PRCMP P08

The Unix shell: regular expressions and find

May 2023

1 Regular expressions

Let's focus on the two Unix regular expression engines:

- Basic Regular Expression (BRE), and
- Extended Regular Expression (ERE).

1.1 BRE

Plain text – String of regular characters that represents the text to be searched. The pattern is case sensitive!

^ – Pattern starts at the beginning of the line.

$ – Pattern occurs at the end of the line.

. – Any single character. Matches any single character except a newline character.

[ ] – Character class. Matches any single character from the group defined inside square brackets.

***** – Repeating character. The character before the asterisk must appear zero or more consecutive times.

1.2 ERE

? – Non-repeating character. The character before the question mark occurs zero or one time.

+ – Repeating character. The character before the plus sign occurs one or more times.

{} – Specifies how many times the character before curly braces occurs in a pattern. The limits are defined inside curly braces.

| – The pipe allows to specify two or more alternatives, making the regex engine to use a logical OR.

() – Grouping expressions. The expression inside parenthesis is treated as a character.

1.3 The `grep` utility

You can use the `grep` utility to search basic regular expressions. You can search extended regular expressions using the `egrep` utility, or adding the `-E` option to `grep`.

Both utilities can receive text in the `stdin`, by piping the output of a previous command:

```
cat file.txt | grep <regex>
```

You can also pass the names of the source file (or files) as arguments to these utilities:

```
grep <regex> file1.txt file2.txt
```

Questions

1. Create a command sequence that lists only subdirectories in `/usr`.

   Hint: use the first character of the permissions group to find the file type.

2. Create a text file with several lines. Make sure some of those lines start with "X", some end in "xpto" and some starting with "principios" and ending with "computacao".

   (a) List all the lines starting with "X".

   (b) List all the lines ending in "xpto".

   (c) List all the lines starting with "X" or end with "xpto".

   (d) List all the lines starting in "principios" and end with "computacao".

3. Consider a file containing a list of marathon classifications called `classifications.csv`. In this file, each line represents the result of a single athlete, following the *position;time;name* format as in the example:

```
2;2:05:25;Manuel Torpedo
1;2:04:23;João Lebre
15;2:20:15;Henrique Labareda
13;2:16:03;Passos Dias Aguiar Mota
(...)
```

   Note that the lines are not necessarily in the order of classification.

   (a) Write a `classifications.csv` file following the format shown. Add several lines with diverse race times.

   (b) Look for the lines where the runners have a surname (last name) of your choice.

   How do you make sure that the text entered is the surname?

   (c) Look for the lines where the runners have a first name of your choice.

   How do you make sure that the text entered is the first name?

   (d) Show the top 5 runners, sorted from first to fifth.

   (e) Repeat the previous question, but this time, show only the race times and the runners' names, separated by a tab.

   (f) Search the runners that finished the race in less than 2 hours and 20 minutes. Show only the race times and the runners' names.

   (g) Show the number of runners (and not the runners) that were found in the previous question.

   Hint: Consult the `grep` utility manual page.

4. Consult the `grep` manual page and find the answer to the following questions.

   (a) Which option makes grep prefix each matching line with the line number within its input file?

   (b) Which option prints *num* lines of trailing context before each match?

   (c) Which option prints *num* lines of trailing context after each match?

   (d) Which option combines the previous two, presenting *num* lines before and after for each match?

   (e) Which option causes grep to stop reading a file after *num* matching lines?

   (f) What could be the usefulness of the `-q` option?

   (g) We often want to look for an expression, regardless of uppercase/lowercase. What is the `grep` option that allows you to ignore letter case?

5. In Portugal, the postal code is a combination of 4 digits with a suffix of 3 digits separated by a hyphen, followed by a space and a postal designation with a maximum of 25 characters.

   Build a regular expression that matches Portuguese postal codes.

6. In Portugal, a complete telephone number is composed of three parts:

   - The international dialling prefix: +
   - The country calling code: 351
   - The phone number: 9 digits.

   The international dialling prefix and the country calling code are written tog The international dialling prefix and the country calling code are with no space between them. However, there is a mandatory space before the phone number.

   For easy reading, it is usual (but not mandatory) to insert two spaces between the nine digits of the phone number. Unfortunately, the place where the two spaces are inserted is not always the same.

   Write one (and only one) regular expression that can match portuguese phone numbers. Test your regex to match these numbers:

   - +351 999 999 999
   - +351 99 999 9999
   - +351 999999999

2 The `find` command

The `find` command is a powerful tool that allows you to locate files that match a certain criteria and perform operations on those files.

The most simple syntax for a `find` call is `find <search_path> ... <criteria>` for example:

```
find /bin -iname 'l*' -size +150k
```

This utility recursively descends the directory tree (starting from the `search_path`) and evaluates the specified criteria for each file in the tree.

The criteria can be formed by any feature of the file: name, size, creation/modification times, etc. You should consult the `find` manual page to access all the options.

Actions on files matching the criteria are specified with the `-exec` option. This option marks the beginning of a command line template where the name of each file found replaces placeholder {}. Following the command line template, the `\;` marks the end of the command line. The command line is immediately executed for each file found.

Example:

```
find . -type f -iname '*.pdf' -exec mv {} my_docs \; -exec echo {} moved to my_docs \;
```

The `-ok` option behaves in the same way as the `-exec` option but interactively asks the user if the action should be performed.

Questions

7. Refer to the manual page of the `find` command and look for the answers to the following questions.
   (a) What is the difference between the `-name` and `-iname` options?
   (b) What is the `-mtime` option for? How could it be used to create periodic backups?
   (c) What is the `-size` option for? How can you use it to select files with size smaller than 10 MB?
   (d) The `-type` option is used to specify the file type. What is the regular file identifier? And what is the directory identifier?

8. Find all the files whose name ends in `.txt`, throughout your home directory.
9. Find all the files that have a size greater than 200 kilobytes in the /bin directory.

   Did the find command search only in the `/bin` directory or did it extend the search?
10. Search all existing directories throughout your home directory.
11. Using `-maxdepth`, repeat the last question, presenting only the directories exactly in your home directory.
12. Find all files that haven't been:
    (a) Modified in the last 3 days in your work directory.
    (b) Accessed in the last 10 days in the `/bin` and `/usr/bin` directories.
13. Copy all files starting with "f", "p" or "c" starting from your work directory to the `/tmp` directory using the find command.
14. Delete all files in the `/tmp` directory that have been accessed in the last 2 days. Make sure you can confirm before executing the operation.

3 Solutions

1. `ls -l /usr/ | grep '^d'`
2. (a)
   (b)
   (c)
   (d)
3. (a) `nano classification.csv`
   (b)
   (c)
   (d) `grep '^[1-5];' classification.csv | sort`
   (e) `grep '^[1-5];' classification.csv | sort | cut -d ';' -f2,3 | tr ';' '\t'`
   (f) `grep ';2:[01][0-9]:' classification.csv | cut -d ';' -f2,3`
   (g) `grep -c ';2:[01][0-9]:' classification.csv`
4. (a) Option `-n`
   (b)
   (c)
   (d)
   (e)
   (f)
   (g)
5.
6. (a)
   (b)
   (c)
   (d)
7. `find $HOME -iname '*.txt'`
8.
9.
10.
11. (a)
    (b)
12.
13.