COMP311
Linux OS Laboratory
Lab8:Text Processing Tools and Regular Expressions

By

Alaa’ Omar

جامعة بيرزيت

BIRZEIT UNIVERSITY

Objectives

1

Identify and use filters as valuable text processing tools.

2

Use simple regular expressions to make text processing more efficient.

Text Processing using Filters

Filters : are commands that take some input and then filter it to produce the requested output without changing the original source of input.

- **head and tail**: used to display lines from the beginning or end of a given input, respectively.
- **cat**: used to view or concatenate files.
- **grep**: used to extract certain rows (lines) from a given input. We will concentrate on the options -i, -l (EL), -v.
- **cut**: used to extract certain columns from a given input. We will use the options -d, -f, and –c.
- **tr**: translates (changes) a given input to a specified output
- **wc**: used to count lines, words, or characters in a given input.
- **sort**: used for sorting a given input. We will present the options -i, -o, -u, -n, -k, and –t.
- **sed**: used for stream editing (changing parts of an input to a specified output)

Filters

Create the following file called **students** using the **vi** command and then save and quit:

```
ah6:506:Ahmad_Hamdan
sh5:345:Suha_HAMDAN
rd7:427:Ribhi_ahmad
hr4:234:hamdan_ribhi
ad6:386:Arwa_Ahmad
ad5:285:ahmadi_Ahmad
```

Head Command

The head Command Syntax

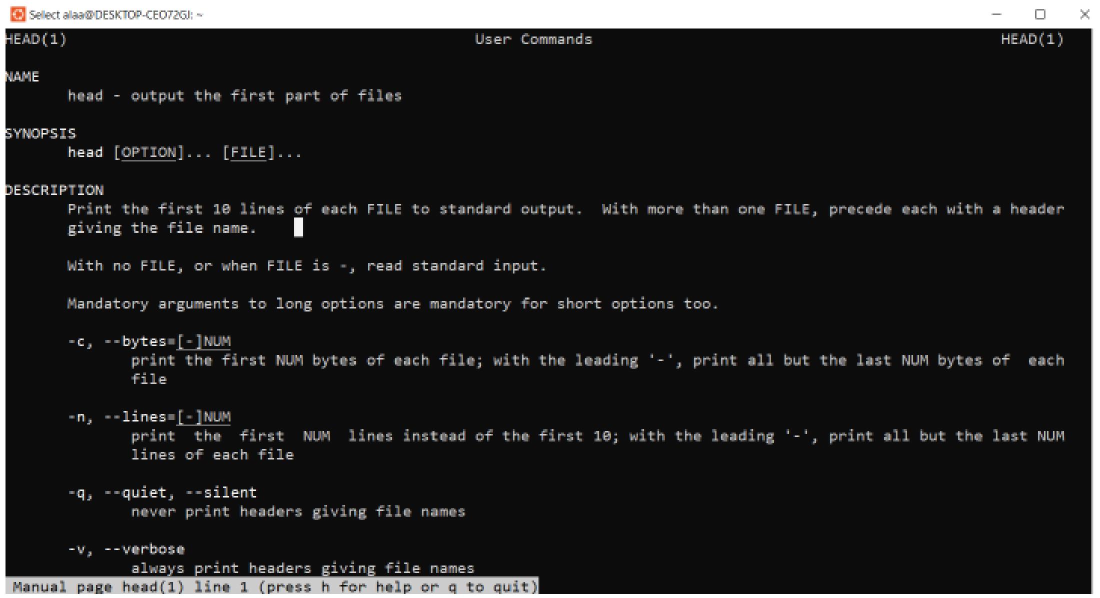
The basic syntax of the head command is

```
$ head [option] [file]
```

Using the Default head Command

By default, the head command prints out the first 10 lines of text in a file.

```
$ head students
```



Select alaa@DESKTOP-CEO72GJ: ~

```
HEAD(1)                                   User Commands                                   HEAD(1)

NAME
       head - output the first part of files

SYNOPSIS
       head [OPTION]... [FILE]...

DESCRIPTION
       Print the first 10 lines of each FILE to standard output.  With more than one FILE, precede each with a header
       giving the file name.

       With no FILE, or when FILE is -, read standard input.

       Mandatory arguments to long options are mandatory for short options too.

       -c, --bytes=[-]NUM
              print the first NUM bytes of each file; with the leading '-', print all but the last NUM bytes of  each
              file

       -n, --lines=[-]NUM
              print  the  first  NUM  lines instead of the first 10; with the leading '-', print all but the last NUM
              lines of each file

       -q, --quiet, --silent
              never print headers giving file names

       -v, --verbose
              always print headers giving file names
 Manual page head(1) line 1 (press h for help or q to quit)
```

Note

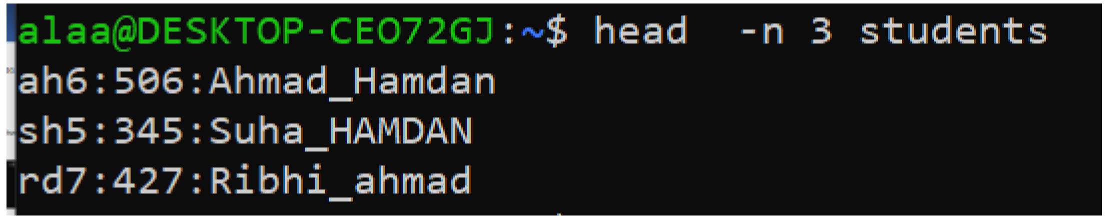
If the file has less than 10 lines, the head command will print all the lines present.

Head Command

Print the First N Number of Lines

You can use the head command to print a specific number of lines instead of the default 10. To print the first three lines of the numbers.txt file, execute this command:

```
$ head  -n 3 students
```



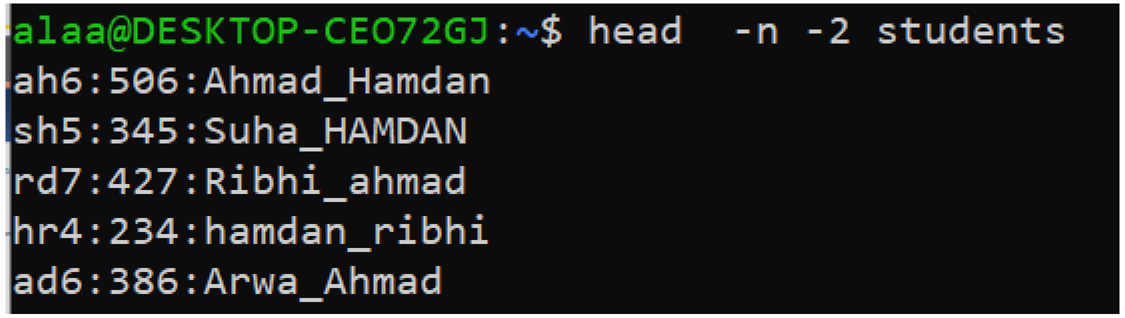
```
alaa@DESKTOP-CEO72GJ:~$ head  -n 3 students
ah6:506:Ahmad_Hamdan
sh5:345:Suha_HAMDAN
rd7:427:Ribhi_ahmad
```

Exclude the Last N Lines Using the head Command

Just like you can print out the first lines of a text, you can also decide to exclude the last N lines when printing. You can do this by using a negative number for the N parameter.

To exclude the last 15 lines of the numbers.txt file, run:

```
$ head  -n -2 students
```



```
alaa@DESKTOP-CEO72GJ:~$ head  -n -2 students
ah6:506:Ahmad_Hamdan
sh5:345:Suha_HAMDAN
rd7:427:Ribhi_ahmad
hr4:234:hamdan_ribhi
ad6:386:Arwa_Ahmad
```

Tail Command

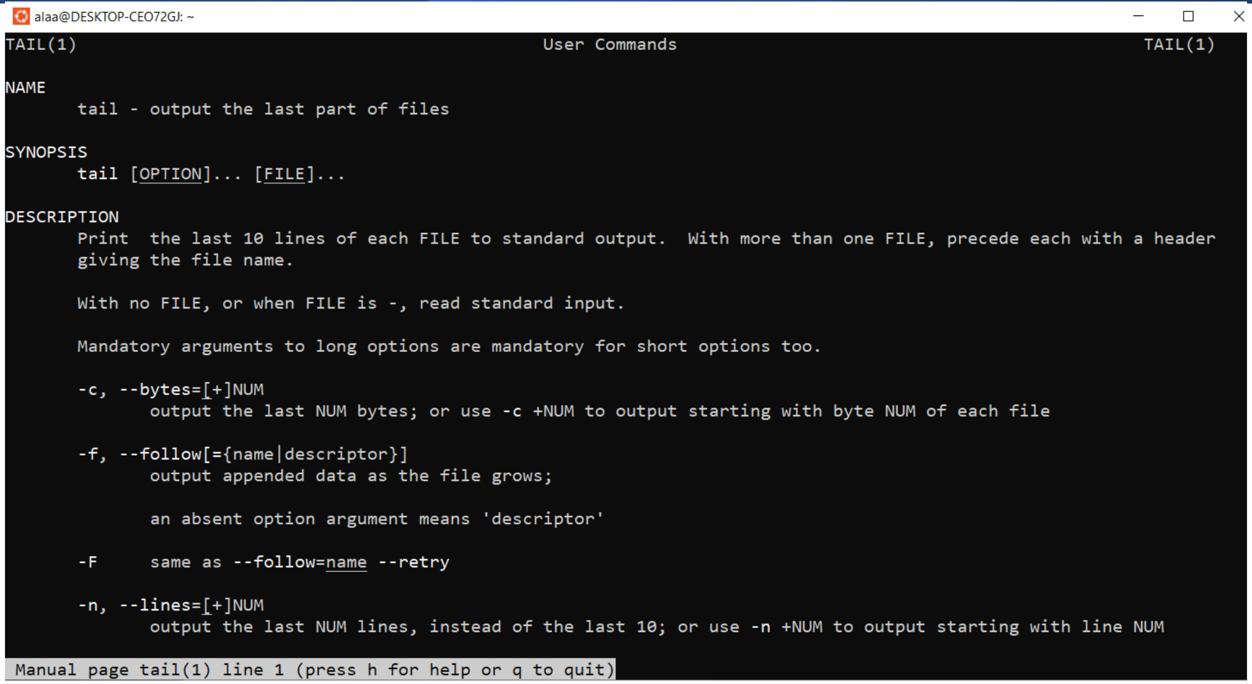
The tail Command Syntax
The basic syntax of the tail command is

```
$ tail [option] [file]
```

Using the Default head Command

By default, the tail command prints out the last 10 lines of text in a file.

```
$ tail students
```



alaa@DESKTOP-CEO72GJ: ~

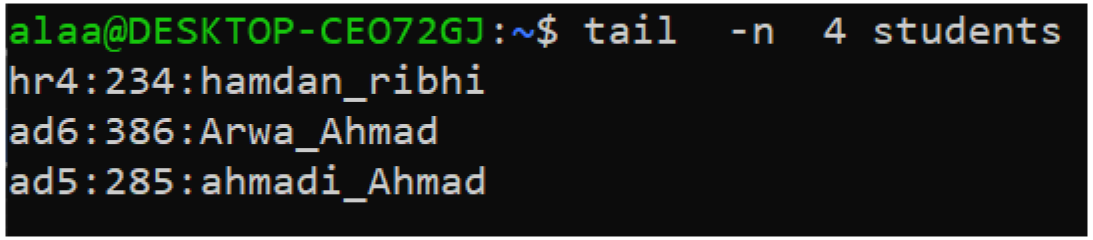
```
TAIL(1)                                    User Commands                                    TAIL(1)

NAME
       tail - output the last part of files

SYNOPSIS
       tail [OPTION]... [FILE]...

DESCRIPTION
       Print  the last 10 lines of each FILE to standard output.  With more than one FILE, precede each with a header
       giving the file name.

       With no FILE, or when FILE is -, read standard input.

       Mandatory arguments to long options are mandatory for short options too.

       -c, --bytes=[+]NUM
              output the last NUM bytes; or use -c +NUM to output starting with byte NUM of each file

       -f, --follow[={name|descriptor}]
              output appended data as the file grows;

              an absent option argument means 'descriptor'

       -F     same as --follow=name --retry

       -n, --lines=[+]NUM
              output the last NUM lines, instead of the last 10; or use -n +NUM to output starting with line NUM

 Manual page tail(1) line 1 (press h for help or q to quit)
```

Head Command

Print the Last N Number of Lines

In a situation where you do not want to print the last 10 lines, but a specific number, you can use the **-n** option to achieve that. To print the last four lines of the **students** file, execute this command:

```
$ tail -n 4 students
```



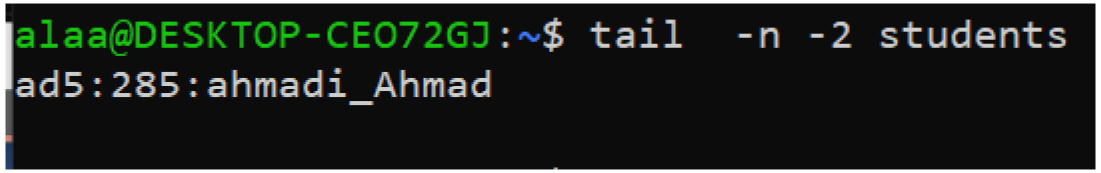
```
alaa@DESKTOP-CEO72GJ:~$ tail  -n  4 students
hr4:234:hamdan_ribhi
ad6:386:Arwa_Ahmad
ad5:285:ahmadi_Ahmad
```

Exclude the Last N Lines Using the head Command

Just like you can print out the first lines of a text, you can also decide to exclude the last N lines when printing. You can do this by using a negative number for the N parameter.

To exclude the last 2 lines of the **students** file, run:

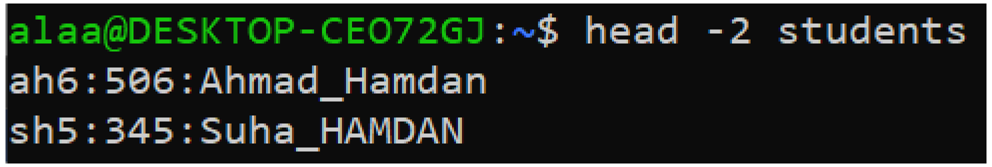
```
$ tail -n -2 students
```



```
alaa@DESKTOP-CEO72GJ:~$ tail  -n -2 students
ad5:285:ahmadi_Ahmad
```

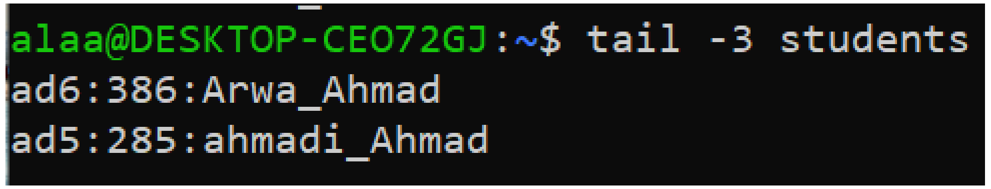
Head and Tail (Practice)

$ head -2 students



```
alaa@DESKTOP-CEO72GJ:~$ head -2 students
ah6:506:Ahmad_Hamdan
sh5:345:Suha_HAMDAN
```

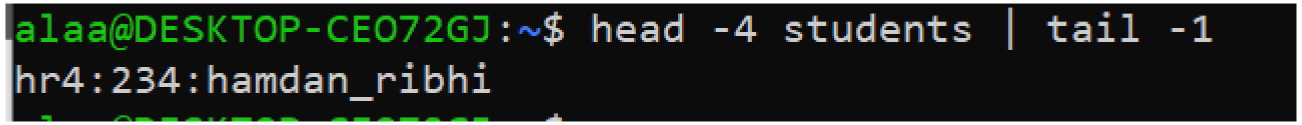
$ tail -3 students



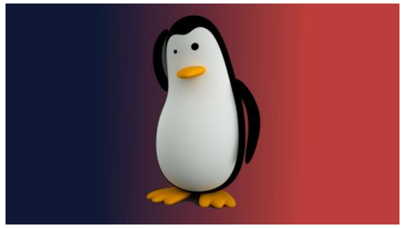
```
alaa@DESKTOP-CEO72GJ:~$ tail -3 students
ad6:386:Arwa_Ahmad
ad5:285:ahmadi_Ahmad
```

What command would you use to get the fourth line only from file students (hint: mix head and tail with pipes):

head -4 students | tail -1



```
alaa@DESKTOP-CEO72GJ:~$ head -4 students | tail -1
hr4:234:hamdan_ribhi
```



cat command

- The cat command is a versatile and frequently used command in Linux/Unix systems. "cat" stands for concatenate, but it has various other functionalities as well.

It is primarily used to:
1. Display the contents of files on the terminal
2. Concatenate multiple files into one
3. Create new files and append content to existing files

Let's explore some common use cases and options of the cat command.

cat command

- **Displaying File Contents:**
  - Syntax: cat [options] [file(s)]
  - Example: cat file.txt
  - Displays the content of "file.txt" on the terminal.

- **Concatenating Files:**
  - Syntax: cat [file1] [file2] > [output_file]
  - Example: cat file1.txt file2.txt > combined.txt
  - Concatenates "file1.txt" and "file2.txt" into "combined.txt".

grep command

- The grep command is a powerful text search tool used in Linux/Unix systems. "grep" stands for Global Regular Expression Print.
- It is primarily used to search for specific patterns or regular expressions within files or command outputs.

Let's explore some common use cases and options of the grep command.

grep command

- The grep command is a powerful text search tool used in Linux/Unix systems. "grep" stands for Global Regular Expression Print.
- It is primarily used to search for specific patterns or regular expressions within files or command outputs.

Let's explore some common use cases and options of the grep command.

grep command

- **Searching in Files:**
  - Syntax: grep [options] pattern [file(s)]
  - Example: grep "keyword" file.txt
  - Searches for the word "keyword" in "file.txt" and displays matching lines.
- **Case Insensitive Search**
  - Syntax: grep -i pattern [file(s)]
  - Example: grep -i "hello" file.txt
  - Performs a case-insensitive search for the word "hello" in "file.txt".

grep command

- The -l option (or --files-with-matches) is used to print only the names of files that contain the matching pattern.
- Instead of displaying the matching lines, it provides a concise output listing the file names.

grep command

- **Inverting Match:Case Insensitive Search**
  - Syntax: grep -i pattern [file(s)]
  - Example: grep -i "hello" file.txt
  - Performs a case-insensitive search for the word "hello" in "file.txt".

cut command

- The cut command is a powerful text processing tool used in Linux/Unix systems. It is primarily used to extract specific fields or columns from files or command outputs.

- Let's explore some common use cases and options of the cut command.

grep command

- **Specifying Delimiter:**
  - Syntax: cut -d DELIMITER -f N [file(s)]
  - Example: cut -d ',' -f 2,4 file.csv
  - Specifies the comma (',') as the delimiter and extracts the second and fourth columns from a CSV file.
- **Outputting Character Ranges:**
  - Syntax: cut -c RANGE [file(s)]
  - Example: cut -c 1-5 file.txt
  - Extracts the characters in the range of 1 to 5 from each line in "file.txt".

tr command

- The tr command is a useful text manipulation tool in Linux/Unix systems. It is primarily used for translating or deleting characters in a given input.

- Let's explore some common use cases and options of the tr command.

tr command

- ➢ Character Translation:
  - Syntax: tr SET1 SET2 [file(s)]
  - Example: echo "Hello" | tr 'l' 'L'
  - Translates all occurrences of lowercase 'l' to uppercase 'L' in the input.

- ➢ Character Ranges:
  - Syntax: tr 'A-Z' 'a-z' [file(s)]
  - Example: echo "HELLO" | tr 'A-Z' 'a-z'
  - Translates uppercase letters to lowercase, converting "HELLO" to "hello".

tr command

- The -s option in the tr command is used to squeeze repeated occurrences of a character to a single occurrence. It helps simplify and condense consecutive repeated characters in the input.

- Let's explore the usage and examples of the tr -s option.

- who | tr –s ‘ ’

cw command

- **Introduction to wc Command:**
  - The **wc** command stands for "word count" and is used to count the number of lines, words, and characters in a file or input stream.
  - It is a handy tool for analyzing text data and generating statistics.

- Let's explore the usage and examples of the tr -s option.

cw command

- **Introduction to wc Command:**
  - The **wc** command stands for "word count" and is used to count the number of lines, words, and characters in a file or input stream.
  - It is a handy tool for analyzing text data and generating statistics.

- **Basic Usage of wc Command:**
  - Syntax: **wc [options] [file(s)]**
  - The command can accept one or more files as arguments or read input from the standard input.
  - By default, it displays the line count, word count, and character count of each file.

- Let's explore some options of the wc command !

cw command

- **Introduction to wc Command:**
  - The **wc** command stands for "word count" and is used to count the number of lines, words, and characters in a file or input stream.
  - It is a handy tool for analyzing text data and generating statistics.

- **Basic Usage of wc Command:**
  - Syntax: **wc [options] [file(s)]**
  - The command can accept one or more files as arguments or read input from the standard input.
  - By default, it displays the line count, word count, and character count of each file.

- Let's explore some options of the wc command !

cw command

- **Counting Lines:**
  - Use the -l option to count the number of lines in a file.
  - Example: wc -l myfile.txt
  - It will output the total number of lines in "myfile.txt".

- **Counting Words:**
  - Use the -w option to count the number of words in a file.
  - Example: wc -w myfile.txt
  - It will output the total number of words in "myfile.txt".

- **Counting Characters:**
  - Use the -c option to count the number of characters in a file.
  - Example: wc -c myfile.txt
  - It will output the total number of characters in "myfile.txt", including whitespace

sort command

- **Introduction to sort Command:**
  - The sort command in Linux is used to sort lines of text or data in a specified order.
  - It provides a flexible way to sort data based on various criteria, such as alphabetical order, numerical values, or custom sorting rules.

- **Basic Usage of sort Command:**

  - Syntax: sort [options] [file(s)]
  - The command can accept one or more files as input or read data from the standard input.
  - By default, it sorts the input data in **ascending** order based on the entire line.

- **Sorting Data in Ascending Order:**

- Use the sort command without any options to sort data in ascending order.
- Example: sort myfile.txt
- It will output the contents of "myfile.txt" sorted in ascending order.

sort command

- **Sorting Numerical Data:**
  - Use the -n option to sort numerical data in ascending order.
  - Example: sort -n numbers.txt
  - It will sort the numbers in "numbers.txt" in ascending numerical order.

- **Sorting Options:**

  - -i or --ignore-case: Performs a case-insensitive sorting.
  - -o <output> or --output=<output>: Specifies the output file to write the sorted result.
  - -u or --unique: Removes duplicate lines from the output.
  - -t <delimiter> or --field-separator=<delimiter>: Specifies a custom field separator for sorting.

sort command

- **Examples of Sort Command:**

  - **sort file.txt**: Sorts the file in ascending order (lexicographically).
  - **sort -n file.txt**: Sorts the file in ascending order (numerically).
  - **sort -r file.txt**: Sorts the file in descending order.
  - **sort -u file.txt**: Sorts the file and removes duplicate lines.
  - **sort -t ',' -k 2,2 file.csv**: Sorts a CSV file based on the second column using comma as the delimiter.

sed command

- **Introduction to sed Command:**
  - The sed command in Linux is a stream editor used for manipulating and transforming text.
  - It reads input line by line, applies specified operations, and produces modified output.
- **Basic Usage of sed Command:**
  - Syntax: sed [options] 'command' [input_file(s)]
  - command specifies the operation(s) to be performed on the input text.
  - The input can be provided through files or piped from other commands.

Regular Expressions (Regex) in Linux

- **Introduction to Regular Expressions (Regex):**
  - Regular expressions are powerful patterns used to match and manipulate text.
  - They provide a concise and flexible way to search, extract, and modify text based on specific patterns.

- **Basic Metacharacters and Quantifiers:**
  - . (dot): Matches any single character.
  - * (asterisk): Matches zero or more occurrences of the preceding character or group.
  - + (plus): Matches one or more occurrences of the preceding character or group.
  - ? (question mark): Matches zero or one occurrence of the preceding character or group.

Regular Expressions (Regex) in Linux

- **Common Operations with sed:**
  - Substitution: Replace text patterns with new content.
  - Insertion: Insert new text at specified positions.
  - Deletion: Remove lines or portions of text.
  - Filtering: Select specific lines based on patterns.
  - Transformation: Perform various transformations on text.

- **Substitution Operation**:
  - Syntax: sed 's/pattern/replacement/[flags]' input.txt
  - Replace the first occurrence of pattern with replacement in each line.
  - Use flags like g (global) to replace all occurrences on a line.

Regular Expressions (Regex) in Linux

- Character Classes and Character Sets:
  - [ ] (square brackets): Matches any character within the specified set.
  - [a-z]: Matches any lowercase letter from a to z.
  - [0-9]: Matches any digit from 0 to 9.
  - [^] (caret): Matches any character not within the specified set.
- **Substitution Operation**:
  - Syntax: sed 's/pattern/replacement/[flags]' input.txt
  - Replace the first occurrence of pattern with replacement in each line.
  - Use flags like g (global) to replace all occurrences on a line.

Regular Expressions (Regex) in Linux

- **Anchors and Boundaries:**
  - ^ (caret): Matches the beginning of a line.
  - $ (dollar sign): Matches the end of a line.
  - \b (word boundary): Matches a word boundary.

- **Substitution Operation**:
  - Syntax: sed 's/pattern/replacement/[flags]' input.txt
  - Replace the first occurrence of pattern with replacement in each line.
  - Use flags like g (global) to replace all occurrences on a line.

The End