

Perl for the Lazy Programmer

Timothy A. Gonsalves
TeNeT Group
Dept. of Computer Science & Engg
I.I.T., Madras

tag@tenet.res.in

What is Perl?

- An interpreted scripting language
- Easy access to utilities and system calls
- Relaxed syntax and semantics for small programs
- Strict checking and OO for large programs
- Good for text processing
- **Quick and dirty *throwaway programs***
- Designed for growth
 - Variables start with \$, @, %
 - => can add keywords

Start small, learn as you grow

*Split fields on space
into array F*

Perl One-liners

Regular expr match

```
>ps aux| perl -ne 'print if /emacs/'
```

apply expr one line at a time to input, like grep

```
>ls -l|perl -ane 'print if $F[4]>5000'
```

examine 5th field in each line, better than grep

```
>ls -l|perl -ane \  
'print if $F[4]>5000 && /May/'
```

```
>ps aux| perl -pe ''
```

print each line, like cat

*Execute at end of
all input*

```
>ps aux| perl -pe '$c++; \  
END print "$c processes\n"'
```

apply expr to each line while printing, better than cat

*Split fields on :
into array F*

Perl One-liners

Print fullname = username for all system users. Use /etc/passwd:

```
root:x:0:0:root:/root:/bin/bash
```

```
rpm:x:37:37::/var/lib/rpm:/sbin/nologin
```

```
tag:x:500:500:Timothy A. Gonsalves:/home/tag:/bin/csh
```

```
guest:x:505:505:Guest:/home/guest:/bin/bash
```

```
>perl -F: -ane 'print "$F[4]=$F[0]\n"\n                if $F[4]' /etc/passwd
```

```
root=root
```

```
Timothy A. Gonsalves=tag
```

```
Guest=guest
```

Perl Scripts

File: userinfo

```
#!/usr/bin/perl -n
($ac, $fn) = (split(':', '))[0, 4];
print "$fn = $ac\n";
```

```
>chmod a+x userinfo
>userinfo /etc/passwd
root=root
Timothy A. Gonsalves=tag
Guest=guest
```

```
>perl userinfo /etc/passwd
```

*Array
assignment*

*Slice elements 0
and 4 from array*

*Split line into
array of fields*

Perl 'finger'

```
>pfinger guest xfs
  guest: Guest
  xfs: X font server
```

File handle

Need input from command-line and /etc/passwd file

File: pfinger

Error handling

```
open(PASSWD, "/etc/passwd") ||
    die "Can't open /etc/passwd";
while (<PASSWD>) {
    foreach $name (@ARGV) {
        if (/^$name/i) {
            ($ac, $fn) = (split(':', $fn))[0,4];
            ($fn)      = split(',', $fn);
            print "$ac: $fn\n";
        }
    }
}
```

Cmd line arg vector

Case insensitive match

Perl 'grep'

```
>pgrep globalVar *.c  
  main.c: globalVar = 0;  
  util.c: if (globalVar > 5)  
  ...
```

*Input from list
of files*

```
>ps -aux |pgrep root
```

Input from stdin

File: pgrep

```
$pat = shift;      # @ARGV is implicit for shift  
while (<>) {       # Same as "perl -n"  
  if (/ $pat/) {  
    print "$ARGV: $_"; }  
}
```

*<>: read from stdin or
from each file on
command line*

Pcount: count processes per user

- For each line of ps output, update list of users and the process count for that user
At the end, print users and counts
 - Tedious using regular arrays
 - Use Perl's *associative arrays* or *hashes*

File: pcount

```
foreach (`ps auxh`) {  
    ($name) = split(' ');  
    $count{$name}++; }  
foreach $name (sort keys %count) {  
    print "$name\t$count{$name}\n"; }  
}
```

*`cmd` -- run
cmd, value is
its stdout*

*Default
variable is \$_
8/1/08*

Pcount: count processes per user

Reverse sorting order:

```
foreach $name (reverse sort keys %count)
{ print "$name\t$count{$name}\n"; }
```

Print in order of decreasing count:

```
foreach $name (reverse sort \
    {$count{$a} <=> $count{$b}} \
    keys %count)
{ print "$name\t$count{$name}\n"; }
```

Optional first argument to sort is the comparison function. \$a and \$b are the two values being compared by sort.

Perl Features

- Hashes or associative arrays
- System programs
- Regular expressions
- File I/O
- Modules
- CPAN

System Programs

- 3 ways to run system programs:

```
$r = system("gcc myprog.c")
```

- \$r gives exit code of gcc
- gcc messages go to stdout of the Perl script

```
$files = `ls *.c`
```

- Exit code is available in \$?

System Programs

Process as a file handle:

- **For input**

```
open(MAILER, "|/bin/mail $to") || die "Cant";  
print MAILER $msg";
```

- **For output**

```
close MAIL; open(LS, "/bin/ls -l|") || die \  
"Cant";  
  
while (<LS>) { print if /2005/;}  
close LS;
```

Regular Expressions

- Search for inexact string:
Ramaswamy or Ramasami or Ramasamy
`/Ramasw?am[iy]/`
- Composed of patterns, each ≥ 1 character
- Like grep, egrep, Posix and Gnu together

RegExp: Syntax

- \ Quote the next metacharacter
- ^ Match the beginning of the line
- .
- \$ Match end of line (or before newline at the end)
- | Alternation
- () Grouping
- [] Character class

RegExp: Syntax

\b – word boundary

\d – digit

\s – space

\w – alphanumeric, ''

\B, \D, \S, \W – *not* **\b, \d, \s, \w**

? - match 0 or 1

***** - 0 or more repetitions

+ - match 1 or more repetitions

{7,10} – between 7 and 10 repetitions

{7,} - ≥ 7 repetitions

{7} - exactly 7 repetitions

RegExp: Examples

- Input contains “Time: 11:27:45”

```
($h, $m, $s) = /Time: (..):(..):(..)/;
```

```
$totalSecs = ($h*60+$m)*60+$s;
```

- Each substring that matches () is assigned to one variable in a list

- More robust:

```
if (/Time: (\d\d):(\d\d):(\d\d)/) {
```

```
    $h = $1;
```

```
    $m = $2;
```

```
    $s = $3; }
```

```
$totalSecs = ($h*60+$m)*60+$s;
```

RegExp: Examples

- Extract the subject from an email:

```
if (/^Subject:\s+(.*)$/ {  
    $subject = $1;  
}
```

OR

```
/^Subject:\s+(.*)$/ && $subject = $1;
```

OR

```
($subject) = /^Subject:\s+(.*)$/;
```

- Matching in a variable:

```
$t = "Time: 12:45:34";  
($h) = $t =~ /Time: (..):(..):(..)/;  
print "$h hours";
```

- Can use any character as delimiter in place of //

RegExp: Name example

- Swap username and fullname:

root System Admin → System Admin = root

- File: pswap

```
@words = split(" ");
```

```
print "@words[1,#words] = $words[0]";
```

- >cp input input.tmp

```
>pswap <input.tmp >input
```

```
>rm input.tmp
```

- Using regexp:

```
perl -ni.tmp -e \
```

```
's/(\w+)\s+(\S.*)$/\2 = \1/' input
```

- s/r1/r2/ substitute r1 by r2

- -i edit file *in place*

RegExp: Mail log Example

- Mail log contains one line per message:
.... from = "tag", ... size = 53467, ...
- Use arrays `users[string]` and `bytes[int]`
split each line to extract from and size
if from not found in `users[]`, add it to end
find index `i` of from in `users[]`
`bytes[i] += size`
- Dozens to hundreds of lines of code!

RegExp: Mail log Example

- Mail log contains one line per message:
.... from = "tag", ... size = 53467, ...
- Using regexp and hashes:

```
while (<MAILLOG>) {
    if (($f, $s) = /from = "[^"]).*\  
                size = (\d+)/) {
        $bytes{$f} += $s;
        $msgs{$f}++;
    }
}
foreach $f (sort keys %bytes) {
    print "$f: $bytes{$f} in $msgs{$f}\n";
}
```

RegExp: Edit C files

- Change “... if (var = expr) ...” to “... if (var == expr) ...”

```
>perl -pi.bak -e \  
  's/(.*if \(\w+\) ={1} (.*$) \  
    /$1 == $2/' *.c
```

- Replace ' ' by '\s*' for robustness

Lazy Perl Programmers Use
Regular Expressions Liberally!

File I/O

- `open` – for read (default), write or append:

```
open(INFO, "datafile") || die("can't: $!");  
open(INFO, "<datafile") || die("can't: $!");  
open(RESULTS, ">runstats") || die("can't: $!");  
open(LOG, ">>logfile") || die("can't: $!");
```

\$! yields the system error number of error string depending on context

- **Mixing read and write on existing file:**

```
open(WTMP, "+< /usr/adm/wtmp")
```

- `opendir`, `readdir`, `closedir` for directories
- Stdout buffered by default
Flush stdout after every write: `$| = 1;`

Modules

CPAN

Advanced Perl

- Eval
- Objects
- References
- Writing modules
- GUI: Perl/Tk

GUI: Perl/Tk

Resources

- Linux: Perl is pre-installed on most distributions
- Windows: ActivePerl recommended, freely downloadable with commercial support
<http://www.activestate.com/products/activeperl/>
- CPAN: collection of a vast number of Perl modules.
Use `perl -MCPAN` to get a shell to search, download and install modules from www.cpan.org

Resources

- R.L. Schwartz, T. Phoenix, b.d. foy, *Learning Perl*, 4th ed., O'Reilly, 2005
The first book used by many Perl programmers
- Larry Wall, Tom Christiansen, Jon Orwant, *Programming Perl*, 3rd ed., O'Reilly, 2000
Definitive book about Perl, the language and culture by the authors of Perl. Covers Perl 5.6.
- S. Spainhour, E. Siever, N. Patwardhan, *Perl in a Nutshell*, 2nd ed., O'Reilly, 2002
Comprehensive reference including CGI, XML, networking, database interaction and GUI
- S. Cozens, *Advanced Perl Programming*, 2nd ed., O'Reilly, 2005
Explains introspection, overriding built-ins, extending Perl's object-oriented model, testing code for greater stability, etc. for sophisticated Perl programs.