2. List files starting with data and ending with .csv

ls data*.csv

3. Copy specific numbered files

cp file[1-3].txt /backup/

4. Move all files except .txt files

mv *[^t] /folder/

Simple Filters

In UNIX/Linux, filters are the set of commands that take input from standard input stream i.e. **stdin**, perform some operations and write output to standard output stream i.e. **stdout**. The stdin and stdout can be managed as per preferences using redirection and pipes. Common filter commands are: grep, more, sort.

1.grep Command:

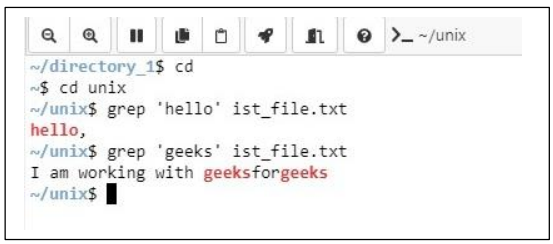
It is a pattern or expression matching command. It searches for a pattern or regular expression that matches in files or directories and then prints found matches.

Syntax:

$grep[options] "**pattern to be matched**" filename

Example:

Cmd→ $grep 'hello' ist_file.txt



S.no	OPTIONS	DESCRIPTiON
01	-v	Returns all lines that do not match the specified regular expression.
02	-n	Returns all lines that match the specified regular expression along with line no.
03	-l	Returns only names of files matching the specified regular expression.
04	-c	Returns count of lines that match regular expression.
05	-i	It is case sensitive option and matches either upper-case or lower-case

Grep command can also be used with meta-characters.

* is a meta-character and returns matching 0 or more preceding characters.

Example:

Input : $grep 'hello' *

Output : it searches for *hello* in all the files and directories.

2. sort Command:

It is a data manipulation command that sorts or merges lines in a file by specified fields. In other words it sorts lines of text alphabetically or numerically, **default sorting is alphabetical**.

Syntax:

$sort[options] filename

Example:

$sort fruits.txt

$sort -n grades.txt

```
~$ cat fruits.txt
mango
orange
kiwi
banana
apple
peach
plum
~$ sort fruits.txt
apple
banana
kiwi
mango
orange
peach
plum
~$ cat grades.txt
1. A
5. A+
6. B
3. C
7. A
2. B+
4. A
~$ sort -n grades.txt
1. A
2. B+
3. C
4. A
5. A+
6. B
7. A
~$
```

S.NO.	OPTIONS	DESCRIPTION
01.	-n	It sorts the lines in numerical order.
02.	-r	It reverses the order of sorting. Eg. If the lines were sort from a to z, it will sort them from z to a.
03.	-f	It sorts uppercase and lowercase together.
04.	+x	It doesn't consider ist x fields while sorting.

3. more Command:

It is used to customize the displaying contents of file. It displays the text file contents on the terminal with paging controls. Following key controls are used:

- To display next line, press the enter key
- To bring up next screen, press spacebar
- To move to the next file, press n
- To quit, press q.

Syntax:

$more[options] filename

Example:

```
cat fruits.txt | more
```

While using more command, the bottom of the screen contains more prompt where commands are entered to move through the text.

Regular Expressions

Regular expressions (regex) in Unix are powerful tools for pattern matching and text processing. They're used in various Unix commands like grep, sed, awk, and in scripting to search, match, and manipulate text.

Basic Syntax:

- . matches any single character except newline.
- * matches zero or more occurrences of the preceding character.
- ^ matches the start of a line.
- $ matches the end of a line.
- [abc] matches any one character inside the brackets (a, b, or c).
- [a-z] matches any one character in the range a to z.
- \ is used to escape special characters.

Common Usage Examples:

- grep '^a' file.txt — lines starting with a.
- grep 'abc$' file.txt — lines ending with abc.
- grep 'a.*b' file.txt — lines containing a followed by b with anything in between.

Extended Regular Expressions (ERE):

Use grep -E or egrep for extended features:

- + matches one or more occurrences.
- ? matches zero or one occurrence.

- | acts as OR.
- () for grouping.

Examples:

- grep -E 'colou?r' file.txt matches color or colour.
- grep -E 'cat|dog' file.txt matches lines with cat or dog.
- sed 's/[0-9]\+//g' removes all numbers.

1.grep
*grep is the simplest of the three: it searches through text and prints lines that match a pattern. Think of it as a really fast filter.
*You give it a pattern (usually a regex), and a file or input stream, and it spits out the lines that match.

Example:
grep 'error' logfile.txt

//This prints every line from logfile.txt that contains the word "error".

features:

- Use -i to ignore case (grep -i 'error' logfile.txt finds “Error”, “ERROR”, etc.)
- -v to invert match (print lines that *don’t* match)
- -r to recursively search directories
- -E or egrep to use extended regex
- Count matches with -c
- Show line numbers with -n

2.awk
*awk is a mini programming language designed for text processing, especially tabular data. It reads input line by line, splits each line into fields (columns), and lets you run scripts to manipulate or extract data.

*****You write patterns and actions. When a line matches the pattern, awk runs the action on that line.

Example:
awk '$3 > 50 { print $1, $3 }' data.txt

//This prints the first and third columns of lines where the third column is greater than 50.

Features:

- Automatically splits lines into fields by whitespace (or any delimiter you choose)
- Built-in variables like $0 (whole line), $1, $2 (fields), NR (line number)
- Supports arithmetic, string functions, conditionals, loops
- Great for reports, summaries, and column extraction

3.sed

*sed is a stream editor. It reads input line by line, applies editing commands (like search-and-replace), and outputs the transformed text. It's perfect for quick text substitutions or more complex editing in scripts.

*You give sed a script of commands (usually just one), and it processes each line accordingly.

Example:

sed 's/foo/bar/g' file.txt

//This replaces every "foo" with "bar" on each line of file.txt.

Features:

- Delete lines (sed '/^#/d' deletes lines starting with #)
- Insert or append lines
- Extract specific lines or ranges
- Complex multi-line editing with advanced scripts

NOTE:

- Use **grep** when you just want to find or filter lines matching a pattern. It's fast and simple.
- Use **awk** when you want to process columns, do calculations, or generate reports from structured text.
- Use **sed** when you want to transform text—replace, delete, insert, or rearrange parts of the text.

Example:

grep 'error' logfile.txt | awk '{print $1, $3}' | sed 's/foo/bar/g'

//This finds lines with “error,” prints certain fields, then replaces “foo” with “bar.”

Grep Family

Overview:

grep stands for **Global Regular Expression Print**. It’s a Unix command-line utility designed to search for patterns in text files or input streams and print the matching lines.

The grep family includes:

- **grep** — the classic tool supporting **Basic Regular Expressions (BRE)**
- **egrep** — stands for **Extended grep**, supports **Extended Regular Expressions (ERE)**
- **fgrep** — stands for **Fixed grep** (or fast grep), searches for **fixed strings** (no regex)

1. grep — Basic Regular Expressions (BRE)

- Search text for patterns using basic regex syntax.
- **features:**
  - . matches any single character.
  - * matches zero or more of the previous character.
  - ^ and $ anchor start and end of line.
  - Character classes like [abc] or [0-9].
  - Grouping and alternation require escaping (\(, \), \|).
- **Example:**
  grep '^a.*b$' file.txt //Finds lines starting with a and ending with b.

- **Features:**
  - Use -i for case-insensitive search.
  - Use -v to invert match (print non-matching lines).

- Use -c to count matches.
- Use -n to print line numbers.

2. egrep (or grep -E) — Extended Regular Expressions (ERE)

- Like grep but supports extended regex syntax without needing to escape special characters.
- **features:**
  - + one or more repetitions.
  - ? zero or one occurrence.
  - | alternation (OR).
  - Grouping with () without escaping.
- **Examples:**
  egrep 'colou?r' file.txt //Matches “color” or “colour”.

  grep -E 'cat|dog' file.txt //Matches lines with “cat” or “dog”.

- **Note:** egrep is technically deprecated on some systems in favor of grep -E, but works the same.

3. fgrep (or grep -F) — Fixed String Search

- Searches for **literal strings** only, no regex processing.
- When you want to search for exact text without worrying about regex special characters.
- When performance matters — fgrep is faster because it skips regex parsing.
- **Example:**
  fgrep 'a+b*c?' file.txt

  //Matches lines that literally contain the string a+b*c?, not treating +, *, or ? as regex symbols.

- **Note:** fgrep is also deprecated on some systems; use grep -F instead.

Summary Table:

Command	**Regex Support**	**Special Characters**	**Usage**
grep	Basic Regular Expressions	*, ., ^, $, [], \(, \)	Standard regex search
egrep / grep -E	Extended Regular Expressions	+, ?, |, () (no escaping)	More powerful regex search

fgrep / grep -F	No regex, fixed string search	None	Literal string search

NOTE:

- If you're unsure which to use, start with grep. If your pattern needs extended features like + or |, switch to grep -E.
- Use fgrep or grep -F when searching for fixed text, especially if your pattern includes characters like * or ? and you don't want them interpreted as regex.

++Example:

- case-insensitive, show line numbers, extended regex.
  grep -i -n -E 'pattern' file.txt

- Find lines containing either "cat" or "dog":
  grep -E 'cat|dog' animals.txt
- Count lines NOT containing "error":
  grep -v -c 'error' logfile.txt # -v Exclude matching lines
- Search for literal string a+b*c? without regex interpretation:
  grep -F 'a+b*c?' file.txt

Advanced filters in UNIX

In UNIX, **advanced filters** are powerful command-line tools used to **process, manipulate, and transform text data**. These filters can be combined using **pipes (|)** to build complex data-processing pipelines.

1. grep – Pattern Searching

Search for lines matching a pattern.

Example:

```
grep "error" logfile.txt
grep -i "warning" logfile.txt      # Case-insensitive
grep -r "TODO" ./src               # Recursive search

grep -v "success" logfile.txt      # Exclude matching lines
```