

# CS3157: Advanced Programming

Lecture #3

Jan 25

Shlomo Hershkop  
*shlomo@cs.columbia.edu*

## Outline

- Feedback
- More Regular Expressions
- Scope
- Hashing
- File handling II
- Complex examples
  
- Reading: Chapter 4,5 (pg-167)

2

## Feedback from last class

- Slide posting
  - Will post slides within 24 hrs after class
  - Outside code will be also posted (links)
  - Reason for not posting prior to class
- General Pace
  - Will try to make it easier to take notes
  - Will divide information so easier to digest
  - Will be very technical at certain points...you will thank me later on when trying to solve labs
  - Will do more elaborate examples

3

## Announcement

- Please see web page for unix training by acis
  
- Will post office hours later tonight
- My office hours: t/th 12-1pm
- Tae M 9:50 - 10:50, T 9:50 - 10:50

4

## Regular Expression

- Review
  - Basics
  - Advanced
- More in examples
  
- So what exactly is a regular expression?

5

## Regular Expression in perl

- Trying to represent patterns to perl
- Very powerful since we can define our program behavior based on general pattern definitions
- Many many shortcuts available

6

## Simplest

- Simplest regular expression is a literal string match

```
if ($name =~ m/white house/ ) {  
  
    do something  
  
}
```

7

## Regular Expressions

- complex regular expressions use *metacharacters* to describe various options in building a pattern.

- `\`
  - Escape character
- `.`
  - Match any single character

- Full list:  
`\ | ( ) [ ] { } ^ $ * + ? .`

8

## Escape shortcuts

- `\w` Match "word" character (alphanumeric plus "\_")
- `\W` Match non-word character
- `\s` Match whitespace character
- `\S` Match non-whitespace character
- `\d` Match digit character
- `\D` Match non-digit character

9

## Other escape codes

- `\t` Match tab
- `\n` Match newline
- `\r` Match return
- `\f` Match formfeed
- `\a` Match alarm (bell, beep, etc)
- `\e` Match escape

10

## Regular expression attributes

- `g` = match globally (all instances)
- `i` = do case insensitive matching
- `e` = evaluate right side as an expression
- `s` = let `.` match newlines
- `m` = `$` and `^` can refer to inside newlines
- `c` = compliment

11

## usages

- 1) `if ( $line =~ /\s.*\S$/ ) {....}`
- 2) `if (not $line =~ /cs3157/ ) {...}`  
`if( $line !~ /cs3157/ ) {....}`
- 3) `while ( $line =~ /\w \w$/`

12

## groups

To allow groups of alternative choices

```
if($string =~ /(A|E|I|O|U|Y)/i)
{ print "String contains a vowel!\n"; }
```

```
if($string =~ /(Clinton|Bush)/)
{ print "President sir!\n"; }
```

13

## Character choices

we can also specify character choices:

```
if( $string =~ /[AEIOUY]/i )
{ print "String contains a vowel!\n"; }
```

Can also specify ranges

```
if( $string =~ /^[^a-e]/l ) {
something
}
```

14

## Groups II

- To allow us to reference for selection and substitution
- Each group can be referred to by scalar \$1, \$2, \$3 ....

Example

- "From [s@aol.com](mailto:s@aol.com) Wed Jun 3 12:12:12 2005"
- If(/^From (.\*) (... ) (... ) (.\*)\$/)

15

## quantifiers

- ba\*b
- ba{3,5}b
- ba{2}b
- /(ab){4,}/

16

## Shortcut 1

- We can say
  - [abcdefgh]
  - [a-h]
  - [a-h] {1,4}

17

1. How would we look for a phone number?
2. What about a street address?

18

## Quick question

- How to indicate the period since period matches any character?

19

```
open MAIL, "mail.txt" or die "cant open
file\n";

while(<MAIL>) {
    print if m/^From: /;
}
```

20

```
open MAIL, "Mail.txt" or die "can't  
open mail file\n";
```

```
while (<MAIL>) {  
    if (/^([^\:]+): ?(.+)\$/ ) {  
        print "Header $1 has val $2\n";  
    }  
}
```

21

## Other shortcuts

```
$name = "advanced programming class"
```

```
if($name =~ /programming/){  
    print `$ ` ;  
    print `$& ` ;  
    print `$' ` ;  
}
```

22

## What is?

```
if($string =~  
    m/^\S+\s+(Hershkop|Stolfo|Aho)/i)  
{print "$string\n";
```

23

## Task

- Given a directory listing in dos, how to backup any file from 2004 ?

|            |          |         |               |
|------------|----------|---------|---------------|
| 08/04/2004 | 05:00 AM | 256,192 | winhelp.exe   |
| 08/04/2004 | 05:00 AM | 283,648 | winhlp32.exe  |
| 08/03/2005 | 03:07 AM | 138     | wininit.ini   |
| 12/14/2005 | 03:05 PM | <DIR>   | WinSyS        |
| 12/14/2005 | 03:03 PM | 12,033  | WMCSetup.log  |
| 01/10/2006 | 09:00 PM | 23,901  | wmsetup.log   |
| 01/10/2006 | 09:00 PM | 459     | wmsetup10.log |

24

```

while(<STDIN>)
{
  my($line) = $_;
  chomp($line);
  if($line !~ /<DIR>/)
  {
    /** only lines with dates at position 28 and (long)
    # filename at pos 44 **

    if ($line =~ /.{28}(\d\d)-(\d\d)-(\d\d).\{8}(.)$/)
    {
      my($filename) = $4;
      my($ymmdd) = "$3$1$2";
      if($ymmdd lt "971222")
      {
        print "move $filename \\backup\n";
      }
    }
  }
}

```

25

## substitutions

- s/pattern/pattern/
- Instead of return t/f we return number of matches
- And will change the applied target

26

## transliteration

- tr/search\_list/replacement\_list/
- -c all characters not in the search list
- -d anything without replacement ...delete
- -s squash duplicates

27

- What is scope?

28

## scope

- Default scope is main
- \$name can also be referred to as \$main::name
- package NAMESPACE
  - Within any block of code, can declare that the rest of the code will belong to a specific namespace

29

## Scope II

- my  
declares the variable and value local to the current scope
- our  
confines the name to local scope
- local  
confines the value to local scope

30

- Remember to place than one variable in parenthesis!!

31

## Security

- Should use pattern matches as a security check on input
- Example:

```
unless ( $year =~ /^d\d$/ ) {  
    die ("problem with year input!");  
}
```

32



## hashes

- A hash function is a function that converts an input from a (typically) large domain into an output in a (typically) smaller range
- Example:
  - Map each name in the class to a somewhat unique number
- Collision = when different keys map to the same output.

33

## Use of hashes

- Hash tables
  - Data structure
  - Unordered list, fast lookup
- Cryptography
- Data processing

34

- Sample code: readfile.pl

35

## Useful commands

- Split

```
split /PATTERN/,EXPR,LIMIT
split /PATTERN/,EXPR
split /PATTERN/
```

```
split Splits a string into a
list of strings and returns that
list. By default, empty leading
fields are preserved, and empty
trailing ones are deleted. ....
```

36