



# Perl

*A Primer*

**Anuradha Weeraman**

anu@taprobane.org

<http://www.linux.lk/~anu/>

21 December 2005



# History & Roadmap

- Created by Larry Wall
- Powerful text handling
- Evolved with the Internet
- CGI / mod\_perl
- OOP
- Matured as a serious platform for development
- Perl 6 / Parrot



# Key Features

- Simple, intuitive, versatile syntax
- Cross platform
- Regular expression support
- Supports OOP semantics
- Open source – dual licensed
- Thriving community
- CPAN



# Supported Platforms

Linux, AIX, IRIX, HP/UX, BSD, Solaris, Tru64,  
DOS, Windows 3.1, Acorn Risc OS,  
Windows 95/98/NT/2000/XP, Apple Macintosh,  
Amiga, BeOS, OS/2, AS/400, OS390, VMS,  
OpenVMS, Stratus, Tandem, EPOC, QNX, Plan  
9, Atari ST, GNU HURD, Cygwin, HP 3000  
MPE etc.



# Practical Extraction & Reporting Language



Pathologically

Eclectic

Rubbish

Lister



There is More Than  
One Way To Do It

TMTOWDI  
(tim-towdi)



# Hello World

```
print "Hello World!\n";
```



# Hello World

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



# Hello World

```
echo 'print "Hello World!\n"' | perl -
```



# Running the First Script

File: HelloWorld.pl

```
#!/usr/bin/perl
```

```
print "Greetings, Earth People!\n";
```

```
$ perl HelloWorld.pl
```

**or**

```
$ perl < HelloWorld.pl
```

**or**

```
$ chmod a+x HelloWorld.pl
```

```
$ ./HelloWorld.pl
```



# Program Structure

- Lenient structure.
- No indentation, whitespace rules.
- Every simple statement ends in a semi-colon.
- All user created objects, except subroutines, are automatically created with a null or zero value.
- Comments are prefixed with a #
- Lines starting with = are interpreted as the start of a section of embedded documentation



# Data Types

- Scalars
- Arrays
- Hashes



# Scalars

- Simple variables
- Preceded by a \$
- Loosely typed

`$var = "random string";`

`$var = "with \n special \t characters";`

`$var = 23;`

`$var = 15.3439;`

`$var = 0xff; # Hexadecimal`

`$var = 033; # Octal`



# Arrays

- Ordered list of scalars
- Preceded by @
- Elements are accessed by subscript
- Elements are numbered starting from 0.
- Negative indexes count backwards

```
@array = (1, 1, 2, 3, 5, 8, 13);  
print $array[4];    # Prints 5
```



# Hashes

- Unordered set of key/value pairs
- Preceded by %
- Keys are used as subscripts to access elements

```
%lives = ( "cat" => 9, "dog" => 1.5);
```

```
%lives = ("cat", 9, "dog", 1.5);
```

```
$lives{'cat'} # Returns 9
```

```
$lives{'dog'} = 2;
```



# Conditionals

**if** (expression) {block} **else** {block}

**unless** (expression) {block} **else** {block}

**if** (expression1) {block}

**elsif** (expression2) {block}

**elsif** (lastexpression) {block}

**else** {block}



# Loops

```
while (<INFILE>) {  
    print OUTFILE, "$_\n";  
}
```

```
for ($i = 1; $i < 10; $i++) {  
    ... # Do something  
}
```



# Loops

```
foreach var (list) {  
    ...  
}
```

- Can be used to iterate through hashes and arrays

```
foreach $element (@array) {  
    print "$element\n";  
}
```



# Modifiers

statement **if** EXPR;

statement **unless** EXPR;

statement **while** EXPR;

statement **until** EXPR;

`$i = $num if ($num < 50); # $i will be less than 50`

`$j = $cnt unless ($cnt < 100); # $j will be >= 100`

`$lines++ while <FILE>;`

`print "$_\n" until /The end/;`



# Modifiers

```
do {  
    $line = <STDIN>;  
    ...  
} until $line eq ".\n";
```

```
do {  
    $line = <STDIN>;  
    ...  
} while length ($line) > 1;
```



# Loop Control

```
LINE: while (<SCRIPT>) {  
    print;  
    next LINE if /^#/;    # discard comments  
}
```

**last** label

**next** label

**redo** label



# Special Variables

- Perl has many special variables
- `$_` - implicit assignment

```
foreach ('hickory','dickory','doc') {  
    print;    # Prints $_  
}
```



# Special Arrays and Hashes

- `@ARGV` – Command line arguments passed into the script
- `@INC` – List of places to look for scripts and modules
- `%ENV` – Hash containing the environment the script is running in



# Special Filehandles

- STDIN
- STDOUT
- STDERR



# Subroutines

**sub** name {block}

**sub** name (prototype) {block}

name(args); # & is optional with parentheses

name args; # Parens optional if predeclared

&name; # Passes current @\_ to subroutine

- The implicit return value is the result of the last evaluated statement



# Switches

- perl -n assumes a  
while (<>) {  
    ... # your script goes here  
}
- around the command line arguments
- perl -p does the same while printing the line

Example: perl -ne “print if /PATTERN/” filename



# More Switches

- To search and replace text in a file:

```
perl -i.bak -p -e 's/match/replace/g' filename
```

- -i switch backs up modified files with a .bak extension.
- See `perl -help` for more switches



# Installing Modules

Download foo-module.tar.gz

```
$ tar zxvf foo-module.tar.gz
```

```
$ cd foo-module
```

```
$ perl Configure.PL
```

```
$ make
```

```
$ make test
```

```
# make install
```

**OR**

use CPAN.



# CPAN

- CPAN.org
- Comprehensive Perl Archive Network
- Mirrors all over the world
- Command line shell
- Bundled with standard Perl distribution
- Intuitive module management



# CPAN

**perl -MCPAN -e shell**

**cpan> install Term::ReadKey**

**cpan> install Term::ReadLine**

**cpan> install Bundle::CPAN**

**cpan> h or ?**



Thank You!