Birzeit University - Faculty of Engineering & Technology
Electrical & Computer Engineering Department - ENCS313
Linux laboratory

Experiment #3
Regular Expressions under Linux

0.1 Introduction

The experiment intends to make students familiar with using *regular expressions* embedded within the basic Linux commands such as `grep`, `tr` and `sed`. Examples on how to use regular expressions will be provided and the `sed` command will be presented[1].

0.2 Objectives

The objectives of the experiment is to learn the following:

- Learn on how to use regular expressions with Linux commands.
- Tackle the `sed` command.

0.3 Regular Expressions

Regular Expressions provide a convenient and consistent way of specifying patterns to be matched.

The shell recognizes a limited form of regular expressions when you use filename substitution. Recall that the asterisk (*) specifies zero or more characters to match, the question mark (?) specifies any single character, and the construct [...] specifies any character enclosed between the brackets. The regular expressions recognized by the aforementioned programs are far more sophisticated than those recognized by the shell. Also be advised that the asterisk and the question mark are treated differently by these programs than by the shell.

0.3.1 Matching Any Character: The Period (.)

A period in a regular expression matches any single character, no matter what it is. So the regular expression:

```
grep '\<c...h\>' *.txt
```

will display all five-character English dictionary words starting with "c" and ending in "h".

The following command will search for all 3-letter words in all text files:

```
grep '\<...\>' *.txt
```

0.3.2 Matching the Beginning of the Line: The Caret (^)

If we want to display only the lines that start with a particular string, we can use the caret as follows:

```
grep ^root /etc/passwd
```

Only the lines that start with the string `root` will be displayed.

[1] Stephen Kochan, Patrick Wood, **Unix Shell programming** Third Edition. Sams Publishing, 456 pages.

0.3.3 Matching the End of the Line: The Dollar Sign ($)

Just as the caret is used to match the beginning of the line, so is the dollar sign $ used to match the end of the line. So the regular expression:

```
grep m$ *.txt
```

will search all lines in all text files that end with the letter `m`

To do

Type exactly the following text in file `exp.txt`:

```
The Unix operating system was pioneered
by
Ken
Thompson and Dennis
Ritchie at Bell
Laboratories in the late 1960s.  Onee
of the primary
goals in the design of the Unix system was to create an environment that
promoted efficient program
development.
[my name]
.Hi man
HHi man
my mamamamamama my
mohammadm
```

Run the following commands and note the output:

```
grep l$ exp.txt
grep y$ exp.txt
grep ^R exp.txt
```

Note:

Run the command:

```
grep .$ exp.txt
```

and note the output that you get.

You will notice that you don't get only the lines that end with a dot but all the lines in the file `exp.txt`. This matches any single character at the end of the line (including a period). recall that a dot matches any character. So how do you match a period?

In general, if you want to match any of the characters that have a special meaning in forming regular expressions, you must precede the character by a backslash (\) to remove that special meaning as follows:

```
grep '\.$' exp.txt
```

You will notice that you get the output that you are looking for.

The following command will search for all lines in file `exp.txt` that start with a dot:

```
grep '^\.'  exp.txt
```

It is worth noting that the regular expression:

```
grep ^$ exp.txt
```

matches any line that contains *no* characters.

0.3.4 Matching a Choice of Characters: The [...] Construct

Suppose you want to display all lines in the file `exp.txt` that contain the string "`The`", you execute:

```
grep '\<The\>' exp.txt
```

If you want to display all lines that contain either the string "`The`" or the string "`the`", you execute:

```
grep '\<[Tt]he\>' exp.txt
```

The regular expression above would match lowercase or uppercase `t` followed immediately by the characters `he`.

If you intend to display all lines that contain numbers, you execute:

```
grep '[0123456789]' exp.txt
```

or, more succinctly, you could simply execute:

```
grep '[0-9]' exp.txt
```

To match an uppercase letter, you execute:

```
grep '[A-Z]' exp.txt
```

And to match an upper- or lowercase letter, you execute:

```
grep '[A-Za-z]' exp.txt
```

To find a line that starts with an uppercase letter, you execute:

```
grep '^[A-Z]' exp.txt
```

If a caret (^) appears as the first character after the left bracket, the sense of the match is *inverted*. For example, the regular expression:

```
grep '[^A-Z]' exp.txt
```

matches any character *except* an uppercase letter. Similarly,

```
grep '[^A-Za-z]' exp.txt
```

matches any nonalphabetic character.

The following command will execute all lines in file `exp.txt` that do not start with an alphabetic character:

```
grep '^[^A-Za-z]' exp.txt
```

0.3.5 Matching Zero or More Characters: The Asterisk (*)

The asterisk is used by the shell in filename substitution to match zero or more characters. In forming regular expressions, the asterisk is used to match zero or more occurrences of the preceding character in the regular expression (which may itself be another regular expression). For example:

```
grep '[t*]' exp.txt
```

matches one or more capital `t`'s, because the expression specifies a single `t` followed by zero or more `t`'s.

The following regular expression matches any alphabetic character followed by zero or more alphabetic characters:

```
grep '[A-Za-z][A-Za-z]*' exp.txt
```

Equally, the following regular expression matches any numeric string followed by zero or more nonalphabetic characters:

```
grep '[0-9][0-9]*' exp.txt
```

Note:

If you want to match a dash character inside a bracketed choice of characters, you must put the dash immediately after the left bracket (and after the inversion character ^ if present) or immediately before the right bracket]. So the expression:

```
grep '[-0-9]' exp.txt
```

matches a single dash or digit character.

Equally, if you want to match a dash character inside a bracketed choice of characters, you must put the dash immediately after the left bracket (and after the inversion character ^ if present) or immediately before the right bracket]. So the expression:

```
grep '[]0-9]' exp.txt
```

matches a right bracket or a digit.

0.3.6 Matching a Precise Number of Characters: \{...\}

In the preceding examples, we've seen how we can specify a certain number of occurences of a character. There is a more general way to specify a precise number of characters to be matched by using the construct:

\{min,max\}

where *min* specifies the minimum number of occurrences of the preceding regular expression to be matched, and *max* specifies the maximum. For example, the regular expression:

```
grep 'T\{1,10\}' exp.txt
```

matches from one to ten consecutive T's.

Whenever there is a choice, the largest pattern is matched; so if the input text contains eight consecutive T's at the beginning of the line, that is how many will be matched by the preceding regular expression. As another example, the regular expression:

```
grep '[A-Za-z]\{8,20\}' exp.txt
```

matches a sequence of alphabetic letters from eight to twenty characters long.

A few special cases of this special construct are worth noting. If only one number is enclosed between the braces, as in

\{2\}

that number specifies that the preceding regular expression must be matched *exactly* that many times. For example, execute the following command:

```
grep '[A-Za-z]\{2\}' exp.txt
```

and note the output. You will be disappointed by the output that you get. Can you explain why you get such an output?

The remedy to that problem is to execute the following command:

```
grep '[.\{2\}' exp.txt
```

and that will match exactly two characters (no matter what they are).

If a single number is enclosed in the braces, followed immediately by a comma, then at least that many occurrences of the previous regular expression must be matched. So, if you execute the following command:

```
grep '[A-Za-z]\{2,\}' exp.txt
```

you will get all the lines that match at least two consecutive characters. Once again, if more

than two exist, the largest number is matched.

0.3.7 Saving Matched Characters: \(...\)

You are surely happy with what you have seen in the previous sections and can sense already the power of regular expressions. However, there are still some limitations in what you have in the above examples. Below are some scenarios:

- How can you display all lines of a file or set of files that start with the same repeated characters?

  Example: Show all lines in file `exp.txt` that start with the letters `aa` or `bb`, `cc`, etc.

- How can you display all lines of a file or set of files that start and end with the same character?

More examples might pop up to your mind. The trick consists of *saving the matched characters* as described below.

It is possible to capture the characters matched within a regular expression by enclosing the characters inside backslashed parentheses. These captured characters are stored in "registers" numbered 1 through 9. For example, the regular expression:

`^\(.\)`

matches the first character on the line, whatever it is, and stores it into register 1. To retrieve the characters stored in a particular register, the construct \n is used, where n is from 1-9.

So, the regular expression:

`^\(.\)\1`

matches the first character on the line and stores it in register 1. Then the expression matches whatever is stored in register 1, as specified by the \1.

Examples

- The following regular expression will display on the standard output all lines of the file `exp.txt` that start with the same repeated characters:

  ```
  grep '^\(.\)\1' exp.txt
  ```

  The net effect of this regular expression is to match the first two characters on a line *if they are both the same character.*

- The following regular expression will display on the standard output all lines of the file `exp.txt` that start and end with the same character:

  ```
  grep '^\(.\).*\1$' exp.txt
  ```

  The above regular expression matches all lines in which the first character on the line (^.) is the same as the last character on the line (\1$). The .* matches all the characters in-between.

  Successive occurrences of the \(...\) construct get assigned to successive registers. So when the following regular expression is used to match some text:

  ```
  ^\(...\)\(...\)
  ```

  the first three characters on the line will be stored into register 1, and the next three characters into register 2.

The below table summarizes the special characters recognized in regular expressions[2].

[2]Stephen Kochan, Patrick Wood, **Unix Shell programming** Third Edition. Sams Publishing, 456 pages.

Notation	Meaning	Example	Matches
.	*any* character	`a..`	`a` followed by any two characters
^	beginning of line	`^wood`	`wood` only if it appears at the beginning of the line
$	end of line	`x$`	`x` only if it is the last character on the line
		`^INSERT$`	a line containing just the characters `INSERT`
		`^$`	a line that contains *no* characters
*	zero or more occurrences of a character	`W.*S`	`W` followed by zero or more characters followed by an `S`
		`.*`	zero or more characters
[*chars*]	any character in *chars*	`[tT]`	lower- or uppercase `t`
[^*chars*]	any character not in *chars*	`[^0 -9]`	any nonnumeric character
		`[^a-zA-Z]`	any nonalphabetic character
\{min,max\}	at least *min* and at most *max* occurrences of previous regular expressions	`x\{1,5\}`	at least 1 and at and at most 5 x's
		`[0-9]\{3,9\}`	from 3 to 9 successive digits
		`[0-9]\{3\}`	exactly 3 digits
		`[0-9]\{3,\}`	at least 3 digits
\(...\)	store characters matched between parentheses in next register (1-9)	`^\(.\)`	first character on line and stores it in register 1
		`^\(.\)\1`	first and second characters on the line if they're the same

0.4 the `sed` command

`sed` is a program used for editing data. It stands for *stream editor*. The general form of the `sed` command is:

```
sed command file
```

where `command` is applied to each line of the specified `file`.

Create the file `exp1.txt` and type into it the following text:

```
The Unix operating system was pioneered by Ken
Thompson and Dennis Ritchie at Bell Laboratories
in the late 1960s. One of the primary goals in
the design of the Unix system was to create an
environment that promoted efficient program
```

```
development.
```

Suppose that you want to change all occurrences of "Unix" in the text to "UNIX." This can be easily done in `sed` as follows:

```
sed 's/Unix/UNIX/' exp1.txt
```

The output will look like:

```
The UNIX operating system was pioneered by Ken
Thompson and Dennis Ritchie at Bell Laboratories
in the late 1960s. One of the primary goals in
the design of the UNIX system was to create an
environment that promoted efficient program
development.
```

The `sed` command `s/Unix/UNIX/` is applied to every line of file `exp1.txt`. Whether or not the line gets changed by the command, it gets written to standard output all the same. Note that `sed` makes no changes to the original input file. To make the changes permanent, you must redirect the output from `sed` into a temporary file and then move the file back to the old one:

```
sed 's/Unix/UNIX/' exp1.txt > temp
mv temp exp1.txt
```

If your text included more than one occurrence of "Unix" on a line, the preceding `sed` would have changed just the first occurrence on each line to "UNIX." By appending the global option `g` to the end of the `s` command, you can ensure that multiple occurrences of the string on a line will be changed. In this case, the `sed` command would read:

```
sed 's/Unix/UNIX/g' exp1.txt
```

0.4.1 The `-n` option

The command `sed` can be used as well if you want to print any part of a file. For such purposed, the command `-n` is used. This option tells `sed` that you don't want it to print any lines unless explicitly told to do so. This is done with the `p` command. By specifying a line number or range of line numbers, you can use `sed` to selectively print lines of text. So, for example, to print just the first two lines from a file, the following could be used:

```
sed -n '1,2p' exp1.txt
```

The output will be as follows:

```
The UNIX operating system was pioneered by Ken
Thompson and Dennis Ritchie at Bell Laboratories
```

If, instead of line numbers, you precede the `p` command with a string of characters enclosed in slashes, `sed` prints just those lines from standard input that contain those characters. The following example shows how `sed` can be used to display just the lines that contain a particular string:

```
sed -n '/UNIX/p' exp1.txt
```

The output will be as follows:

```
The UNIX operating system was pioneered by Ken
the design of the UNIX system was to create an
```

0.4.2 Deleting lines

To delete entire lines of text, use the `d` command. By specifying a line number or range of numbers, you can delete specific lines from the input. In the following example, `sed` is used to delete the first two lines of text from `exp1.txt`:

```
sed '1,2d' exp1.txt
```

The output will be as follows:

```
in the late 1960s. One of the primary goals in
the design of the UNIX system was to create an
environment that promoted efficient program
development.
```

By preceding the `d` command with a string of text, you can use `sed` to delete all lines that contain that text. In the following example, `sed` is used to delete all lines of text containing the word "UNIX":

```
sed '/UNIX/d' exp1.txt
```

The output will be as follows:

```
Thompson and Dennis Ritchie at Bell Laboratories
in the late 1960s. One of the primary goals in
environment that promoted efficient program
development.
```

The power and flexibility of `sed` command goes far beyond what we've shown here. In the below table, we show some examples of `sed` commands:

`sed` **command**	**Description**
`sed '5d'`	Delete line 5
`sed '/[Tt]est/d'`	Delete all lines containing Test or test
`sed -n '20,25p' text`	Print only lines 20 through 25 from text
`sed '1,10s/unix/UNIX/g' exp1.txt`	Change unix to UNIX wherever it appears in the first 10 lines of `exp1.txt`
`sed '/jan/s/-1/-5/'`	Change the first -1 to -5 on all lines containing jan
`sed 's/.../' exp1.txt`	Delete the first three characters from each line of `exp1.txt`
`sed 's/...$//' exp1.txt`	Delete the last 3 characters from each line of `exp1.txt`
`sed -n 'l' text`	Print all lines from text, showing nonprinting characters as \nn (where nn is the octal value of the character), and tab characters as \t