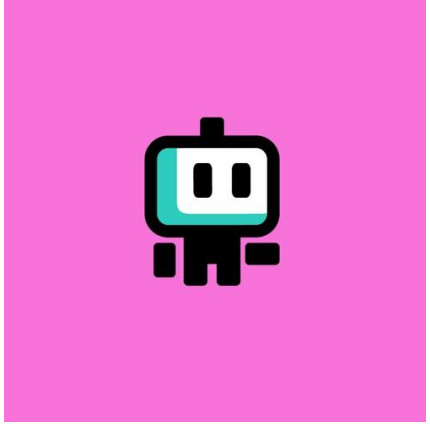

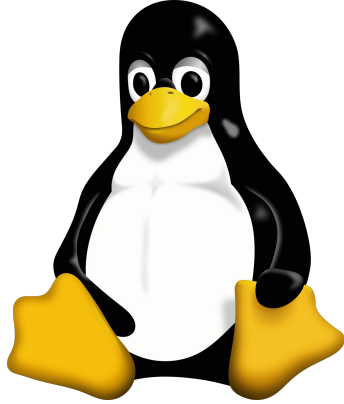

Learn How to write Bash

The sed command

About

1. `**sed**`, short for **stream editor**, is a command-line utility designed for text processing and manipulation.
2. Acting as a filter, it reads text line by line, applies specified operations, and outputs the modified text.
3. Its primary function is to automate text transformations, making it a powerful tool for scripting and streamlining repetitive tasks.
4. `**sed**` is especially adept at handling large datasets and excels in tasks like **search-and-replace**, **text substitution,** and **line deletion.**
5. Known for its simplicity and efficiency, `**sed**` operates on patterns and actions, enabling users to define rules for text modification.
6. It plays a crucial role in Unix-like operating systems, offering a flexible and scriptable approach to text editing.
7. Whether used interactively or as part of scripts, `**sed**` proves invaluable for tasks ranging from simple text edits to complex processing pipelines in the world of **command-line text manipulation**.

Usage

1. In `**sed**`, a command follows a basic structure that consists of patterns and corresponding actions.
2. The command is structured as follows: `**sed [options] 'pattern action' [input-file]**`.
3. The pattern defines the condition to match in the input text, while the action specifies the operation to perform upon a match.
4. Multiple patterns and actions can be combined within a single `**sed**` command.
5. Understanding the order of execution is crucial: `**sed**` processes input sequentially, applying patterns and actions in the order they appear.
6. Patterns dictate when an action should be triggered, and actions determine what transformation occurs when a pattern is matched.
7. The concise syntax, combining patterns and actions, enables powerful and flexible text processing.
8. Users can perform tasks like **substitution, deletion, insertion**, or **complex transformations** by leveraging the synergy between patterns and actions in a `**sed**` command.
9. This structural simplicity, coupled with its versatile capabilities, makes `**sed**` an efficient tool for automating various text manipulation tasks on the command line.

Command line Options:

`**sed**` excels in simple text substitution, a fundamental feature for transforming data.

The syntax for substitution is `s/pattern/replacement/`, where "pattern" represents the text to match, and "replacement" is the new content.

For example, to replace the first occurrence of "apple" with "orange" in a file named `fruits.txt`, the command would be

`sed 's/apple/orange/' fruits.txt`. This will work on the first occurrence for each line of the file

To substitute globally (all occurrences on a line), add the `g` flag: **`sed 's/apple/orange/g' fruits.txt`.**

Patterns employ regular expressions, enabling versatile matching. For instance, **`sed 's/[0-9]\+/<NUMBER>/' data.txt`** replaces any sequence of digits with the "**<NUMBER>**"

Mastery of these substitution basics equips users to efficiently manipulate text, making `sed` a powerful tool for automating repetitive tasks involving textual content.

Addressing in **`sed`** specifies lines to operate on.

To target a specific line, use its number (e.g., **`sed '2s/old/new/' file.txt`** replaces in the second line). Ranges, like `1,3`, affect lines within that range.

Patterns, such as `/pattern/`, match lines containing the specified pattern. Combining these addressing mechanisms allows precise control over text transformations in **`sed`** commands.

Example 1: Replacing

Assume the content of file.txt:

Line 1: apple

Line 2: banana

Line 3: cherry

Replace "banana" with "orange" only on the second line

sed '2s/banana/orange/' file.txt

 # Result:

Line 1: apple

Line 2: orange

Line 3: cherry

In this example, the `**sed '2s/banana/orange/' file.txt**` command targets only the second line (addressed by line number 2) for the substitution, resulting in the replacement of "banana" with "orange" on that specific line.

Example 2: In-Place Editing using sed -i

The `-i` option in `sed` allows in-place editing of files. When using this option, it's important to exercise caution, as it modifies the original file. To make backups before editing, add a suffix to the `-i` option (e.g., `-i.bak`). Here's an example:

Assume the content of file.txt:

Line 1: apple

Line 2: banana

Line 3: cherry

Perform in-place editing, making a backup with the .bak suffix

sed -i.bak 's/banana/orange/' file.txt

 # Result:

Line 1: apple

Line 2: orange

Line 3: cherry

Original file (file.txt) is modified, and a backup (file.txt.bak) is created

Example 3: Regular and Global Substitution(½)

Global substitution in `**sed**` involves replacing all occurrences of a pattern in each line. Unlike regular substitution, which replaces only the first occurrence, global substitution uses the `g` flag.

Here's an example:

Assume the content of file.txt:

Line 1: apple banana apple

Line 2: cherry apple banana

Regular substitution (replaces only the first occurrence)

sed 's/apple/orange/' file.txt

 # Result:

Line 1: orange banana apple

Line 2: cherry orange banana

Example 4: Regular and Global Substitution(2/2)

Global substitution (replaces all occurrences)

sed 's/apple/orange/g' file.txt

 # Result:

Line 1: orange banana orange

Line 2: cherry orange banana

In the first example, `**sed 's/apple/orange/' file.txt**` replaces only the first occurrence of "apple" on each line.

In this example, `**sed 's/apple/orange/g' file.txt**` globally substitutes all occurrences of "apple" on each line. Understanding when to use global substitution is key for comprehensive text transformations in `sed`.

Example 5: Append Insert Delete in sed(⅓)

Append

Assume the content of file.txt:

Line 1: apple

Line 2: banana

Line 3: cherry

Append "grape" to the end of each line **sed 's/$/ grape/' file.txt**

 # Result:

Line 1: apple grape

Line 2: banana grape

Line 3: cherry grape In this example, `sed 's/$/ grape/' file.txt` appends "grape" to the end of each line.

However it doesn't modify the file. To modify use : -i **Eg: sed -i 's/$/ grape/' file.txt**

Example 5: Append Insert Delete in sed(⅔)

Insert

Assume the content of file.txt:

Line 1: apple

Line 2: banana

Line 3: cherry

Insert "orange" before the first line **sed '1i orange' file.txt**

 # Result:

Line 1: orange

Line 2: apple

Line 3: banana

Line 4: cherry Here, `sed '1i orange' file.txt` inserts "orange" before the first line.

However it doesn't modify the file. To modify use : -i

Example 5: Append Insert Delete in sed(3/3)

Delete

Assume the content of file.txt:

Line 1: apple

Line 2: banana

Line 3: cherry

Delete the second line **sed '2d' file.txt**

 # Result:

Line 1: apple

Line 3: cherry

In this example, `sed '2d' file.txt` deletes the second line.

Example 3: Editing with Regular Expressions

In `**sed**`, regular expressions (regex) enhance pattern matching capabilities. Basic regex symbols include `.` (matches any character), `*` (matches zero or more occurrences), and `^` (matches the start of a line). Here's an example:

Assume the content of file.txt:

Line 1: apple

Line 2: banana

Line 3: cherry

Use a basic regex to match lines starting with "a"

sed -n '/^a/p' file.txt **-n suppresses auto printing || p instructs to print the matching patterns**

 # Result:

Line 1: apple

In this example, `**sed -n '/^a/p' file.txt**` uses a regex (`^a`) to match lines starting with "a" and prints only those lines (`-n` suppresses automatic printing). Understanding regular expressions in `sed` empowers users to create intricate patterns for more sophisticated text manipulations.

Example 3: Usage in config management

1. Let's have a file with Database config
2. Use sed to modify the file content
3. sed -i 's/db_host="backendhost"/db_host="localhost"/' db.conf

```
user@InfinityEngine MINGW64 ~/sed-learning (master)
$ ls -lrt
total 5
-rw-r--r-- 1 user 197121 69 Jan 31 22:27 fruits.txt
-rw-r--r-- 1 user 197121 37 Jan 31 22:29 data.txt
-rw-r--r-- 1 user 197121 48 Jan 31 22:32 file.txt.bak
-rw-r--r-- 1 user 197121 83 Jan 31 22:47 file.txt
-rw-r--r-- 1 user 197121 70 Feb  4 11:57 db.conf
```

Exercise:

1. /etc/sudoers:
2. Append 1 new user: **dev**
3. **Append 1 more user: test**
4. **Update dev to Dev**
5. **Update test to Test**