Systems Programming & Scripting

Lecture 12: Introduction to Scripting & Regular Expressions

Goals of the Scripting Classes

- Learn how to easily do common operations on the command-line
- Learn how to generate simple scripts of repetitive activities
- Understand the structuring constructs available in the bash scripting language

Non-Goals of Scripting Classes

- Give an introduction to Unix commands.
- Cover the full-range of bash
- Compare bash with other scripting languages (tcsh, python, …)
- Advocate scripting for large scale programming

Introduction to Shell Scripting

- Scripts are sequences of repetitive commands, usually executed on the command-line.
- Scripting languages provide only an impoverished set of programming abstractions
- A focus of scripting languages is at easily combining simple commands to perform a more complex job.
- Powerful tools to achieve that are 'pipes' and 'regular expressions'.

Regular Expressions

- *Regular expressions* provide a powerful, efficient and flexible text processing technique.
- They form the basis of text and data processing tools.
- They also commonly used to select files.

History of Regular Expressions

- Based on mathematical notation developed by McCulloch and Pitts to describe neural networks.
- Formally introduced in 1956 by Kleene in his paper “Representation of Events in Nerve Nets”
- Ken Thompson, the inventor of Unix, used regular expressions in search algorithms.
- Their first practical use was in the Unix editor *qed.*

Using Regular Expressions

- Integrated in many tools and languages
  - vi, grep, Perl, PHP.
- Facilitates a search engine.
  - Match upper and lower case.
  - Either or string matching.
  - Quantify character repeats.
  - Match classes of characters.
  - Match any character.
  - Expressions can be combined.
  - You can match anything using regular expressions.
- Syntax is simple.

The Regular Expressions language

- Regular expressions are constructed using two types of characters:
  - Special characters or meta characters
  - Literal or normal text.
- Can think of regular expressions as a language:
  - Grammar: meta characters.
  - Words: literal text.

Basic Syntax

- . The dot matches any single character
  - E.g. ab. matches aba, abb, abc, etc.
- [ ] A bracket expression matches a single character contained within the bracket.
  - E.g. [abc] matches a, b or c
  - [a-z] specifies a range which matches any lowercase letter from a to z.
  - [abcx-z] matches a, b, c, x, y and z.

Cont. Basic Syntax

- [^] negation of [ ]
  - Matches a single character not contained in bracket.
  - E.g. [^abc] matches any character other than a, b or c.
- ^ matches the starting position of a string.
- $ matches the ending position of a string.
- * matches the previous element zero or more times.
  - E.g. abc*d matches abd, abcd, abccd, etc.

POSIX regular expressions

- [:alnum:] matches alpha-numerical characters
- [:alpha:] matches alphabetical characters
- [:digit:] matches numerals
- [:upper:] matches upper case characters
- [:lower:] matches lower case characters

Examples

- Searching (in) files
- Using find
- Using version control
- Doing stream processing with sed