Day 12: Recipes II

Writing Files, User Input, and Configuration

Suggested Reading:

Perl Cookbook (2nd Ed.) *[skim parts]*

Chapter 1: Strings
Chapter 7: File Access
Chapter 8: File Contents

Homework Review

Homework Preview

Weather Analysis, Part II

- Save our data!
  - Need to add to the data file each time
  - What should be saved?
  - Must be sure not to lose historical data
  - What happens if the script is run twice in one day?
- Read program configuration
  - Hardcoded defaults
  - Configuration file
  - Command-line options

Writing Files

Seriously?

What is so hard about writing a file?

>_<

Writing Files **Robustly**

aka

What if the power goes off in the middle of a file write?

Atomic File Writes

Key idea: Write to separate file, move in place

```
sub safe_write {
  my ($filename, $contents) = @_;
  open(NEW, '>', "$filename.NEW")     or return;
  print NEW $contents                 or return;
  close(NEW)                          or return;
  rename($filename, "$filename.BAK") or return;
  rename("$filename.NEW", $filename) or return;
  return 1;
}
```

Atomic File Writes — Further Challenges

- `rename()` not (always) atomic
- Gap between renames
- Multiple safe-writes overwrite backups
- All errors treated the same way
- Does not handle appends
- Temporary filename not safe…

Better Temporary Files

Use `File::Temp` to open file and give name

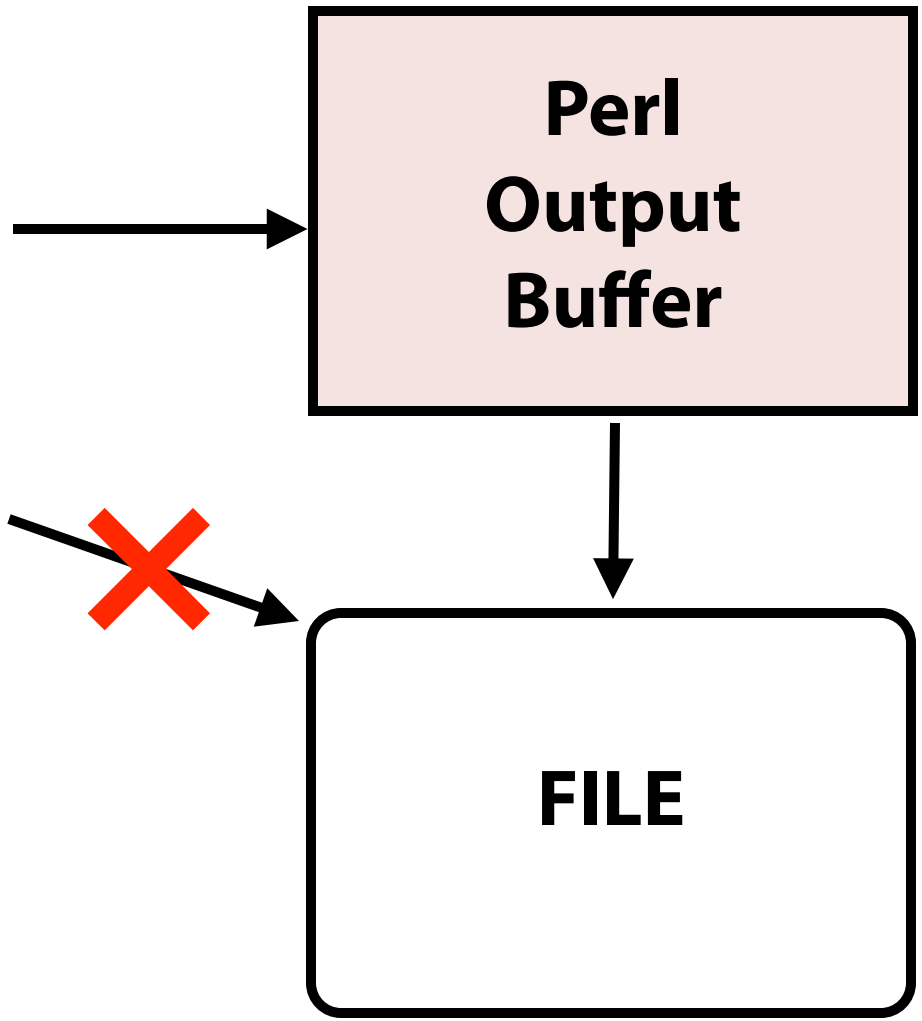
```
use File::Temp qw/tempfile/;
my $temp_fh = tempfile();

my ($fh, $temp_filename) = tempfile();
print $fh $contents;
close($fh);

rename($filename, "$filename.BAK");
rename($temp_filename, $filename);
```

Perl Output Buffers

```
open(OUT, '>', $filename) ...
print OUT "first line\n";
sleep(30);
foreach my $thing (@things) {
  sleep(10);
  print OUT calculate($thing);
}
close(OUT);
```



Flushing Your Buffers

- Set **$|** (dollar-pipe) to true
- Affects ***all*** output buffers
- Can significantly affect performance

```
$| = $ARGV[0];                  # try 0, then try 1

print "Start of output... ";
sleep(2);
print "and now we are done!\n";
```

A Few Scalar Tricks

String Trimming

- Input strings may have leading, trailing whitespace
- User input, data files, configuration, arguments
- Can mess up REs, comparisons, etc.

```
sub trim {
    my $string = shift;
    $string =~ s/^\s+//;
    $string =~ s/\s+$//;
    return $string;
}

trim("\t oops  \t\n");  => 'oops'
```

Making Sure a Variable is Defined

- The obvious (if verbose) way:

```
my $foo = get_value();  # can return undef
if (not defined $foo) {
    $foo = default value here;
}
```

- The Perl way:

```
my $foo = get_value();  # can return undef
$foo ||= default value here;
```

- Any problems with the Perl way?

Long String Literals

Use special Perl syntax called a "here-document"

```
my $long_string = <<EOF;
This is the start of the string.
It can go on for many lines.
When done, terminate on the next line.
EOF

safe_write($filename, <<CONTENTS);
* File contents!
  - Formatting is preserved!
* Interpolation even works: $foo!!!
CONTENTS
```

Really Long String Literals

- Use **__DATA__** section at end of script
- **Pro:** Part of script… **Con:** Part of script

```
while (<DATA>) {
    # do stuff with lines here
}

__DATA__
This is the start of a very long data block.
It must be the last part of the script file.
Perl will not run code after __DATA__.
No $variable interpolation here.
```

Configuration

Sources of Configuration Settings

1. Standard input
2. Command-line options
3. Configuration file(s)
4. Hardcoded values in the script (defaults)

Simple Configuration File Layout

```
# Comment lines:
# So that you can describe config settings,
# right in the file

# Blank lines are OK

setting_name_1 = 42
setting_name_2 = Thanks For All the Fish

another_setting =
# some_setting = is commented out
```

Configuration File Parsing

```
my %config;
while (<CONFIG>) {
  s/#.*//;                  # no comments
  s/^\s+//;                 # no leading white
  s/\s+$//;                 # no trailing white
  next unless length;       # anything left?
  my ($var, $value) =
    split(/\s*=\s*/, $_, 2);
  $config{$var} = $value;
}
```

Better yet, for unit testing: Write a sub that takes a string (or array of lines) and returns a hash

Implementing Priority Order

- Process in reverse order, allowing overrides:

```
my %config = ( 'setting' => 42 );   # default
read_config_file($filename, \%config);
GetOptions('setting' => \$config{'setting'});
```

- Process in order, checking whether defined

```
GetOptions('setting' => \$config{'setting'});
$config{'setting'} ||= $cfg_file{'setting'};
$config{'setting'} ||= 42;  # default
```

Homework

Weather Analysis, Part II

- Save our data!
  - Need to add to the data file each time
  - Must be sure not to lose historical data
  - What happens if the script is run twice in one day?
- Read program configuration
  - Hardcoded defaults
  - Configuration file
  - Command-line options