PIPE

A pipe | takes the output of the command before it and passes it as an input for the command after it.
For example, if u want to output the command sort, to the input of the command head:

sort a.txt | head -n 5

sort a.txt - sort lines of text file
head -n 5 - output the first n lines of the file

REGULAR EXPRESSIONS

They are patterns used to match strings in text. Powerful way to search, replace and manipulate text by specific patterns of characters.

.	Matches any single character
\	Changes the meaning of the character following it, between normal and special
[abc]	Matches any single character that appears in the list (a, b or c)
[a-z]	Matches any single character that belongs to the range (a ... z)
[^0-9]	Matches any single character that does not appear in the list $\notin(0 \dots 9)$
^	Beginning of line
$	End of line
\<	Beginning of word
\>	End of word
( )	Allows to group more characters into an expression, example : +(abc)
*	Previous expression zero or more times
+	Previous expression one or more times
?	Previous expression zero or one times
{min,max}	Previous expression at least min and at most max times
|	Logical OR between parts of an regular expression

Examples :

1. .* - any sequence of characters
2. [a-zA-Z02468] - any letter and any even digit
3. [,] - space or comma
4. ^[^0-9]+$ - the first ^ and the last $ are ensuring the match begins at the start of the string and goes until the end of the string
   [^0-9] is going to match strings that are only one character and the character is not 0-9
   + is going to match strings that are one or multiple characters that are not 0-9

GREP

The names breakdown is :

- re - regular expression
- p - print matching lines
- g - global, meaning the command should be applied to all lines in the file

Options arguments for grep :

- -E - use extended regular expressions, so the shell doesn't misinterpret characters in the command, for example grep "cat|dog" file treats | as a pipe, in contrast with grep -E "cat|dog" file which treats | as the logical OR
- -v - displays lines that do not match the given regular expression
- -i - case-insensitiv, meaning it ignores upper/lower case when matching
- -q - do not display matching lines, just exiting with 0 if found, or 1 if not found

SED

It is used for searching/replacing/adding/deleting text from file.
By default, it does not modify the file, but displays result of processing the input file.
To modify the file we need to use the -i option for in-place editing : sed -i
Features :

a. Search/Replace for strings
   sed -E "s/old/new/flags" file.txt
   s - is the search/replace command
   flags can be g, i or both
   1. g - replacement everywhere on the line, without it, only the first appearance is replaced
   2. i - perform a case-insensitive search
   Exemplu :
   bash : echo "ana are mere" | sed 's/mere/pere/'
   output : ana are pere
b. Search.Replace for individual characters
   sed -E "y/oldchar/newchar/" file.txt
   y - is the transliteration command
   Exemplu
   bash : echo "ana are mere" | sed 'y/aeiou/AEIOU/'
   output : AnA ArE mErE
c. Swapping strings using regex
   sed -E "s/(grup1) (grup2) ... (grupN)/\ordine/flags" file.txt
   Exemplu :
   bash : echo "hello world" | sed "s/(hello) (world)/\2 \1/"
   output : world hello
d. Delete lines matching a regulat expression
   sed -E "/regex/d" file.txt
   d - is the line deletion command
   Exemplu :
   bash : sed -E "/[0-9]/d" file.txt
   output : sterge liniile care contin cel putin o cifra

AWK

1. It is a programming language usually used in terminal. The structure of and AWK Command is awk 'pattern { action }' file, where pattern can be an expresion : awk '$3 > 100 { print $3 }' file or regular expressions used in the forward slashes (/ ... /) : awk '/^John/ { print $2 }' file or the pattern can be empy, so the action block runs for every line and inside the action, you can explicitly test the condition : awk '{ if ($3 > 100) print $3 }'
2. Awk treats the input text as a table, with each line being a row, and the fields of each rows are separated by the delimiter (default is space). You can change the delimiter using -F option.
3. Selectors :
   a. BEGIN - the block associated with this selector is executed before any input has been processed
   b. END - the block associated with this selector is executed after all input has been processed
4. Special variabes :
   a. NR - number of the current line of input
   b. NF - the number of fields on the current line
   c. $0 - the entire input line
   d. $1,$2, .. - the fields of the current line
5. The AWK program can be written in a file, or provided directly on the command line between apostrophes

Example :

angajati.txt

```
Ion 3000
Maria 4000
Andrei 2500
```

AWK script :

```
awk 'BEGIN { suma = 0; print "Salarii angajati:" }
 { suma += $2; print $1, $2 }
 END { print "Suma salariilor:", suma }' angajati.txt
```