**Department of Computer Science and Engineering**

|  |  |
| --- | --- |
| **Course Code:CSE321** | **Credits: 1.5** |
| **Course Name: Operating Systems** | **Semester: Sum’18** |

**Lab 03**

**Advanced Shell Programming**

1. **Topic overview:**

Students will be introduced with the Advanced Linux shell command to manipulate the content of a file. Use of command ***grep, sed,awk*** etc will be introduced.

**2. Lesson Fit:** Knowledge of commands of shell programming in linux is required.

**3. Learning outcome:**

After this lab students will be able to:

* insert , append, update content of a file
* Perform basic operation on the content of file,

**4.** **Anticipated Challenges and Possible Solutions :**

Students might face problems to use different regular expression.

**Solutions:** Before starting the basics of the advanced commands, Instructor has to discuss about regular expressions.

**5.**  **Acceptance and Evaluation**

Students will show their progress as they complete each task. They will be

marked according to their lab performance.

**6. Activity details:**

**First hour: i.** Discussion on ***grep*** command

ii. discuss on **regular expression.**

iii. Show different option of command ***grep***.

**Grep command**

## **About grep**

grep, which stands for "global regular expression print," processes text line by line and prints any lines which match a specified pattern.

## **grep syntax**

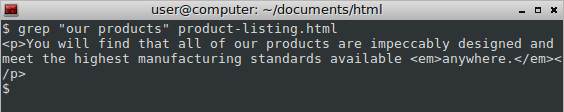
**grep [*OPTIONS*] *PATTERN* [*FILE*...]**

## Overview

Grep is a powerful tool for matching a [regular expression](https://www.computerhope.com/jargon/r/regex.htm) against text in a file, multiple files, or a stream of input. It searches for the *PATTERN* of text that you specify on the command line, and outputs the results for you.

## Example Usage

Let's say want to quickly locate the phrase "our products" in HTML files on your machine. Let's start by searching a single file. Here, our *PATTERN* is "our products" and our *FILE* is product-listing.html.

****

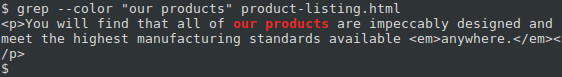
A single line was found containing our pattern, and grep outputs the entire matching line to the terminal. The line is longer than our terminal width so the text wraps around to the following lines, but this output corresponds to exactly one line in our *FILE*.

Note: The *PATTERN* is interpreted by grep as a regular expression. In the above example, all the [characters](https://www.computerhope.com/jargon/c/charact.htm) we used (letters and a space) are interpreted [literally](https://www.computerhope.com/jargon/l/literal.htm) in regular expressions, so only the exact

phrase will be matched. Other characters have special meanings, however — some punctuation marks.

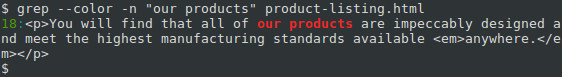
**Viewing grep output in color**

If we use the --color option, our successful matches will be highlighted for us:

****

**Viewing line numbers of successful matches**

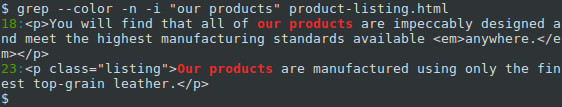
It will be even more useful if we know where the matching line appears in our file. If we specify the -n option, grep will prefix each matching line with the line number:



Our matching line is prefixed with "18:" which tells us this corresponds to line 18 in our file.

**Performing case-insensitive grep searches**

What if "our products" appears at the beginning of a sentence, or appears in all uppercase? We can specify the -i option to perform a *case-insensitive* match:

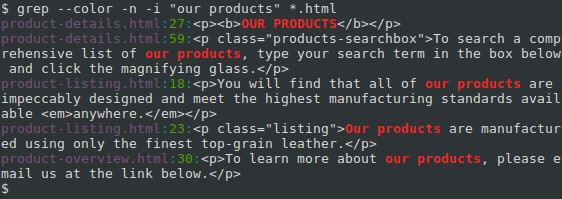
****

**Using the -i option, grep finds a match on line 23 as well.**

**Searching multiple files using a wildcard**

If we have multiple files to search, we can search them all using a [wildcard](https://www.computerhope.com/jargon/w/wildcard.htm) in our *FILE*name. Instead of specifying product-listing.html, we can use an asterisk ("\*") and the .html

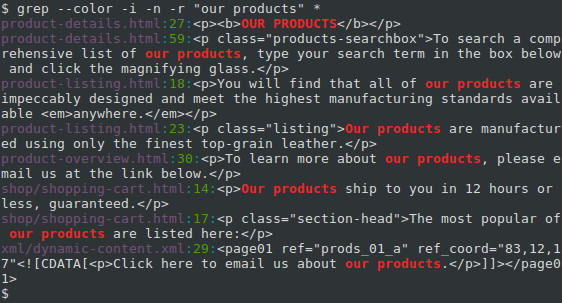
extension. When the command is executed, the [shell](https://www.computerhope.com/jargon/s/shell.htm) will expand the asterisk to the name of any file it finds (within the current directory) which ends in ".html".

****

Notice that each line starts with the specific file where that match occurs.

**Recursively searching subdirectories**

We can extend our search to [subdirectories](https://www.computerhope.com/jargon/s/subdirec.htm) and any files they contain using the -roption, which tells grep to perform its search [recursively](https://www.computerhope.com/jargon/r/recursive.htm). Let's change our *FILE* name to just an asterisk ("\*"), so that it will match any file or directory name, and not just HTML files:

****

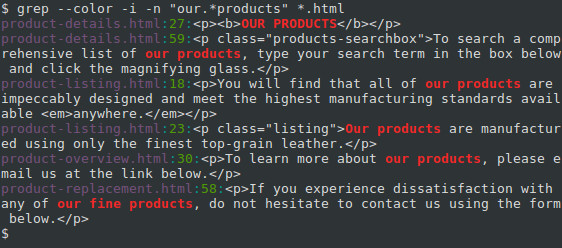
This gives us three additional matches. Notice that the directory name is included for any matching files that are not in the current directory.

**Using regular expressions to perform more powerful searches**

The true power of grep is that it can be used to match regular expressions. (That's what the "re" in "grep" stands for). Regular expressions use special characters in the *PATTERN*string to match a wider array of strings. Let's look at a simple example.

Let's say you want to find every occurrence of a phrase *similar* to "our products" in your HTML files, but the phrase should always start with "our" and end with "products". We can specify this *PATTERN* instead: "our.\*products".

In regular expressions, the period (".") is interpreted as a single-character wildcard. It means "any character that appears in this place will match." The asterisk ("\*") means "the preceding character, appearing zero or more times, will match." So the combination ".\*" will match *any number* of *any* character. For instance, "our amazing products", "ours, the best-ever products", and even "ourproducts" will match. And because we're specifying the -i option, "OUR PRODUCTS" and "OuRpRoDuCtS will match as well. Let's run the command with this regular expression, and see what additional matches we can get:

****

Here, we also got a match from the phrase "our fine products".

Grep is a powerful tool that can help to work with text files, and it gets even more powerful when you become comfortable using regular expressions.

## Technical Description

grep searches the named input *FILE*s (or [standard input](https://www.computerhope.com/jargon/s/stdin.htm) if no files are named, or if a single dash ("-") is given as the file name) for lines containing a match to the given *PATTERN*. By default, grep prints the matching lines.

Also, three variant programs [egrep](https://www.computerhope.com/unix/uegrep.htm), [fgrep](https://www.computerhope.com/unix/ufgrep.htm) and rgrep are available:

* **egrep** is the same as running grep -E. In this mode, grep evaluates your *PATTERN*string as an extended regular expression (ERE). Nowadays, ERE does not "extend" very far beyond basic regular expressions, but they can still be very useful.
* **fgrep** is the same as running grep -F. In this mode, grep evaluates your *PATTERN*string as a "fixed string" — every character in your string is treated literally. For example, if your string contains an asterisk ("\*"), grep will try to match it with an actual asterisk rather than interpreting this as a [wildcard](https://www.computerhope.com/jargon/w/wildcard.htm). If your string contains multiple lines (if it contains newlines), each line will be considered a fixed string, and any of them can trigger a match.
* **rgrep** is the same as running grep -r. In this mode, grep will perform its search [recursively](https://www.computerhope.com/jargon/r/recursive.htm). If it encounters a directory, it will traverse into that directory and continue searching. ([Symbolic links](https://www.computerhope.com/jargon/s/symblink.htm) are ignored; if you want to search directories that are symbolically linked, you should use the -R option instead).

In older operating systems, egrep, fgrep and rgrep were distinct programs with their own executables. In modern systems, these special command names are shortcuts to grep with the appropriate flags enabled. They are functionally equivalent.

**Practice problems: Task 01-05**

**Hour 2: discussion on sed command**

**Activities :**  1.Discussion on ***sed*** command.

2. Show different option of command ***sed***.

sed is a *stream editor*. A stream editor is used to perform basic text transformations on an input stream (a file, or input from a [pipeline](https://www.computerhope.com/jargon/p/pipe.htm)). While in some ways similar to an editor which permits [scripted](https://www.computerhope.com/jargon/s/script.htm) edits (such as [ed](https://www.computerhope.com/unix/ued.htm)), sed works by making only one pass over the input(s), and is consequently more efficient. But it is sed's ability to filter text in a pipeline which particularly distinguishes it from other types of editors.

#### **Example :1) Displaying partial text of a file**

With sed, we can view only some part of a file rather than seeing whole file. To see some lines of the file, use the following command,

**[linuxtechi@localhost ~]$ sed -n 22,29p testfile.txt**

here, option ‘n’ suppresses printing of whole file & option ‘p’ will print only line lines from 22 to 29.

#### **Example :2) Display all except some lines**

To display all content of a file except for some portion, use the following command,

**[linuxtechi@localhost ~]$ sed 22,29d testfile.txt**

Option ‘d’ will remove the mentioned lines from output.

#### **Example :3) Display every 3rd line starting with Nth line**

Do display content of every 3rd line starting with line number 2 or any other line, use the following command

**[linuxtechi@localhost ~]$ sed -n '2-3p' file.txt**

#### **Example :4 ) Deleting a line using sed command**

To delete a line with sed from a file, use the following command,

**[linuxtechi@localhost ~]$ sed Nd testfile.txt**

where ‘N’ is the line number & option ‘d’ will delete the mentioned line number. To delete the last line of the file, use

**[linuxtechi@localhost ~]$ sed $d testfile.txt**

#### 

#### **Example :5) Deleting a range of lines**

To delete a range of lines from the file, run

**[linuxtechi@localhost ~]$ sed '29-34d' testfile.txt**

This will delete lines 29 to 34 from testfile.txt file.

#### **Example :6) Deleting lines other than the mentioned**

To delete lines other than the mentioned lines from a file, we will use ‘!’

**[linuxtechi@localhost ~]$ sed '29-34!d' testfile.txt**

here ‘!’ option is used as not, so it will reverse the condition i.e. will not delete the lines mentioned. All the lines other 29-34 will be deleted from the files testfile.txt.

#### **Example :7) Adding Blank lines/spaces**

To add a blank line after every non-blank line, we will use option ‘G’,

**[linuxtechi@localhost ~]$ sed G testfile.txt**

#### **Example :8) Search and Replacing a string using sed**

To search & replace a string from the file, we will use the following example,

**[linuxtechi@localhost ~]$ sed 's/danger/safety/' testfile.txt**

here option ‘s’ will search for word ‘danger’ & replace it with ‘safety’ on every line for the first occurrence only.

#### **Example :9) Search and replace a string from whole file using sed**

To replace the word completely from the file, we will use option ‘g’ with ‘s’,

**[linuxtechi@localhost ~]$ sed 's/danger/safety/g' testfile.txt**

#### **Example :10) Replace the nth occurrence of string pattern**

We can also substitute a string on nth occurrence from a file. Like replace ‘danger’ with ‘safety’ only on second occurrence,

[linuxtechi@localhost ~]$ sed ‘s/danger/safety/2’ testfile.txt

To replace ‘danger’ on 2nd occurrence of every line from whole file, use

**[linuxtechi@localhost ~]$ sed 's/danger/safety/2g' testfile.txt**

#### Example :11) Replace a string on a particular line

To replace a string only from a particular line, use

**[linuxtechi@localhost ~]$ sed '4 s/danger/safety/' testfile.txt**

This will only substitute the string from 4th line of the file. We can also mention a range of lines instead of a single line,

**[linuxtechi@localhost ~]$ sed '4-9 s/danger/safety/' testfile.txt**

#### **Example :12) Add a line after/before the matched search**

To add a new line with some content after every pattern match, use option ‘a’ ,

**[linuxtechi@localhost ~]$ sed '/danger/a "This is new line with text after match"' testfile.txt**

To add a new line with some content a before every pattern match, use option ‘i’,

**[linuxtechi@localhost ~]$ sed '/danger/i "This is new line with text before match" ' testfile.txt**

#### **Example :13) Change a whole line with matched pattern**

To change a whole line to a new line when a search pattern matches we need to use option ‘c’ with sed,

**[linuxtechi@localhost ~]$ sed '/danger/c "This will be the new line" ' testfile.txt**

So when the pattern matches ‘danger’, whole line will be changed to the mentioned line.

Tasks: TASK 1-4 of sed sections.

**Third hour:** solving rest of the tasks of LABTASKS.

**Lab tasks**

**GREP**

Task 1: Search for the given string in a single file

Task 2:Checking for the given string in multiple files.

Task 3: Case insensitive search using grep -i

Task 4: Match regular expression in files

Task 5:Checking for full words, not for sub-strings using grep -w

Task 6: Displaying lines before/after/around the match using grep -A, -B and -C

Task 7: Display N lines after match

Task 8: Display N lines before match

Task 9: Highlighting the search using GREP\_OPTIONS

Task 10: Invert match using grep -v

Task 11: Counting the number of matches using grep -c

Task 12: Show the position of match in the line

**SED**

Task 1: Replace every occurrence of Nick with John in report.txt

Task 2: Replace every occurrence of Nick or nick with John.

Task 3: Add 8 spaces to the left of a text for pretty printing.

Task 4: Display only one paragraph, starting with "Of course"

Task 5: Show only lines 12-18 of file.txt

Task 6: Delete the last line

Task 7: Replace foo with bar only for the first instance in a line.