Perfect Perl CLIs

Building Perl comand-line programs that are obvious and easy-to-use.

10 September 2020

https://bit.ly/perfect-perl-clis

[01] All on the same page?

- "Unix," the Unix philsophy, Unix "toolkit"
- "pipe"; "pipe-and-filter programming"
- experience with GetOpts::Short, or GetOpts::Long?
- ever written a command-line utility professionally?
  - …or to scratch an itch?

[02] Unix philosophy

- write programs that do one thing and do it well
- write programs to work together
- write programs to handle text streams
  - because that is a universal interface

–Doug McIlroy

[03] What does "discoverability" mean for CLIs?

- will it show up if I `man -k somekeyword`?
  - (probably not, because `mkwhatis` only works on system paths)
- will it show up if I `something[Tab]`?
- does it *do* something (besides crash) if I run it w/ no options?
- does it have a `--help` option?
  - does **that** guide me toward (more) help?
- does it behave like other programs I'm used to?

[04] What does it mean to be a "Unix program"?

- Don't try to do things that other standard Unix utils already do
  - *e.g.*, sorting, printing matching lines
- Use linefeed-delimited text streams as your IPC/RPC
  - read from standard input when used in a pipe
  - write **normal** (expected, requested) output to standard output

  - write **exceptional** output to standard error
- Return zero for success, non-zero for failure
- Have a manual page (it's cake with `pod2man`)

[05] The situation for Perl

Perl *is* part of the Unix toolkit.

pod2man comes with Perl and makes it easy to

- turn POD into a manual page
- so your users can `man yourcmd` just like any other

Getopt::Long and Pod::Usage make it easy to

- parse command line options the conventional way
- produce nicely-formatted, standard-looking `--help` output

[06] Ten recommendations for user-friendly (bioinformatics) CLIs

1. Print something if no parameters are supplied.

```
# after GetOptions pulls out what it needs...
die "ERROR: need a input filename" unless @ARGV;
```

2. Always have a `-h` or `-help` switch.

```
# after e.g., GetOptions('help' => \$help, 'manual' => \$manual)
pod2usage(0)                            if $help;
pod2usage(-exitval => 0, -verbose => 2) if $manual;
```

3. Have a `-v` [sic] or `-version` switch.

```
# right inside the call to GetOptions
GetOptions('version|V' => sub { print "$VERSION\n"; exit });
```

4. Do not use stdout for messages and errors.

[07] Ten recommendations for user-friendly CLIs (cont'd)

5. Always raise an error if something goes wrong (`or die`!)
6. Validate your parameters (Getopt::Long can do this!)
7. Don't hard-code any paths (no "magic numbers" as the boss would say)
8. Don't pollute the `PATH` (corrollary: install into the `$PATH`).
   - provide a standard `Makefile.PL`
   - provide a `Makefile` that installs to `~/.local`
   - look into something like Environment Modules
9. Check that your dependencies are installed.

10. Don't distribute bare JAR files. (Ha!)

[08] What does it mean to be a "filter"?

Take the example script `scrubbed`: it reads (potentially malformed) input and writes sanitized output.

It has a name ("scrub") that suggests it behaves like a filter: reading from and writing to standard out.

Some Unix toolkit programs (`cat`, `awk`, `grep`) will appear to "hang" if you forget the input filename.

You can either handle this one of two ways:

- by requiring `-` as the input filename to read from stdin
- by waiting for input (forever)

[09] POD to man pages

Unix man pages are grrreat. Their lack of features is a feature.

POD is basically a man page with slightly different section names from the Unix / POSIX standard. They show up in section "3pm" of the manual.

You can access them with `man cmdname` or `perldoc cmdname`.

You can process POD from a Perl module into a man page with `pod2man`.

[10] A minimum-viable POD section for Unix man page

```
=head1 NAME

mycmd - Do the thing

=head1 SYNOPSIS

B<mycmd> [options] [arg1 arg2 arg3 ...] [FILE] [< FILE]

    Options:
      --help      brief help message


=head1 OPTIONS

=over 10

=item B<--help>

Prints this help message.
```

[11] A minimum-viable Makefile

```
# Makefile for myprogram
PREFIX ?= /usr/local  # try 'make install PREFIX=$HOME/.local'

help:
    @echo "Type 'make install [PREFIX=path; default /usr/local]'." >&2

install: install-bin install-man

install-bin: myprogram
    install $< $(PREFIX)/bin

install-man: myprogram.1
    install -m644 $< $(PREFIX)/share/man/man1

myprogram.1: myprogram
    pod2man $< > $@
```

FIXME: Do a proper `Makefile.PL` for non-Unix OSes.

[12] ANSI colors with Term::ANSIColor

```
use v5.10;
use Term::ANSIColor;
say colored('Bold and blue', 'bold blue');

# or…
use Term::ANSIColor ':constants';
say BOLD RED "OH NOES!", RESET;
```

Why not ANSI all the things? You don't want ANSI escapes in your output if your output is getting piped to another program! So, instead…

```
BEGIN { $ENV{ANSI_COLORS_DISABLED} = 1 unless -t STDOUT }

use Term::ANSIColor ':constants';

my $ERR = BOLD RED 'ERROR', RESET;
die "$ERR: Everything is broken and terrible.";
```

(Perl::Critic will probably tell you to use IO::Interactive. It's OK.)

h/t: https://github.polettix.it/ETOOBUSY/2020/08/08/term-ansicolor

[13] BONUS: Stupid Perl tricks

Executable Perl modules (kinda like Python `__main__.py` inside a package):

```
# mymodule.pm
package MyModule;
use v5.10;
```

```
main() if not caller();

sub main  { say "I say, I say."; }
sub func1 { say "This is 'func1' doing its function thing."; }
1;
```

Now you can run `MyModule.pm` as a command, *or* `use` its functions:

```
$ perl module.pm
$ PERL5LIB=. perl -Mmymodule -e 'MyModule::func1();'
```

h/t: https://perlmaven.com/modulino-both-script-and-module

[14] BONUS: Stupid Bash tricks - programmable completion

If you have bash-completion installed:

```
# bash-completion provides the '_longopt' shell function
complete -F _longopt mycommand
```

Otherwise:

```
# make your own function (put this in your ~/.bashrc)
_mycommand() {
      mycommand --help | perl -ne 'print if s/\s+(--\w+).*/$1/'
}
complete -o default -F _mycommmand mycommand
```

Plenty more examples here: https://github.com/scop/bash-completion/tree/master/completions

[FIN] Thanks!

Thanks for your kind attention.

All materials (incl. a PDF handout) will be up shortly at https://github.com/ernstki/perfect-perl-clis