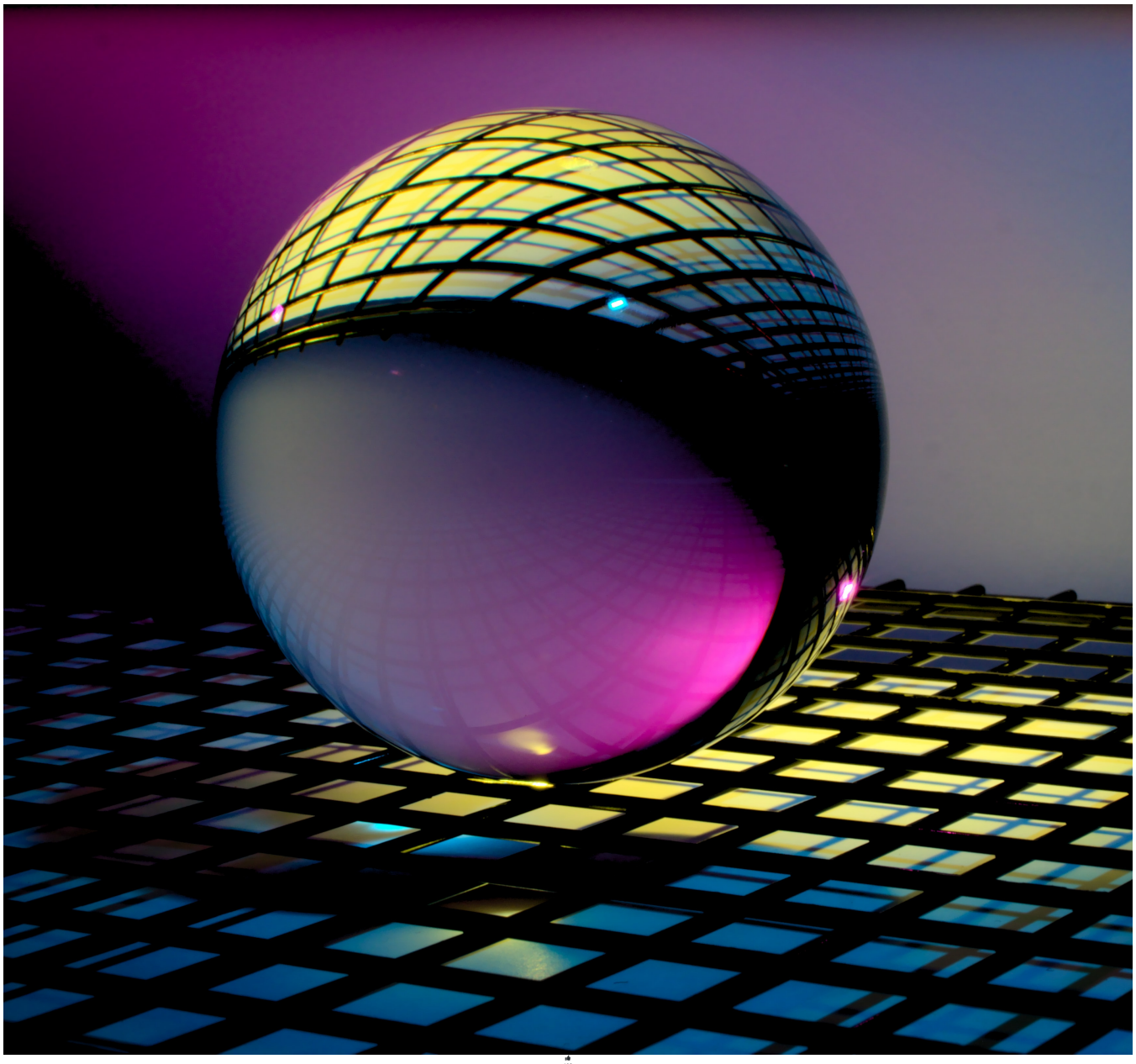
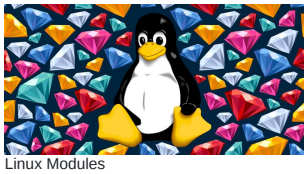


103



Linux Modules

Learn linux modules in a fun way

Start AttackBox Awards Help Options

Chart Scoreboard Discuss Writeups More

Difficulty:

Active Machine Information

Loading... Loading... Loading... Loading...

100%

Task 1 Let's Introduce

Task 2 Mountain du

Task 3 Owl

Task 4 Did someone said STROPS?

Task 5 Shall I try morsecode?

Task 6 AWWKAY

Task 7 That's what she sed

Sorry for the pun. But this has been on my mind since I started creating this room. Nvm, so sed. is THE. 2nd most powerful tool of all. I especially consider using sed most of the time, because that's what she sed... jk. it's because it offers good number of strops in short commands. Easy to use, once get a habit of it.

Download Task Files

The sed life

sed(Stream EDitor) is a tool that can perform a number of string operations. Listing a few, could be: FIND AND REPLACE, searching, insertion, deletion. I think sed of a stream-oriented vi editor... Ok so a few questions popped up, like how? and what is stream-oriented? Let's not dive deep into streams, just keep in mind that I said it in contrast with "orientation with input stream". You can't call vi stream oriented, because it doesn't work with neither of input or output stream.

On the other hand, you can easily perform operations with sed command by either piping the input or redirecting(<) the input from a file. I compared it with vi, because while learning this command I kinda missed using vim, since I shifted to sublime(No offence to vim fanboys/fangirls out there, I just use sublime to keep things common between my windows and linux machine and it's a great lightweight editor, so uk).

Syntax: `sed [flags] [pattern/script] [input file]`

Important Flags

Flags	Description
-e	To add a script/command that needs to executed with the pattern/script(on searching for pattern)
-f	Specify the file containing string pattern
-E	Use extended regular expressions
-n	Suppress the automatic printing or pattern spacing

The sed command

There are endless ways of using sed. I am gonna walk you through a very detailed general syntax of (mostly all) sed patterns, with some general examples. Rest is your thinking and creativity, on how YOU utilize this tool.

'[condition(s)(optional)] [mode(s)(optional)/][source/to-be-searched pattern(mandatory)]/[to-be-replaced pattern(depends on mode you use; if no mode then by default it is optional)]/[args/flags to operate on the pattern searched(optional)]'

Hope these colors could have helped you identify the parts. If you have any previous knowledge of sed, feel free to co-relate. Again, this is just the pattern inside sed command (excluding external flags). Also, note the single quotes at the start/end.

Hmm, but may be, it's still not clear. Alright let's take a simple example to relate this.

sed -e '1,3 s/john/JOHN/g' file.txt

Let's not care about what's meaning of 1,3 all that slashes, that s,g. And focus on the color codes. Hope the syntax is now making a ∞ sense... Great. Moving forward to flags.

The purple gang

Modes	Description
s	(Most used)Substitute mode (find and replace mode)
y	Works same as substitution; the only difference is, it works on individual bytes in the string provided(this mode takes no arguments/conditions)

The green gang

Flags/Args	Description
/g	globally(any pattern change will be affected globally, i.e. throughout the text; generally works with s mode)
/i	To make the pattern search case-insensitive(can be combined with other flags)
/d	To delete the pattern found(Deletes the whole line; takes no conditions/modes/to-be-replaced string)
/p	prints the matching pattern(a duplicate will occur in output if not suppressed with -n flag.)
/1,/2,/3.../n	To perform an operation on an nth occurrence in a line(works with s mode)

Let's see these in action... Explaining the previously taken command, (sed -e '1,3 s/john/JOHN/g' file.txt)

- Starting with the sed keyword itself, initializes the sed command.
- With -e flag specifying that following is a script command.(you don't need to specify -e if it's a single command; as it will be automatically interpreted by sed as a positional argument)
- Then comes the pattern. Starting with the yellow portion is the condition(which is/are btw, generally on lines inside a file), specifying to take range of lines 1,3 (line index starts from 1) and execute the following code on that range of lines. Following a space comes the mode, specifying that we need to use a substitution mode(as we are substituting a value) by using s. Then we specify / as a delimiter to differentiate between the parts of code. After the first slash came the pattern we want to operate the substitution on(you may choose to use regex in this region too). Following the 2nd slash comes the string we want to replace the pattern with. Finally, after the last slash was an arg/flag, /g specifying to operate this operation globally, wherever the pattern was found.
- Finally was the filename we want to take input from and apply operation/code that we specified beside it.

Hope there is no confusion as per sed is concerned. Hence, the output for the above command would be like:

```
pjroxy@kali ~/Desktop$ sed -e '1,3 s/john/JOHN/g' file.txt
tpsec youtube hackthebox 34024
JOHN youtube ctf 50024
thecybermentor tcmsec courses 25923
tlveoverflow youtube ctf 45345
nahansec hackerone bughunting 12365
stok hackerone bughunting 1234
```

Let's view a few more examples to get the concept clear:

- Viewing a range of Lines

```
pjroxy@kali ~/Desktop$ sed -n '3,5p' file.txt
thecybermentor tcmsec courses 25923
tlveoverflow youtube ctf 45345
nahansec hackerone bughunting 12365
```

-n flag suppressed the output and we got the duplicates created by p arg.

- Viewing the entire file except a given range

```
pjroxy@kali ~/Desktop$ sed '3,5d' file.txt
tpsec youtube hackthebox 34024
john youtube ctf 50024
stok hackerone bughunting 1234
```

- Viewing multiple ranges of lines inside a file

```
pjroxy@kali ~/Desktop$ sed -n -e '1,2p' -e '4,5p' file.txt
tpsec youtube hackthebox 34024
john youtube ctf 50024
tlveoverflow youtube ctf 45345
nahansec hackerone bughunting 12365
```

- To start searching from nth pattern occurrence in a line you can use combination of /g with /1,/2,/3.

```
pjroxy@kali ~/Desktop$ sed 's/youtube/YOUTUBE/1' file.txt
tpsec YOUTUBE hackthebox 34024
john YOUTUBE ctf 50024
thecybermentor tcmsec courses 25923
tlveoverflow YOUTUBE ctf 45345
nahansec hackerone bughunting 12365
stok hackerone bughunting 1234
pjroxy@kali ~/Desktop$ sed 's/youtube/YOUTUBE/2' file.txt
tpsec youtube hackthebox 34024
john youtube ctf 50024
thecybermentor tcmsec courses 25923
tlveoverflow youtube ctf 45345
nahansec hackerone bughunting 12365
stok hackerone bughunting 1234
pjroxy@kali ~/Desktop$
```

You can see when I specified /1 it gave a change in the text, with /2 it didn't. This is because there was only 1 occurrence of the string "youtube", and the 2nd occurrence couldn't be found. Also I didn't used 1g or 2g because there were no further occurrences of the pattern, so there is no need to use it. Still it would have worked the same, if used. Try it on your own.

- If you have log files to view which have trailing white spaces, and it is hard to read them, then you can fix that using regex.

```
pjroxy@kali ~/Desktop$ cat file1.txt
tpsec        youtube hackthebox 34024
john         youtube        ctf     50024
thecybermentor    tcmsec        courses     25923
tlveoverflow            youtube ctf     45345
nahansec            hackerone bughunting        12365
stok        hackerone        bughunting 1234
pjroxy@kali ~/Desktop$ sed 's/ *$//g' file1.txt
tpsec youtube hackthebox 34024
john youtube ctf 50024
thecybermentor tcmsec courses 25923
tlveoverflow youtube ctf 45345
nahansec hackerone bughunting 12365
stok hackerone bughunting 1234
```

Let's take one last example on this sed command.

- More on regex can be: Making every line to start with a bullet point and enclose the digits in square brackets... Ok, but how? Let's first view it, and then we'll take a look at the explanation.

```
pjroxy@kali ~/Desktop$ cat file.txt
tpsec youtube hackthebox 34024
john youtube ctf 50024
thecybermentor tcmsec courses 25923
tlveoverflow youtube ctf 45345
nahansec hackerone bughunting 12365
stok hackerone bughunting 1234
pjroxy@kali ~/Desktop$ sed 's/\(^\b[[:alpha:]]*\)\([[:digit:]]*\)/\=\> \1\[\2\]/g' file.txt
=> tpsec youtube hackthebox [34024]
=> john youtube ctf [50024]
=> thecybermentor tcmsec courses [25923]
=> tlveoverflow youtube ctf [45345]
=> nahansec hackerone bughunting [12365]
=> stok hackerone bughunting [1234]
pjroxy@kali ~/Desktop$
```

"What is this?! where's that :alpha: came from? I understand the \b is part of regex you used, following those, un-identifiable escape characters and some \1 and \2 referenced either wrong, as it's /1, /2 to identify the nth occurrence. I mean, it's so confusing".

I agree, it's so noisy, and hard to read. But believe 70% of it is nothing to do with `sed` , it's all regex, so take your time. Try to understand what the regex is doing. Rest the "to-be-replaced" part is just a way sed is assigning the groups to it's default variables we created within regex.

Explanation

sed 's/\(^\b[[:alpha:]]*\)\([[:digit:]]*\)/\=\> \1\[\2\]/g' file.txt

Hope things are a little clear, by undertaking the knowledge of previous color coding. You can easily differentiate the mode, pattern, to-be-replaced string. *Later I removed the background from the part belonging to the keywords in sed. As there was some issue in changing the font color, it just won't persist.*

- Starting with the regex part. Opening a group with escape character, ^ to put the cursor at the starting of the line, and then \b represents to search for beginning of a word, and then defines a set of characters to include, following a "*" to specify 'n' number of characters. Then closes the group by escaping the closing brackets. Creating another regex group, using escape sequence, we then initialized another set and specified * at the end of the set to take n characters of that set, at last group is closed using escape sequence.
- At the replaced end, we are using escape sequences to make a bullet(it's just a good practice to use escape sequence with every symbolic character; even if the output is same), then we have escape characters for the square brackets enclosing a sed variable /2 (after /1 which is coming up).
- Now its turn for the sed's keyword part. We used [:alpha:] in the set defined by regex, which is nothing but another representation of using `a-zA-Z` in regex, which means to capture any alphabetic characters. sed offers such keywords(calling them "bracket expressions"), which we can use to make the input code look cleaner. Similarly we used the bracket expression for specifying digit as well which we specified using [:digit:].

Note: There's a space after the first bracket expression inside the regex set ([[:alpha:]]space[]). As you see, this space was to indicate the regex set *so that * could take multiple words until the digits start occurring in the text(regex logic).*

- Then there are some in-built variables as we saw in ~~awk~~ awk, that we used in the to-be-replace part of sed. \1 depicted the first group which selected everything until the first character occurred. The second group comprised of a set consisting decimal characters, which were enclosed with [\2] with the use of escseq.

Here, we finished learning about sed variables, the number of groups you create with regex, can be later indexed as variable \n in sed.

Well this is pretty much it, on the sed command. If you want to learn more, check-out the resources on the sed command.

Resources:

- Sed Command in Linux/Unix with examples - GeeksforGeeks
- sed, a stream editor (gnu.org) (Official Documentation)
- 15 Useful 'sed' Command Tips and Tricks for Daily Linux System Administration Tasks (tecmint.com)

Again, if there is not much you could learn about this command don't feel bad, just go through the resources and try practicing by making your own texts and play with this command.

How would you substitute every 3rd occurrence of the word 'hack' to 'back' on every line inside the file file.txt?

sed 's/hack/back/3g' file.txt

Correct Answer

Hint

How will you do the same operation only on 3rd and 4th line in file.txt?

sed '3,4 s/hack/back/3g' file.txt — Correct Answer — Hint

Download the given file, and try formatting the trailing spaces in sed1.txt with a colon(:).

sed 's/ *$/:/g' sed1.txt — Correct Answer

View the sed2 file in the directory. Try putting all alphabetical values together, to get the answer for this question.

congratulations you made it through this small little challenge — Correct Answer — Hint

What pattern did you use to reach that answer string?

's/[[:digit:]]//g' — Correct Answer — Hint

Alternatively, you can use tr to remove all the digits, and then pipe the output in sed to remove trailing whitespaces.

cat sed2.txt | tr '[:digit:]' ' ' | sed 's/ *$//g'

No answer needed — Correct Answer

What did she sed?(In double quotes)

"That's What" — Correct Answer — Hint

Task 8 My x-args a lot

Task 9 There's always a special mention... Isn't it?

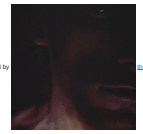
Task 10 cURL or cRAWl

Task 11 WebGoat or webGet

Task 12 ROFL

Task 13 The Last of Us

Task 14 Is it night yet?

Created by  thegrigs

This is a **free** room, which means anyone can deploy virtual machines in the room (without being subscribed)! 1228 users are in here and this room is 29 days old.