Lecture 7: Regular Expressions

CS2042 - UNIX Tools

October 15, 2008

Lecture Outline

1 Intro to Regexps
- Meet Grep
- Meet RegExps

2 RegExp Specifics
- Basic RegExp Syntax
- RegExp Repetition
- Special Characters

Grep

Let's say we have a large record of user logons. How could we find the history of a single user?

Searching Text

grep <**string**> **[file]**

- Searches [file] for all lines containing <string>

grep -v <**string**> **[file]**

- Searches [file] for all lines NOT containing <string>

More Grep

Using **grep** on a file:

Example:

grep "logging" /var/log/up2date

- Shows server's logins to up2date server

Using **grep** with piped input:

Example:

history | grep grep

- When have **I** used **grep** recently?

Lecture Outline

1 Intro to Regexps
- Meet Grep
- Meet RegExps

2 RegExp Specifics
- Basic RegExp Syntax
- RegExp Repetition
- Special Characters

Intro to RegExps

Occasionally we need to search for less specific strings. For example, your address book catches fire - why not search your saved e-mail for anything formatted like an address? We can do this (fairly) easily using RegExps!

What is a RegExp?

Stands for "Regular Expression"

- Similar to wildcard strings, but more powerful
- Also uses different syntax (sorry!)

Scope

RegExps are used all over the place. We've already seen **grep**, which takes RegExp search strings. Later we'll see some other fun commands which use them.

- search documents in emacs/vi
- write scripts in Perl/Python/Ruby/...

Lecture Outline

1 Intro to Regexps
- Meet Grep
- Meet RegExps

2 RegExp Specifics
- Basic RegExp Syntax
- RegExp Repetition
- Special Characters

Some Wildcard Analogues

Some of the RegExp patterns perform the same tasks as our earlier wildcards.

Single Characters

Wild card: ? RegExp: .

- Matches any single character.

Wild card: [a-z] RegExp: [a-z]

- Matches one of the indicated characters
- Don't separate multiple characters with commas in RegExp form (e.x. [a,b,q-v] becomes [abq-v])

A Note on Ranges

Like shell wildcards, RegExps are case-sensitive. What if you want to match any letter, regardless of case?

- What will **[a-Z]** match?

Character Sorting

Different types of programs sort characters differently. In the C language, characters A-Z are assigned numbers from 65 to 90, while a-z are 97-122. Thus, the range **[a-Z]** would equate to [122-65]. Though this is bad enough, there are non-alphabet characters within that range. To specify the range of all letters safely, use **[a-zA-Z]**.

Note that not everything treats sorting like C. For example, a dictionary program might sort its characters aAbBcC.... To be on the safe side though, always use **[a-zA-Z]**

New Stuff

Positional Operators

ˆ and **$** allow us to match strings occuring at the beginning and end of a line, respectively. The positions of these two characters (relative to the search string) matter.

- ˆ comes before its string
- **$** comes after its string

Example:

grep o$

- Matches lines ending with "o"

grep ˆ[A-Z]

- Matches lines beginning with a capital letter

Your Shell's Special Treatment

Note:

Since you usually type regular expressions within shell commands, it is good practice to enclose the regular expression in single quotes (') to stop the shell from expanding it before passing the argument to your search tool.

Example:

Instead of **grep [qxz] test**, use **grep '[qxz]' test** - these single quotes are unnecessary when using RegExps outside the shell (within **nano**, for example).

Lecture Outline

1 Intro to Regexps
- Meet Grep
- Meet RegExps

2 RegExp Specifics
- Basic RegExp Syntax
- RegExp Repetition
- Special Characters

Matching Any Number

RegExps gain a lot of their flexibility through their handling of repeated expressions. A RegExp pattern followed by one of these repetition operators defines how many times that pattern should be matched.

Free Repeat

<**expr**>*

- Matches any number of repetitons of <expr>
- "any number" includes zero!

- We can use * to match repeated strings, such as ab*a = abba
- Can also combine with other RegExps!
- .* is equivalent to the * wildcard

Matching One or Fewer

Let's say we have an item which is optional - either it's there or not. We don't care, as long as it's not there more than once.

Optional Expression

<**expr**>**?**

- An expression followed by a **?** will be matched *at most* once.

Note:

grep requires special care when handling many special characters on the command line. For now, if you want to practice these types of RegExps, use the search tool within **less** (by typing / followed by the RegExp) or your favorite editor.

Matching One or More

We can already match a pattern many times by using *****, but what if we want to ensure that the pattern gets matched at least once?

Required Expressions

There are two ways to accomplish this:

- The **+** symbol matches the preceding pattern *at least* once.
- Repeat the pattern to be matched once, following the second with a *****

Example:

To match a line beginning with one or more capital letters:

- **^[A-Z]+** or
- **^[A-Z][A-Z]***

Matching a Range of Repetitons

More Specific Repetitions

- {**n**} : preceding item is matched exactly **n** times
- {**n,**} : item is matched **n** or more times
- {**n,m**} : preceding item is matched at least **n** times, but no more than **m** times

How would you use these RegExp patterns to search a document for words longer than 6 letters?
What if you wanted to exclude proper names?

Two More Expression Types

Grouping Expressions

(expr) : Matches *expr*

- Useful for grouping expressions together
- Can do things like **k(abb)*** to find *k*, *kabb*, *kabbabb*, etc.

The 'or' Operator

<**expr1**>|<**expr2**> : Matches *expr1* or *expr2*

- Think [xy] for multi-character expressions
- **(dos)|(DOS)** matches *dos* or *DOS*, but not *Dos* (as **[dD][oO][sS]** would)

Lecture Outline

1. Intro to Regexps
   - Meet Grep
   - Meet RegExps
2. RegExp Specifics
   - Basic RegExp Syntax
   - RegExp Repetition
   - Special Characters

Using Reserved Characters

Okay, we can match a bunch of patterns numbers and letters.

- How can we match all lines beginning with a period (.)?

Escaping Characters

The backslash (\) is used in UNIX to "escape" the following character, either to give it special meaning or (in this case) to remove the special meaning that it has. If you want to search for a special symbol using a RegExp, it must be preceded by a backslash.

grep '^\.' will match all lines beginning with a period.

Escaping Characters in Grep

From the grep Manual:

In basic regular expressions the metacharacters ?, +, {, }, |, (, and) lose their special meaning; instead use the backslashed versions \?, \+, \{, \}, \|, \(, and \).

Now we can use **grep** to search for some more interesting patterns:

Example:

- **grep '[0-9]\{3\}-[0-9]\{4\}'** : phone number syntax (i.e. 555-5291)
- **grep '"\?Hello?"\?'** : Matches Hello? with or without quotes