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

08/08/05 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**

-- Maybe there's a better use for TMTOWTDI?

Minimal Perl

a carefully crafted dialect of Perl

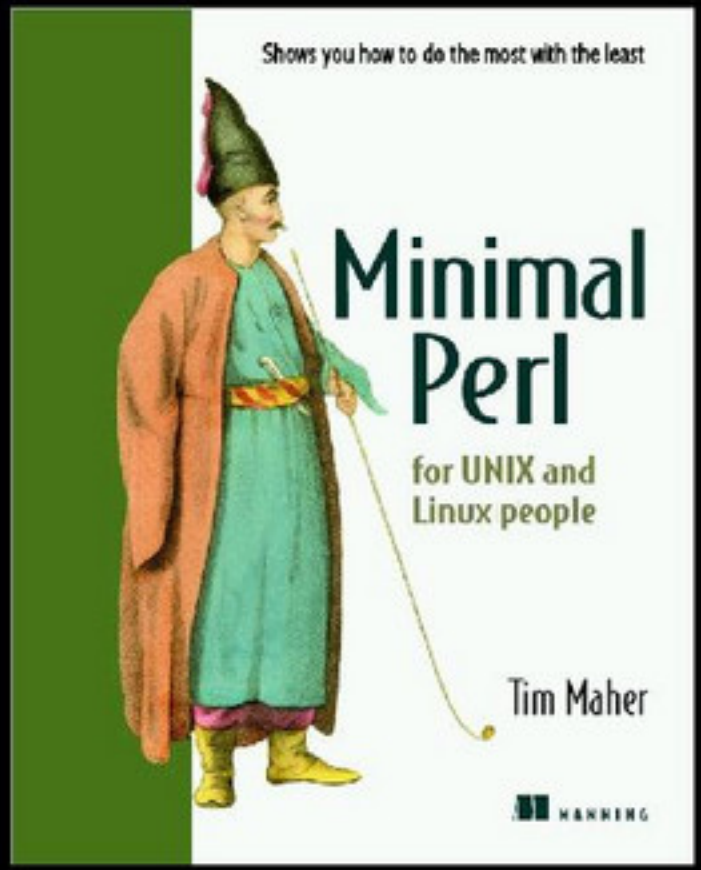
MORAL:

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

ALTERNATIVE, for UNIX People:

- **learn the "Minimal Perl" dialect**
  - used in this talk, and upcoming book

Coming in Fall, 2005



www.MinimalPerl.com

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"*

Target Audience for Part II
"UNIX Shell Programmers"

UNIX Shell Programmers

- **who are skilled in using**
  - variables,
  - conditionals,
  - loops, etc.
- **with the**
  - Bourne,
  - Bash,
  - and/or Korn shell

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**

But first, what is Perl?
a Command, or a Language?

The answer is: *BOTH!*

Perl can be used for

- **easily writing insanely powerful one-liners**

or

- **"hardly" writing large systems that might drive you insane**

Dealing with Invocation Options

Common Obstacles
aka Stumbling Blocks

- **"The options are too complicated"**
  - Do I use -wlne, or -plwe, -or -weln?"
    - Er, yes!
- **Don't worry, help is on the way!**

Simplifying Perl Invocation Options

via aliases!

- **Output only**
  - alias `perl_o='  perl -wl'`
- **Input only, or input/output**
  - alias `perl_io='  perl -wnl'`
- **Input only, or input/output with printing**
  - alias `perl_iop='perl -wpl'`
- **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!

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 '127 / 3'
42.3333333333333
```

Filter Program

Grepping for stuff

UNIX command

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

Perl alternative

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

Field-oriented Program

UNIX command

- `awk '{ print $1 }'` *F*
  - prints 1st field for each line

Perl alternative

- **loads field into named variable, for easier access**
- `perl_f -e '($F1) = @F; print $F1;'` *F*

Perl as a (better) `grep` command

Grep Shortcomings

- **Main Deficiencies of UNIX grep/egrep**
  - can't show much context for matches, like:

```
line above match
line containing match
line below match
```

    - can't match across lines
  - can't highlight matches
  - some lack word-boundary metacharacter
    - makes it hard to avoid sub-string matches!

Grep Shortcomings
(continued)

- **no legible control-character representation**
- **no custom record definitions**
- **no later access to match components:**
  - the match alone (as opposed to line)
  - individual components of match
  - data pre- and post- match

Surprise!

- **Perl corrects all these deficiencies**

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
Metacharacter quoting	Y	Y	Y+
Advanced RE features	-	-	Y
Backreferences	-	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

Perl as (a better) Grep

using word-boundary metacharacter

EXAMPLES

```
$ grep 'BOB' tv
SPONGEBOB SQUAREPANTS
BOB HOPE
$ perl_io -e '/\bBOB\b/ and print;' tv
```

OR

```
$ cat tv |
> perl_io -e '/\bBOB\b/ and print;'
BOB HOPE
$
```

\b: Perl's **"Word-Boundary"** Metacharacter

Perl as (a better) Grep

How it Works

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

/RE/: Match regex against current line

and: print is conditional on match

print: print current line (that contains match)

F: file to be examined for matches

Displaying the Match Only

via "match" variable, $&

Problem: Want to see US postal codes only

Solution: Use "match" variable, $&

```
$ cat members
Bruce Cockburn M5T 1A1
Matthew Stull 98115
$ perl_io -e '/\d\d\d\d\d$/ and
                print $&;' members
98115
$
```

NOTE: \d represents a digit

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* alias variations

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

Perl's Character Generators

Character Generator	Name
\n	newline
\r	return
\t	tab
\f	form-feed
\e	escape
\NNN	octal value
\xNN	hex value
\cX	control character

Outlines for Slashdot
a web-scraping application

- **Character \#267 marks bullet items on many 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 control-character codes

Repetition Metacharacters

- **Features Common to Egrep & Perl**

Syntax	Name
X|Y	Alternation
(X)	Capturing parentheses
\1, \2, ...	Backreference

- **Perl Enhancements**

$1, $2, ...	Numbered variable
(?:X)	Non-capturing parentheses

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

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 '/\QSTRING\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 easily modify original file
    - *serious drawback for an editor!*
  - match replacement not easily customizable
  - some versions, no word-boundary metacharacter
    - makes it hard to avoid sub-string matches:

Perl as (a better) Sed

Stream Editing Applications:

- **One-liner for typing directly to shell**
  - Benefit: Perl's enhanced *RE* metacharacters

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

How it works:

s/*old***/***new***/g:** change occurrences of *old* to *new*

Perl as (a better) Sed
(continued)

```
$ cat M
It was problematic; do you have a problem?

$ sed 's/problem/issue/g' M
It was issueatic; do you have a issue?

$ perl_iop -e 's/\ba problem\b/an issue/g;' M
It was problematic; do you have an issue?
$
```

Perl as a Better Sed Command

Mass Editing: the Webmaster's Friend

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

```
$ cd HTML       # 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
```

NOTE:

-i.bak: in-place editing; original now file **.bak**

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

$ special_perl_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 match by `X`'s result

`\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 one is:**
  - no obvious way to disable input parsing
  - very inefficient, if fields aren't used
- **another is:**
  - no way to specify range of fields
- **Perl has same capabilities and more**
  - but Perl solutions are rarely as compact

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";' directory
```

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 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
JanC 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**
  - by simply adding `-i.bak` option

Examples

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

File Editing Applications

NEW POLICY: Change "pants" to "trousers"

```
$ cat F
WORLDWIDE PANTS
SPONGEBOB SQUAREPANTS
$ sed 's/PANTS/TROUSERS/g' F # file unchanged
WORLDWIDE TROUSERS
SPONGEBOB SQUARETROUSERS
$ perl_iop -i.bak -e 's/\bPANTS\b/TROUSERS/g;'
  F
$ cat F # Perl can actually change original file!
WORLDWIDE TROUSERS
SPONGEBOB SQUAREPANTS
$
```

Custom Field Separators

- **to specify non-standard field separators**
  - provide literal characters, or regular expression after `-F` option

```
$ tail -2 /etc/passwd
timm:x:213:100:Tim
Maher:/home/timm:/bin/bash
spug:x:10012:200:SPUG:/home/spug:/bin/ksh
  0   1    2     3     4         5          6 <- field numbers
$ perl_f -F':' -e 'print "@F[0,6]";'
timm /bin/bash
spug /bin/ksh
. . .
$
```

SUMMARY
Part I

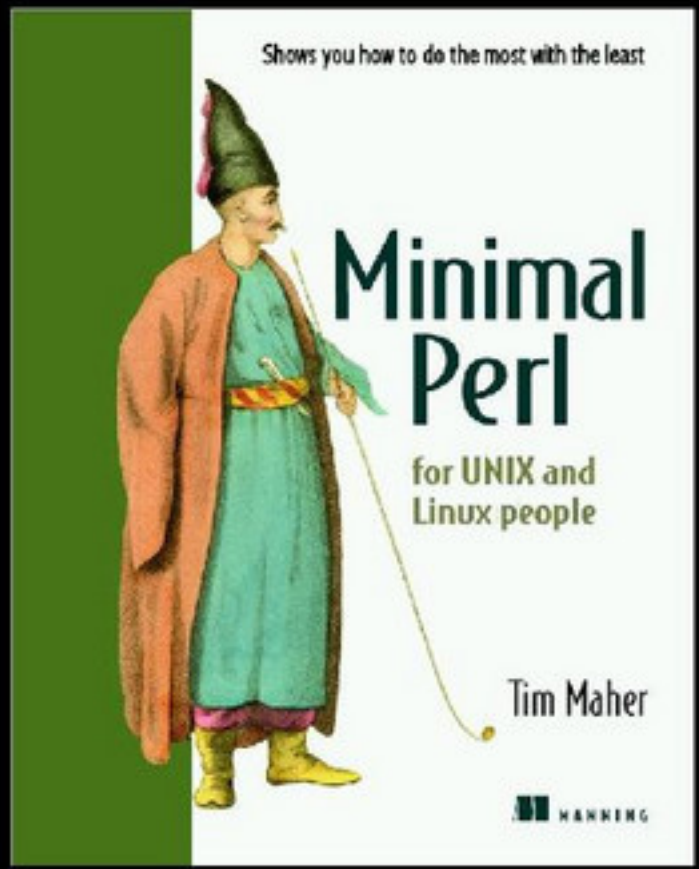
With a prior understanding of UNIX

- **and knowledge of a few basic Perl techniques**
- **you can write simple Perl commands that are superior to their UNIX equialents**
  - whether you're a UNIX *User* or *Programmer*

CONCLUSION
and Shameless Plug

- **I hope you enjoyed the presentation!**
- **To learn more along these lines,**
  - watch for my book!

Coming in Fall, 2005



www.MinimalPerl.com

That's All, Folks!

Thanks for your interest.

Tim Maher

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