Regular Expressions

OPS102 Week 12 Class 1

Tiayyba Riaz/John Sellens

April 1, 2024

Seneca Polytechnic

Outline

What Are Regular Expressions?

Regular Expressions

- Regular expressions (regex) are powerful tools used for pattern matching in text.
- You have used grep previously to find specific strings in file. You might also have used the find functionality in Microsoft Word.
- Sometimes we want to search for strings that follow a complex pattern instead of specific string, like finding all postal codes or email addresses from a database file.

Regular Expressions - 2

- In order to work with patterns, as opposed to specific strings, we use regular expressions.
- Regular expressions are widely used in applications, such as:
  - searching and filtering log files
  - manipulating configuration files
  - scripting
  - text processing

Regular Expressions - 3

- The simplest regular expression is a string of characters that matches itself. This is like finding a string like `"Hoopla"` in a text file.
- Complex regular expressions use letters, numbers, and, special characters to define many different strings which follow a pattern.
- A regular expression can match a string in more than one place.
  - There can be multiple matches on a single line.
- The (simple) regular expression `"aps"` will match the following lines

```
This bridge may collapse.
I do not like capsicum.
Did you see that snapshot of the caps with snaps?
```

Regular Expressions - 4

- Linux and Windows provide various command-line tools that support regular expressions e.g. `"grep"`, `"egrep"`, `"sed"`, `"awk"`, `"vim"`.
- We will be using `"egrep"` for this lesson.
- `"egrep"` is an “extended” version of `"grep"`.

Regular Expressions - Simple Examples

`egrep friend poem2.txt`	Matches all occurrences of the word friend. It will match any string having `"friend"` as a substring in it, like `"friend"`, `"friendly"`, `"friendship"`
`egrep 'friend' poem2.txt`	It is recommended to enclose the search pattern in quotes to avoid shell expansion.
`egrep '[fF]riend' poem2.txt`	Matches both `"friend"` and `"Friend"`

Components of Regular Expressions

Components of Regular Expressions

- The following are the seven most commonly used components of a regular expression:
  - Atoms
  - Wildcards
  - Character Classes
  - Repetition
  - Alternation
  - Groups
- These components can be used to define simple to complex patterns.
- In the next slides we will look at each of these and their examples.

Atoms

- Atoms are the building blocks of regular expressions.
- They can be individual characters, metacharacters, or escape sequences.
- For example:
  - `"a"` matches the character `"a"`.
  - `"."` matches any single character (except a newline).
    - Newlines are normally not considered – just record separators.

`egrep 'a.' testfile`	Matches `"ax"` or `"a0"` or `"a "` (a followed by space) i.e. `"a"` followed by any character.
`egrep 'c.t' testfile`	Matches `"cat"`, `"cet"`, `"cut"`, but not `"can"` or `"ct"`
`egrep '.....' testfile`	Matches any 5 characters. e.g. `"chair"`, `"c123gh"`

Wildcards

- Wildcards are special characters that represent repeating patterns in a string.
- `"*"` matches zero or more occurrences of the preceding atom.
- `"+"` matches one or more occurrences of the preceding atom.
- `"?"` matches zero or one occurrence of the preceding atom.

`egrep 'ab*' testfile`	Matches `"ab"` or `"abb"` or `"abbbbb"` (any number of b even zero)
`egrep 'ab?' testfile`	Matches `"a"`, or `"ab"`
`egrep 'a+b' testfile`	Matches `"ab"`, or `"aab"`, or `"aaaaaaab"` (any number of a but not zero)

- Question: What does `"a.*"` match?

Character Classes

- Character classes allow you to specify a set of characters to match against.
  - Similar to bash globbing character classes.
- `"[abc]"` matches any one of the characters `"a"`, `"b"`, or `"c"`.
- `"[0-9]"` matches any one digit.
- `"[A-Z]"` matches any uppercase letter, `"[A-Za-z]"` matches any letter.
- If a character class starts with a caret (`"^"`), the class is negated – it matches any character *not* in the class.
- Wildcard characters lose their special meanings inside a class.

`egrep '[aeiou]' testfile`	Matches any single vowel.
`egrep '[aeiou]*' testfile`	Matches zero or more vowels.
`egrep '[()*{}?]' testfile`	Matches opening and closing parenthesis, braces, question mark or asterisk.

Repetition

- Repetition specifies how many times a preceding atom can occur in a match.
  - Like ?, *, and + but more flexible.
- We can specify an exact number, a range, or an upper or lower bounds to the amount of times an atom is matched.
- `"{n}"` matches exactly n occurrences.
- `"{n,}"` matches at least n occurrences.
- `"{n,m}"` matches at least n and at most m occurrences.
- `"{,m}"` matches at most m occurrences.

`egrep '[aeiou]{3}' testfile`	Matches exactly 3 vowels together.
`egrep '[aeiou]{3,5}' testfile`	Matches exactly 3 or 4 or 5 vowels together.

Alternations

- Alternation allows you to specify multiple alternatives for a pattern.
- This is implemented with the `or` (`"|"`) operator.
- e.g. `"cat|dog"` matches either `"cat"` or `"dog"`.

`egrep 'ford|chevy' cars`	will match lines having either ford or chevy
`egrep 'Mr|Mrs Smith' file`	Will match `"Mr"` or `"Mrs Smith"`. Will not match `"Smith"` if it comes after `"Mr"`. How are you Mr Smith and Mrs Smith?

Groups

- Matching patterns can be grouped to be treated as one unit by using parenthesis.
  - i.e. A group is treated as a regular expression atom.
- The parentheses group the contents but are not part of the search pattern.

`egrep '(ab)+' file`	Matches one or more occurrences of the sequence `"ab"`, e.g. 3 matches in Mabmabmababababmmmm
`egrep '(Mr|Mrs) Smith' file`	Matches `"Mr Smith"` or `"Mrs Smith"`
`egrep 'a(abc)*z' file`	Matches: `"az"`, `"aabcz"`, `"aabcabcz"`

More Special Rules

- `"^"` – Anchors a match to the start of a line.
- `"$"` – Anchors a match to the end of a line.
- `"\"` – Quotes (escapes) special characters.
- `"\<"` – slightly non-standard – Anchors to the beginning of a word.
- `"\>"` – slightly non-standard – Anchors to the end of a word.

`egrep '^a' file`	Matches a letter `"a"` at the start of a line.
`egrep '^a.*\?.*b$' file`	Matches an entire line that starts with `"a"` and ends with `"b"` and has a question mark somewhere in the middle.

Regular Expressions in Bash Scripts

Regular Expressions in Bash Scripts

- Bash does not support regular expressions directly
- But bash scripts often use other commands that do
  - e.g. grep, egrep, sed, awk, ...
- Our scripts may want to select or modify data from files
- Or, we might want to use a regex to validate input!

Validating Script Input

- We learned about simple checking of user inputs in some of our scripts
  - e.g. Use `test` to compare an input variable against a known value.
- We often want to use a regular expression to ensure that our input is in the expected format.
  - e.g. Check that an input is a valid number, or that it looks like a possible postal code, etc.
- Design Pattern: use `echo` piped into `egrep` and check exit status.

Validating Script Input Example

The typical method to check an input against a validating regular expression is to use `echo` to pipe the value into `egrep` and check the exit status.

For example:

```
read -p "Please enter a number:" num
# read will (usually?) trim leading and trailing spaces (if any)
echo "$num" | egrep -q '^[0-9]+$'
if [ $? -eq 0 ]; then
  echo "this is an unsigned int: $num"
fi
```

You can of course use any suitable regex.
Remember that the status of a pipe is that of the final command.

More Interesting Validating Regular Expressions

Check for a signed integer:

```
echo "$num" | egrep -q '^[+-]?[0-9]+$'
```

Check for a (possibly) floating point number:

```
echo "$num" | egrep -q '^[+-]?[0-9]+[.]?[0-9]*$'
```

Check for a Canadian postal code (format):

```
echo "$code" | egrep -q '^[A-Z][0-9][A-Z] *[0-9][A-Z][0-9]$'
```

And, of course, far more complicated variants.

Summary

Summary of Regular Expressions

- Regular expressions are a way to describe patterns used to match strings of text.
  - Also commonly referred to as a “regex”.
- Regular expressions are very commonly used in system administration, text processing, text editing, input validation, and other programming tasks.
- Lots of tools in Linux and Windows make use of regular expressions.

Summary of Regex Syntax

`"."`	Match any character
`"+"`	Match one or more occurrences of preceding pattern
`"*"`	Match zero or more occurrences of preceding pattern
`"?"`	Match zero or one occurrence of preceding pattern
`"[abc]"`	Match any of the characters from the sequence given in brackets
`"{n}"`	Matches the preceding atom 'n' times exactly
`"{n,m}"`	Matches the preceding atom 'n' times but not more than 'm'
`"{n,}"`	Matches the preceding atom 'n' or more times
`"(xyz)+"`	Match one or more occurrences of the group xyz
`"(xyz|abc)"`	Match either xyz or abc
`"(xy|ab)c"`	Match either xyc or abc