SHREE UTTAR GUJARAT BCA COLLEGE

Unit 4: Advanced Text Processing Tools

4.1 Introduction to Regular Expressions (Basic and Extended)

4.2 Pattern Matching using grep, egrep, and fgrep

4.3 Stream Editing with sed (search, replace, line deletion, insertion)

4.1 Introduction to Regular Expressions (Basic and Extended)

What are Regular Expressions?

Regular Expressions Sequence of characters that defines a search pattern, it helps to validate a series of characters for matching. So, regular expression basically helps to describe the complex pattern in the text. We can use this to search a particular string, pattern, replace text, data.

Regular expressions are also called Regex or RegexP.

A sequence of characters that defines a search pattern.

What is the use of Regular Expressions?

1. Can be used to validate a form.
2. To extract a part of the file.
3. To clean up a string.
4. Used in database or command line as well.
5. And used for other multiple purposes and list is on and on

Basic Components of a Regex Pattern

At the heart of every regex pattern are literals (like letters and numbers), meta-characters (special characters with specific meanings), and operators (which dictate how matches are made).

Common Applications

Regex patterns are prevalent in tasks such as input validation, string parsing, and text replacement, among other operations in programming and text editing.

Some Basic Examples

If you want to match all strings that contain the word computer, the easiest way to achieve this is to simply write computer inside a regular expression. Using this method, your regular expression will match computer mouse , computer software my computer , and so on. However, it will not match COMPUTER since regular expressions are case sensitive by default.

4.2 Pattern Matching using grep, egrep, and fgrep

What is the difference between grep, egrep, and fgrep?

- **grep:** The basic grep command uses basic regular expressions (BRE) to perform text searches. It's case-sensitive and doesn't support some advanced regex features.
- **egrep:** egrep (or grep -E) uses extended regular expressions (ERE) and offers more advanced regex features. It's case-sensitive by default but can be made case-insensitive with the -i option.
- **fgrep:** fgrep (or grep -F) searches for fixed strings, treating the search pattern as literal text without interpreting it as a regular expression. It's suitable for plain text searches, and it doesn't support regex features.

grep

The grep command is one of the most useful tools in Linux and Unix systems. It is used to search for specific words, phrases, or patterns inside text files, and shows the matching lines on your screen.

Syntax of grep Command in Unix/Linux

The basic syntax of the **`grep`** command is as follows:

```
grep [options] pattern [files]
```

- [**options**]: These are command-line flags that modify the behavior of grep.
- [**pattern**]: This is the regular expression you want to search for.

- [**file**]: This is the name of the file(s) you want to search within. You can specify multiple files for simultaneous searching.

Example of grep Command in Linux
grep Command is useful when you need to quickly find certain keywords or phrases in logs or documents.

Example : Search for a word in a file
If you have a file called notes.txt and you want to find all lines containing the word Python, you can use:

```
grep "hello" file1grep .txt
```

Options in grep Command in Linux
Its name comes from an old editor command: *g/re/p*, which stands for "globally search for a regular expression and print. Here are some use cases of grep command:

1. Case insensitive search
The -i option enables to search for a string case insensitively in the given file. It matches the words like "UNIX", "Unix", "unix".

```
grep -i "Word" file1.txt
```

2. Displaying the Count Matches Using grep
We can find the number of lines that matches the given string/pattern

```
grep -c "word" file1.txt
```

3. Display the Matching Filenames Using grep
We can just display the files that contains the given string/pattern.

```
grep -l "unix" *
```

or

```
grep -l "unix" file1.txt file2.txt file3.xt file4.txt
```

4. Checking Whole Words Using grep
By default, grep matches the given string/pattern even if it is found as a substring in a file. The -w option to grep makes it match only the whole words.

```
grep -w "unix" geekfile.txt
```

5. Display Matched Pattern Using grep

By default, grep displays the entire line which has the matched string. We can make the grep to display only the matched string by using the -o option.

```
grep -o "unix" geekfile.txt
```

6. Show Line Numbers with grep -n

To show the line number of file with the line matched.

```
grep -n "unix" geekfile.txt
```

7. Inverting the Pattern Match Using grep

You can display the lines that are not matched with the specified search string pattern using the -v option.

```
grep -v "unix" geekfile.txt
```

8. Match Lines Starting with a String using grep

The ^ regular expression pattern specifies the start of a line. This can be used in grep to match the lines which start with the given string or pattern.

```
grep "^unix" geekfile.txt
```

9. Match Lines Ending with a String using grep

The $ regular expression pattern specifies the end of a line. This can be used in grep to match the lines which end with the given string or pattern.

```
grep "os$" geekfile.txt
```

10. Specifies expression with -e option

Can use multiple times :

```
grep –e "Agarwal" –e "Aggarwal" –e "Agrawal" geekfile.txt
```

11. Use -f to Read Patterns from a File

It is used to search for multiple patterns, listed in a separate file, within another target file.

```
cat pattern.txt
```

Agarwal
Aggarwal
Agrawal

```
grep –f pattern.txt  geekfile.txt
```

12. Print Specific Lines from a File with grep

-A prints the searched line and n lines after the result, -B prints the searched line and n lines before the result, and -C prints the searched line and n lines after and before the result.

Syntax:

```
grep -A[NumberOfLines(n)] [search] [file]
grep -B[NumberOfLines(n)] [search] [file]
grep -C[NumberOfLines(n)] [search] [file]
```

Example:

```
grep -A1 learn geekfile.txt
```

egrep

What is egrep in Linux?

egrep is a command-line utility that searches text using extended regular expressions. It is an enhanced version of the grep command, providing additional features and flexibility for matching patterns in text. Regular expressions are sequences of characters that define search patterns, making them powerful tools for text manipulation.

Basic Syntax of egrep in Linux

The basic syntax of the egrep command is:

```
egrep [options] pattern [file(s)]
```

- **options:** These are optional flags that modify egrep's behavior.
- **pattern:** This is the regular expression pattern you want to search for.
- **file(s):** These are the optional files you want to search within. If omitted, egrep will read from standard input.

Common Options

egrep provides a wide array of options to fine-tune its behavior. Here are some of the most common ones:

- **-i (ignore case):** This flag makes the pattern matching case-insensitive.
- **-v (invert match):** It returns lines that do not match the pattern.
- **-r (recursive):** Searches directories and their subdirectories.
- **-l (list filenames):** Displays only the names of files that contain matching lines.
- **-o (only matching):** Prints only the part of the line that matches the pattern.
- **-c (count):** Shows the count of matching lines.
- **-n (line numbers):** Displays line numbers along with matching lines.
- **-E (extended regular expressions):** This option allows you to use extended regular expressions. It is not required since egrep always uses extended regular expressions, but it's good practice to include it for clarity.

Examples of egrep Command in Linux:

You can use egrep to search for lines containing the word "Linux":

```
egrep 'Linux' sample.txt
```

Example 2: Case-Insensitive Search
To perform a case-insensitive search for the word "linux," use the -i option:

```
egrep -i 'linux' sample.txt
```

Example 3: Invert Match
To find lines that do not contain the word "Linux," use the -v option:

```
egrep -v 'Linux' sample.txt
```

Example 4: Searching Multiple Files
You can search for a pattern in multiple files simultaneously. Let's say you have two files, file1.txt and file2.txt, and you want to find lines containing the word "example" in both files:

```
egrep 'example' file1.txt file2.txt
```

fgrep

What is the 'fgrep' Command?

The '**fgrep**' command is part of the grep family, designed to search for fixed strings rather than patterns defined by regular expressions. It's particularly helpful when dealing with strings that include characters typically reserved for regular expressions, as '**fgrep**' treats these characters as literals. The command can be used to search through one or multiple files, providing results that match the specified string exactly as given.

Syntax:

```
fgrep [options] [ -e pattern_list] [pattern] [file]
```

where,

- '**[options]**': Command-line options that modify the behavior of fgrep.
- '**-e pattern_list**': A list of patterns to search for.
- '**[pattern]**': The string to be searched in the file(s).
- '**[file]**': The file or files in which to search for the pattern. If no file is specified, fgrep reads from standard input.

Common Options of the 'fgrep' Command

Below are the commonly used options for the '**fgrep**' command along with their descriptions and examples:

Option	Description
-c	Prints only a count of the lines that contain the pattern.
-h	Displays the matched lines without prefixing them with the filename.
-i	Ignores case distinctions during the search, making the search case-insensitive.
-l	Lists only the names of files with matching lines, each listed once.
-n	Prefixes each matched line with its line number in the file.

Option	Description
-s	Suppresses error messages about nonexistent or unreadable files.
-v	Inverts the match, displaying lines that do not contain the pattern.
-x	Prints only lines that match the pattern exactly, without partial matches.
-e	Allows specifying multiple patterns. This is useful when the pattern begins with a dash (-).
-f	Takes patterns from a file.
--help	Displays help information about the fgrep command.
file	A path name of a file to be searched for the patterns. If no file operands are specified, the standard input will be used.

‘fgrep' Command Examples in Linux

1. '-c' option:
Displaying the count of number of matches. We can find the number of lines that match the given string.
Example:

```
$fgrep -c "usin.g" para
```

2. '-h' option
To display the matched lines.
Example:

```
fgrep -h "usin.g" para
```

3. '-i' option
Used in case insensitive search. It ignore upper/lower case distinction during comparisons. It matches words like: "geeks*forgeeks", "Geeks*forgeeks".
Example:

```
fgrep -i "geeks*forgeeks" para
```

4. '-l' option
It will display the file names that match the pattern. We can just display the files that contains the given string/pattern.
Example:

```
fgrep -l "geeks*forgeeks" para para2
```

5. '-n' option
Precede each line by its line number in the file. It shows line number of file with the line matched.
Example:

```
$ fgrep -n "learni\ng" para
```

6. '-v' option
It is used to display all lines except those that contain the pattern. It will print all lines except those that contain the pattern.
Example:

```
fgrep -v "@re" para
```

7. '-x' option
It will display only lines matched entirely.
Example 1:

```
fgrep -x "@re" para
```

4.3 Stream Editing with sed (search, replace, line deletion, insertion)

What is the Sed Command

As we mentioned, SED or **Stream Editor** is used for both simple and complex text transformations, such as **finding and replacing**, **inserting,** or **deleting** text. It's commonly used in shell scripts to automate file editing tasks.

Sed Command Syntax:

The **basic syntax** for using the **SED command** in Linux is:

```
sed [OPTIONS] 'COMMAND' [INPUTFILE...]
```

where,

- **'OPTIONS':** These are optional flags that modify the behavior of the sed command.
- **'COMMAND'**: This defines the command or sequence of commands to execute on the input file.
- **'INPUTFILE'**: One or more input files to be processed.

Commonly Used SED Command Options

Below are some of the most frequently used **SED command options,** let's check them out:

Option	Description
-i	Edit the file in-place (overwrite)
-n	Suppress automatic printing of lines.
-e	Allows multiple commands.
-f	Reads sed commands from a file.
-r	Use extended regular expressions.

Practical Examples of SED Command Usage

Here are some basic **SED commands** that will help you get started with text manipulation.

Consider the below text file as an input.

```
cat > geekfile.txt
unix is great os. unix is opensource. unix is free os.
learn operating system.
unix linux which one you choose.
unix is easy to learn.unix is a multiuser os.Learn unix .unix is a powerful.
```

1. Sample Commands

Replacing or substituting string: Sed command is mostly used to replace the text in a file. The below simple sed command replaces the word "unix" with "linux" in the file.

```
sed 's/unix/linux/' geekfile.txt
```

2. Replacing the nth Occurrence of a Pattern in a Line

To replace only the **nth occurance** of a word in a line, use the following syntax:

```
 sed 's/old_word/new_word/n' filename

sed 's/unix/linux/2' geekfile.txt
```

3. Replacing all the Occurrence of the Pattern in a Line

Here, we will use the **g** flag to replace **all the occurances** of a pattern in a line. Let's check out the syntax below:

```
sed 's/old_word/new_word/g' filename
```

The substitute flag /g (global replacement) specifies the sed command to replace all the occurrences of the string in the line.

```
sed 's/unix/linux/g' geekfile.txt
```

4. Replacing from nth Occurrence to all Occurrences in a Line

Use the combination of /1, /2 etc and /g to replace all the patterns from the nth occurrence of a pattern in a line. The following sed command replaces the third, fourth, fifth... "**unix**" word with "**linux**" word in a line.

```
sed 's/unix/linux/3g' geekfile.txt
```

5. Parenthesize First Character of Each Word

This sed example prints the first character of every word in parenthesis.

```
echo "Welcome To The Geek Stuff" | sed 's/\(\b[A-Z]\)/\(\1\)/g'
```

6. Replacing String on a Specific Line Number

You can restrict the sed command to replace the string on a specific line number. An example is

```
sed '3 s/unix/linux/' geekfile.txt
```

7. Duplicating the Replaced Line with /p flag

The **/p** print flag prints the replaced line twice on the terminal. If a line does not have the search pattern and is not replaced, then the **/p** prints that line only once.

```
sed 's/unix/linux/p' geekfile.txt
```

8. Printing Only the Replaced Lines

Use the -n option along with the /p print flag to display only the replaced lines. Here the **-n** option suppresses the duplicate rows generated by the /p flag and prints the replaced lines only one time.

```
sed -n 's/unix/linux/p' geekfile.txt
```

9. Replacing String on a Range of Lines

You can specify a range of line numbers to the sed command for replacing a string.

```
sed '1,3 s/unix/linux/' geekfile.txt
```

10. Deleting Lines from a Particular File

SED command can also be used for deleting lines from a particular file. SED command is used for performing deletion operation without even opening the file

Examples:

1. To Delete a particular line say n in this example

Syntax:

sed 'nd' filename.txt

Example:

sed '5d' filename.txt

2. To Delete a last line

Syntax:

sed '$d' filename.txt

3. To Delete line from range x to y

Syntax:

sed 'x,yd' filename.txt

Example:

sed '3,6d' filename.txt

4. To Delete from nth to last line

Syntax:

sed 'nth,$d' filename.txt

Example:

sed '12,$d' filename.txt

5. To Delete pattern matching line

Syntax:

sed '/pattern/d' filename.txt

Example:

sed '/abc/d' filename.txt

Advanced SED Command Examples

Below are some of the advanced SED commands that can be used for handling various editing tasks. let's check them out:

1. Regular Expressions

SED supports regular expressions that allows it to **handle more complex pattern matching**. To **enable regular expressions**, you need to use **-r** option.

Example:

Here, we have matched any word start with '**u**' and replacing it with "**Linux".** Let's check this out in the provided syntax below:

```
sed -r 's/\bu\w+/Linux/g' geekfile.txt
```

2. Insert Text

You can use this option to insert any text before or after any specific line. Here's the syntax provided below:

```
sed '3i\new text' filename  # Insert text before line 3
sed '3a\new text' filename  # Insert text after line 3
```

Example: To insert a new line before line2 in **geekfile.txt**

```
sed '2i\This is a new line' geekfile.txt
```