Day 4: Basic I/O

Suggested reading: *Learning Perl* (6th Ed.)

Chapter 5: Input and Output

Chapter 12: File Tests

Chapter 13: Directory Operations

Turn In Homework

Homework Review

Write code.
At least a little.
Every day*, even on the weekend*.
Play around!

Perl Documentation

Perldoc

- Online:
  - **http://perldoc.perl.org/**
  - Be sure to select Perl version (upper left)
- From command line:

```
perldoc -f function
```

- Many useful, well-written (and long!) man pages:

```
man perlop          # all operators (eg, +)
man perlsyn         # core syntax
man perldata        # data types, VERY thoroughly
man perlfunc        # all functions in one
```

Files

What Is a File?

- Sequence of bytes on disk
- Up to Perl and your script to interpret the bytes

```
% cat hw4.txt
This
is
the
Speech
Thought
and
Writing
Corpus
compiled
by
...
```

```
54 68 69 73 0a 69 73 0a  |This.is.|
74 68 65 0a 53 70 65 65  |the.Spee|
63 68 0a 54 68 6f 75 67  |ch.Thoug|
68 74 0a 61 6e 64 0a 57  |ht.and.W|
72 69 74 69 6e 67 0a 50  |riting.P|
72 65 73 65 6e 74 61 74  |resentat|
69 6f 6e 0a 43 6f 72 70  |ion.Corp|
75 73 0a 63 6f 6d 70 69  |us.compi|
6c 65 64 0a 62 79 0a 45  |led.by.E|
...
```

Opening & Closing

- Open file and make filehandle: **`open()`**
- Filehandle variable name is usually uppercase
- Optional middle argument is for *mode*

```
open(FILE, 'input.txt');
open(DATA_SET, '<', $my_filename);
```

- Close filehandle: **`close()`**

```
close(DATA_SET);
```

Failing

`open()` returns ***false*** upon failure:

```
unless (open(FILE, $filename)) {
    die "Could not open '$filename': $!\n";
}
```

… or the Perl way:

```
open(FILE, $filename) or
    die "Could not open '$filename': $!\n";
```

Reading

- Read one line using **<*FILEHANDLE*>**

```
my $line = <RAW_DATA_1>;
```

- Returns ***false*** when file is done
- Use **chomp** to strip trailing newline

```
while (my $line = <RAW_DATA_1>) {
   chomp $line;
   # do something with $line
}
```

Interlude: Scalar vs. List Context

```
my $foo = ____________;
my @foo = ____________;
```

```
1 + 2 + ____________ + 3 + 4;
```

```
push @stack, ____________;
```

Reading Whole Files

- In *list* context, **<>** yields one list element per line

```
my @lines = <INPUT>;
```

- **chomp** works on arrays

```
my @lines = <INPUT>;
chomp @lines;
```

Writing

- Open for (over)writing:

```
open(OUTPUT_FILE, '> output.txt');
open(OUTPUT_FILE, '>', 'output.txt');
```

- Open for appending:

```
open(MY_DATA_OUT, '>> output.txt');
open(MY_DATA_OUT, '>>', 'output.txt');
```

- Write to filehandle:

```
print OUTPUT "What could be easier?\n";
```

File Example

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

open(INPUT, 'input.txt') or
    die "Open 'input.txt' failed: $!\n";
open(OUTPUT, '>', 'output.txt') or
    die "Open 'output.txt' failed: $!\n";

my $number = 1;
while (my $line = <INPUT>) {
    print OUTPUT "$number: $line";
    $number++;
}

close(INPUT); close(OUTPUT);
```

Automatic Filehandles

- **STDIN** = standard input (default for **<>**)
- **STDOUT** = standard output (default for **print**)
- **STDERR** = standard error (output)

```
chomp(my $user_input = <STDIN>);
chomp(my $user_input = <>);

print STDOUT "Hello, world!\n";
print "Hello, world!\n";

print STDERR "Critical error\n";
```

The Magic of $_

- Automatically created (e.g., by **<>**)
- Used as default (e.g., by **chomp**, **print**, …)
- Can use **$_** explicitly
- Check perldoc for details

```
while (<>) {                    # STDIN is implied
  chomp;                        # $_ is implied
  print "input: $_\n";
}
```

we will see $_ again another day…

Directories

File Test Operators

`-r`	readable	`-e`	exists	`-f`	regular file
`-w`	writeable	`-z`	zero length	`-d`	directory
`-x`	executable	`-s`	size (*bytes*)	`-l`	symbolic link

```
if (-r $path) {
    if (-d $path) {
        print "skip directory: $path\n";
    }
    elsif (-f $filename) {
        open(INPUT, $filename);
        # ...
    }
}
```

Directory Contents I

Can read directory contents directly

```
opendir(DIR, $dirname) or
    die "Could not open $dirname: $!\n";

foreach my $path (readdir(DIR)) {
    # do something with $path
}

closedir(DIR);
```

Note: Includes `.` and `..` (on Unix, Linux, Mac OS X)

Directory Contents II

- `glob('...')` lists files matching a (shell) pattern
- `<...>` syntax works, but can be confusing

```
foreach my $path (glob("$dir/*.html")) {
  open(INPUT, $path) or die "...";
  # do stuff
  close(INPUT);
}

foreach my $path (<$dir/*.html>) {…}
```

Shell-Like Operations

- **chmod**, **chown** : change file permissions, owner
- **mkdir**, **rmdir** : make and remove directories
- **rename**, **unlink**, **symlink** : move, remove, and link

```
chmod 0644, $filename if -x $filename;

mkdir $new_directory_name;

if (-e $old_name) {
   rename $old_name, $new_name;
   symlink $new_name, $old_name;
}
```

Last 2 Slides!

Other Scripting Languages

- Most have similar I/O operations
- Check for different or additional:
  - **Operation names** (**`-d`** vs. **`isdir()`** vs. **`directory?()`**)
  - **Syntax**
  - **Operations**
- Some environments are more restricted

Homework

- Read a large file of words (or subset, for practice)
- Count words, except for UPPERCASE
  - What collection type works well here?
- Find and print, e.g., *freq*("Yes") > *freq*("yes")

```
#!/usr/bin/perl

# Homework for CS 368-3
# Assigned on Day 04, 2012-06-28
# Written by Your Name Here

use strict;
use warnings;
```