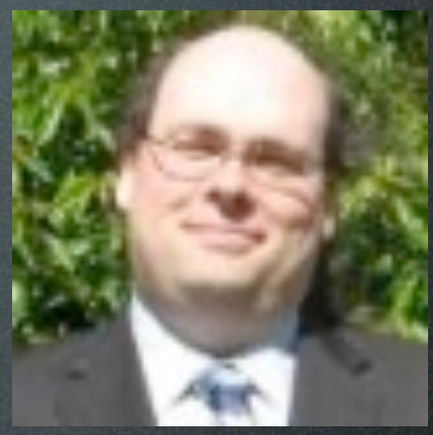
Cross Platform Perl

Write Once, Perl Everywhere

Presentation Info

Louis Erickson

<laufeyjarson@laufeyjarson.com>



- Programming in Perl for 15 years.
- Currently works at NVIDIA.

Presented on 12 August 2014 for the SF.pm user's group.

Thanks to eVault for having us tonight!

Introduction

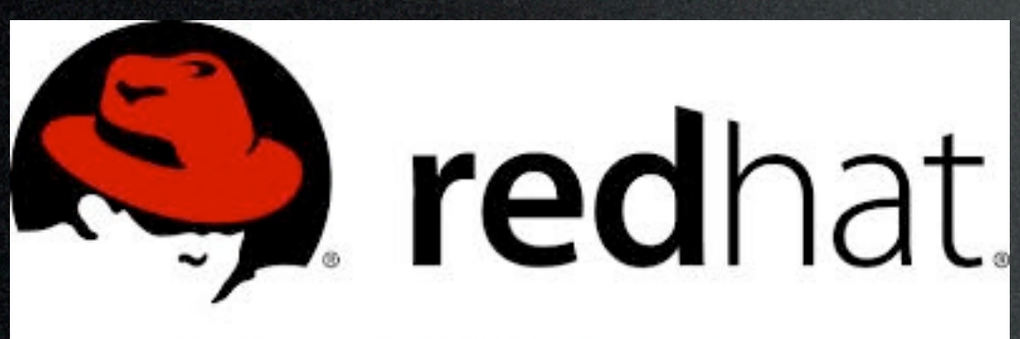
- Perl Platforms and History
- Writing Cross-Platform Perl
- Specific Gotchas!
- Final Demonstration
- Q&A

Perl Platforms and History

- 1.0: December 1987
- 2.0: June 1998
- 3.0: October 1989
- 4.0: March 1991
- 5.0: October 1994 (with many betas)

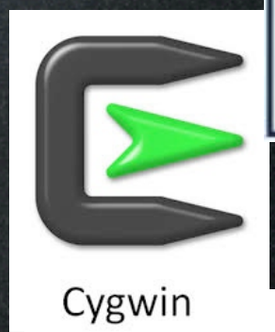
Versions You May Run Into

- 5.6: Microsoft distributes this with the Windows Resource Kit.
- 5.8.8: Used by RedHat and derivatives
- 5.10: First new features in a while
- 5.16: Just obsoleted
- 5.18: Supported. 5.18.2 from 6 Jan 2014
- 5.20: Supported. 5.20.0 from 27 May 2014



Platforms

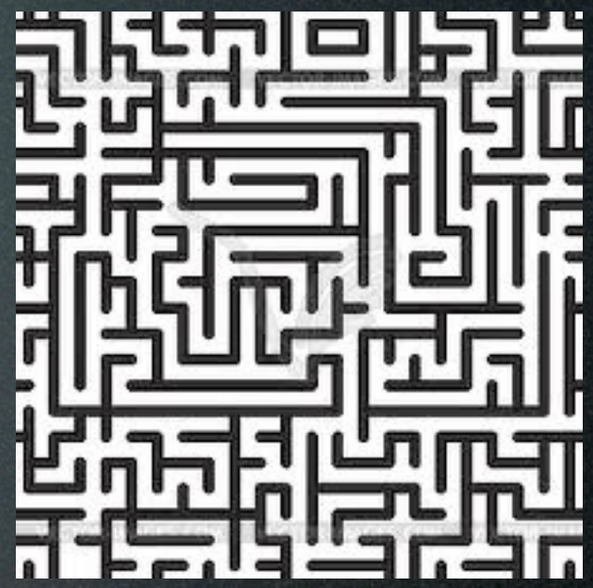
- Started on Unix machines.
- Ported to:
  - msdos
  - beos
  - cygwin
  - win32
  - vms
  - acorn (RISC)
  - AS/400 (EBCIDIC!)
- Many others.



Microsoft!

- In 1999 Microsoft funded ActiveState to port Perl to Windows.
- Resulted in a 5.6 port for Microsoft, and help keep ActiveState updating Perl.

Complex Perl API



- Perl's complex API offers enough interface to do most work.
- As long as a port supplies most of that interface, many Perl scripts Just Work.
- perldoc perlport has details

Writing Cross Platform Perl

- Stay in the interpreter
- Be careful accessing system resources
- Follow all the rules

Stay in the interpreter

- System-agnostic:
  - Logical operators (if/then/else/for/next/continue/etc.)
  - Arithmetic
  - Variables and Data Structures
  - Network I/O
  - Most File I/O <-- Some pitfalls

What Won't Port Easily

- Calls to external programs (system, backticks)
- Calls to external libraries (XS modules)
- System calls (Win32 API or fcntl calls)

Be Careful Accessing System Resources

- Avoid calls outside Perl when possible.
- File Names need special attention.
- Isolate required system-specific code.
  - system/backtick command lines
  - Compiled modules

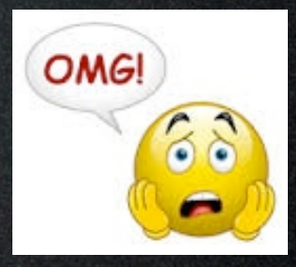
Specific Gotchas

- File Names
- File I/O
- Shelling out (system, backticks, open “|”)
- XS Modules
- Networking

File Names

- Different systems use different naming conventions.
- Sorting this out yourself is hard and error-prone.
- Use libraries!
- DON'T just glue stuff together and hope!

Pathy Details

Platform	Separator	Volume?	Notes
Unix/ Linux	/	No	Mac OSX is Unixy. Cygwin is Unixy.
Windows/ MSDOS	\	Yes	Windows API accepts / internally; Command line requires \
Old Mac System	:	Yes	

Spaces In Paths

- All modern systems can have spaces in path components!
- Need to be escaped to get them past the system shell.
- Escaping is very system specific; avoid it when possible by using multi-parameter system, backtick, etc.

File::Spec

- Platform independent path name handling
- Core Perl
- Provides logical ways to find paths
- Somewhat awkward, makes a single path three parts

Path::Class

- Nice object-oriented tool for path handling.
- Based on File::Spec internally - very compatible.
- Stringifies to correct path.
- Handles reading dir contents.
- Has to be installed from CPAN.

File I/O

- Text Translation
- binmode()
  - Everyone should use binmode!
- Open modes
- Redirection may not work
- Temp file - names + locations
  - use File::Temp if possible, Path::Class if not

Text Translation

- Perl tries to make every platform's idea of text match.
  - Internally, they all match Unix.
- Converts on input and output.
- This means "\n" means different things on different platforms.

Unixy Text

- \n read in stored as \n in strings
- \n written from string written as \n

Windowsy Text

- \r\n read in to \n in string
- \n written out to \r\n

Simple Text Works!

- Every platform uses “\n” in strings and regular expressions.
- I/O looks about as expected...
- ... until you get someone else’s text.
- Or binary data.

Stuff Is Easy on one Platform

```
#! env perl

while (my $line = <>) {
   if($line =~ /^(\d\d\d\d-[A-Z][a-z][a-z]-\d\d)\s+/) {
      print "Found date: $1\n";
   }
}
```

Enter binmode()

- Binmode sets a file handle to ':raw' I/O.
  - No translation at all.
    - Bytes left alone, assuming you are doing the right thing.
  - Can also set text modes for Unicode handling! Neat, but complex.

Shelling Out - Don't!

- system("mkdir -p /some/long/path");
- system ("cp filea fileb");
- system ("rm -rf /foobarbaz");
- system("touch file.tmp");
- my $hostname = `hostname`;
- my $now = `date`;
- my $result = `p4 submit -c 12345`;

Instead:

- Path::Class::Dir->mkpath
- File::Copy
- Path::Class::Dir->rmtree
- File::Touch
- Sys::Hostname
- POSIX::asctime
- p4perl module

Or even:

- File::Spec and File::Path::make_path (or while and mkdir)
- sysread, syswrite, close
- File::Spec and File::Path::remove_tree (or File::Find, unlink)
- open, utime
- Sys::Hostname - harder than it sounds
- time, sprintf
- p4perl module - harder than it sounds

Platform Specificity

- Perl knows what platform it's on - ask it!
- Isolate platform-specific code.

Exploring The Platform

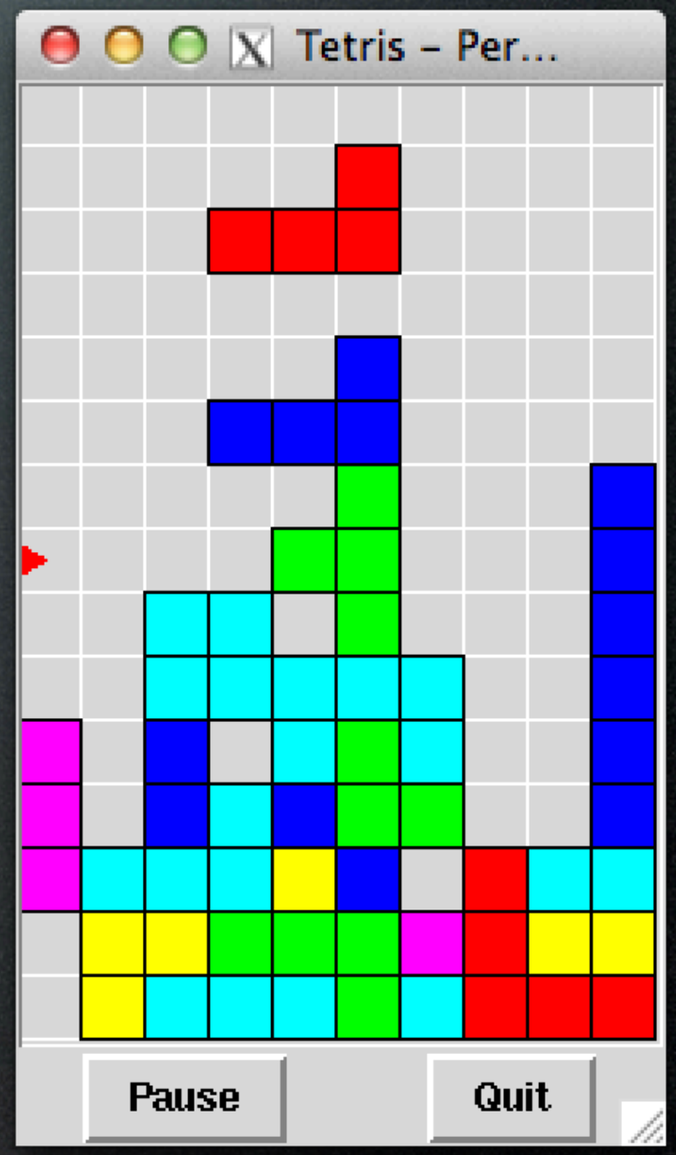
- The $^O/$OSNAME variable contains the name of the platform.
- The $^V/$PERL_VERSION variable contains the version of Perl running.
- $^X/$EXECUTABLE_NAME contains the path to the running Perl.
- Config module has TONS of information, word size, byte alignment, host info, etc.

Making Platform Decisions

- Isolate code
  - Functions or objects
- Test $OSNAME
  - Have to handle unknown case

GUI

- Several useful libraries.
  - Tk
  - Wx



Follow All The Rules

- binmode!
- File and path names.
- Avoid shelling out whenever possible
- Isolate system-specific code

Compiling Modules

- perl Makefile.PL
- make
- make test
- make install

Compiling, cont'd

- Make may be system dependent
  - Windows: nmake or dmake
- Lots of work on the part of authors
  - MakeMaker, Module::Build, Module::Install, DistZilla, etc!

Version Problems

- Getting identical versions everywhere is hard.
  - ActiveState's 5.18 can't build modules, so use 5.16...
- Libraries will vary across systems.

Distribution

- Sharing Perl scripts can be hard
  - Requirements
- Build a single package
  - FatPacker
  - PAR

Avoid System Perl

- Problems adding modules
  - Requires root/admin access
  - System updates can corrupt additions
- Vendors apply strange patches
- Version may be very old - or very new!

Build Your Own Perl

- perlbrew is your friend.
- Local::Lib can help
- CPAN::Mini to keep a copy of CPAN
- Pinto to control library dependencies

To Perl or Not to Perl

- Not To Perl:
  - Device Drivers
  - Plugins, other things that load as DLLs
  - Speed or size dependent
- When To Perl:
  - Everything else

Demonstration

- Same code, shared to several platforms:
  - Mac OSX
  - Windows, ActiveState Perl
  - Linux
- Same Perl version.

File::Spec

- Demo

Path::Class

- Demo

binmode()

- Demo

Determining Platform

- Demo

TK

- System independent GUI library.
- Written for use with TCL, ported lots of places.
- Pretty system independent.
- Demo

Resources

- perldoc perlhist
- perldoc -f binmode
- http://www.perl.com/pub/1999/06/activestate.html