

CS306: Introduction to Perl

Section 5: I/O

U. of Alabama at Birmingham
Dept. of Computer & Information Sciences

Slide 1

Section 5: I/O

Standard input and output

<> operator

Formatted output

Working with files

Working with directories

Slide 2

STDIN

- STDIN is Perl's handle for standard input (usually the keyboard)
- `chomp($line = <STDIN>);`
 - Read one line from keyboard, remove `\n` character
- `chomp(@array = <STDIN>);`
 - Read all lines up to EOF and chomp them all

Slide 3

STDIN in loops

- `while ($line = <STDIN>) { ... }`
- `while (chomp($line = <STDIN>)) # BROKEN`
 - Doesn't work because you'll end up trying to chomp undef, which will throw an error. Chomp in the loop.

Slide 4

The diamond operator <>

- <STDIN> is just an example of the <> operator
- You can use the <> operator to make your programs more flexible
- while (\$line = <>) { ... }
 - # ./myprogram file1 file2
 - Will read each line from file1, then file2, etc...
 - # ./myprogram
 - Will use STDIN

Slide 5

The <> operator 2

- Larry did this to make his perl programs act like other UNIX utilities such as cat, sed, awk, sort, grep, lpr, etc...
- You can indicate STDIN as part of a list of files using the -
 - ./myprogram.pl file1 - file2

Slide 6

@ARGV

- @ARGV will hold the list of files that were passed into the program.
- ./myprogram.pl file1 file2 file3
 - \$ARGV[0] is "file1", etc...

Slide 7

STDOUT

- print "\$diskusage{fran}"; #goes to STDOUT
- print "@array\n";
- print <>; # Print every line of input – i.e. "cat"
- print sort <>; # Unix sort, anyone?

Slide 8

Formatted output with printf

- `printf "%s is using %d MB of disk space.", $user, $disk{$user};`
 - The % placeholders are called *conversions*
- %10s – A 10 character string (will pad with spaces if necessary, but not truncate)

Slide 9

Sample Conversions

- `printf "%8d", 16;` # “ 16”
- `printf "%-8s", "foo";` # “foo ”
- `printf "%8.3f", 3.1415926;` # “ 3.142”
- `printf "Rate: %.2f%%", 6.273;` # “Rate: 6.27%”
- `printf "(%g) (%g) (%g)\n", 8/4, 17/5, 2**8;`
“(2) (3.4) (1.84467e+19)”
 - %g does something reasonable with a number

Slide 10

Creating Filehandles

- `open IN, "<filename";`
`while ($line = <IN>) { ... };`
`close IN;`
- The IN is called a filehandle. You can name it anything you'd like
- The `close()` is optional but good practice. Perl will close all filehandles on exit, or when they are reopened.

Slide 11

Creating Filehandles 2

- `open IN, "<filename"; # or open IN, "filename"`
- `open OUT, ">filename";`
- `open APPEND, ">>filename";`
- `$success = open LOG, ">>access_log";`
`if (! $success) { print "Can't open file.\n"; exit; }`

Slide 12

Standard Filehandles

- STDIN – The standard input, usually but not always the keyboard
- STDOUT – The standard output, often but not always the screen. In a pipeline, STDOUT becomes the next program's STDIN
- STDERR – The standard error, usually the screen

Slide 13

Understanding STDIN and STDOUT

- Perl trusts that the user has done the right thing with STDIN and STDOUT
- `# ./myprogram.pl <datafile >output.txt`
 - Perl will use datafile as STDIN and output.txt as STDOUT.
 - STDIN is no longer the keyboard, and STDOUT is no longer the screen. But the program doesn't have to change.

Slide 14

The Pipeline Model

- `# cat "datafile" | ./yourprogram.pl | lpr`
 - Stringing together UNIX utilities using the pipe
 - Your perl program's STDIN is the output of 'cat "datafile"'
 - Your program's STDOUT becomes the input to lpr (the UNIX command line print utility)
- STDIN and STDOUT are really an OS/Shell function and Perl just trusts it

Slide 15

When STDERR Is Used

- If you are pipelining, you don't want error messages to get sucked up into the input of the next program
- `cat "datafile" | myprogram.pl | lpr`
- ```
while ($line = <>) {
 if (length($line) == 0) {
 print STDERR "Empty line!\n";
 }
 ...
}
```

Slide 16

## File Tests

- print “What's the filename? “;  
chomp(\$fname = <STDIN>);  
unless (-e \$filename) {  
    print “File doesn't exist!\n”;  
    exit;  
}  
open IN, “<\$fname”;
- -e is a *file test* – it checks for existence of the file or directory name

Slide 17

## Common File Tests

- -r File is readable
- -w File is writable
- -e File or directory name exists
- -f Is a file
- -d Is a directory
- -T Appears to be a text file
- -B Appears to be a binary file
- Many more! See Programming Perl

Slide 18

## Working With Directories

- opendir DIR, “/home/fran”;  
foreach \$file (readdir DIR) {  
    print “Filename is \$file\n”;  
}  
close DIR;

Slide 19

## Eliminating . and ..

- Each directory on UNIX includes two special files, . (itself) and .. (parent directory). You usually don't want to process these.
  - while (\$file = readdir DIR) {  
    next if \$file eq “.” or \$file eq “..”;  
    ...  
}

Slide 20

## The End

- Any questions on I/O?