CSCI 2132: Software Development

Wildcards and Regular Expressions

Norbert Zeh

Faculty of Computer Science
Dalhousie University

Winter 2019

Searching

Problem:

- Find all files whose names match a certain pattern
- Find all files that contain a certain text pattern
- …

Tools:

- Wildcards (shell)
- Regular expressions (`grep` and other tools)

Filename Substitution (Wildcards)

- Also known as **pathname substitution** or **pathname expansion**
- Used to specify patterns that match multiple pathnames
- Makes use of **wildcards** (metacharacters expanded by the shell)

Some wildcards:

- `?` matches any single character
- `*` matches any string
- `[a-z_]` matches any character in the range `‘a’ .. ’z’` and `‘_’`
- `[!a-z]` or `[^a-z]` matches any character not in the range `‘a’ .. ’z’`

File Substitution Examples

`[0-9]`	any digit
`[a-zA-Z]`	any English letter
`[unix]`	any of the characters `‘u’, ’n’, ‘i’, ‘x’`

- `ls ~/csci2132/lab1/*.java`
  List Java files in `csci2132/lab1`
- `ls *.????`
  List all files with 4-character extension
- `ls lab[1–9]`
  List all files with names `lab1, ... , lab9`
- `ls [!0-9]*`
  List all files whose names don’t start with a digit
- `cp lab1.bk/*.java lab1/`
  Copy Java files from the `lab1.bk` directory to the `lab1` directory

More Examples

- `ls ~/csci2132/lab1/H????World.java`
- `ls H*`
- `ls [!A-Z]*`
- `ls */*/*.java`
- `ls *.java */*.java`
- `echo .*`
  (`echo` prints out its command line arguments, useful in scripts)
- `cat *.txt > allfiles`

Regular Expressions

- Patterns used to match strings
- Used in fast and flexible text search tools
- Name comes from regular sets defined by Stephen Kleene
- Can be matched using deterministic finite automaton (DFA)
- Kleene's notation implemented in `QED` editor to match patterns (author Ken Thompson)
- Thompson later added this to the UNIX editor `ed`
- Led to the tool `grep`
  (Name comes from `ed` command `g/re/p`: global search for regular expression and print matching lines.)

Reading about Regular Expressions

- The Unix book
  - Chapter 3, “Filtering Files” (page 84)
  - Appendix, “Regular Expressions” (page 665)

Two Types of Regular Expressions

Basic regular expressions follow exactly the definition of regular sets by Kleene and can be matched using a DFA.

Extended regular expressions add extensions that

- Make regular expressions more powerful
- Cannot be matched using a DFA but …
- … can still be matched efficiently.

Basic Regular Expressions

- Made up of characters and metacharacters:
  - **Metacharacters:** `. ( ) [ ] * ? ^ $ \`
  - Anything that is not a metacharacter matches itself

Metacharacters:

- `.` matches any character
- `[ … ]` matches a character class analogously to wildcards
  (metacharacters are not special; negation using only `^`, not `!`)
- `(expr)` matches the `expr` (grouping)
- `expr*` matches any sequence of strings that match `expr`
- `expr?` matches 0 or 1 string that matches `expr`
- `\char` matches `char` even if `char` is a metacharacter
- `^` matches the beginning of the line
- `$` matches the end of the line

Examples of Basic Regular Expressions

- One or more spaces: “␣␣*”
- Empty line: “^$”
- Formatted dollar amount: “\$[0-9][0-9]*\.[0-9][0-9]”

Filters

A **filter** is a program that reads text from `stdin`, transforms it, and outputs the result to `stdout`.

Often used as elements of pipelines.

`grep`

`grep` is a filter that reads its input line by line and prints all lines that match a given pattern

Input:

- `stdin` if no files given on command line
- Otherwise, the listed files

General use: `grep [options] <pattern> [files]`

`grep` Options

- None: Pattern is interpreted as a basic regular expression
- `-E`: Pattern is interpreted as an extended regular expression
- `-F`: Pattern is interpreted as a fixed string
- `-n`: Precede each output line by its line number
- `-i`: Ignore case (lowercase/uppercase) when looking for matches
- `-v`: Output the lines that do not match
- `-w`: Restrict matches to whole words

grep Example

Consider the following file prices:

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

If we enter

```
$ grep ‘\$[0-9][0-9]*\.[0-9][0-9]’ prices
```

what is the output?

```
Chocolate $1.23 each
Jacket $278.00</pre>
<pre>$44.00
```

Another grep Example

The file /usr/share/dict/linux.words contains a dictionary of English words.

What grep command can we use to find all 5-letter words that start with ‘a’ or ‘b’ and end with ‘b’?

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

What grep command can we use to find all words that start with ‘a’ or ‘b’ and end with ‘b’?

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

What command do I add to my pipeline to count how many such words there are?

```
$ grep ‘^[ab].*b$’ /usr/share/dict/linux.words | wc -l
```

Extended Regular Expressions

Every basic regular expression is an extended regular expression.

Additional features:

- More repetition specifiers:
  - `expr?`: Match expression `expr` 0 or 1 time
  - `expr+`: Match expression `expr` at least once
  - `expr{m}`: Match expression `expr` exactly `m` times
  - `expr{m,n}`: Match expression `expr` between `m` and `n` times
  - `expr{,n}`: Match expression `expr` up to `n` times
  - `expr{m,}`: Match expression `expr` at least `m` times
- Back references:
  - `(subexpr1) ... (subexpr2) ... \2\1`: `\1` and `\2` match copies of the strings that matched the subexpressions `subexpr1` and `subexpr2` enclosed in parentheses

Examples of Extended Regular Expressions

- A string that consists of one or two digits followed by at least one letter:
  - `[0-9]?[0-9][a-z]+`
  - `[0-9]{1,2}[a-z]+`
- At least one occurrence of `Mon`, `Wed` or `Fri`:
- An IP address:
- A string that ends with the same two characters it starts with, in reverse order:

Similarities and Differences Between Wildcards and Regular Expressions

Most of the time, wildcards are good enough for file matching:

- All Java files:

```
$ ls *.java
$ ls | grep ‘*.java$’
```

Some patterns cannot easily be matched using wildcards but can be matched using regular expressions:

- All files whose names contain exactly one dash:

```
$ ls | grep ‘^[^-]*-[^-]*$’
```