Day 4: Basic File Text Processing

Suggested Reading:
Learning Perl (4th Ed.),
Chapter 5: Input and Output

Reminders

- Turn in homework at **START** of class
- Writing code is **fun**!
  - Write at least a little every day
  - The more you do, the easier it gets
- When in doubt, ask questions!
  - Zathras does not want you being confused
- Think about last class session
  - What would you like it to cover?
  - Any particular ideas?

Homework Code Formatting

- Look at the formatting of the code examples in Learning Perl (4th Ed.)
  - Use this as a template for your homework
- Indention is vital to making the code you hand in to us readable!
- Example:

```
if ($whatever) {
  # Do something
}
foreach my $var (@array) {
  if ($var eq "blah") {
    print "var is $var\n";
  }
}
```

Homework Feedback

- Simple is almost always better
- Do this:

```
while (1) {
  if (whatever) {
    last;
  }
}
```

- Not this:

```
my $loop = 1;
while ($loop) {
  if (whatever) {
    $loop = 0;
  }
}
```

(More) Homework Feedback

- Do this:

```
while (1) {
  if ($foo eq "quit") {
    last;
  }
  # do more stuff
}
```

- Not this:

```
print "Enter command:";
my $foo = <STDIN>;
while ($foo ne "quit") {
  # do some stuff
  print "Enter command:";
 $foo = <STDIN>;
}
```

die

- (From Learning Perl, 4th Ed.):
  - The die function prints out the message you give it (to the standard error stream, where such messages should go) and ensures that your program exits with a nonzero exit status.
- Example:

```
if ($error) {
  die "An error has ocurred!";
}
```

if `$error` is non-zero, this will produce:

```
An error has occurred at foo.pl line 5.
```

Opening a File

- Use `open()` to open the file and create a file handle:
  - By convention, file handle names are UPPERCASE

```
open(INPUT, "my_input.txt", "r");
open(FILE, "input.txt");
```

- Use `close()` to close the file:

```
close(INPUT);
```

Checking for Failures

- `open()` returns 0 if it fails:

```
unless (open(FH, $file)) {
  print "Failed to open $file\n";
} else {
  # read the file via file handle FH
  close(FH);
}
```

```
unless (open(FH, $file)) {
  die "Couldn't open $file";
}
```

```
open (FH, $file) or die "Couldn't open $file";
```

Reading From the File

- Use the `<>` operator to read from a file handle:

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

- Use `chomp()` to strip off newline characters:

```
while (my $line = <FH>) {
  chomp $line;
  print $line;
}
```

Reading from STDIN

- STDIN is a file handle that is automatically opened for you

```
my $input = <STDIN>;
chomp $input;
```

- STDIN is the default file handle, so this will do the same thing:

```
my $input = <>;
chomp $input;
```

Reading a Whole File

- The <> operator can also be used in an array context to read a whole file in a single operation:

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

- Similarly, `chomp()` can be used to strip off newline characters of the whole array:

```
my @lines = <FH>;
chomp @lines;
print $lines[0];
```

File Writing Operations

- To open a file for writing:

```
open(OUTPUT, ">output.txt");
```

- To append to an existing file:

```
open(OUTPUT, ">>output.txt");
```

- To write to the file:

```
print OUTPUT "This is really cool\n";
```

  – Notice: No comma (",") between the descriptor and the string to print!

File Writing Snippets

- Putting it all together:

```
unless (open(OUTPUT, ">$file")) {
  die "Couldn't write to $file";
}
print OUTPUT "Line of output\n";
close(OUTPUT);
```

- Or:

```
unless (open(OUT, ">>$file")) {
  print "Can't write to $file\n";
} else {
  print OUT "Added this line\n";
  close(OUT);
}
```

Writing to STDOUT and STDERR

- STDOUT and STDERR are file handles that is automatically opened for you

```
print STDOUT "This goes to STDOUT\n";
print STDERR "This goes to STDERR\n";
```

- Like STDIN for <>, STDOUT is the default output of `print`:

```
print "More to STDOUT\n";
```

- STDERR is typically use to report errors:
  - die and warn print their message to STDERR

```
print STDERR "Error detected!\n";
```

The Magic of $_

- $_ is a automatic variable in perl
  - Unless otherwise specified:
    - <> assigns to $_
    - `chomp()` and many other functions and operators operate on $_
    - `print` prints the contents of $_

The Magic of $_: Example

- Thus:

```
while (<>) {    # Reads from STDIN into $_
  chomp;        # chomp $_
  print;        # print contents of $_
}
```

- Is equivilent to:

```
while ($_ = <STDIN>) {
  chomp $_;
  print $_;
}
```

Sample File Reading Script

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

my $file = "example-01.txt";
unless (open(IN, $file) ) {
  die "Can't read input file '$file'";
}
while (my $line = <IN>) {
  if (index($line, "Important") >= 0) {
    print $line;
  }
}
close(IN);
```

- Open the file
- Generate an error if it fails
- Read the file line by line
- Look for the string "Important"
- Print the line
- Close the file

"Idoiomatic" File Reading Script

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

my $file = "example-01.txt";
open (IN, $file ) or
  die "Can't read input file '$file'";
while (<IN>) {
  print if(/Important/);
}
close(IN);
```

- Open the file
- Generate an error if it fails
- Read the file line by line (into $_)
- Look for the string "Important" (in $_)
- Print the line ($_)
- Close the file

More Fun With open()

- open() can also be used to run a process with a pipe:

```
my $cmd = "/bin/ls -l";
unless open(INPUT, "$cmd|") {
  die "Can't run $cmd";
}
while (my $line = <INPUT>) {
  print if index($line, "nleroy") >= 0;
}
close(INPUT);
```

More Operations

- The @ARGV array contains the command line:

```
scalar(@ARGV)
$ARGV[0]
```

- File test operators:
  - "-f $name": detect if $name exists and is a file
  - "-d $name": detect if $name exists and is a directory
  - "-s $name": Returns the size (in bytes) of $name

- "Glob" operator: <>

```
my @files = <*>;
foreach my $path(<$dir/*>) { something(); }
while (my $line = <FH>) { something(); }
```

Files & Directories

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

die "usage: example-03 directory" unless scalar(@ARGV) == 1;
my $dir = shift(@ARGV);
die "$dir isn't a directory" unless -d $dir;
foreach my $path (<$dir/*>) {
  if (-d $path) {
    print "$path is a directory\n";
  } elsif (-f $path) {
    print "$path is a file\n";
  } else {
    print "I don't know what $path is\n";
  }
}
```

Files & Directories (Python)

```
#! /usr/bin/env python
import sys
import os
import glob

if len(sys.argv) != 2 :
  print >>sys.stderr, "usage: example-03 directory"
  sys.exit(1)
dirpath = sys.argv[1]
if not os.path.isdir( dirpath ) :
  print >>sys.stderr, dirpath, "isn't a valid directory"
  sys.exit(1)
for path in glob.glob( dirpath+"/*" ) :
  if os.path.isdir( path ) :
    print path, "is a directory"
  elif os.path.isfile( path ) :
    print path, "is a file"
  else :
    print "I don't know what", path, "is"
```

Files & Directories (Ruby)

```
#! /usr/bin/env ruby
unless ARGV.size == 1
  warn "usage: example-02 string"
  exit 1
end
dirpath = ARGV[0]
unless File.directory?(dirpath)
  warn "#{dirpath} isn't a valid directory"
  exit 1
end
Dir.glob("#{dirpath}/*").each do |path|
  if File.directory?(path)
    puts "#{path} is a directory"
  elsif File.file?(path)
    puts "#{path} is a file"
  else
    puts "I don't know what #{path} is"
  end
end
```