Common Text Processing Commands in Linux

1. `cat`

Definition: The `cat` command is used to display the contents of a file, concatenate files, or create a new file.

Usage/Formula:

```
cat [OPTION] FILE
```

Examples:

- Display the contents of a file with non-printing characters and $'s, equivalent of using -vE :

  ```
  cat -e file.txt
  ```

- Concatenate two files into a third file:

  ```
  cat file1.txt file2.txt > combined.txt
  ```

- Create a new file by typing content directly(exit with Ctrl+C):

  ```
  cat > shopping_list.txt
    buy sweet potatoes
    ^C
  ```

2. `tac`

Definition: The `tac` command displays the contents of a file in reverse order (line by line).

Usage/Formula:

```
tac [OPTION] FILE
```

Examples:

- Display the contents of a file in reverse order suppressing repeating empty lines to a single empty line:

  ```
  tac -s file.txt
  ```

- Reverse the order of multiple concatenated files:

```
tac file1.txt file2.txt
```

- Combine `tac` with redirection to save the reversed content:

```
tac file.txt > reversed_file.txt
```

3. `head`

Definition: The `head` command displays the first few lines of a file (default is 10 lines).

Usage/Formula:

```
head [OPTION] FILE
```

Examples:

- Display the first 10 lines of a file(default behaviour):

```
head file.txt
```

- Display the last 20 lines of a file:

```
head -n 20 file.txt
```

- Display the last 10 lines of multiple files:

```
head sys32.txt camote.csv
```

4. `tail`

Definition: The `tail` command displays the last few lines of a file (default is 10 lines).

Usage/Formula:

```
tail [OPTION] FILE
```

Examples:

- Display the last 10 lines of a file(default behaviour):

```
tail file.txt
```

- Display the last 20 lines of a file:

```
tail -n 20 file.txt
```

- Display the last 10 lines of multiple files using wildcards:

```
tail *.txt *.csv
```

5. `cut`

Definition: The `cut` command is used to extract specific sections (fields or columns) from a file or input.

Usage/Formula:

```
cut [OPTION] FILE
```

Examples:

- Extract the second and third field from a CSV file:

```
cut -d',' -f2,3 file.csv
```

- Extract the first field from a CSV file but changing the delimiter in the output:

```
cut -d',' -f1 --output-delimiter=' = ' file.csv
```

- Extract from a file excluding the third and fourth fields:

```
cut -d',' --complement -s -f3,4 file.csv
```

6. `sort`

Definition: The `sort` command sorts lines in a file in ascending or descending order.

Usage/Formula:

```
sort [OPTION] FILE
```

Examples:

- Sort lines alphabetically:

  ```
  sort file.txt
  ```

- Sort lines in reverse order:

  ```
  sort -r file.txt
  ```

- Sort a CSV file by the second field in numerical order:

  ```
  sort -nt',' -k2 file.csv
  ```

7. `wc`

Definition: The `wc` command is used to count lines, words, and characters in a file.

Usage/Formula:

```
wc [OPTION] FILE
```

Examples:

- Count the number of lines in a file:

  ```
  wc -l file.txt
  ```

- Count words in a file:

  ```
  wc -w file.txt
  ```

- Display line, word, and character counts for a file:

```
wc file.txt
```

8. `tr`

Definition: The `tr` command is used to translate or delete characters from input.

Usage/Formula:

```
tr [OPTION] SET1 [SET2]
```

Examples:

- Convert lowercase letters to uppercase:

```
echo "hello world" | tr 'a-z' 'A-Z'
```

- Replace spaces with underscores:

```
echo "hello world" | tr ' ' '_'
```

- Remove digits from input:

```
echo "abc123" | tr -d '0-9'
```

- Convert tabs into space:

```
cat file.py | tr -s "[:space:]" ''
```

9. `diff`

Definition: The `diff` command is used to compare two files line by line and display their differences.

Usage/Formula:

```
diff [OPTION] FILE1 FILE2
```

Examples:

- Compare two files and display differences:

```
diff db2019.txt db2020.txt
```

- Compare two files and display differences in a column format:

```
diff -y db2019.txt db2020.txt
```

- Suppress identical lines in the output:

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

- Create a unified diff (used in patches):

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

10. `grep`

Definition: The `grep` command searches for patterns in files and displays matching lines.

Usage/Formula:

```
grep [OPTION] PATTERN FILE
```

Examples:

- Search for a word in a file (can be contained inside another word):

```
grep "word" file.txt
```

- Search for the full specified word in a file:

```
grep -w "keyword" file.txt
```

- Perform a case-insensitive search:

```
grep -i "kEywORd" file.txt
```

- Search recursively in all files within a directory:

```
grep -R "keyword" /path/to/directory
```

- Search for every line that starts with a specific word in a file:

```
grep "^keyword" file.txt
```

- Search and display the number of times a word appears in a file:

```
grep -c "keyword" file.txt
```