Perl

- Introduction
- Scalars
- Lists and arrays
- Control structures
- I/O
- Associative arrays/hashes

- Regular expressions
- Subroutines and objects
- Dealing with files
- Directory and file manipulation

Perl history

PERL stands for "Practical Extraction and Report Language" (although there is the alternative "Pathologically Eclectic Rubbish Lister".)

It was created by Larry Wall and became known in the 1990s.

It was available both from ucbvax and via Usenet.

Perl is released under the Artistic License and under the GNU General Public and License.

Perl's Artistic License

6. The scripts and library files supplied as input to or produced as output from the programs of this Package do not automatically fall under the copyright of this Package, but belong to whomever generated them, and may be sold commercially, and may be aggregated with this Package. If such scripts or library files are aggregated with this Package via the so-called "undump" or "unexec" methods of producing a binary executable image, then distribution of such an image shall neither be construed

as a distribution of this Package nor shall it fall under the restrictions of Paragraphs 3 and 4, provided that you do not represent such an executable image as a Standard Version of this Package.

7. C subroutines (or comparably compiled subroutines in other languages) supplied by you and linked into this Package in order to emulate subroutines and variables of the language defined by this Package shall not be considered part of this Package, but are the equivalent of input as in Paragraph 6, provided these subroutines do not change the language in any way that would cause it to fail the regression tests for the language.

Advantages of Perl

- Perl 5 is a pleasant language to program in.
- It fills a niche between shell scripts and conventional languages.
- It is very appropriate for system administration scripts.
- It is very useful for text processing.
- It is a high level language with nice support for objects.

A Perl program often will take far less space than the equivalent C or C++ program.

Perl is Interpreted

- Perl is first "compiled" into bytecodes; those bytecodes are then interpreted. Ruby, Python, and Java all do essentially the same thing.
- This is faster than shell interpretation, particularly when you get into some sort of loop. It is still slower than standard compilation.
- On the machine I tested, an empty loop in bash for 1 million iterations takes 34 seconds; 1 million iterations

of an empty loop in Perl takes 0.47 seconds. 1 million iterations of empty loop in C run in 0.001 to 0.003 seconds.

A Perl Program

```
#!/usr/bin/perl -w
# 2006 09 18 - rdl
use strict;
print ‘‘Hello, World!\n’’;
exit 0;
```

The first line indicates that we are to actually execute "/usr/bin/perl". (The "-w" indicates "please whine".) The second line is a comment. The third line makes it mandatory to declare variables. (Notice that statements are terminated with semicolons.) The 4th line does our

Hello World, and 5th line terminates the program.

Basic concepts

- There is no explicit "main", but you can have subroutines.
- Features are taken from a large variety of languages, but especially shells and C.
- It is very easy to write short programs that pack a lot of punch.

Similarities to C

- Many operators
- Many control structures
- Supports formatted i/o
- Can access command line arguments
- Supports access to i/o streams, including stdin, stdout, and stderr.

Similarities to shell programming

- Comment syntax of #
- $variables
- Interpolation of variables inside of quoting.
- Support command line arguments.
- Implicit conversion between strings and numbers.
- Support for regular expressions.

- Some control structures.

- Many specific operators similar to shell commands and Unix command syntax.

Scalars

Scalars represent a single value:

```
my $var1 = ‘‘some string’’;

my $var2 = 23;
```

Scalars are strings, integers, or floating point numbers.

There are also “magic” scalars which appear in Perl code. The most common one is $_, which means the “default” variable, such as when you just do a `print` with

no argument, or are looping over the contents of a list. The "current" item would be referred to by $_.

Numbers

Both integers and floating point numbers are actually stored as double precision values —unless you invoke the "use integer" pragma:

```
#!/usr/bin/perl -w
# Script19.pl
# 2006-09-18 - rdl. Illustrate use of "use integer"
use strict;
use integer;
my $w = 100;
my $x = 3;
print "w / x = " . $w/$x . "\n";
[langley@sophie 2006-Fall]$ ./Script19.pl
w / x = 33
```

Floating point literals

- Floating point literals are similar to those of C.
- All three of these literals represent the same value:

```
12345.6789
123456789e-4
123.456789E2
```

Integer decimal literals

☞ Similar to C:

```
0    -99    1001
```

☞ Can use underscore as visual separator:

```
2_333_444_555_666
```

Other integeral literals

- Hexadecimal:

```
0xff12   0x991b
```

- Octal:

```
0125    07611
```

- Binary:

```
0b101011
```

C-like operators

Operator	Meaning
=	Assignment
+ - * / %	Arithmetic
& | << >>	Bitwise
< >≤ ≥	Relational
&& || !	Logical
+= -= *=	Binary assignment
++ –	Increment/Decrement
? :	Ternary
,	

Operators not similar to C operators

Operator	Meaning
*	Exponetiation
¡=¿	Numeric comparison
x	String repetition
.	String concatenation
eq ne lt gt ge le	String relations
cmp	String comparison
=¿	Like comma but forces first left word to be string

Strings

Strings are a base type in Perl.

Strings can be either quoted to allow interpolation (both metacharacters and varialbes), or quoted so as not to be. Double quotes will allow this, single quotes prevent interpolation.

Single quoted strings using '

Single quoted strings are not subject to most interpolation.

However, there are two to be aware of: (1) Use \' to indicate a literal single quote inside of a single quoted string that was defined with '. (You can avoid this by using the q// syntax.) (2) Use \\ to insert a backslash; other \SOMECHAR are not interpolated inside of single quoted strings.

Double quoted strings

You can specify special characters in double quoted strings easily:

```
print "this is an end of line\n";

print "there are \t tabs \t embedded \t here \n";

print "embedding double quotes \" are easy \n";

print "that costs \$1000 \n";

print "the variable \$variable ";
```

String operators

- The period "." is used to indicate string concatenation.
- The "x" operator is used to indicate string repetition:

  `‘‘abc ’’ x 4` $\longrightarrow$ `‘‘abc abc abc abc ’’`

Implicit conversions atwixt numbers and strings

Perl will silently convert numbers and strings where appropriate.

For instance:

`"5" x "10"` $\longrightarrow$ `"5555555555"`

`"2" + "2"` $\longrightarrow$ `4`

`"2 + 2" . 4` $\longrightarrow$ `"2 + 24"`

Scalars

- Ordinary scalar variables begin with $
- They correspond to the regular expression $[a-zA-Z][a-zA-Z0-9_]*
- Scalar can hold integers, strings, or floating point numbers.

Declaring scalars

- I recommend you use the pragma `use strict;` – and if you do so, then you will have to explicitly declare all of your variables before using them. Use `my` to declare your variables.

- You can declare and initialize one or more variables with `my`:

```
my $a;
my ($a,$b);
my $a = ‘‘value’’;
my ($a,$b) = (‘‘a’’, ‘‘b’’);
```

☞ Variable declarations can occur almost anywhere

Variable interpolation

You can use the special form `${variablename}` when you are trying to have a variable name interpreted when it is surrounded by non-whitespace:

```
[langley@sophie 2006-Fall]$ perl
$a = 12;
print "abc${a}abc\n";
abc12abc
```

Undef value

A variable has the special value `undef` when it is first created (it can also be set with the special function `under()` and can be tested with the special function `defined()`).

An `undef` variable is treated as zero if it is used numerically.

An `undef` variable is treated as an empty string if it is used as a string value.

The `print` operator

- The `print` operator can print a list of expressions, such as strings, variables, or a combination of operands and operators.
- By default, it prints to stdout.
- The general form is `print [expression [, expression]*];`

The line input operator <STDIN>

- You can use <STDIN>to read a single of input:

```
$a = <STDIN>
```

- You can test for end of input with `defined($a)`.

The `chomp` function

You can remove the newline from a string with `chomp`:

```
$line = <STDIN>;
chomp($line);

chomp($line = <STDIN>);
```

The `chomp` function

```
[langley@sophie 2006-Fall]$ perl
chomp($line = <STDIN>);
print $line;
abcdefghijk
abcdefghijk[langley@sophie 2006-Fall]$
```

String relational operators

The string relational operators are `eq`, `ne`, `gt`, `lt`, `ge`, and `le`.

Examples:

```
100 lt 2
"x" le "y"
```

String length

You can use the `length` function to give the number of characters in a string.