A Field Guide To The Perl Command Line

Andy Lester
andy@petdance.com
http://petdance.com/perl/

Where we're going

- Command-line == super lazy
- The magic filehandle
- The -e switch
- -p, -n: Implicit looping
- -l, -0: Record separator handling
- -a, -F: Field handling
- -i: Editing in place

-e: Your program

- The -e is your program. It's repeatable.
- Anything can go in here, even BEGIN blocks.
- Mind the quoting
- Mind the semicolons

-e examples

```
$ perl -e'print "Hello, World!\n"'
Hello, World!

# Perl as your calculator
$ perl -e'print 1024*1024/80, "\n"'
13107.2

# Mind the quotes: WRONG
$ perl -MCGI -e"print $CGI::VERSION"
# print ::VERSION

# Better & best
$ perl -MCGI -e"print \$CGI::VERSION"
$ perl -MCGI -e'print $CGI::VERSION'

# Windows
C:\> perl -e"print \"Hello, World!\n\""
```

The magic filehandle

- Perl does lots of the common stuff for you
- Diamond operator takes STDIN or file input from @ARGV files
- Modify your @ARGV before reading
  - Do command-line option parsing
  - Modify @ARGV on your own
- Currently-read filename is in $ARGV

Magic filehandle

```
for my $file ( @ARGV ) {
    open( my $fh, $file )
        or die "Can't open $file: $!\n";
    while ( my $line = <$fh> ) {
        # do something with $line
    }
    close $fh;
}

$ perl myprog.pl file1.txt file2.txt file3.txt

# Instead, do this:
while ( my $line = <> ) {
    # do something
}

# Also automatically works with redirection
$ grep blah blah blah | perl myprog.pl
```

-p & -n: Implicit looping

```
# -n wraps your code in this loop (basically)
while (<>) {
    # Your code goes here
}

# -p wraps your code in this loop (basically)
while (<>) {
    # Your code goes here

    print;
}
```

-p examples

```
# Program to print output with line numbers
# (in case cat -n doesn't do it for ya)
while (<>) {
    $_ = sprintf( "%05d: %s", $., $_ );
    print;  # implicitly print $_
}

# Try this instead
#!/usr/bin/perl -p

$_ = sprintf( "%05d: %s", $., $_ );


# or even shorter as:
$ perl -p -e'$_ = sprintf( "%05d: %s", $., $_ )'
```

-n examples

```
# Print commented lines
$ perl -n -e'print if /^\s*#/'

# Print values that look like dollars, like "$43.50"
#!/usr/bin/perl -n
while ( /\$(\d+\.\d\d)/g ) {
    print $1, "\n";
}

# Or total 'em up
#!/usr/bin/perl -n
BEGIN { $total=0 }
END { printf( "%.2f\n", $total ) }

while ( /\$(\d+\.\d\d)/g ) {
    $total += $1;
}
```

-l: line-ending handling

- Automatically adds or removes '\n'
- In effect:
  - chomp()s everything on input
  - Adds '\n' to each print
- A godsend for one-liners

-0: Input record sep

```
# That's hyphen-zero, not hyphen-oh.
# Lets you specify $/ from the command line.
# Value is in octal.
# You could use -e'BEGIN { $/="whatever"}'

# Work on a Mac file with chr(13) as the separator
perl -015 -e.....

# Special values:
-00 (zero zero) = paragraph mode (same as $/="")
-0777 = slurp mode (same as $/=undef)

# Print out all non-literal POD code:
$ perl -n -00 -e'print unless /^\s+/;' article.pod
```

-i: edit in place

- Opens each file, reads from it, and replaces it with STDOUT.
- Avoids the "make a foo file" dance
- Can specify a backup file like -i.bak
  - Old file foo.txt becomes foo.txt.bak

-a and -F: Autosplitting

- -a makes Perl split $_ into @F on whitespace
- Implicitly turns @F into a list of fields
- -F specifies what to split on if not whitespace

-a and -F examples

```
# Print total of 10th column from an Apache log
# (total number of bytes transferred)
$ perl -l -a -n -e'$n+=$F[9];END{print $n}' access_log


# Print all users that have a login shell
$ perl -l -n -a -F: \
  -e'print $F[0] unless $F[-1] eq "/bin/false"' \
  /etc/passwd

# Note that even though there are no slashes,
# -F: still means that the split regex is /:/
```

Option stacking

You can combine options on the command line, if they're not ambiguous.

```
$ perl -l -n -a -F: -e'....'
$ perl -lnaF: -e'....'
```

But don't do it. It adds complexity and potential bugs.

```
$ perl -p -i -l -e'$_=substr($_,0,40)' myfile.txt
$ perl -pil -e'$_=substr($_,0,40)' myfile.txt
```

What you think is -l is actually telling -i to append "l" to the backup file.

-m & -M: Module loading

- -mFoo does a "use Foo();"
  - Doesn't import any symbols
- -MFoo does a "use Foo;"
  - Imports any default symbols.
- -M-Foo does a "no Foo;"
  - But who uses "no" anyway?

-m/-M examples

```
# What version of CGI do I have?
$ perl -MCGI -le'print $CGI::VERSION'
2.89


# Some modules are meant for the command line
$ perl -MCPAN -e'install "Module::Name"'


# Text::Autoformat exports autoformat() by default
$ perl -MText::Autoformat -e'autoformat'
```

Wrapping up

- Perl respects command-line options on the #!perl line

```
$ perl -i -pe's/FOO/BAR/g'
```

```
#!/usr/bin/perl -i -p
s/FOO/BAR/g;
```

- This works on Windows, even though Windows doesn't use the shebang line itself.
- One-liner to convert Mac files:

```
$ perl -i.bak -l015 -pe1 *.txt
```