Lab: Regular Expressions

Linux has two regular expression engines:

1. An engine that supports the **POSIX Basic Regular Expression (BRE)** standard.
2. An engine that supports the **POSIX Extended Regular Expression (ERE)** standard.

POSIX BRE Regular Expressions

Perhaps the simplest BRE pattern is a regular expression for finding the exact occurrence of a sequence of characters in a text. This is what sed and awk look like for a string:

```
elliot©ubuntu:-$ echo "This is a test" | sed -n '/test/p'
This is a test
elliot©ubuntu:-$ echo "This is a test" | awk '/test/{print $0}'
This is a test
```

When working with regular expressions, keep in mind that they are case-sensitive:

```
elliot©ubuntu:-$ echo "This is a test" | awk '/test/{print $0}'
This is a test
```

The first regular expression did not match, since the word “test” starting with a capital letter does not occur in the text. The second, tuned to search for a capitalized word, found a matching string in the stream.

In regular expressions, you can use not only letters but also spaces and numbers:

```
elliot©ubuntu:-$ echo "This is a test 2 again" | awk '/test 2/{print $0}'
This is a test 2 again
```

Spaces are treated by the regular expression engine as regular characters.

Special symbols

There are a few things to keep in mind when using different characters in regular expressions. So, there are some special characters, or metacharacters, which require a special approach to use in a template. Here they are:

```
.*[]^${}\+?|()
```

If one of them is needed in the template, it will need to be escaped with a backslash (backslash) — .

For example, if you need to find a dollar sign in the text, it must be included in the template, preceded by an escape character. Let’s say you have a file `myfile` with this text:

```
There is 10$ on my pocket
```

The dollar sign can be detected using a pattern like this:

```
elliot©ubuntu:-$ cat myfile
There is 10$ on my pocket
elliot©ubuntu:-$ awk '/\$/{print $0}' myfile
There is 10$ on my pocket
```

Also, a backslash is also a special character, so if you want to use it in a pattern, you will need to escape it too. It looks like two forward slashes:

```
elliot©ubuntu:-$ echo "\ is a special character" | awk '/\\/{print $0}'
\ is a special character
```

Point symbol

A period is used to search for any single character except for line feed. Let's pass to such a regular expression a file myfile, the contents of which are given below:

Note: Make sure to update `myfile` content as below:

```
elliot©ubuntu:-$ cat myfile
this is a test
This is another test
And this is one more
start with this
elliot©ubuntu:-$ awk '/.st/{print $0}' myfile
this is a test
This is another test
```

As you can see from the output, only the first two lines from the file match the pattern, since they contain the sequence of characters "st", preceded by one more character, while the third line does not contain a suitable sequence, and in the fourth it is, but is at the very beginning of the line.

Character classes

The period matches any single character, but what if you need more flexibility to limit the set of characters you are looking for? In such a situation, you can use character classes.

Thanks to this approach, you can organize a search for any character from a given set. Square brackets are used to describe a character class — `[]`:

```
elliot©ubuntu:-$ cat myfile
this is a test
This is another test
And this is one more
start with this
elliot©ubuntu:-$ awk '/[oi]th/{print $0}' myfile
This is another test
start with this
```

Here we are looking for the "th" character sequence preceded by the "o" or "i" character.

Classes come in handy when looking for words that can start with both uppercase and lowercase letters:

```
elliot©ubuntu:-$ echo "this is a test" | awk '/[Tt]his is a test/{print $0}'
this is a test
elliot©ubuntu:-$ echo "This is a test" | awk '/[Tt]his is a test/{print $0}'
This is a test
```

Character classes are not limited to letters. Other symbols can be used here. It is impossible to say in advance in what situation the classes will be needed — everything depends on the problem being solved.

POSIX ERE Regular Expressions

The POSIX ERE templates that some Linux utilities support may contain additional characters. As already stated, awk supports this standard, but sed does not.

❓ Question mark

The question mark indicates that the preceding character may appear once in the text or not at all. This character is one of the repetition metacharacters. Here are some examples:

```
elliot©ubuntu:-$ echo "tet" | awk '/tes?t/{print $0}'
tet
elliot©ubuntu:-$ echo "test" | awk '/tes?t/{print $0}'
test
elliot©ubuntu:-$ echo "tesst" | awk '/tes?t/{print $0}'
```

As you can see, in the third case the letter “s” occurs twice, so the regular expression does not respond to the word “tesst”.

The question mark can be used with character classes as well:

```
elliot©ubuntu:-$ echo "tst" | awk '/t[ae]?st/{print $0}'
tst
elliot©ubuntu:-$ echo "test" | awk '/t[ae]?st/{print $0}'
test
elliot©ubuntu:-$ echo "tast" | awk '/t[ae]?st/{print $0}'
tast
elliot©ubuntu:-$ echo "taest" | awk '/t[ae]?st/{print $0}'
elliot©ubuntu:-$ echo "teest" | awk '/t[ae]?st/{print $0}'
```

If there are no characters from the class in the string, or one of them occurs once, the regular expression works, however, as soon as two characters appear in the word, the system no longer finds a match for the pattern in the text.

➕ Plus symbol

The plus symbol in the pattern indicates that the regular expression will find the desired one if the preceding character occurs one or more times in the text. At the same time, such a construction will not react to the absence of a symbol:

```
elliot©ubuntu:-$ echo "test" | awk '/te+st/{print $0}'
test
elliot©ubuntu:-$ echo "teest" | awk '/te+st/{print $0}'
teest
elliot©ubuntu:-$ echo "tst" | awk '/te+st/{print $0}'
```

In this example, if there is no e in a word, the regular expression engine will not find a match in the text. The plus symbol also works with character classes, which makes it look like an asterisk and a question mark:

```
elliot©ubuntu:-$ echo "tst" | awk '/t[ae]+st/{print $0}'
elliot©ubuntu:-$ echo "test" | awk '/t[ae]+st/{print $0}'
test
elliot©ubuntu:-$ echo "teast" | awk '/t[ae]+st/{print $0}'
teast
elliot©ubuntu:-$ echo "teeast" | awk '/t[ae]+st/{print $0}'
teeast
```

In this case, if the string contains any character from the class, the text will be considered to match the pattern.