Perl Programming

WeeSan Lee <weesan@cs.ucr.edu>
http://www.cs.ucr.edu/~weesan/cs183/

Roadmap

- How to Create a Perl Script?
- Here Documents
- Three basic data types: Scalars, Arrays, Hashes
- Branching
- Iterations
- File IO
- Function/Subroutine
- String Manipulation
- Regular Expression

How to Create a Perl Script?

- Edit hello.pl
  - #!/usr/bin/perl
  - print "Hello World!\n";
- Make hello.pl an executable
  - $ chmod +x hello.pl
- Run hello.pl
  - $ ./hello.pl
    - Hello World!

Here Documents

- #!/usr/bin/perl
- print <<EOF;
- Hello World!
- EOF

Scalar Variables

- $a = 1;
- $a = 1 + 2;
- $a = 2 ** 10;
- $a++;
- $a--;

- $a = "foo";
- $b = "bar";
- $c = $a . $b;
- $d = $a x 3;
- print $d;
- print "$a $b";

- $a=`date`;
- print "$a\n";

Comparisons

- Numbers
  - $a == $b # Is $a numerically equal to $b?
  - $a != $b
  - $a < $b
  - $a <= $b
  - $a > $b
  - $a >= $b
- Strings
  - $a eq $b # Is $a string-equal to $b?
  - $a ne $b # Is $a string-unequal to $b?

Arrays

- @fruit = ("apples", "oranges");
- print "$fruit[0]\n";
  - apples
- push(@fruit, "banana");
- print $fruit[2] . "\n";
  - banana
- $a = pop(@food);
- print $a
  - banana
- $len = @fruit;
- print $len
  - 2
- $str = "@fruit";
- print $str
  - apples oranges
- $" = "/";
- print "@fruit\n";
  - apples/oranges
- ($a, $b) = @fruit;
- print "$a - $b\n";
  - apples - oranges
- print "last index = $#fruit\n";
  - last index = 1
- $#fruit = 0;
- print @fruit
  - apples

Arrays

- @ARGV is a special array
  - $ARGV[0]: 1st command-line argument
  - $ARGV[1]: 2nd command-line argument
  - …

Hashes/Associative Arrays

```
%scores = ("Alice", 80,
           "Bob", 90,
           "Claire", 92,
           "David", 60);
print "The score for Claire = $scores{'Claire'}\n";

%scores = ("Alice"   => 80,
           "Bob"     => 90,
           "Claire"  => 92,
           "David"   => 60);
print "The score for Claire = $scores{Claire}\n";
```

Hashes/Associative Arrays

- foreach $i (keys %scores) {
- print "The score for $i = $scores{$i}\n";
- }
  - The score for Bob = 90
  - The score for Claire = 92
  - The score for Alice = 80
  - The score for David = 60

- print "\nAll the scores are:";
- for $i (values %scores) {
- print " $i";
- }
- print "\n\n";
  - All the scores are: 90 92 80 60

Hashes/Associative Arrays

- How to display all sorted by the students?
  - for $i (sort keys %scores) {
  - printf("%10s: %d\n", $i, $scores{$i});
  - }
    - Alice: 80
    - Bob: 90
    - Claire: 92
    - David: 60

Hashes/Associative Arrays

- How to display all sorted by the scores?
  - for $i (sort { $scores{$b} <=> $scores{$a} } keys %scores) {
  - printf("%10s: %d\n", $i, $scores{$i});
  - }
    - Claire: 92
    - Bob: 90
    - Alice: 80
    - David: 60

A subroutine for the sort function. $a and $b are the elements to be compared.
<=> is a special 3-way numeric comparison. Eg.
3 <=> 1 returns 1
1 <=> 1 returns 0
1 <=> 3 returns -1

Hashes/Associative Arrays

- while (($person, $score) = each(%scores)) {
- print "The score for $person = $score\n";
- }
  - The score for Bob = 90
  - The score for Claire = 92
  - The score for Alice = 80
  - The score for David = 60

Hashes/Associative Arrays

- exists() can be used to check the existence of a hash key
  - If (!exists($scores{weesan})) {
  - print "No score for weesan\n";
  - }
- %ENV is a special hash variable
  - print "$ENV{USER} is using $ENV{SHELL}\n";
    - weesan is using /bin/tcsh

Arrays vs. Hashes

- #!/usr/bin/perl
- @a = (0);
- %a = (0 => 1);
- print "\$a[0] = $a[0], \$a{0} = $a{0}\n";
  - What's the output?
  - $a[0] = 0, $a{0} = 1

Branching - if

- Syntax
  - **if** (*<condition>*) {
  - *<stmts>*
  - } **elsif** (*<condition>*) {
  - *<stmts>*
  - } **else** {
  - *<stmts>*
  - }

- Example
  - if (!$a) {
  - print "Empty string\n";
  - } elsif (length($a) == 1) {
  - print "Len = 1\n";
  - } elsif (length($a) == 2) {
  - print "Len = 2\n";
  - } else {
  - print "Len > 2\n";
  - }

Branching - unless

- Syntax
  - **unless** (*<condition>*) {
  - *<stmts>*
  - } **else** {
  - *<stmts>*
  - }

- Example
  - unless ($my_grade >= 60) {
  - print "I failed CS183!\n";
  - } else {
  - print "I passed CS183!\n";
  - }

Branching - switch or not?

```
$_ = <STDIN>;
chop($_);

SWITCH: {
   /[a-z]/ && do {
         print "$_ is a lower case.\n";
         last SWITCH;
   };
   /[A-Z]/ && do {
         print "$_ is a upper case.\n";
         last SWITCH;
   };
   /[0-9]/ && do {
         print "$_ is a number.\n";
         last SWITCH;
   };
   print "$_ is a punctuation.\n"
}
```

for loop

- Syntax
  - **for** (*<c style for loop>*) {
  - *<stmts>*
  - }
- Examples
  - for ($i = 0; $i < 10; ++$i) {
  - print "$i\n";
  - }

- Syntax
  - **foreach** *<var>* (*<list>*) {
  - *<stmts>*
  - }
- Example
  - foreach $i (0..7, 8, 9) {
  - print "$i\n";
  - }
  - foreach $i (@fruit) {
  - print "$i\n";
  - }

Range operator

while loop

Chop off the last char. of a string regardless. chomp() remove the trailing \n.

- Syntax
  - **while** (*<condition>*) {
  - *<stmts>*
  - }
- Example

Lower case function

```
print "Password? ";                # Ask for input
$a = <STDIN>;                      # Get input
chop $a;                           # Remove the newline
while (lc($a) ne "weesan") {
    print "sorry. Again? ";        # Ask again
    $a = <STDIN>;                  # Get input again
    chop $a;                       # Chop off newline again
}
```

until loop

- Syntax
  - **until** (*<condition>*) {
  - *<stmts>*
  - }
- Example

```
print "Password? ";              # Ask for input
$a = <STDIN>;                    # Get input
chop $a;                         # Remove the newline
until (lc($a) eq "weesan") {
    print "sorry. Again? ";      # Ask again
    $a = <STDIN>;                # Get input again
    chop $a;                     # Chop off newline again
}
```

do … while loop

- Syntax
  - **do** {
  - *<stmts>*
  - } **while** (*<condition>*);
- Examples
  - do {
  - print "Password? "; # Ask for input
  - $a = <STDIN>; # Get input
  - chop $a; # Chop off newline
  - } while (lc($a) ne "weesan"); # Redo while wrong input

File IO

- #!/usr/bin/perl
- $file = '/etc/passwd';
- open(PASSWD, $file) or
- die "Failed to open $file: $!\n";
- @lines = <PASSWD>;
- close(PASSWD);
- print @lines;

Open the file and assign the file handle to PASSWD.

Similar to || in Bash.

$! returns the error string.

Read everything into an array.

Close the file.

Print the whole array.

File IO

```
#!/usr/bin/perl
$file = '/etc/passwd';
open(PASSWD, $file) or
      die "Failed to open $file: $!\n";
foreach $line (<PASSWD>) {
    print $line;
}
close(PASSWD);
```

Read one line at a time.

Print one line at a time.

File IO

- open(INFO, $file); # Open for input
- open(INFO, ">$file"); # Open for output
- open(INFO, ">>$file"); # Open for appending
- open(INFO, "<$file"); # Also open for input
- open(INFO, '-'); # Open standard input
- open(INFO, '>-'); # Open standard output

- print INFO "This line goes to the file.\n";

File IO

- Useful trick
  - @ARGV = ("-") unless @ARGV;
  - open(LOG, $ARGV[0]) or
  - die "Failed to open $ARGV[0]: $!\n";

Function/Subroutine

```
#!/usr/bin/perl

sub foo {
    print "foo()\n";
}

# Older version requires &
&foo;
```

Function/Subroutine

```
#!/usr/bin/perl

sub bar {
    print "@_\n";
    print "1st argument = $_[0]\n";
}

bar("1", "2", "3");
bar(1, 2, "3", 4);
bar();
```

Function/Subroutine

```
#!/usr/bin/perl

sub max {
    if ($_[0] > $_[1]) {
        $_[0];              # return is optional
    } else {
        $_[1];
    }
    #return ($_[0] > $_[1] ? $_[0] : $_[1]);
}

$i = max(3, 1);
print "$i\n";
```

Function/Subroutine

```
sub foo {
    $a = 2;
}
$a = 1;
print "\$a = $a\n";
foo();
print "\$a = $a\n";
```

- $a = 1
- $a = 2

```
sub bar {
    my $a = 2;
}
$a = 1;
print "\$a = $a\n";
bar();
print "\$a = $a\n";
```

- $a = 1
- $a = 1

Function/Subroutine

```
#!/usr/bin/perl

$labs = 98;
$projects = 91;
$ave = int(compute_grade($labs, $projects));

print "Labs = $labs, Projects = $projects, ",
   "Ave. = $ave (extra credit included)\n";

sub compute_grade {
   my ($labs, $projects) = @_;
   # With extra credit
   $labs += 2;
   my ($total, $ave);
   $total = $labs + $projects;
   $ave = ($labs + $projects) / 2;
   return ($ave);
}
```

A good way of using formal parameters inside a Perl function.

Local variables declaration.

Function/Subroutine

- $ man perlsub
- $ man perlfunc

String Manipulation - substr

- #!/usr/bin/perl
- $a = "Welcome to Perl!\n";
- print substr($a, 0, 7), "\n"; # "Welcome"
- print substr($a, 7), "\n"; # " to Perl!\n"
- print substr($a, -6, 4), "\n"; # "Perl"

String Manipulation - split

```
#!/usr/bin/perl

$a = "This is a test";
@b = split(/ /, $a);
#@b = split(/ /, $a, 2);     # 2 strings only, ie. “This” and “is a test”

foreach $i (@b) {
    print "$i\n";
}

$_ = "This is a test";
@b = split(/ /);             # Take $_ as input

foreach $i (@b) {
    print "$i\n";
}
```

String Manipulation - join

- #!/usr/bin/perl

- @a = ("This", "is", "a", "test");
- $b = join(' ', @a);
- $c = join(' / ', @a);
- print "\$b = $b\n";
- print "\$c = $c\n";

  - $b = This is a test
  - $c = This / is / a / test

String Manipulation - substitution

```
#!/usr/bin/perl

$a = "I love Perl!";
if ($a =~ s/love/hate/) {
    print "New \$a = $a\n";
} else {
    print "Not found!\n";
}
$_ = $a;
if (s{hate}{love}) {
    print "New \$_ = $_\n";
} else {
    print "Not found!\n";
}
```

- New $a = I hate Perl!
- New $_ = I love Perl!

```
#!/usr/bin/perl

$url = "http://www.cs.ucr.edu/";
$url =~ s{/$}{/index.html};

print "$url\n";
```

- http://www.cs.ucr.edu/index.html

String Matching Using Regular Expression (Regexp)

```
#!/usr/bin/perl

$a = "This is a test";
if ($a =~ /is/) {
    print "Found!\n";
} else {
    print "Not Found!\n";
}

$_ = $a;
if (/is/) {
    print "Found!\n";
} else {
    print "Not Found!\n";
}
```

Regular Expression - Metacharacters

- . # Any single character except a newline
- ^ # The beginning of the line or string
- $ # The end of the line or string
- * # Zero or more of the last character
- + # One or more of the last character
- ? # Zero or one of the last character

- Eg.
- /^http:.+html$/
- /^http:.+\.html$/
- /^http:.+\.(html|htm)$/

Regular Expression - Metasymbols

- \d # [0-9]
- \w # [0-9a-zA-Z_]
- \s # [\t\r\n]

- \D # Reverse of \d or [^0-9]
- \W # Reverse of \w or [^0-9a-zA-Z_]
- \S # Reverse of \s or [^[\t\r\n]

Regular Expression - Metasymbols

```
unless ($phone =~ /\d\d\d-\d\d\d\d/) {
    print "That's not a phone number!\n";
}

unless ($phone =~ /\d{3}-\d{4}/) {
    print "That's not a phone number!\n";
}

/\d{3,5}/        # 3-5 digits long

unless ($passwd =~ /\w{8,}/) {
    print "Passwd too short!\n";
}
```

Regular Expression - Captured Strings

- Also called backreferences, eg.
  - $a = "07/Jul/2007:08:30:01";
  - $a =~ /([^:]+):(.+):(.+):(.+)/;
  - ($hour, $min, $sec) = ($2, $3, $4);
  - print "$hour:$min:$sec\n";

Regular Expression - is Greedy

- $a = "12345678901234567890";
- $a =~ /(1.*0)/;
- print "$1\n";
  - 12345678901234567890
- $a =~ /(1.*?0)/;
- print "$1\n";
  - 1234567890

Regular Expression - Misc

- To ignore case when matching
  - /\w+/i
- More info
  - $ man perlre
  - http://www.comp.leeds.ac.uk/Perl/matching.html
  - http://www.perl.com/pub/a/2000/11/begperl3.html

Debug Perl

- $ perl -w debug.pl
- $ perl -d debug.pl
- print() or printf() is your friend ☺

References

- Programming Perl
  - Ch 2-5
- Beginner's Introduction to Perl
  - http://www.perl.com/pub/a/2000/10/begperl1.html
- Perl Tutorial
  - http://www.comp.leeds.ac.uk/Perl/start.html