Faculty of Computer Science, Dalhousie University *19-Sep-2018*

CSCI 2132 — Software Development

Lecture 7: Wildcards and Regular Expressions

Location: Chemistry 125 Instructor: Vlado Keselj
Time: 12:35 – 13:25

Previous Lecture

- Pipes
- Inodes
- Hard links
- Soft links
- Filename Substitution (Wildcards) (started)

Filename Substitution (Wildcards)

- Also known as **pathname substitution**
- Used to specify multiple filenames (i.e., pathnames)
- Makes use of "wildcards"; i.e., metacharacters expanded by the shell
- Some wildcard types:
  - `?`: matches any single character
  - `*`: matches any string, including empty string
  - `[...]`: matches any single character in the set
  - `[!...]`: any character except characters from the set
  - we can use ranges with '`-`' in brackets

File Substitution Examples

- `[0-9]`: any digit between 0 and 9
- `[a-zA-Z]`: any English alphabet character
- `[unix]`: matches either 'u', 'n', 'i', or 'x'
- `ls ~/csci2132/lab1/*.java` — list Java files
- `ls *.????` — list all files with 4-character extension
- `ls lab[1-9]` — list all files with the name consisting of word `lab` and a digit from 1 to 9
- `ls [!0-9]*` — list all files which name does not start with a digit
- `cp lab1.bk/*.java lab1/` — copy Java files from one directory to another

More Examples

- `ls ~/csci2132/lab1/*.java`
- `ls ~/csci2132/lab1/H????World.java`
- `ls H*`
- `ls [!A-Z]*`
- `ls */*/*.java`
- `ls *.java */*.java`
- `echo .*`

- command `echo` — prints out command line arguments
- `cat *.txt > allfiles`

The following command will list any files whose names end with .java in the lab1 directory we created for Lab 1:
`ls ~/csci2132/lab1/*.java`
The following command will list any file, in the same directory, whose name start with H, followed by any 4 characters, which is then followed by World.java:
`ls ~/csci2132/lab1/H????World.java`
The following command will list files whose names start with 'H' in the current working directory:
`ls H*`
The following command will list files whose names do not begin with an uppercase letter:
`ls [!A-Z]*`

7.2 Regular Expressions

Regular Expressions

- Regular Expressions are patterns used to match strings, and thus used in fast and flexible text search
- The name comes from Regular Sets defined by the mathematician Stephen Kleene
- Implemented as DFA (Deterministic Finite Automata) or NFA (Non-deterministic Finite Automata)
- Kleene's notation implemented by Ken Thompson into the editor QED to match patterns
- Thompson later added this to the Unix editor ed
- Eventually led to the command `grep`, coming from ed command `g/re/p` (Global search for Regular Expression and Print matching lines)

Regular expressions are used to address the problem of supporting fast and flexible text search: regular expression search can be done using a DFA (you will learn this in a 3rd year course) which is fast, and they can be used to specify a range of patterns and are thus flexible.

A regular expression is a sequence of normal and special characters specifying a pattern to match against strings. It is used in many programs such as grep, sed, and perl. Again, we need learn metacharacters that can be used to specify matching rules. Pay attention to the difference between these characters and wildcards in filename substitution.

Reading about Regular Expressions

- The Unix book: Chapter 3, Filtering Files (p.84)
- Appendix: Regular Expressions (p.665)
- Regular expressions
  - Patterns used for searching and replacing text
  - Used in many contexts, but we will focus on the grep command
  - There are two kinds of regular expressions: basic regular expressions and extended regular expressions

---

Regular expressions are used to express textual patterns for search or search-and-replace. They are useful in many context, but we will focus on their use in the grep command.

Basic Regular Expressions

Metacharacters:

- `.`: Matches any single character. For example, some strings that match a.b are: acb, a b, a-b, ...
- `[]`: Matches any of the single characters enclosed in brackets. We can use - to specify a range, and ^ to compliment the set (e.g. `[^ ]`). Other special characters, including ., *, ^ (note that ^ has other special meanings outside `[]`), `$` and \ lose their special meanings inside `[]`.

– `*`: 0 or more occurrences of the character that precedes it. For example, a* can be used to specify the following set of strings: empty string, a, aa, aaa, aaaa, ...
– `^`: Matches the beginning of a line.
– `$`: Matches the end of a line. `^` and `$` are useful if we want a pattern to occur as the prefix/suffix of a line.
– `\`: Inhibits the meaning of any metacharacters.

Many of these characters also have special meanings in the shell, so quoting (placing the regular expressions within a pair of single quotation marks ' ') may be necessary when using a regular expression as part of a command line.

BRE Examples

– BRE = Basic Regular Expressions
– One or more spaces: spacespace* (replace space by a space character): ' *'
– Empty line: `^$`
– Formatted dollar amount: `\$[0-9][0-9]*\.[0-9][0-9]`

Filters, grep command

– Filter is a program that is mostly used to read stdin, process data, and write to stdout
– Often used as elements of pipelines
– One such program is `grep`
– grep reads a file or stdin and outputs lines matching a regular expression
– grep syntax
```
grep [options] BRE [file(s)]
```

A **filter** is a program that gets most of its data from stdin and writes its main results to stdout. As pipes use one program's stdout as another's stdin, it is natural that filters are often used as elements of pipelines. Now let us learn a filter command that uses BRE.

The filter **grep** scans a file and outputs all lines that contain a given (regular expression) pattern. The syntax of using grep is
```
grep [options] BRE [file(s)]
```

When a file (or files) is not present, grep reads from stdin.

For example, consider the following file named price:

```
Chocolate $1.23 each
Candy $.56 each
Jacket $278.00</pre>
<pre>$44.00
$44
```

If we enter the following command
```
grep '\$[0-9][0-9]*\.[0-9][0-9]' price
```
The output will be the following three lines:

```
Chocolate $1.23 each
Jacket $278.00
$44.00
```

Let's see more examples. In most UNIX/Linux distributions, there is a dictionary file that contains English words in a dictionary, and each line is one English word. This file is /usr/share/dict/linux.words on bluenose. The following command will make use of this file to find all the 5-letter dictionary words that start with a or b and end with b:

```
grep '^[ab]...b$' /usr/share/dict/linux.words
```

The following command will find all the dictionary words that start with a or b and end with b:

```
grep '^[ab].*b$' /usr/share/dict/linux.words
```

Similarity between Wildcards and Regular Expressions

– We can get similar results with wildcards and regular expressions; e.g.:
```
ls *.java
ls | grep '\.java$'
```
– List of all files in `/bin`, whose names contain exactly one minus sign (-):
```
ls /bin | grep '^[^-]*-[^-]*$'
```

The following two commands produce equivalent results, although in one we use filename wildcards and in another one regular expressions:

```
ls *.java
ls | grep '\.java$'
```

The use of wildcards directly matches files in the folder and because of this they are safer in terms of accidental errors in matching some other part of the output of an 'ls' command. On the other hand, the regular expressions are more expressive. for example the following command will list all the files in `/bin` whose names contain exactly one minus sign (-):

```
ls /bin | grep '^[^-]*-[^-]*$'
```

This cannot be done using wildcards. Then, why do we need wildcards now that we can use regular expressions to perform these tasks? There are a few reasons. First, in the example of listing java files, one 'ls' command with wildcars is sufficient and we do not need to use pipes, so this is a simpler operation. Second, wildcards can also be used with many other commands such as 'cp' to perform operations on a set of files, while regular expressions require use of 'grep', which cannot be directly used in these situations.