

ADVANCED PERL

a language by Larry Wall

Practical Extraction and Report Language or Pathologically Eclectic Rubbish Lister

*Tom Christiansen
CONVEX Computer Corporation
<tchrist@convex.com>*

Copyright 1992, 1993

There's more than one way to do it!

TSC - perl.00

Sept 18, 1992

Slide 0

Eval for Code Generation

- Sometimes you want to generate your own code.

```
print "Enter perl code: ";  
$code = <STDIN>;  
print "DEBUG: $code\n" if $debug;  
eval $code;
```

- An `eval` counts as a block, so any local declared within it are freed upon its return, but subroutine and format definitions will persist.
- The return value is the last expression evaluated.

TSC - perl.01

Sept 18, 1992

Slide 1

Emulating Arrays of Arrays

- While you can't make an array of arrays, you can make an array of array names.
- For example, let's say what you really want is `$name[$i][$j]` and you don't want to use `$name{$i,$j}`. If `@name` contains a list of names of arrays, you can do this:

```
$ary = $name[$i];  
$val = eval "\$$ary[\$j]";
```

TSC - perl.02

Sept 18, 1992

Slide 2

Rename Example

- Think how many times you've wanted to translate all filename to lower case from upper, or rename all "`*.f`" files to "`*.f.bak`" using just one command.
- Here's a mini-program of Larry Wall's called *rename* that can be used in all these ways:

```
rename 's/\.orig$//' *.orig  
rename 'y/A-Z/a-z/ unless /^Make/' *  
rename '$_ .= ".bak"' *.f  
rename 'print "$_: "; s/foo/bar/  
      if <STDIN> =~ /^y/i' *  
find /tmp -name '*~' -print |  
      rename 's/^(.+)\~$/.#$1/'
```

TSC - perl.03

Sept 18, 1992

Slide 3

Rename Code

```
($op = shift)
|| die "Usage: rename expr [files]\n";
if (!@ARGV) {
    @ARGV = <STDIN>;
    chop(@ARGV);
}
for (@ARGV) {
    $was = $_;
    eval $op;
    die $@ if $@;
    rename($was, $_) unless $was eq $_;
}
```

TSC - perl.04

Sept 18, 1992

Slide 4

Exception Handling

- Sometimes you want to do something that would otherwise result in a fatal run-time error and still live to see the next line of code.
- Good candidates for this are features that may not be on all systems, like symbolic links, dbm files, and networking functions.
- If you wrap that code with an `eval`, then any fatals are caught. The error message will be found in the `$@` variable.
- `eval` may also take a block, making syntax errors appear at compile time instead of run time.

TSC - perl.05

Sept 18, 1992

Slide 5

Eval Examples

- Test for system-dependent features:

```
eval { symlink($x, $y); };  
warn "No symlinks ($@)" if $@;
```

- Catch fatal run-time errors:

```
# make divide-by-zero non-fatal  
eval { $answer = $a / $b; };  
warn $@ if $@;
```

TSC - perl.06

Sept 18, 1992

Slide 6

Exception Handling

- Where in other languages you might use some sort of catch and throw mechanism, in Perl you use an `eval` for `setjmp()` and a `die` for `longjmp()`.
- Here's an example to time out an input routine:

```
$SIG{ALRM} = TIMEOUT;  
sub TIMEOUT { die "restart input\n"; }  
do {  
    eval { &realcode() }  
} while $@ =~ /^restart input/;  
sub realcode {  
    alarm 15; $ans = <STDIN>; alarm 0;  
}
```

TSC - perl.07

Sept 18, 1992

Slide 7

The do Operator

- The operator **do** has three uses in Perl:
 - Calling a function — obsoleted by **&**.
 - Creating a **do { } while/until** loop.
 - As a raw form of **require** that doesn't do error checking.
- The last form is usually used when generating an include file on the fly, and is a lot like

```
eval 'cat $file'
```

except that it searches **@INC**.
- The **do { }** operator can be used to introduce a block anywhere in an expression, but is seldom used except in case-statement lookalikes.

TSC – perl.08

Sept 18, 1992

Slide 8

The Perl Library

- Perl comes with a many useful library routines; examples include **getopts** parsing, **ctime**, debugging routines, find emulation, an infinite precision arithmetic library, and many others.
- To access these, use the **require** statement.

```
require 'getopts.pl';  
&Getopts("ab:c:") || die "...";
```
- The **require** will raise a fatal exception if
 - The file is not found.
 - There is a syntax error compiling the file.
 - The file does not return a true value.

TSC – perl.09

Sept 18, 1992

Slide 9

Library Search Path

- Perl's library search path is in `@INC`, a list of directories.

```
print "@INC\n";  
/usr/local/lib/perl .
```

- The `PERLLIB` environment variable is a colon-delimited list of pathnames which will be prepended to `@INC` if set.

```
$ PERLLIB=/tmp/a:/tmp/b; export PERLLIB  
$ perl -e 'print "@INC\n"'  
/tmp/a /tmp/b /usr/local/lib/perl .
```

TSC - perl.10

Sept 18, 1992

Slide 10

Other Features of `require`

- Perl will only include a file once under the same name. It maintains a table in `%INC` of what you've included.
- The key is the filename you specified, and the value is the location of the file actually found.

```
require 'getopts.pl';  
print "$INC{'getopts.pl'}\n";  
/usr/local/lib/perl/getopts.pl
```

TSC - perl.11

Sept 18, 1992

Slide 11

find2perl

- The *find2perl* tool translates a *find* command into Perl code.
- Advantages:
 - Increased portability because it understands the same syntax on all platforms.
 - You can tweak the Perl code to do things yourself you couldn't do with plain *find*.
 - The resulting code is easier to maintain than very long *find* commands.
- Disadvantages:
 - Somewhat slower than ordinary *find*.

TSC - perl.12

Sept 18, 1992

Slide 12

find2perl Mechanism

- Call *find2perl* as you might call *find*. It will output Perl code that does what *find* would have done.
- The key is the *find.pl* library routine. All *find2perl* code contains:

```
require 'find.pl';
```
- The **wanted** function will be called on each file encountered. It is called with the full path name of the file to examine in `$name` and just the basename in `$_`.

TSC - perl.13

Sept 18, 1992

Slide 13

find2perl Example

- Let's clean out anything in /tmp over 3 days old:

```
$ find2perl /tmp -mtime +3 \  
    -exec /bin/rm -f {} ;  
  
#!/usr/bin/perl  
require "find.pl";  
&find('/tmp');  
  
sub wanted {  
    (($dev,$ino,$mode,$nlink,$uid,$gid) =  
        lstat($_) && int(-M _) > 3  
        && unlink($_);  
}
```

TSC - perl.14

Sept 18, 1992

Slide 14

Further Customization

- To change this from 3 days to 1/4 day (6 hours), simply change this line:

```
    int(-M _) > 3 &&  
to  
    -M _ > 0.25 &&
```

- To delete files created in the future (perhaps with *touch* or the Perl `utimes` function) test for a negative modification time:

```
-M _ < 0
```

TSC - perl.15

Sept 18, 1992

Slide 15

More Find Examples

- Locate symlinked files that point nowhere:

```
require 'find.pl';
&find('.');
sub wanted {
    -l && !-e && print "$name\n";
}
```

- To locate files with funky filenames:

```
sub wanted {
    tr/\001-\040\177//
    && print "<<<$name>>>\n";
}
```

TSC - perl.16

Sept 18, 1992

Slide 16

Mirroring a Directory

- This program duplicates a directory tree, creating actual directories but symlinks for everything else. NB: most error checking omitted for brevity.

```
for (($srcdir, $dstdir) = @ARGV) {
    -d && next;
    warn "$0: $_ is not a directory\n";
    &usage;
}

chdir $srcdir;
require 'find.pl';
&find('.');
```

TSC - perl.17

Sept 18, 1992

Slide 17

&wanted for *shadow*

```
sub wanted {  
    (($dev, $ino, $mode) = lstat($_));  
    $mode &= 0777;  
    $name =~ s!^\.\/!!;  
    if (-d _) {  
        mkdir("$dstdir/$name", $mode);  
    } else {  
        symlink("$srcdir/$name",  
            "$dstdir/$name");  
    }  
    1;  
}
```

TSC - perl.18

Sept 18, 1992

Slide 18

Pack and Unpack

- Perl strings can hold arbitrary binary data.
- Occasionally, you need to convert binary data into strings Perl can work with.
- Sometimes you need to convert Perl strings into binary data C can work with.
- The **pack** and **unpack** functions do this.
- Each takes a “template” describing the conversion; these templates are usually the layouts of a C struct declaration.

TSC - perl.19

Sept 18, 1992

Slide 19

Templates

- Syntax:
`$var = pack(TEMPLATE, LIST);`
`@ary = unpack(TEMPLATE, EXPR);`
- Each format character may be followed by a numeric repeat count; * means to gobble up the rest of the values.
- Converting between longs and doubles and back again may lose precision.
- Spaces are ignored, and so should be used to improve readability.

TSC - perl.20

Sept 18, 1992

Slide 20

Structure Templates

a/A	ascii string, null/blank padded
c/C	signed/unsigned char
s/S	signed/unsigned short
i/I	signed/unsigned integer
l/L	signed/unsigned long
n/N	short/long in "network" order
v/V	short/long in "Vax" order
f/d	float/double (native mode)
b/B	bits, asc./dec. bit order
h/H	nybbles low/high first
x/X	null byte/back-up a byte
@	null-fill until absolute position
u	uuencoded string
%n	n-bit checksum

TSC - perl.21

Sept 18, 1992

Slide 21

Template Examples

Template Interpretation

S4	four unsigned shorts
i f c8	an integer , a float, and 8 chars
A10 A20	10-byte string, 20-byte string
c3 x i	3 chars, skip one, an int
A5 A*	5-byte string, rest of string
%16C*	16-bit checksum

TSC - perl.22

Sept 18, 1992

Slide 22

Unpack vs Substr

- In dealing with fixed fields, it's faster to use **pack** and **unpack** than repeated calls to **substr**.
- Here's an example that pulls apart (and reconstructs) lines from the *ps* program:

```
# 15158 p5 T 0:00 perl foo.pl
$ps_t = 'A6 A4 A7 A5 A*';
open(PS, "ps|");
while (<PS>) {
    ($pid, $tt, $stat, $time, $command) =
        unpack($ps_t, $_);
    print "pid $pid took time $time\n";
}
```

TSC - perl.23

Sept 18, 1992

Slide 23

Udecode Example

- Perl supports the 'u' format for uuencoding. Here's a simple *udecode* program:

```
# udecode
while (<>) {
    next if 1 .. /begin/;
    print unpack ('u', $_);
    last if /end/;
}
```

TSC - perl.24

Sept 18, 1992

Slide 24

Binary Data

- Usually you use `read` to input binary data. This really calls *fread* (3S).
- This code acts much like the *who* program:

```
# needed for converting localtime() months
@mo = (Jan, Feb, Mar, Apr, May, Jun, Jul,
        Aug, Sep, Oct, Nov, Dec);

open(UTMP, '/etc/utmp')
|| die "can't open /etc/utmp: $!";
```

TSC - perl.25

Sept 18, 1992

Slide 25

Who Clone

```
while (read(UTMP,$utmp,36)) {
    ($line,$name,$host,$time) =
        unpack('A8 A8 A16 l', $utmp);
    if ($name) {
        $host = "($host)" if $host;
        ($sec,$min,$hour,$mday,$mon)
            = localtime($time);
        printf "%-9s%-8s%s %2d %02d:%02d"
            . " %s\n",
            $name,$line,$mo[$mon],
            $mday,$hour,$min,$host;
    }
}
```

TSC - perl.26

Sept 18, 1992

Slide 26

Bit Strings

- To help in manipulating binary data efficiently and to support the BSD `select` call, there is a special kind of string called a *bit vector*.
- A bit vector is just a string of arbitrary length that you can perform certain bitwise logical operations on.
- The `vec` function is used to manipulate these data: `vec(EXPR, OFFSET, BITS)`.
- The bitwise operators `|`, `&`, `^`, and `~` will assume bit-vector mode if their operands are strings.
- Like `substr`, `vec` may be used as an lvalue.

TSC - perl.27

Sept 18, 1992

Slide 27

Using vec() in Assignments

- **WARNING:** If you ever evaluate those strings in a numeric context, they lose their magic, so make sure you use **eq** and not **==** on them.
- The following code constructs a bit mask for use with **select**, assuming input may come from either the tty or a socket:

```
$rin = $win = $ein = '';  
vec($rin,fileno(TTYIN),1) = 1;  
vec($rin,fileno(SOCKET),1) = 1;  
vec($win,fileno(TTYOUT),1) = 1;  
$ein = $rin | $win;
```

TSC - perl.28

Sept 18, 1992

Slide 28

Using select()

- Wait for a while; use **undef** for **\$timeout** indefinitely.

```
( $nfound, $timeleft ) =  
    select( $rout=$rin, $wout=$win,  
           $eout=$ein, $timeout );
```
- Now check each one:

```
&readsock if vec( $rout, fileno( SOCKET ), 1 );  
&readtty  if vec( $rout, fileno( STDIN ), 1 );  
&writetty if vec( $wout, fileno( STDOUT ), 1 );
```
- You may also use **select** for sleeps:

```
select( undef, undef, undef, 2.25 );
```

TSC - perl.29

Sept 18, 1992

Slide 29

Dynamic Scoping

- Remember that `local()` introduces a new dynamically scoped identifier.
- Dynamic scope means that the visibility of a variable is according to the run-time call stack.
- A routine may see its own locals plus anything the calling routine had been able to see.
- Local protects you from your ancestors, not from your decedents!

TSC – perl.30

Sept 18, 1992

Slide 30

Package Scoping

- Static (lexical) scoping is available via the `package` facility. All identifiers actually “live” in some package.
- A fully-qualified identifier includes a package specifier, which is just two identifiers separated by a tick mark:

```
$packname'variable++;  
open(APACK'FILE, "< $somefile") || die;  
for (keys %network'listeners) { }
```

TSC – perl.31

Sept 18, 1992

Slide 31

Current Package

- Identifiers that are not fully-qualified are considered to be in the current package.
- The null package is the same as the main package: `$'foo` is the same as `$main'foo`.
- Perl programs begin in the `main` package.

TSC – perl.32

Sept 18, 1992

Slide 32

Package Examples

- Use the `package` statement to switch to another package:

```
package config;  
push(@files, $main'curfile);  
  
package main;  
push(@config'files, $curfile);  
  
push(@config'files, $main'curfile);
```

- The scope of a package statement is lexical, not dynamic, and lasts until the end of the current block.

TSC – perl.33

Sept 18, 1992

Slide 33

Libraries

- When writing library routines, you sometimes want to keep variables around between invocations, so you can't use `local()` for it.
- If you don't "declare" them, you will be using "global" variables at the outer scope, polluting your name space. By changing the namespace, you hide the names from the rest of your program.
- There's still no arbiter of package names, so if two files both say they're in `package syslog` then they'll share a common namespace.

TSC - perl.34

Sept 18, 1992

Slide 34

Static Data Example

- Perl supports module initialization code. This is any code outside a subroutine declaration in a required module.

```
package bigfloat;
require "bigint.pl";

$div_scale = 40;
$rnd_mode = 'even';

sub main'fneg { }
sub main'fabs { }

sub norm { }

1;
```

TSC - perl.35

Sept 18, 1992

Slide 35

Perl Pointers

- The ***ident** notation, known as a type-glob, refers to all types of the given identifier: **\$ident**, **@ident**, **%ident**, **&ident**, as well as formats and file and directory handles.
- While most often used to pass an aggregate datatype or file handle to a subroutine, with care, this construct can also be used much as one would use a pointer in C.
- Assigning ***foo** to ***bar** makes all **foo** objects aliases for all **bar** objects.

TSC – perl.36

Sept 18, 1992

Slide 36

The Perl Symbol Table

- You can examine (and even change) your own symbol table directly. The **main** symbol table is in **%_main**, the **foobar** symbol table is in **%_foobar**, etc.
- The identifier ***some_pack' ident** is exactly the same as **\$_some_pack{' ident' }**

TSC – perl.37

Sept 18, 1992

Slide 37

A Simple Dumpvar

- This finds all the symbols in a package:

```
&dumpsyms('main');  
sub dumpsyms {  
    local($package) = shift;  
    local(*stab) = eval("_$package");  
    while ($key, $val) = each(%stab) {  
        print "${package}' $key\n";  
    }  
}
```

- See `dumpvar.pl` for a more complete treatment of this.

TSC - perl.38

Sept 18, 1992

Slide 38

Using Pointers for Aliasing

- Notice that both `$color` and `&color` will be aliased below:

```
sub color {  
    print "color is $color\n";  
}  
$color = "red"; &color;  
*pen = *color;  
$pen = 'green'; &pen();  
  
color is red  
color is green
```

TSC - perl.39

Sept 18, 1992

Slide 39

Synthetic Data Structures

- While you can't have lists of lists or arrays of arrays, you can have lists or arrays of pointers, which you can then dereference.
- There's no way to declare a complex structure with one declaration — you have to declare each piece separately, then use a pointer to it in the larger structure.
- This becomes cumbersome after more than one or two levels of indirection.

TSC – perl.40

Sept 18, 1992

Slide 40

Example Synthetic Data Structure

```
@english = ('zero', 'one', 'two');
@spanish = ('zero', 'uno', 'dos');
%words = ( England, *english,
           Spain,   *spanish,);

for $place ( England, Spain ) {
    *nums = $words{$place};
    for $i (1..2) {
        print "$i in $place is $nums[$i]\n";
    }
}
```

TSC – perl.41

Sept 18, 1992

Slide 41

Output of Previous Program

```
1 in England is one
2 in England is two
1 in Spain is uno
2 in Spain is dos
```

TSC – perl.42

Sept 18, 1992

Slide 42

Fun with Scoping

- The program on the following page is an arguably abusive demonstration of some of the subtler aspects of Perl's scoping behavior. It takes *find -ls* output and makes the symbolic IDs numeric. Do *not* try this one at home, folks.
- See whether you can answer the following questions:
 - Why is/isn't this program recursive?
 - How many associative arrays does it maintain?
 - What are their names?

TSC – perl.43

Sept 18, 1992

Slide 43

Scoping Abuse in Action

```
$fmt = 'a26 A9 A8 a*';
while (<>) {
    ($pre, $login, $group, $post) =
        unpack($fmt, $_);
    print pack($fmt, $pre, &id(*login),
        &id(*group), $post);
}
sub id { local(*id) = @_;
    $id{$id} = $id =~ /^\\d+$/ ? $id : &id
        unless defined $id{$id};
    $id{$id};
}
sub login { (getpwnam($id))[2]; }
sub group { (getgrnam($id))[2]; }
```

TSC - perl.44

Sept 18, 1992

Slide 44

Symbol Generation

- Sometimes you want to create a new object with a unique name.
- We'll generate symbol names of the form "symbol" with a generation count appended to it.

```
sub gensym {
    eval '*symbol' . ++$gensym'symbol;
}
```

- Wondering what happened to the `return` statement? Remember that a function's return value, like a `do` block's, is simply the last expression evaluated.

TSC - perl.45

Sept 18, 1992

Slide 45

Generating a List of Lists

- If you don't know how many lists you're going to want, you can generate new ones on the fly using *gensym*.

```
@metalist = (); # list of pointers

for ('a' .. 'z') {
    &applist(*metalist, $_ .. "$_$_");
}

for (@metalist) {
    *list = $_;
    print "@list\n";
}
```

TSC - perl.46

Sept 18, 1992

Slide 46

List of Lists Workhorse

- Create a pointer to a new list
- Append this to master list of pointers
- Save remaining element in new list

```
sub applist {
    local(*plp) = shift;
    local(*nlist);
    push(@plp, *nlist = &gensym);
    @nlist = @_;
}
```

TSC - perl.47

Sept 18, 1992

Slide 47

Still More Elaborate Constructs

- There's no restriction that what you assign to a perl point be a representable identifier. However, if you do this, you'll only be able to get at it through dereferencing.
- The *dutree* program uses this property to maintain a recursive data structure. It read *du* output and delivers the output in a tree-like fashion, sorted according to how much disk space each subtree is using. See appendices for program.

TSC - perl.48

Sept 18, 1992

Slide 48

Anonymous Functions

- You can write a function that returns the name of a newly created function.

```
sub addto {  
    local($fname) = 'func' .++$addto'sym;  
    local($bump) = shift;  
    eval "sub $fname { shift + $bump; }";  
    $fname;  
}  
$func = &addto(3);  
print &$func(7);  
10
```
- The *plum* program (on convex.com) uses generated symbols for creating functions on the fly.

TSC - perl.49

Sept 18, 1992

Slide 49

Anonymous Packages

- *plum* maintains an associative array of anonymous packages that all contain members like `$Name`, `@Lines`, and `%Mappings` in each package.
- To switch active records, these symbols are reset to the local pointer in `main`:

```
eval "  
    package $folders{$name};  
    *'Name      = *Name;  
    *'Lines     = *Lines;  
    *'Mappings  = *Mappings;  
";
```

TSC - perl.50

Sept 18, 1992

Slide 50

Include File Conversion

- Sometimes you need to get at definitions from C include files, such as `EX_UNAVAILABLE`, `NOFILE`, or `ETXTBSY`.
- The *h2ph* program turns things like

```
#define ETXTBSY 26  
into  
sub ETXTBSY { 26 }
```

- You could then use this like so:

```
if ($! == &ETXTBSY) { }
```

TSC - perl.51

Sept 18, 1992

Slide 51

Translating Include Files

- To translate from *cpp* `#defines` into Perl:
`h2ph < foo.h > foo.ph`
- If you are the superuser, do this to install all system include files:
`cd /usr/include`
`h2ph *.h */*.h`
- Now include this in your program this way:
`require 'sys/errno.ph';`

TSC - perl.52

Sept 18, 1992

Slide 52

Problems with h2ph

- Anything but simple cases is hard:

```
#define n_hash n_desc
#define MNTTAB "/etc/fstab"
#if KERNEL
```

becomes

```
sub n_hash { &n_desc; }
sub MNTTAB { "/etc/fstab"; }
if (&KERNEL) { ... }
```

TSC - perl.53

Sept 18, 1992

Slide 53

Eval Protection

- If you call something that's not defined, you'll raise a fatal exception.
- Code in a `#if` will be protected by an `eval`. Something that's only incompletely defined may set but not raise the exception.
- Therefore you really should check `$@` for any `eval` errors.

```
require 'nlist.ph';  
$N_DATA = &N_DATA;  
die $@ if $@;
```

TSC - perl.54

Sept 18, 1992

Slide 54

Translating sys/ioctl.h

- Because of the complex macros in `sys/ioctl.h`, always check to make sure the calls are ok.
- *h2ph* doesn't know about ANSI-style *cpp* arguments employing `#` concatenation.
- Some extra work will almost always be needed to make these work.

TSC - perl.55

Sept 18, 1992

Slide 55

Massaging sys/ioctl.ph

- To get some ioctls to work with sys/ioctl.ph, you will need to either construct a %sizeof table of byte sizes of types.

```
#define TIOCGETD    _Ior('t',0,int)
#define TIOCGWINSZ _Ior('t',104,struct winsize)
```

become

```
eval 'sub TIOCGETD {
    &_Ior(ord(\'t\'),0,&int);
}';
eval 'sub TIOCGWINSZ {
    &_Ior(ord(\'t\'),104,struct winsize);
}';
```

TSC - perl.56

Sept 18, 1992

Slide 56

Massaging sys/ioctl.ph (continued)

- You'll need to change &int and struct winsize into 'int' and 'struct winsize'.
- Eventually someone will say in C

```
sizeof(int)
sizeof(struct winsize)
```

which will become after *h2ph* gets done with it,

```
$sizeof{'int'}
$sizeof{'struct winsize'}
```
- That means that you'd need to build a %sizeof table of everything that your include files try to take the sizeof.
- An alternative is to use *c2ph*.

TSC - perl.57

Sept 18, 1992

Slide 57

Return Values for ioctl and fcntl

- Most functions in Perl return some true value if they work, and a false one, namely `undef`, if they fail. (Two exceptions are `system` and `syscall`.)
- This is nice, because you can always do
`function || die;`
- However, `ioctl` and `fcntl` may need to return a success value of 0, so they really return the string "0 but true". Because of the way string-to-numeric conversions work (the `atof()` function), this is 0 numerically, but true in a logical context, which is precisely what you want.

TSC - perl.58

Sept 18, 1992

Slide 58

The Quick and Dirty Way

- If you don't want to be portable, or care more about speed, you can always just hardcode in the value for the `ioctl`. For example, to find that value of `TIOCSTI`, compile and run this:

```
#include <sys/ioctl.h>
main() {
    printf("0x%08x\n", TIOCSTI);
}

$ cc tio.c
$ a.out
0x80017472
```

TSC - perl.59

Sept 18, 1992

Slide 59

A Sample Quick and Dirty Program

- This stuffs its argument as a string into your input buffer if your system support the TIOCSTI ioctl.

```
sub jam {
    $TIOCSTI = 0x80017472 # sys dependent
    unless defined $TIOCSTI;
    for (split(//, $_[0])) {
        ioctl(STDERR, $TIOCSTI, $_)
        || die "bad TIOCSTI: $!";
    }
}
```

TSC - perl.60

Sept 18, 1992

Slide 60

More Hardwiring

- Here's another hardwired ioctl use. This is the *chop* program, which throws away characters beyond the current window length.

```
$TIOCGWINSZ = 0x40087468; $cols = 80;
ioctl(STDERR, $TIOCGWINSZ, $winsize) &&
    $cols = (unpack('S4', $winsize))[1];
$ARGV[0] =~ /^-(\d+)$/ && (shift, $cols = $1);
$cols--;
while (<>) {
    chop;
    $_ = substr($_, 0, $cols) . "0";
    print;
}
```

TSC - perl.61

Sept 18, 1992

Slide 61

Rolling Your Own Syscalls

- Many systems have their own system calls. If your C library supports the *syscall* function for indirectly calling a system call, then you can get at (most of) them.
- The rules for arguments to syscalls are that numeric arguments are passed as ints, and strings are passed as pointers.
- You must pre-extend your string long enough to receive any result that may be written into it.
- **syscall** returns 0 on success, -1 on failure.

TSC - perl.62

Sept 18, 1992

Slide 62

Syscall Example

- If we didn't have a **syswrite** function, you could emulate it this way;

```
require 'syscall.ph';  
syscall( &SYS_write, fileno(STDOUT),  
        "hi!\n", 4);
```

- Here's one to get the hostname:

```
require 'sys/syscall.ph';  
$h = " " x 100; # long enough  
syscall(&SYS_gethostname, $h, length($h))  
    && die "gethostname: $!";
```

TSC - perl.63

Sept 18, 1992

Slide 63

Using c2ph

- A serious problem in translating a C structure into Perl is the way people hardcode structure layouts, types, and sizes.
- The *c2ph* program offers a way to use your C compiler to automate this process. A program using the *c2ph* interface has a chance of portability. Hardwired layouts that assume they know sizes and types and padding haven't no chance.
- *c2ph* works by having your C compiler spit out debugging stab information, which it then converts into Perl code. If you don't have a BSD-style compiler, you'll need to use *gcc*.

TSC - perl.64

Sept 18, 1992

Slide 64

Setting up c2ph

- After running *h2ph* on an *.h include file, run *c2ph* on it as well, appending the output to the *.ph file.
- For *sys/ioctl.ph*, change the reference to `$sizeof{some_time}` to be `&some_time' sizeof()` instead.
- For more details about this, consult the file *c2ph.doc* in the Perl source directory.

TSC - perl.65

Sept 18, 1992

Slide 65

c2ph Entry Points

- For each struct or union type, a package of that name is created, with a standard set of standard entry points. In this way, it functions like a class with member function.
- Append the output from *c2ph* to all the *.ph files that *h2ph* made.
- The following entry points are supported:

sizeof	byte count
typedef	format for pack/unpack
offsetof	byte offset for vec()
typeof	the original C type
(field)	each field name

TSC - perl.66

Sept 18, 1992

Slide 66

rusage Example

- Let's say you want to do a call in Perl to find out how many signals you'd received. First, you need to set up the system call:

```
require 'syscall.ph';
require 'sys/resource.ph';
$SIG{ALRM} = CRIER;
sub CRIER { print "Ouch!\n"; }
for (1..10) { kill 'ALRM', $$; }
$buff = " " x &rusage'sizeof();
$rc = syscall(&SYS_getrusage,
              &RUSAGE_SELF, $buff);
die "getrusage: $!" if $rc;
@ru = unpack(&rusage'typedef()', $buff);
printf "you took %d signals\n",
      $ru[&rusage'ru_nsignals];
```

TSC - perl.67

Sept 18, 1992

Slide 67

A Better Who Clone

- The previous *who* program had the format and fields hardwired in. Unfortunately, not all systems represent a `struct utmp` the same way. See the reference material for a more portable version of this.
- Another way of unpacking to save repeating the package name is as follows:

```
($line,$name,$time,$host) = do {  
    package utmp;  
    (unpack(&typedef(),$'utmp')) [  
        &ut_line, &ut_name, &ut_time, &ut_host  
    ];  
};
```

TSC - perl.68

Sept 18, 1992

Slide 68

Network Programming

- Perl supports both the Berkeley socket mechanism and the System V `shm*` IPC calls.
- These are pretty much direct translations from the C routines, in that you have to pack things into binary formats. Since perl understands the length of its string, those arguments aren't required.
- The Perl man page has some simple TCP client and server code in it. The included *statmon* program demonstrates how to set up UDP socket and multiplex between the socket and the tty.

TSC - perl.69

Sept 18, 1992

Slide 69

gethostbyaddr Example

- Like many networking calls, the `gethostbyaddr` function requires that you pass it a binary-coded argument. You can't just say:
`gethostbyaddr("130.45.2.1");`
- This program shows how to pack up the argument correctly:

```
$AF_UNIX= 2; $IP_FMT = 'C4';
for (@ARGV) {
    split(/./);
    splice(@_, $#_, 0, (0) x (4-$#_));
    @hostent = gethostbyaddr(pack($IP_FMT, @_),
                             $AF_UNIX);
    printf "[%s] is %se0, join('.', @_),
           $hostent[0] || "<UNKNOWN>";
}
```

TSC - perl.70

Sept 18, 1992

Slide 70

Inefficient Sorting

- If you're going to sort a lot of records, and you can express this as a call to the *sort* program, you should do so. It's going to be faster.
- Sometimes, however, the sort criteria are too complex for this. Such cases often involve regular expression matches or crossreferencing auxiliary data.
- Consider this sort function:

```
@list = sort {
    ($a1) = $a =~ /(\w+-\w+)/;
    ($a2) = $b =~ /(\w+-\w+)/;
    $table{$a1} cmp $table{$a2};
} @list;
```

TSC - perl.71

Sept 18, 1992

Slide 71

Efficient Sorting

- Remember that we end up comparing every record with every other one in an $O(n \log(n))$ qsort algorithm. That means we perform the same CPU-intensive test more than once.
- It's far better to build up an index of keys and use those for the comparison.

```
for (@list) { push(@idx, /(\w+-\w+)/); }  
@list = @list [  
    sort {  
        $table{$a} cmp $table{$b};  
    } 0..$#list  
];
```