Option and Config Processing with Perl

Jon Allen

http://perl.jonallen.info - jj@jonallen.info

Command-line basics

- Many programs take options on the command-line

```
myscript --message "Hello, World" hello.txt
```

- Command-line arguments are passed to your program in the @ARGV array

```
$ARGV[0] = '--message'
$ARGV[1] = 'Hello, World'
$ARGV[2] = 'hello.txt'
```

- So, how do we make use of them?
  - Process the @ARGV array manually or use a module

Using command-line options

- CPAN has 43 “Getopt” modules, and 168 “Config” modules
  - “Mmmm, choice!” or “Aaargh, choice!” ???
- Getopt::Long is a core module
  - No dependencies to worry about
- Wide range of facilities
  - Can also be extended
- Should be all you need

Getopt::Long facilities

- Many different types of options:
  - Boolean (**`--verbose`**)
  - Single value (**`--name JJ`**)
  - Multiple values (**`--name Brian Barbie Steve`**)
  - Hash value (**`--name Barbie=Director JJ=Member`**)
- Flexible formatting
  - Can use **`--name=JJ`**, **`--name JJ`**, **`--Name=JJ`**, etc.
  - Allows option names to be abbreviated, e.g. **`--n JJ`**
  - Single or double dash (**`-name`** or **`--name`**)
- Values stored in scalars or hash

Getopt::Long usage

- Build up a list of “option specifiers”
  - Defines name and type, e.g. **`file=s`** defines an option called ‘**`file`**’ that must have a string value ‘**`=s`**’
  - See http://perldoc.perl.org/Getopt/Long.html

```
use Getopt::Long;

my ($name,$verbose);
GetOptions( 'name=s'  => \$name,
            'verbose' => \$verbose );
```

- Any unused arguments will be left in @ARGV

More Getopt::Long examples

- Store option values in a hash

```
my %options;
GetOptions( \%options,
            'name=s',
            'verbose' );
```

- Trigger code blocks

```
my %options;
GetOptions( \%options,
            'name=s',
            'verbose',
            'usage' => sub {
                         print "RTFM...\n";
                         exit;
                       } );
```

Using longer option names

- What should the option for “Output File” be?

```
--output-file JJ.xml
--output_file JJ.xml
--outputfile JJ.xml
--OutputFile JJ.xml
```

- All of them!
  - Getopt::Long allows aliases for option names to be defined using the “|” character in the option spec:

```
my $options;
GetOptions( \%options,
            'output_file|output-file|outputfile=s' );
```

That's a lot of typing though!

- It would be nicer if these alternatives could be generated automatically…

```
use Getopt::Long;

my %specifiers = ( 'source-directory' => '=s',
                   'verbose'          => ''    );

my %options;
GetOptions( \%options, optionspec(%specifiers) );
```

- The **`%options`** hash always contains the underscored version of option names

The optionspec() subroutine

```
sub optionspec {
  my %option_specs = @_;
  my @getopt_list;
  while (my ($option_name,$spec) = each %option_specs) {
    (my $variable_name = $option_name) =~ tr/-/_/;
    (my $nospace_name  = $option_name) =~ s/-//g;
    my $getopt_name = ($variable_name ne $option_name) ?
      "$variable_name|$option_name|$nospace_name" :
      $option_name;
    push @getopt_list,"$getopt_name$spec";
  }
  return @getopt_list;
}
```

The next step: Config files

- For programs with lots of options, it improves usability to let users save their settings
- Lots of possible formats
  - INI, XML, Apache, YAML, etc.
  - Lots of CPAN modules as well
- Downside - users need to learn a new syntax
- And you have to write more code!

- There must be an easier method…

Getopt::ArgvFile

- From CPAN - http://search.cpan.org/dist/Getopt-ArgvFile
- Uses config files with exactly the same format as command-line options

```
--name "Jon Allen"
--verbose
```

- Only one extra line of code!

```
use Getopt::ArgvFile home=>1;  # Loads '.scriptname' from
use Getopt::Long;              # user's home directory

my %options;
GetOptions( \%options,
            'name=s',
            'verbose' );
```

Further usage of Getopt::ArgvFile

- User-specified config files - the “@” directive

```
myscript @/path/to/config/file
```

- Comment lines are allowed in config files
- Can override config using command-line options

```
cat ~/.myscript
# Set the user's name
--name JJ

# Enable verbose logging
--verbose

myscript --name Barbie
```

Changing the default filename

- In previous example a default config file was loaded (~/.scriptname) - this can be changed:

```
use Getopt::ArgvFile qw/argvFile/;  # Disables automatic loading
use Getopt::Long;                   # of config files

# Prepend @ARGV with '@' directive(s)
# listing our desired filename(s)
unshift @ARGV,'@/path/to/config/file';

argvFile();  # Resolves any '@' directives in @ARGV

my %options;
GetOptions( \%options,
            'name=s',
            'verbose' );
```

Fin!

Thank you for listening!

Any questions?

http://perl.jonallen.info/talks/optionprocessing