Lecture 9

Introduction to Perl

COP 3344 Introduction to UNIX

Fall 2007

Acknowledgment: These slides are modified versions of Prof. Sudhir Aggarwal's slides

Advantages of Perl

- Fills the gap between shell programming and a conventional compiled programming language like C or C++
- Perl code is very dense
  - Size is often 30% to 70% that of similar C code
- Good for simple programs that you want to code quickly
  - Good for text manipulation
- It is very portable

A Sample Perl Program

hello
```
#!/usr/bin/perl -w
#Test program: hello world
print “hello world!\n”;
exit 0;
```

```
$ chmod 700 hello
$ ./hello
Hello world!
```

- Program features
  - #! specifies the program that executes the file
    - The `-w` flag prints warnings
  - Comments start with #
  - White space can be used almost anywhere
  - Statements end with a `;`

Scalar Data in Perl

- A scalar is a single item of data
  - A number, for example `255` or `3.1416e2`
  - A string, for example `‘hello\n’` or `“good bye”`
    - `‘the \n does not have a special meaning here’`
    - `“the \n here represents the newline character”`
    - `“the $var variable is replaced by its value here”`
- Perl uses strings and numbers almost interchangeably
  - Implicit conversion in performed between strings and numbers depending on the operations performed on the scalar data

hello
```
#!/usr/bin/perl -w
#Test program: hello world
print ‘hello world!\n’;
exit 0;
```

```
$ chmod 700 hello
$ ./hello
Hello world!\n$
```

Scalar Variables

- Names preceded by $ regardless of its use on the left or right side of an assignment
- Examples

  `$sum = 14`

  `$sum = $var + 47.3`

pprog2
```
#!/usr/bin/perl -w
$help="aid";
$s="band" . $help;
print "$s\n";
```

```
$./pprog2
bandaid
```

Examples of Operators

`=`	assignment
`+, -, *, …`	arithmetic
`<, <=, …`	relational
`&&, ||, !`	logical
`++, --`	increment, decrement
`eq, ne, lt, gt, le, ge`	string relational
`cmp`	string comparison
`.`	concatenation
`x`	string repetition
`“fred” x 3`	result is `“fredfredfred”`

Line Input Operator `<STDIN>`

- The `<STDIN>` operator reads line of input
  - Read from standard input, up to and including the next newline character

  `$line = <STDIN>;`
  - If the end-of-file is reached, then `<STDIN>` returns `undef`, which acts like `0` or the empty string
  - The `chomp` operator is used to remove a newline from the end of a string

    `chomp ($line = <STDIN>);`

pprog3
```
#!/usr/bin/perl -w
$line = <STDIN>;
if($line eq "\n"){
  print "Blank line!\n";}
else{
  print "The line was: $line";}
```

```
$./pprog3

Blank line!
$ ./pprog3
sdf
The line was: sdf
```

Acting on Each Line

```
#!/usr/bin/perl -w -n
print;
```

- The `-n` causes the program to be executed on each line

pprog4
```
#!/usr/bin/perl -w -n

print;
```

datafile
```
Name    GPA
asd     4.0
sdf     3.2
fghsd   3.6
qwer    4.0
```

```
$./pprog4 < datafile
Name    GPA
asd     4.0
sdf     3.2
fghsd   3.6
qwer    4.0
```

Pattern Matching

- Match patterns using `m/Pattern/`
  - Usually used with the binding operator =~
  - Example: `$mystring =~ m/cat+/` has the value `true` if `$mystring` has any of the following values: `cat, catt, cattt`, …

pprog5
```
#!/usr/bin/perl -w -n
if($_ =~ m/4\.0/)
{
   print $_;
}
```

datafile
```
Name    GPA
asd     4.0
sdf     3.2
fghsd   3.6
qwer    4.0
```

```
$./pprog5 < datafile
asd     4.0
qwer    4.0
```

Pattern Matching with Substitution

- Substitute patterns using `s/Pattern/Substitute/`

pprog6
```
#!/usr/bin/perl -w -n
$line = $_;
$line =~ s/cat+/dog/;
print $line;
```

```
$./pprog6 < datafile2
dogs are good
dogs are good
dogs are good, good cat
```

datafile2
```
dogs are good
cats are good
catts are good, good cat
```

pprog7
```
#!/usr/bin/perl -w -n
$line = $_;
$line =~ s/cat+/dog/g;
print $line;
```

```
$./pprog7 < datafile2
dogs are good
dogs are good
dogs are good, good dog
```