Notes 7

1. cat

Definition: `cat` (concatenate) reads files sequentially and writes their contents to standard output, or concatenates multiple files.

Usage / Syntax:

```
cat [OPTIONS] [FILE]...
```

Examples:

```
# 1) Display the contents of a file
cat file.txt

# 2) Concatenate two files into a new one
cat part1.txt part2.txt > combined.txt

# 3) Number non-blank output lines
cat -b file.txt

# 4) Squeeze multiple blank lines into one
cat -s file.txt
```

2. tac

Definition: `tac` (reverse cat) displays a file line by line in reverse order.

Usage / Syntax:

```
tac [OPTIONS] [FILE]...
```

Examples:

```
# 1) Show file lines in reverse
tac file.txt

# 2) Save reversed output to another file
tac log.txt > reversed_log.txt
```

```
# 3) Number lines in reverse order
tac -n file.txt
```

---

3. head

Definition: `head` outputs the first part of files, by default the first 10 lines.

Usage / Syntax:

```
head [OPTIONS] [FILE]...
```

- `-n N`: print the first N lines (default is 10)
- `-c N`: print the first N bytes

Examples:

```
# 1) Show the first 10 lines (default)
head file.txt

# 2) Show the first 5 lines
head -n 5 file.txt

# 3) Show the first 20 bytes
head -c 20 file.txt
```

---

4. tail

Definition: `tail` outputs the last part of files, by default the last 10 lines.

Usage / Syntax:

```
tail [OPTIONS] [FILE]...
```

- `-n N`: print the last N lines (default is 10)
- `-c N`: print the last N bytes
- `-f`: follow the file as it grows (real-time)

Examples:

```
# 1) Show the last 10 lines
 tail file.txt

# 2) Show the last 20 lines
```

```
 tail -n 20 file.txt

# 3) Follow a log file in real time
 tail -f /var/log/syslog
```

---

5. cut

Definition: `cut` removes sections from each line of files, based on delimiters or character positions.

Usage / Syntax:

```
cut [OPTIONS] [FILE]...
```

- `-d 'DELIM'`: specify the delimiter character (default is tab)
- `-f LIST`: select fields (columns) separated by delimiter
- `-c LIST`: select characters by position

Examples:

```
# 1) Extract the first field from a comma-separated file
cut -d ',' -f 1 data.csv

# 2) Extract fields 2 to 4 from /etc/passwd\ ncut -d ':' -f 2-4 /etc/passwd

# 3) Extract characters 1 through 5
cut -c 1-5 file.txt

# 4) Extract non-consecutive fields 1,3,5
cut -d',' -f1,3,5 data.csv
```

---

6. sort

Definition: `sort` sorts lines of text files according to various criteria.

Usage / Syntax:

```
sort [OPTIONS] [FILE]...
```

- `-r`: reverse the results of comparisons
- `-n`: compare according to string numerical value
- `-k N,M`: sort based on keys (fields) from N to M

Examples:

```
# 1) Sort alphabetically
sort names.txt

# 2) Sort in reverse order
sort -r names.txt

# 3) Sort numerically
sort -n numbers.txt
```

---

7. wc

Definition: `wc` (word count) counts lines, words, and bytes in files.

Usage / Syntax:

```
wc [OPTIONS] [FILE]...
```

- `-l`: print the newline counts
- `-w`: print the word counts
- `-c`: print the byte counts

Examples:

```
# 1) Count lines, words, and bytes
wc text.txt

# 2) Count only lines
wc -l text.txt

# 3) Count only words
wc -w text.txt
```

---

8. tr

Definition: `tr` (translate) translates or deletes characters from standard input, writing to standard output.

Usage / Syntax:

```
tr [OPTIONS] SET1 [SET2]
```

- `-d SET`: delete characters in SET
- `-s SET`: replace repeated characters in SET with a single

Examples:

```
# 1) Convert lowercase to uppercase
tr 'a-z' 'A-Z' < file.txt

# 2) Delete spaces
tr -d ' ' < file.txt

# 3) Squeeze repeated tabs into one
tr -s '\t' < file.txt
```

---

9. diff

Definition: `diff` compares files line by line and displays the differences.

Usage / Syntax:

```
diff [OPTIONS] FILE1 FILE2
```

- `-u`: output in unified format
- `-y`: display side-by-side

Examples:

```
# 1) Compare two files
diff v1.txt v2.txt

# 2) Show unified diff
diff -u v1.txt v2.txt

# 3) Show side-by-side comparison
diff -y v1.txt v2.txt

# 4) Ignore whitespace changes
diff -w v1.txt v2.txt
```

---

10. grep

Definition: `grep` searches for patterns in files, printing matching lines.

Usage / Syntax:

```
grep [OPTIONS] PATTERN [FILE]...
```

- `-i`: ignore case distinctions
- `-r`: read all files under each directory, recursively
- `-n`: prefix each line of output with the line number

Examples:

```
# 1) Search for 'error' in a log file
 grep 'error' log.txt

# 2) Case-insensitive search for 'warning'
 grep -i 'warning' log.txt

# 3) Recursive search in current directory
grep -r 'TODO' .

# 4) Show line numbers with matches
 grep -n 'function' script.sh
```