Minimal Perl for UNIX & Linux People

Part I: For all UNIX & Linux Users

Tim Maher
www.TeachMePerl.com
tim(AT)TeachMePerl.com
(866) DOC-PERL

11/13/06 Printing

Maximal Perl

Is the traditional view of Perl

Perl's famous motto:

- *There's More Than One Way to Do It!*

But nobody *really* needs

- **several different ways to express each common operation**

Minimal Perl
a carefully crafted dialect of Perl

POLICY:

- **there's no need for a UNIX user to learn *all* of Perl!**
  - at least, not initially

ALTERNATIVE:

- **concentrate on the UNIX-like features of Perl**
- **so you can learn quickly by capitalizing on your existing knowledge**

Target Audience for Part I
"UNIX/Linux People"

UNIX users

- **who have used `grep`**
  - to extract lines that match
- **maybe also `sed`**
  - to change text non-interactively
- **probably also `awk`**
  - maybe just for field processing

. . . but aren't necessarily *"Programmers"*

UNIX Shell Skills
help you learn Perl

Commands

- **have options and arguments**

Input is read from

- **filename arguments**
- **or STDIN -- pipe or keyboard**

Quoting

- **SQs disallow processing**
- **DQs allow some substitutions**

More UNIX Shell Skills
help you learn more Perl!

Many other elements same or similar

- **file tests**
  - `-r`, `-w`, etc.
- **here docs**
  - *something* `<<Eof`
- **logical operators**
  - `||`, `&&`

Even More UNIX Shell Skills
help you learn even more Perl!

Many other elements same or similar

- **filename generation**
  - `*.txt`, `[aeiou]*`
- **regular expressions**
  - `.*txt$`, `^[aeiou].*`
- **vi/sed-like commands**
  - `s/old/new/g`

Goals of this Talk

- **to teach you some Perl**
  - and that Perl is worth learning
- **to impress you with how much you can do with Perl**
  - while learning so little
- **to *inspire* you to learn more Perl later**

Dealing with Invocation Options

Common Obstacles
aka Stumbling Blocks

I'm confused about Perl's options; which ones should I use?

perhaps:

- `perl -wlne`

or maybe:

- `perl -wnl -e`

or how about:

- `perl -00 -Fwlnea`

Never fear, help is on the way!

Simplifying Perl Invocation Options
via aliases!

- **For commands that will only do output:**
  - alias `perl_o=' perl -wl'`
- **For input only, or input/output:**
  - alias `perl_io=' perl -wnl'`
- **For input/output with automatic printing:**
  - alias `perl_iop=' perl -wpl'`
- **For input only, or input/output, with fields:**
  - alias `perl_f=' perl -wnla'`

Simplifying Perl Invocation Options (cont.)
for Paragraph mode

- alias `Perl_io=' perl -00 -wnl'`
- alias `Perl_iop='perl -00 -wpl'`
- alias `Perl_f='  perl -00 -wnla'`

What Invocation Options Mean

`-wl:`	warnings, automatic carriage returns
`-wnl:`	adds input processing
`-wnla:`	adds field processing
`-00:`	enables "paragraph" mode
`-p:`	adds automatic printing
`-e:`	execute program in following argument

Okay, now forget those details; use the aliases!

Sample Output Program
performing a calculation

- **The `-e` argument introduces the program**
  - the aliases are incomplete without it
  - needs SQs around following program argument

UNIX command

```
$ expr 127 / 3
42
```

Perl alternative

```
$ perl_o -e 'print 127 / 3'
42.3333333333333
```

Sample Filter Program

Grepping for stuff

UNIX command

- `grep 'error'` *F1 ...*

Perl alternative

- `perl_io -e '/error/ and print;'` *F1 ...*

Perl as a (better) `grep` command

Capabilities of greppers vs. Perl

CAPABILITY	Classic greppers	POSIX greppers	Perl
Word boundary metacharacter	-	Y	Y
Compact character class shortcuts	-	?	Y
Control character representation	-	-	Y
Binary file matching	Y	Y	?
Line spanning matches	-	-	Y
Repetition ranges	Y	Y	Y
Metacharacter quoting	Y	Y	Y+
Advanced RE features	-	-	Y
Backreferences	Y	Y	Y+
Arbitrary record definitions	-	-	Y
Access to match components	-	-	Y
Match highlighting	-	Y	?
Custom output formatting	-	-	Y
Embedded commentary	-	-	Y
Directory file skipping	-	?	Y

Grep Shortcomings

- **can't match across lines** *(all greps)*
- **can't arbitrarily define records** *(all greps)*
- **can't show match in custom context, e.g., paragraph or page** *(all greps)***:**

```
    lines of paragraph above match
    line containing match
    lines of paragraph below match
```

- **no unambiguous way to represent control-characters** *(all greps)*
- **can't highlight matches** *(UNIX greps)*

Grep Shortcomings
(continued)

- **no later access to:** *(all greps)*
  - match itself (as opposed to its record)
  - individual components of match
  - pre- and post-match data
- **!! No *Standard Collection* of Metacharacters !!**
  - UNIX & GNU versions of `grep` & `egrep` are different

Surprise!

- **Perl corrects all these deficiencies**

Grep-like Perl Commands
how they work

```
perl_io -e '/RE/ and print;' F
```

`/RE/`: match regex against current line

`and`: makes print conditional on match

`print`: print current line (that contains match)

`F`: file to be examined for matches

Matching in Paragraph Mode
to see match context

Lines are matched by default:

```
$ perl_io -e '/Muddy/ and print ;' F
Muddy Waters (aka McKinley Morganfield)
```

Paragraphs matched using P* aliases

```
$ Perl_io -e '/Muddy/ and print ;' F
Muddy Waters (aka McKinley Morganfield)
was born in Rolling Fork, MS
```

NOTE: **grep** can't do this!

Displaying the Match Only

via "match" variable, `$&`

Problem: Want to see US postal codes only

Solution: Use "match" variable, `$&`

```
$ cat members
Jeff Healey M5T 1A1
Matthew Stull 98115

$ perl_io -e '/\d{5}$/ and
               print $&;' members
98115
$
```

NOTE: `\d` represents a digit; `{`*num*`}` specifies a quantity

Perl's String Escapes

STRING ESCAPE	NAME
\n	newline
\r	return
\t	tab
\f	form-feed
\e	escape
\NNN	octal value
\x*NN*	hex value
\c*X*	control character

Matching Using String Escapes

- **Can be hard to get a SQ into a SQ'd string:**

```
$ perl_io -e '/D'A/ and print;' data
>    # Shell is waiting for closing quote!
```

- **Effective, but difficult solution:**

```
$ perl_io -e '/D'"'"'A/ and print;' data
D'Addario & Company Inc.
```

- **Easier way:**

```
$ perl_io -e '/D\047A/ and print;' data
```

`\047`: "string escape" for apostrophe

Outlining Slashdot

a web-scraping application

- **Character #267 marks bullet items on some web-sites**
  - can be used to extract outline

```
$  lwp-request -o text slashdot.org |
>   perl_io -e '/\267/ and print;'
     · Microsoft Tracking Newsgroup Posters
     · SCO Prepares To Sue Linux End Users
     · Talk About A Security Hole, Go To Jail?
```

NOTE: **`grep`** lacks string escapes

The Matching Operator

format variations

Form	Meaning
/*RE*/	Match against $_
m:*RE*:	Match against $_
string =~ /*RE*/	Match against *string*
string =~ m:*RE*:	Match against *string*
string !~ m:*RE*:	Non-match against *string*
string !~ /*RE*/	Non-match against *string*

Advanced Matching
with line-spanning match

The following command

- **reads, and matches against, a paragraph at a time**
- **matches across lines within paragraphs**
  - using the "don't care" `.*` sequence
    - with `/s`, which allows `.` to match newline
- **captures (`(RE)`) and retrieves (`$1`) a portion of the match**

Advanced Matching (cont.)
with line-spanning match

```
$ ifconfig # Linux system
eth0  Link encap:Ethernet  HWaddr 00:D0:59:33:5F:60
      …
      UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
      …

lo    Link encap:Local Loopback
      …
      UP LOOPBACK RUNNING  MTU:16436  Metric:1
      …
$ ifconfig |
>   Perl_io '/^eth0 .*MTU:(\d+)/s and
         print "MTU for eth0 is: $1";'
MTU for eth0 is: 1500
```

NOTE: **grep** can't do line-spanning matches

Grep-like Perl commands

A Summary

grep command	Perl counterpart
grep '*RE*' file	perl -wnl -e '/*RE*/ and print;' file
grep -i '*RE*' file	perl -wnl -e '/*RE*/i and print;' file
grep -v '*RE*' file	perl -wnl -e '/*RE*/ or print;' file
grep -l '*RE*' file	perl -wnl -e '/*RE*/ and print $ARGV and close ARGV;' file
fgrep '*string*' file	perl -wnl -e '/\Q*string*\E/ and print;' file

Perl as a (better) `sed` command

The Sed Command
(not as famous as grep)

sed

- **main text processing command of early UNIX**
- **AWK replaced it in 1977 for most uses**
- **still used for text substitutions**

```
$ date | sed 's/Sat/Saturday/'
Saturday Apr 19 15:14:52 PDT 2003
$
```

Why Awk Replaced Sed

```
$ cat N   # : is field separator
Mr. Spongebob:Squarepants:SPONGE
Mr. Squidward:Tentacles:SQUID

$ awk -F':' '{ print $2 ", " $1 }' N
Squarepants, Mr. Spongebob
Tentacles, Mr. Squidward

$ sed 's/^\([^:][^:]*\):\([^:][^:]*\):.*$/\2, \1/' N
Squarepants, Mr. Spongebob
Tentacles, Mr. Squidward
```

IS THAT sed COMMAND A JOKE?

- No, we really used to process fields like that!

Sed Shortcomings

- **Deficiencies of UNIX sed**
  - can't match across lines
  - can't define custom records
  - match replacement not easily customizable
  - no string escapes to represent special characters
  - no "ignore-case" option *(UNIX* **sed***)*
  - can't modify original file *(UNIX* **sed***)*
    - *serious drawback for an editor!*

Perl as a Better Sed Command

Mass Editing: the Webmaster's Friend

- **Help! Our company's domain name just changed!**

```
$ cd HTML      # 5,362 files here!
$ perl_iop -i.bak -e '
>   s/\bacme.com\b/yakme.com/g;
>   ' *.html
$   # All done!
```

Perl as a Better Sed Command

How it Works

```
perl_iop -i.bak -e 's/old/new/g;' F
```

-i.*bak*: requests "in-place" editing

F.*bak*: stores copy of original file

Even More Better Perl Sed-er
Using Computed Replacements

- **eval**
  - is a Perl built-in function
  - compiles and executes Perl source code
- `s/RE/code/e`
  - `e` modifier on substitution operator
  - invokes Perl's `eval` facility
    - replaces RE with `code`'s *computed result*

Converting Miles to Kilometers
Using Perl's "eval" in a Substitution

```
$ cat drive_dist
            Van        Win        Tor
Vancouver   0          1380       2790
Winnipeg    1380       0          1300
Toronto     2790       1300       0

$ next_page_command drive_dist
            Van        Win        Tor
Vancouver   0          2208       4464
Winnipeg    2208       0          2080
Toronto     4464       2080       0
$
```

Converting Miles to Kilometers (cont.)

Using Perl's "eval" in a Substitution

- **can replace numeric values by ones 8/5ths greater**
  - using calculation on `$&`, which contains what was matched
- **can use `|` as alternate delimiter for `/`**

```
perl_iop -e 's|\d+| $& * (8/5) |ge;'
```

How does it Work?

```
perl_iop -e 's|\d+| $& * (8/5) |ge;' F
```

`s|`*`RE`*`|`*`X`*`|ge:` replace each match by result of ***X***

`\d+:` matches one or more digits

`$&:` contents of last match

Perl as a (better) `AWK` command

The Awk Command

The "Swiss Army Knife" of UNIX

AWK

- **combines** *Pattern Matching* **with** *Conditional Execution*
- **is designed for** *Data Validation*, *File Conversion*, *Report Generation*
- **automatically splits input into fields**
- **most common use:**
  - *field processing*

The Awk Command
Deficiencies

Deficiencies of AWK

- **few, given brilliance of its design; main ones are:**
  - no way to specify a range of fields
  - variable substitution doesn't occur within quotes

Awk vs. Perl
how they match up

Perl Advantages:

- **Perl has nearly all of AWK's capabilities**
- **Plus a whole lot more**

Perl Disadvantage:

- **Perl solutions are *never* as compact as their AWK counterparts**

Perl as AWK

Problem

Print first two fields in reverse order

AWK Solution

```
awk '{ print $2, $1 }' F
```

Perl Solution

```
perl_f -e '($A,$B)=@F;  # load fields
     print "$B  $A"' F
```

`@F`: field container, used by `-a`

`($A,$B)=@F`: copies field 1 into `$A`, 2 into `$B`

Perl as AWK
continued

- **Print first field if second matches pattern**
  - TAB character (\t) is field separator

```
perl_f -F'\t' -e '($A,$B)=@F;
      $B =~ /\b98107\b/  and
           print $A;' file
```

Input

```
Torbin Ulrich--->98107
Yeshe Sherpa---->98117
```

Output

```
Torbin Ulrich
```

Perl as (a better) Awk

Extracting Fields

- **Simple Perl field extractor**
  - by default, field separators are SPs and TABs
  - can list field numbers within [ ] in desired order
    - first field in `@F` array is #0
  - ascending range 1-3 specified as 1..3, etc.

Examples

```
perl_f -e 'print "@F[4,1..3]";' F
```

Extracting Fields: Example

```
$ cat staff
NAME PHONE DEP
Joel x3210 715
Jane x2046 229
 0    1    2       <= Field Numbers
$ perl_f -e 'print "@F[2,0,1]";' staff
DEP NAME PHONE
715 Joel x3210
229 Jane x2046
$
```

Perl as (a better) Awk
File Editing Applications

Unlike AWK,

- **Perl can do in-place editing on input file**
  - using `-i.bak` option

Examples

```
perl_f -i.bak -e 'print "@F[4,1..3]";' F
```

NOTE: `@F[4,1..3]`) is a shortcut for `@F[4,1,2,3]`;

AWK has no such shortcut

Perl has String Interpolation

AWK lacks it

AWK line-numbering program:

```
awk      '{ print NR  ": "  $0 }' file
```

Perl counterpart written AWKishly:

```
perl_io '  print $., ": ", $_; ' file
```

Perl counterpart written Perlishly:

```
perl_io '  print "$.: $_"; '     file
```

NOTE: The double-quoted string acts as a "template" for the output, making it easier to visualize

Perl has Pattern Ranges

Format:

```
/RE1/ ... /RE2/ and print;
```

Result:

- **Prints records between first that matches *RE1* and next that matches *RE2***

Example:

```
$ perl_io -e  '/^Oops:/ ... /^Code:/
>                and print;' messages
```

NOTE: *For simplicity, the timestamp-prefix has been removed from each line in* `messages`

Perl has Pattern Ranges (cont.)
like AWK

Match for `/^Oops:/ ... /^Code:/`

```
Oops: 0001
CPU:    0
EIP:    0010:[__remove_inode_page+79/144]
...
Process kswapd (pid: 4, stackpage=c9f31000)
...
Call Trace:    [shrink_cache+656/896] ? ...
...
Code: 89 50 24 89 02 c7 43 24 00 00 00 ...
```

NOTE: *An "Oops" message documents a problem with the Linux kernel.*

Perl as a (better) `find` command

`find | xargs` *`cmd`*

- **fiendishly clever technique for**
  - converting `find`'s output into `cmd`'s arguments
  - works around argument-size limitations
  - minimizes invocations of `cmd`

`find | xargs` *cmd* example

Problem:

- **Find most recently modified regular file**
  - latest payment, newest subscriber, etc.

Solution:

```
 find . -type f  | # collect filenames
   xargs ls -lrt | # sort by mod-time
    tail -1        # show newest
-rw-r--r-- ...  2006-11-09 12:22 ./rygel.xvi
```

Cool!

- *Or maybe not ...*

`find | xargs` limitations

With this command:

```
find | xargs sorting-command | tail -1
```

- **no guarantee that all files will be sorted by a** *single invocation* **of** *sorting-command*
- **result will be the most recent file from the** *last batch processed*!

Perl as a (better) `find`/`xargs` Command
the solution

```
#! /usr/bin/perl -wnl
# most_recent_file

BEGIN { $newest=0; }

$mtime=(stat $_)[9]; # get file's mod-time

if (defined $mtime and $mtime > $newest) {
    $newest=$mtime; # save time
    $name=$_;       # save name
}

END { print $name; }  # report results
```

Perl as a (better) **`find`**/**`xargs`** Command *example*

```
$ find . -type f | most_recent_file
-rw-r--r-- ...  2006-11-09 12:22 ./scorpius
```

- **Not only *cool*, but this time it's *correct*!**

SUMMARY
Part I

With a prior understanding of UNIX

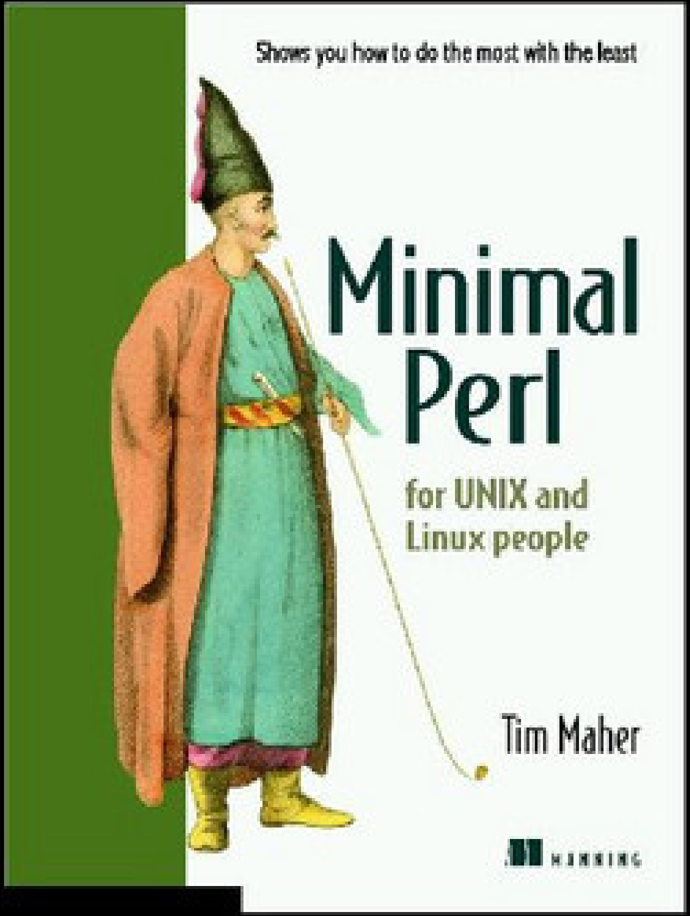
- **and knowledge of a few basic Perl techniques**
- **you can write simple Perl commands that are superior to their UNIX/Linux equivalents**

In Part II,

- **we'll show Shell Programmers how to write powerful Perl scripts**

CONCLUSION
and Shameless Plug

- I hope you enjoyed the presentation!
- To learn more along these lines,
  - check out my book!


Shows you how to do the most with the least
Minimal Perl
for UNIX and Linux people
Tim Maher
MANNING

www.MinimalPerl.com

That's All, Folks!

Thanks for your interest.

CONSULTIX
UNIX &
PERL
Training

Tim Maher

www.TeachMePerl.com
tim(AT)TeachMePerl.com
(866) DOC-PERL