**: Compares `file1.txt` and `file2.txt`, showing the differences and indicating their positions.
61. **diff -y file1.txt file2.txt**: Displays both files side by side, highlighting their differences.
62. **diff -u**: Shows the differences between files with context around each change.
63. **find . -name '*.py'**: Finds files with the `.py` extension and displays their locations.
64. **find . -type d**: Finds all directories within the current directory.
65. **find . -type d -name '*py'**: Finds directories with names containing 'py'.

66. **find . -type f**: Finds all files within the current directory.
67. ***find . -type f -name 'e*' or -name 'f*'***: Finds files with names starting with 'e' or 'f'.
68. **find . -type f -size +100k -size -1M**: Finds files larger than 100 kilobytes but smaller than 1 megabyte.
69. **find . -type f -mtime +3**: Finds files modified more than 3 days ago using modification time (`mtime`), or `-ctime` for changed time.
70. **find . -type f -mtime -1**: Finds files modified within the last 24 hours.
71. **find . -type f -mtime -1 -delete**: Deletes all files modified within the last 24 hours.
72. **find . -type f -exec cat {} ;**: Concatenates the contents of all files found.
73. **grep**: Helps find text inside files.
74. **grep bat cricket.txt**: Searches for the word 'bat' in the file `cricket.txt`.
75. **grep -n bat cricket.txt**: Searches for the word 'bat' in `cricket.txt`, displaying line numbers.
76. **grep -r "bat"**: Searches for 'bat' recursively in all files (default is current directory).
77. **du**: Calculates the size of directories (disk usage).
78. **du folder1**: Displays the size of `folder1`.
79. **du -m**: Displays disk usage in megabytes.
80. **du -g**: Displays disk usage in gigabytes.
81. **du -h | sort -h**: Finds sizes in human-readable format and sorts them accordingly.
82. **df**: Displays disk usage information.
83. **df text.txt**: Shows space allocation, usage, and available space for `text.txt`.
84. **df -h text.txt**: Shows human-readable disk usage information for `text.txt`.
85. **history**: Shows command history with numbers.
86. **!2343**: Executes the command with the specified number from history.
87. **history | grep 'cricket'**: Filters command history to show commands containing 'cricket'.
88. **ps**: Displays information about running processes (stands for process status).
89. **kill -9 85746**: Terminates the process with the specified process ID (PID) forcefully.
90. **killall -9 python**: Terminates all processes named 'python' forcefully.
91. **sleep 1000 &**: Starts a process (in this case, `sleep 1000`) in the background.
92. **sleep 1000**: Starts a process (in this case, `sleep 1000`) in the foreground.
93. **jobs**: Lists all currently running background jobs.
94. **jobs -l**: Provides more detailed information about background jobs.
95. **sleep 100**: Pauses the execution of a script or command for a specified amount of time.
96. **Control + z**: Suspends a currently running process and puts it into the background; it can be continued later.
97. **fg 2**: Resumes the process with job number 2 in the foreground.
98. **bg %job_id**: Resumes a suspended job in the background.
99. **gzip text.txt**: Compresses the file `text.txt`, deleting the original and creating `text.txt.gz`.
100. **gzip -c text.txt > giptexed.txt** or **gzip -k text.txt**: Compresses `text.txt` without deleting the original, creating `giptexed.txt.gz`.
101. **gzip -d gipfile.gz** or **gunzip**: Decompresses `gipfile.gz` to `gipfile`.

102. **tar**: Groups multiple files into a single file.
103. **tar -cf archive.tar file1 file2**: Creates an archive named `archive.tar` containing `file1` and `file2`.
104. **tar -xf archive.tar**: Extracts files from `archive.tar`.
105. **tar -xf archive.tar -C directory**: Extracts files from `archive.tar` to the specified `directory`.
106. **nano**: A text editor.
107. **alias**: Creates a custom command to represent a longer command.
108. Example: `alias l='ls -ltr'` creates a shortcut `l` to run `ls -ltr`.
109. **nano ~/.bashrc**: Opens the `.bashrc` file to save aliases.
110. **xargs**: Uses the output of one command as input for another.
111. Example: `cat files.txt | xargs rm` removes the files listed in `files.txt`.
112. **ln main.txt hardlink.txt**: Creates a hard link named `hardlink.txt` to `main.txt`, acting as a mirror image; changes to one affect the other bidirectionally.
113. **rm main.txt**: Even if `main.txt` is removed, `hardlink.txt` remains unaffected as it's a hard link.
114. **ln -s main.txt softtext.txt**: Creates a symbolic link (`softtext.txt`) to `main.txt`; if `main.txt` is removed, `softtext.txt` becomes invalid.
115. **who**: Displays the number of users logged in to the system.
116. **sudo**: Used to execute commands with root permissions.
117. **sudo -i**: Starts a root shell session.
118. **sudo useradd newuser**: Creates a new user with username `newuser`.
119. **sudo useradd -m newuser**: Creates a new user with username `newuser` and a home directory.
120. **passwd**: Used to change a user's password.
121. **sudo passwd newuser**: Sets a password for a new user or changes the password for an existing user.
122. **su user_name**: Switches to the specified user.
123. **su - ashish**: Switches the user to 'ashish' and sets up the environment similar to when 'ashish' logs in.
124. **exit**: Exits the current user session and returns to the previous user or session.
125. **passwd**: Used by users to change their own passwords.
126. **chown**: Changes the owner of a file or directory.
127. Example: `chown ashish1 folder1` changes the owner of `folder1` to 'ashish1'.
128. **sudo chown ashish1 folder1**: Changes ownership of `folder1` to 'ashish1' with root permissions.
129. **sudo chown -R ashish1 folder1**: Changes ownership of `folder1` and its contents to 'ashish1' recursively.
130. **groups**: Displays the list of groups the current user belongs to.
131. **sudo chown ashish**
gym/: Changes the group owner of the 'gym' folder to 'gymgroup', with 'ashish' as the owner.

132. **sudo groupadd gymgroup**: Creates a new group named 'gymgroup'.
133. **sudo usermod -aG gymgroup ashish**: Adds user 'ashish' to the 'gymgroup' group.
134. **File Permissions**:

- The first character indicates whether it's a file (-) or directory (d).
- The next three characters represent permissions for the owner (read, write, execute).
- The following three characters represent permissions for the group (read, write, execute).
- The last three characters represent permissions for others (read, write, execute).

135. **chmod mode file**: Changes file permissions.

- Mode can be who (u for user, g for group, o for others, a for all), what (- for minus, + for plus to add, = to set explicitly), and which (r for read, w for write, x for execute).

136. **chmod u+w file1**: Adds write permission to the user for `file1`.
137. **chmod u-rwx file1**: Removes all permissions (read, write, execute) for the user from `file1`.
138. **chmod ug-r file1**: Removes read permission for both user and group from `file1`.
139. **chmod a=r file1**: Sets read-only permission for all (user, group, others) on `file1`.
140. **chmod u=r file1**: Sets read-only permission for the user on `file1`.