Perl Tutorial

Study group material prepared by Adrian

5th February 2005

1 Background

- Pratical Extraction and Reporting Language, by Larry Wall
- http://www.perl.com/ is the official site of Perl, maintained by O'Reilly and http://www.perldoc.com/ contains documentations
- Current versions: Perl 5.8 / Perl 6
- Books: Many are published by O'Reilly & Associates.
  - Steve Oualline, *Perl for C Programmers*, New Riders, 2003
    An excellent text for new users
  - Ellen Siever et al, *Perl in a Nutshell*, O'Reilly & Associates, 2001
    The encyclopaedia of Perl
- Strength of Perl: Fast to code, built-in hash, tons of built-in functions and modules, *regular expressions*

2 Command line

- Let me give you a million "thanks! ":
  `$ perl -e 'print "thanks! "x1000000'`
- Show me the file `test.txt`, but replace all `ABC` to `DEF`
  `$ perl -pe 's/ABC/DEF/g' test.txt`
- That's good enough! How about replace the content and save it?
  `$ perl -pi -e 's/ABC/DEF/g' test.txt`
- Further command line stuff: read manpage of Perl or Nutshell Chapter 3

3 Language

- Think it as C:
  - Semicolon `;` ends an statement
  - Braces `{}` encloses compound statements
  - Whitespaces separates tokens
  - You can present number in any form you can understand
  - You can enclose a string with single `'` or double `"` quotes (single-character string *is* a string)
  - *but*, `#` starts a comment until end-of-line
- Perl is easier to handle then C
  - Simple variables = scalar, e.g. `$var`
  - Vectors or 1D ordered list = array, e.g. `@var`
  - A set of key-value pair = hash, e.g. `%var`
  - An item of an array: `$var[1]`
  - An item of a hash: `$var{'cuhk'}`

Example 1: Variables

```
#!/usr/bin/perl
use strict;
use warnings;

my $name='Adrian';
my $num=1.5;
my @array = ('Kai','Adrian','Humphrey','Gao Yan');
my %hash = ('Apple' => 'red', 'Banana' => 'yellow');

$num++;
print "I am $name\n";
print "Apple is ".$hash{'Apple'}."\n";
print "$num - $array[1]\n";
```

```
$ perl ex1.pl
I am Adrian
Apple is red
2.5 - Adrian
```

- First line specifies the interpreter
- `use strict` make Perl care about variable declarations, which avoids bugs-by-typos and `use warning` ask Perl to warn everything, which helps debug
- `++` increases a number by 1, whether you are an integer or not
- Numerical or string values are all variables! Perl don't differentiate them
- Joining strings by a dot
- You can even use `printf` as in C

3.1 Control structures: if statements

- `if (`*EXPR*`) {`*BLOCK*`} else {`*BLOCK*`}`
- `unless (`*EXPR*`) {`*BLOCK*`} else {`*BLOCK*`}`
- `if (`*EXPR*`) {`*BLOCK*`}`
  `elsif {`*BLOCK*`}`
- `if (`*EXPR*`) {`*BLOCK*`}`
  `elsif {`*BLOCK*`}`
  `....`
  `else {`*BLOCK*`}`
- They are "postfix" version of conditioning statements:

  ```
  $n=1 if ($n==0);
  $n=1 unless ($n!=0);
  ```

3.2 Control structures: Loops

```
open INFILE,"<test.txt";
open OUTFILE,">test.old";
while (<INFILE>) {
   print OUTFILE, "$_\n";
}
close INFILE; close OLDFILE;
```

- This is an example of copying a file, `<INFILE>` reads a line from the file handle
- `$_` is a *special variable* in Perl that correspond to the unnamed data, i.e. a line in this example
- `<INFILE>` returns false when end-of-file, hence the while loop ends
- we also have `do {...} until (...)` and `do {...} while (...)` loops

- ```
  $num=1;
  for ($i = 1; $i < 10; $i++) {
      $num *= $i;
  }
  ```
  - This calculates the factorial 10!
  - Just like C, right?!
- ```
  foreach $var (@list) {
      print "$var\n";
  }
  ```
  - This loop prints the items of `@list` each in a line
  - `foreach` loop scans the whole array and put the item into a variable in each round
- ```
  $n++ while ($n<100);
  $n++ until ($n>=100);
  ```
  - They are “postfix” version of loops

3.3 Control structures: Breaking a loop

- `last` is same as `break` in C
- `next` is same as `continue` in C
- `redo` is same as `continue` in C except it do not re-evaluate the conditions as C does

4 Special variables

- `$_` is the “default” variable, for example, they are the same:

```
foreach ("Apple","Orange","Banana") {
    print;
};
foreach ("Apple","Orange","Banana") {
    print $_;
};
```

- `$_` is the variable automatically applied when a function needs an argument but you didn’t gave anything
- `@ARGV` is the array containing all arguments to the script, use as `char**argv` in C

```
#!/usr/bin/perl
use strict;
use warnings;

for (my $i=0;$i<=$#ARGV;$i++) {
    print "$i: $ARGV[$i]\n";
};
```

  - `$#ARGV` gives the size of an array
- `@_` is similar to `@ARGV`, but it is the array for arguments to a function

```
#!/usr/bin/perl
use strict;
use warnings;

showall(@ARGV);

sub showall
{
for (my $i=0;$i<=$#_;$i++) {
print "$i: $_[$i]\n";
};
};
```
```

- %ENV is a hash to environments. Used extensively in CGI!
- STDIN, STDOUT, STDERR are not variables, but file handlers

5 Operators

Ordinary operators (precedence order). See "Nutshell" Chapter 4.5 for more.

Associativity	Operators
Left	Terms and list operators (leftward)
Left	-> (method call, dereference)
Nonassociative	++ -- (autoincrement, autodecrement)
Right	** (exponentiation)
Right	! ~ \ and unary + and - (logical not, bit-not, reference, unary plus, unary minus)
Left	=~ !~ (matches, doesn't match)
Left	* / % x (multiply, divide, modulus, string replicate)
Left	+ - . (addition, subtraction, string concatenation)
Left	<< >> (left bit-shift, right bit-shift)
Nonassociative	Named unary operators and file-test operators
Nonassociative	< > <= >= lt gt le ge (less than, greater than, less than or equal to, greater than or equal to, and their string equivalents.
Nonassociative	== != <=> eq ne cmp (equal to, not equal to, signed comparison, and their string equivalents)
Left	& (bit-and)
Left	| ^ (bit-or, bit-xor)
Left	&& (logical AND)
Left	|| (logical OR)
Nonassociative	.. ... (range)
Right	?: (ternary conditional)
Right	= += -= *= and so on (assignment operators)
Left	, => (comma, arrow comma)
Nonassociative	List operators (rightward)
Right	not (logical not)
Left	and (logical and)
Left	or xor (logical or, xor)

6 Functions

Scalar manipulation	chomp, chop, chr, crypt, hex, index, lc, lcfirst, length, oct, ord, pack, q//, qq//, reverse, rindex, sprintf, substr, tr///, uc, ucfirst, y///
Regular expressions and pattern matching	m//, pos, qr//, quotemeta, s///, split, study
Numeric functions	abs, atan2, cos, exp, hex, int, log, oct, rand, sin, sqrt, srand
Array processing	pop, push, shift, splice, unshift
List processing	grep, join, map, qw//, reverse, sort, unpack
Hash processing	delete, each, exists, keys, values
Input and output	binmode, close, closedir, dbmclose, dbmopen, die, eof, fileno, flock, format, getc, print, printf, read, readdir, rewinddir, seek, seekdir, select, syscall, sysread, sysseek, syswrite, tell, telldir, truncate, warn, write
Fixed-length data and records	pack, read, syscall, sysread, syswrite, unpack, vec
Filehandles, files, and directories	chdir, chmod, chown, chroot, fcntl, glob, ioctl, link, lstat, mkdir, open, opendir, readlink, rename, rmdir, stat, symlink, sysopen, umask, unlink, utime
Flow of program control	caller, continue, die, do, dump, eval, exit, goto, last, next, redo, return, sub, wantarray
Scoping	caller, import, local, my, package, use
Miscellaneous	defined, dump, eval, formline, local, my, prototype, reset, scalar, undef, wantarray
Processes and process groups	alarm, exec, fork, getpgrp, getppid, getpriority, kill, pipe, qx//, setpgrp, setpriority, sleep, system, times, wait, waitpid
Library modules	do, import, no, package, require, use
Classes and objects	bless, dbmclose, dbmopen, package, ref, tie, tied, untie, use
Low-level socket access	accept, bind, connect, getpeername, getsockname, getsockopt, listen, recv, send, setsockopt, shutdown, socket, socketpair
System V interprocess communication	msgctl, msgget, msgrcv, msgsnd, semctl, semget, semop, shmctl, shmget, shmread, shmwrite
Fetching user and group information	endgrent, endhostent, endnetent, endpwent, getgrent, getgrgid, getgrnam, getlogin, getpwent, getpwnam, getpwuid, setgrent, setpwent
Fetching network information	endprotoent, endservent, gethostbyaddr, gethostbyname, gethostent, getnetbyaddr, getnetbyname, getnetent, getprotobyname, getprotobynumber, getprotoent, getservbyname, getservbyport, getservent, sethostent, setnetent, setprotoent, setservent
Time	gmtime, localtime, time, times

The above are functions listed by category. Search "Perl functions" in Google gaves you a lot of links of the same content, i.e. the `perlfunc` page. One of them is:
`http://www.sunsite.ualberta.ca/Documentation/Misc/perl-5.6.1/pod/perlfunc.html`

7 Regular Expressions

- In Perl, we have 3 regex calls
  - `m/regex/` search for a pattern
  - `s/regex/replace/` replace a pattern with some specified string
  - `tr/pattern/pattern/` replace a set of character to another set of character
- Example:

```
#!/usr/bin/perl
use strict;
use warnings;

my $string1="Hello World!";

print "String1 starts with an H\n" if ($string1=~ m/^H/);

$string1 =~ tr/a-zA-Z/n-za-mN-ZA-M/;
print "String1 in ROT-13: $string1\n";

$string1 =~ tr/a-zA-Z/n-za-mN-ZA-M/;
print "String1 in ROT-13 again: $string1\n";

$string1 =~ s/Hello/Hi to the/;
print "String1 is: $string1\n";
```

```
$ perl ex4.pl
String1 starts with an H
String1 in ROT-13: Uryyb Jbeyq!
String1 in ROT-13 again: Hello World!
String1 is: Hi to the World!
```

- Among them, the most useful is the substitution, `s///`. You can add some modifiers as well:

Modifier	Meaning
`s///g`	Replace globally, i.e., all occurrences.
`s///i`	Do case-insensitive pattern matching.
`s///e`	Evaluate the right side as an expression.
`s///o`	Only compile pattern once.
`s///m`	Treat string as multiple lines.
`s///s`	Treat string as single line.

- Regular expression has three major variations, namely, POSIX standard (grep), POSIX extended (egrep), Perl
- Usually, a string is what it is, but some characters are bearing some special meanings:

Metacharacter	Meaning
`\`	Escapes the character(s) immediately following it
`.`	Matches any single character except a newline (unless `/s` is used)
`^`	Matches at the beginning of the string (or line, if `/m` used)
`$`	Matches at the end of the string (or line, if `/m` used)
`*`	Matches the preceding element 0 or more times
`+`	Matches the preceding element 1 or more times
`?`	Matches the preceding element 0 or 1 times
`{...}`	Specifies a range of occurrences for the element preceding it
`[...]`	Matches any one of the class of characters contained within the brackets
`(...)`	Groups regular expressions
`|`	Matches either the expression preceding or following it

- where those quantifiers has the following interpretations:

Maximal	Minimal	Allowed Range
`{n,m}`	`{n,m}?`	Must occur at least `n` times but no more than `m` times
`{n,}`	`{n,}?`	Must occur at least `n` times
`{n}`	`{n}?`	Must match exactly `n` times
`*`	`*?`	0 or more times (same as `{0,}`)
`+`	`+?`	1 or more times (same as `{1,}`)
`?`	`??`	0 or 1 time (same as `{0,1}`)

- and escape (\) means the following:

Code	Matches
`\a`	Alarm (beep)
`\n`	Newline
`\r`	Carriage return
`\t`	Tab
`\f`	Formfeed
`\e`	Escape
`\007`	Any octal ASCII value
`\x7f`	Any hexadecimal ASCII value
`\cx`	Control-x
`\d`	A digit, same as `[0-9]`
`\D`	A nondigit, same as `[^0-9]`
`\w`	A word character (alphanumeric), same as `[a-zA-Z_0-9]`
`\W`	A nonword character, `[^a-zA-Z_0-9]`
`\s`	A whitespace character, same as `[\t\n\r\f]`
`\S`	A non-whitespace character, `[^\t\n\r\f]`
`\b`	Matches at word boundary (between `\w` and `\W`)
`\B`	Matches except at word boundary
`\A`	Matches at the beginning of the string
`\Z`	Matches at the end of the string or before a newline
`\z`	Matches only at the end of the string
`\G`	Matches where previous `m//g` left off

- That's enough, let's read some examples

8 More?

- Perl is renowned for its power in text manipulation
- You can even compile Perl into binary for closed-source distribution, by the `perlcc` command
- Many languages (e.g. PHP, Java) has implemented built-in regex functions to mimic Perl
- C doesn't for its simplicity, but there are alternatives
  - Use POSIX library for POSIX regex, whose function is limited and complicated to use
  - Use PCRE, Perl Compatible Regular Expressions, a library available for all UNIX platforms (Windows user: install a DLL for yourself)
  - If you use C++, use PCME, *PCRE Made Easy*, by using C++ string type, PCME is even easier to use then PCRE
- Perl is easy to learn, but hard to be a master
  (but a master writes only obscure code, e.g. two MIT guys writes a DVD descrambler like this in 2001:)

```
#!/usr/bin/perl
# 472-byte qrpff, Keith Winstein and Marc Horowitz <sipb-iap-dvd@mit.edu>
# MPEG 2 PS VOB file -> descrambled output on stdout.
# usage: perl -I <k1>:<k2>:<k3>:<k4>:<k5> qrpff
# where k1..k5 are the title key bytes in least to most-significant order

s''$/=\2048;while(<>){G=29;R=142;if((@a=unqT="C*",_)[20]&48){D=89;_=unqb24,qT,@
b=map{ord qB8,unqb8,qT,_^$a[--D]}@INC;s/...$/1$&/;Q=unqV,qb25,_;H=73;O=$b[4]<<9
|256|$b[3];Q=Q>>8^(P=(E=255)&(Q>>12^Q>>4^Q/8^Q))<<17,O=O>>8^(E&(F=(S=O>>14&7^O)
^S*8^S<<6))<<9,_=(map{U=_%16orE^=R^=110&(S=(unqT,"\xb\ntd\xbz\x14d")[_/16%8]);E
^=(72,@z=(64,72,G^=12*(U-2?O:S&17)),H^=_%64?12:0,@z)[_%8]}(16..271))[_]^((D>>=8
)+=P+(~F&E))for@a[128..$#a]}print+qT,@a}';s/[D-HO-U_]/\$$&/g;s/q/pack+/g;eval
```