

An Introduction to Perl

Jason Stajich
Duke University

1

The Perl Programming Language



- Perl is an interpreted language
- Initially designed and written by Larry Wall
- Language of System Administrators and “The Duct Tape of the Internet”
- Flexible, fast text processing
- Motto - “There is More Than One Way to Do It”

2

Jason Stajich, Duke University 2003

Why use Perl for Biology?

- Fast, efficient text manipulation
- Compact programs
- “Shell out” to run other programs with in Perl
- Suite of tools including Bioperl

3

Jason Stajich, Duke University 2003

Hello World

- `print "hello world\n";`

4

Jason Stajich, Duke University 2003

Getting situated

- all statements end with a ;
- comments start with #
- Brackets {} enclose code blocks

5

Jason Stajich, Duke University 2003

First Steps

- Syntax
 - variables: `$var`, `@array`, `%hash`
 - strings: `'literal'`, `"interpolateable"`
 - numbers: `0.11`, `2e-3`, `234`
 - scalars (`$var`), arrays (`@array`), associative array or hash (`%hash`)

6

Jason Stajich, Duke University 2003

How to use variables

- Store string value
 - `$species = 'Homo sapiens';`
- Print out
 - print `"my species is $species\n";`
- Assign to another variable
 - `$speciesa = $speciesb`

7

Jason Stajich, Duke University 2003

Strings

- Quotes
 - single quotes - `'literally $3.00'`
 - double quotes - `"value=$var\n"`
 - back quotes - ``execute in shell``
 - formatting -
`$str = sprintf("%s
%d", 'formatting', 9);`

8

Jason Stajich, Duke University 2003

String Functions

- **uppercase, lowercase** - `$A = uc($a), $b = lc($B)`
- **equality** - `if($a eq $b) {...}`
- **length** - `$len = length($str)`
- **substring** - `$substr = substr($str,3,5)`
- **index of a substring** - `$ind = index($str,"b")`
- **concatenate** - `$sum = $part1 + $part2`
- **reverse** - `$revstr = reverse($str);`

9

Jason Stajich, Duke University 2003

Easy String Things In Perl

- **Append to a string**
`-$str .= "stuff to add at the end";`
- **Remove the end if it is whitespace**
 - `chomp($str);`
- **Type in a bunch of text**
`-$str = qq{ This line and the next line will all be part of the string};`

10

Jason Stajich, Duke University 2003

Numerics and arithmetic

- **Integers** - `$num = 2030`
- **Floating points** - `$float = 0.400323`
- **Scientific notation** - `$value = 1.32e-23`
- **+, -, /, *** arithmetic, **^** for power, **log()** (base e)
- `$var++`, `$var--` (add or sub 1)
- `$a += 3; $num /= $num2;`

11

Jason Stajich, Duke University 2003

Arrays and Lists

- Array variable starts with **@**
- holds a list of items, can mix types (numbers, strings, objects, even other arrays)
- `@array = ('bannana', 'apple', 'cherry');`
- **Automatically resized**
`$array[1000] = '1';`
- Lists are anonymous arrays (unassigned)

12

Jason Stajich, Duke University 2003

Manipulating arrays

- Get the 3rd item in the array
`$item = $array[2];`
- Append a string to 1st array item
`$array[0] .= "-split";`
- Swap values
`$tmp = $array[2]; # save value`
`$array[2] = $array[3];`
`$array[3] = $tmp;`

13

Jason Stajich, Duke University 2003

Array functions

- Remove last item - `$item = pop @array;`
- Add to the end - `push @array, $item;`
- Remove 1st item - `$item = shift @array;`
- Add to the front - `unshift @array, $item;`
- Length of array - `$len = scalar @array;`
- `@revarray = reverse @array;`

14

Jason Stajich, Duke University 2003

From String to Array and back

- Turn string into array, based on delimiter
 - `@words = split(" ", "the quick brown fox")`
- Array back into a string
 - `$str = join(" ", @words)`

15

Jason Stajich, Duke University 2003

Sorting an Array

- `@sorted = sort @list;`
- By default will sort in lexical order
- Can specify a function
- Reverse lexical
`@sorted = sort { $b cmp $a } @list`
- Numeric
`@sorted = sort { $b <=> $a } @list`

16

Jason Stajich, Duke University 2003

Easy Array things in Perl

- Make an array of strings

```
—@spring_months = qw(Mar Apr May);
```

17

Jason Stajich, Duke University 2003

Associative Arrays or Hashes

- Key => value pairs
- Like arrays, except can use anything to index, rather than just integers
- No implicit order
- Each key has one value, special tricks to store more than one value per key

18

Jason Stajich, Duke University 2003

Manipulating Hashes

- Add a key,value pair

```
$hash{'dog'} = 'beagle';
```
- Initialize at once

```
%hash = ('dog' => 'beagle',  
         'cat'  => 'calico');
```
- Remove a value

```
delete $hash{'cat'};
```
- Test if a key has been set

```
if( exists $hash{'dog'} ) {}
```

19

Jason Stajich, Duke University 2003

Hash functions

- Get the keys

```
@keys = keys %hash;
```
- Get the values

```
@values = values %hash;
```
- Loop through both

```
while( ($key,$value) = each %hash) {}
```

20

Jason Stajich, Duke University 2003

References

- Represent memory address of variable (pointers)
- Either with leading \ or for arrays []
 - Note: These do slightly different things
- `$ref = \ $var;`
- `$arrayref = [@array]; $aref = \@array;`
- `$hashref = [%hash]; $href = \%hash;`

21

Jason Stajich, Duke University 2003

De-Referencing

- Get the data from the reference
- Have to know what was stored there (scalar, array, hash)
- `%$val, @$val, $$val`
- `@{ $val{ $key } }`

22

Jason Stajich, Duke University 2003

Most common use of references

- Storing arrays of arrays, hashes of arrays
- ```
my @month;
my @days = (M Tu W Th F Sa Su);
for (1..5) { # weeks
 push @month, [@days];
}

for my $week (@month) {
 print join(" ", @$week), "\n";
}
```

23

Jason Stajich, Duke University 2003

## Logic Construction

---

- ```
if ( something ) {
    something is true
} elsif ( somethingelse ) {
    somethingelse is true
} else {
    neither something nor somethingelse are true
}
```

24

Jason Stajich, Duke University 2003

Boolean operators

- equivalent - (numeric) == (characters) eq
- and (&&, and), or (||, or), not (!, not)
- unless is !if
 - if(!\$a){} same as unless(\$a)

25

Jason Stajich, Duke University 2003

Truth?

```
$var = "";  
if( $var ) { print "var is true\n";}  
$var = "1";  
if( $var ) { print "var is true\n";}  
$var = "0";  
if( $var ) { print "var is true\n";}  
$var = "greenmush";  
if( $var ) { print "var is true\n";}
```

26

Jason Stajich, Duke University 2003

Loops

- while (boolean){} or until (!boolean){}
- for(initialize; boolean test; eachincr){}
for(\$i=0;\$i<\$n;\$i++){}
for(\$i=\$n-1;\$i>=0;\$i--){}
for(\$i=\$n;\$i-->0){}
- foreach \$var (@list){} or for \$var (@list){}

27

Jason Stajich, Duke University 2003

Using Loops

- foreach \$g (qw(GENE1 GENE2 GENE3)){
print "g is \$g\n";
}
- @genes = qw(GENE1 GENE2 GENE3);
- for(\$i=0;\$i < scalar @genes;\$i++){
print "g is \$genes[\$i]\n";
}

28

Jason Stajich, Duke University 2003

Jumping around a loop

```
$counter = 0;
while( 1 ) { # infinite loop
    print "count is $counter\n";
    last if $count++ > 100;
}
foreach $file ( @files ) {
    if( index($file,"gene1") < 0 )
    {
        next; #go to the next item
    }
    print "file is $file\n";
}
```

29

Jason Stajich, Duke University 2003

Context

- Perl tries to be clever - will do different things depending on context
- `$len = @array;`
- `($first) = @array;`
- `(undef,$second) = @array;`

30

Jason Stajich, Duke University 2003

Subroutines

- Encapsulate a routine, defined input, and output
- Can return a list, hash, or single value
- Calling a subroutine
`$rc = &run();`
- Writing a sub routine
`sub run { #input args are in @_; }`

31

Jason Stajich, Duke University 2003

Subroutine Anatomy 101

- `@_` are the stack of input arguments to a subroutine
- 'return' will return scalar or list of items
- If no 'return', the results of the last entry will be returned:
 - ```
sub log_10 {
 ($a) = @_;
 log($a)/log(10);
}
```

32

Jason Stajich, Duke University 2003

## Subroutine writing

---

- `wantarray` - will return true if subroutine is called in array context, false if scalar context.
- `$a = &run;`
- `($a) = &run;`

33

Jason Stajich, Duke University 2003

## Input/Output

---

- `STDOUT`, `STDERR` output streams
- `STDIN` input stream
- `print "hello world\n"` uses `STDOUT`
- `print STDERR "hello world\n"`
- `open(OUT, ">filename") || die("$filename: $!");`

34

Jason Stajich, Duke University 2003

## Filehandles OUTPUT

---

- `open(FH, ">filename") || die($!);`
- `open($input, "| gzip -c > outfile.gz");`  
`print $input "a message\n";`
- `close(FH);`

35

Jason Stajich, Duke University 2003

## Filehandles Input

---

- Diamond operator `<>`
- `open(IN, $filename)`  
`|| die("$filename: $!");`  
`while(<IN> ) {`  
 `print "line is $_"`  
`}`
- `open($in, "grep 'gene' $datafile`  
`|");`  
`while(<$in>) {}`
- `close(IN);`

36

Jason Stajich, Duke University 2003

## Embedding data in a script

---

```
while(<DATA>) {
 @line = split;
 next if ($line[0] eq
'SYMBOL');
 print $line[0], " range:",
 $line[2] -
$line[3], "\n";
}
```

```
DATA
SYMBOL PRICE HIGH LOW
CNN 20.35 25.35 19.75
MSFT 31.27 108.61 29.13
AMZN 37.10 95.21 15.91
```

Jason Stajich, Duke University 2003

## Miscellanea

---

38

## Is it there, or not?

---

- defined \$var will test if a variable is defined
- For hashes 'exists' can test if a key has been set.
- defined \$hash{\$key}  
exists \$hash{\$key}
- \$hash{\$key} = undef;  
delete \$hash{\$key}

39

Jason Stajich, Duke University 2003

## "I'm out of here"

---

- die("exit with this message");
- exit(0); exit program here
- use Carp;  
warn("warnings can be printed here");  
croak("I'm croaking here");
- use Coy; # Haiku error messages  
warn("There is a problem");  
die("I've gone and died");

40

Jason Stajich, Duke University 2003

## Making Perl stricter

- Perl was written by lazy people, for lazy people
- Autovivication (auto creating variables) can be useful for throw away scripts, not so for mission critical things
- To make it stricter about variable declaration add these pragma to the top of your code
  - `use warnings;`
  - `use strict;`

41

Jason Stajich, Duke University 2003

## Scope

- Variables are valid within a certain scope of a program
- Scope is limited to blocks which are defined by `{ }`
- use `strict`; to insure respected scope
- requires all variables to be declared  
`my $var;` OR `my ($var1,@var2, %var3);`

42

Jason Stajich, Duke University 2003

## Context

- Perl can be sneaky (because it was written by/for lazy people)
- These are different contexts:  
`my @array = &routine();# list`  
`my $num = &routine();# scalar`  
`my ($first) = &routine();# list`

43

Jason Stajich, Duke University 2003

## Special variables

- `$_`, `$_` - implicit variable
- `@ARGV` - cmd line arguments
- `$,` - list separator for printing arrays
- In Regular expressions
  - `$`` - (prematch) string preceding last match
  - `$'` - (postmatch) string following last successful match

44

Jason Stajich, Duke University 2003

## Gotchas

- Indexes into Arrays and string start at 0
- use `strict`; to get Perl to respect scope
- Asking for things in wrong context (scalar in array context will give array size)
- Printing variable references will show mem location
- single quotes " will no interpolate variables, double quotes "" will
- Don't close `STDERR`, `STDOUT` filehandles

Jason Stajich, Duke University 2003

## Good Habits

- use `strict`;
- Comment as you go
- Start small - write in blocks/modules, test.
- Debug with print statements or perl debugger (`perl -d script.pl`)

46

Jason Stajich, Duke University 2003

## Help Resources

- `perldoc` - LOTS in there - Most of Learning Perl is examples from `perldoc`  
try running `'perldoc perldoc'`
- Learning Perl, Programming Perl
- [www.cpan.org](http://www.cpan.org) - Perl module Archive
- [search.cpan.org](http://search.cpan.org)
- [perl.com](http://perl.com) - (O'Reilly) has manuals, articles

47

Jason Stajich, Duke University 2003

## Regular Expressions

- Part of "amazing power" of Perl
- Allow matching of patterns
- Syntax can be tricky
- Worth the effort to learn!

48

Jason Stajich, Duke University 2003

## A simple regexp

- ```
if( $fruit eq 'apple' ||
    $fruit eq 'Apple' ||
    $fruit eq 'pear' ) {
    print "got a fruit $fruit\n";
}
```
- ```
if($fruit =~ /[Aa]pple|pear/){
 print "matched fruit
 $fruit\n";
}
```

49

Jason Stajich, Duke University 2003

## Regular Expression syntax

- use the =~ operator to match
- `if( $var =~ /pattern/ ) {}` - scalar context
- `my ($a,$b) = ( $var =~ /(\\S+)\\s+(\\S+)/ );`
- `if( $var !~ m// ) {}` - true if pattern doesn't
- `m/REGEXPHERE/` - match
- `s/REGEXP/REPLACE/` - substitute
- `tr/VALUES/NEWVALUES/` - translate

50

Jason Stajich, Duke University 2003

## m// operator (match)

- Search a string for a pattern match
- If no string is specified, will match \$\_  
\$0
- Pattern can contain variables which will be interpolated (and pattern recompiled)

```
while(<DATA>) {
 if(/A$num/) { $num++ }
}
while(<DATA>) {
 if(/A$num/o) { $num++ }
}
```

51

Jason Stajich, Duke University 2003

## Pattern extras

- `m//` -if specify m, can replace / with anything  
e.g. `m###`, `m[]`, `m!!`
- `/i` - case insensitive
- `/g` - global match (more than one)
- `/o` - compile regexp once
- `/x` - extended regexps (allows comments and whitespace)

52

Jason Stajich, Duke University 2003

## Shortcuts

- `\s` - whitespace (tab,space,newline, etc)
- `\S` - NOT whitespace
- `\d` - numerics ([0-9])
- `\D` - NOT numerics
- `\t`, `\n` - tab, newline
- `.` - anything

53

Jason Stajich, Duke University 2003

## Regex Operators

- `+` - 1 -> many (match 1,2,3,4,... instances)  
`/a+/` will match 'a', 'aa', 'aaaaa'
- `*` - 0 -> many
- `?` - 0 or 1
- `{N}`, `{M,N}` - match exactly N, or M to N
- `[]`, `[^]` - anything in the brackets, anything but what is in the brackets

54

Jason Stajich, Duke University 2003

## Saving what you matched

- Things in parentheses can be retrieved via variables `$1`, `$2`, `$3`, etc for 1st,2nd,3rd matches
- ```
if( /(\S+)\s+([\d\.\+\-]+)/ ) {  
  print "$1 --> $2\n";  
}
```
- ```
my ($name, $score) =
 ($var =~
 /(\S+)\s+([\d\.\+\-]+)/);
```

55

Jason Stajich, Duke University 2003

## Putting it all together

- A parser for output from a gene prediction program

56

Jason Stajich, Duke University 2003

GlimmerM (Version 3.0)  
Sequence name: BAC1Contig11  
Sequence length: 31797 bp

Predicted genes/exons

| Gene # | Exon # | Strand | Exon Type | Exon Range  | Exon Length |
|--------|--------|--------|-----------|-------------|-------------|
| 1      | 1      | +      | Initial   | 13907 13985 | 79          |
| 1      | 2      | +      | Internal  | 14117 14594 | 478         |
| 1      | 3      | +      | Internal  | 14635 14665 | 31          |
| 1      | 4      | +      | Internal  | 14746 15463 | 718         |
| 1      | 5      | +      | Terminal  | 15497 15606 | 110         |
| 2      | 1      | +      | Initial   | 20662 21143 | 482         |
| 2      | 2      | +      | Internal  | 21190 21618 | 429         |
| 2      | 3      | +      | Terminal  | 21624 21990 | 367         |
| 3      | 1      | -      | Single    | 25351 25485 | 135         |
| 4      | 1      | +      | Initial   | 27744 27804 | 61          |
| 4      | 2      | +      | Internal  | 27858 27952 | 95          |
| 4      | 3      | +      | Internal  | 28091 28576 | 486         |
| 4      | 4      | +      | Internal  | 28636 28647 | 12          |
| 4      | 5      | +      | Internal  | 28746 28792 | 47          |
| 4      | 6      | +      | Terminal  | 28852 28954 | 103         |

## Putting it together

```

• while(<>) {
 if(/^(Glimmer\S*)\s+\((\.\+)\)/ {
 $method = $1; $version = $2;
 } elsif(/^(Predicted genes)|(\s+\#)/ ||
 /\s+$/) { next
 } elsif(# glimmer 3.0 output
 /\s+(\d+)\s+ # gene num
 (\d+)\s+ # exon num
 ([\+|-])\s+ # strand
 (\S+)\s+ # exon type
 (\d+)\s+(\d+) # exon start, end
 \s+(\d+) # exon length /ox)
 {
 my ($genenum, $exonnum, $strand, $type, $start, $end,
 $len) = ($1,$2,$3,$4,$5,$6,$7);
 }
}

```

58 Jason Stajich, Duke University 2003

## s/// operator (substitute)

- Same as m// but will allow you to substitute whatever is matched in first section with value in the second section
- \$sport =~ s/soccer/football/
- \$addto =~ s/(Gene)/\$1-\$genenum/;

59 Jason Stajich, Duke University 2003

## The tr/// operator (translate)

- Match and replace what is in the first section, in order, with what is in the second.
- lowercase - tr/[A-Z]/[a-z]/
- shift cipher - tr/[A-Z]/[B-ZA]/
- revcom - \$dna =~ tr/[ACGT]/[TGCA];  
\$dna = reverse(\$dna);

60 Jason Stajich, Duke University 2003

## (aside) DNA ambiguity chars

---

- **aMino** - {A,C}, **Keto** - {G,T}
- **puRines** - {A,G}, **prYmidines** - {C,T}
- **Strong** - {G,C}, **Weak** - {A,T}
- H (Not G)- {ACT}, B (Not A), V (Not T), D(Not C)
- `$str =~  
tr/acgtrymkswbvdnxACGTRYMKSWHBV  
DNX/tgcayrkmswdvbhnxTGCA YRKMSWDV  
BHNX/;`

61

Jason Stajich, Duke University 2003

## Advanced Perl and Cookbook recipes

---

62

## Advanced Functions

---

- `grep` - search a list, only return elements which match
  - `@b = grep { $_ > 1 } @array;`
- `map` - apply some function to each item in an array
  - `@b = map { $lookup{$_} } @array;`

63

Jason Stajich, Duke University 2003

## Schwartzian transformation

---

- Useful if sorting on computed value
- `@data = ( 'spec:start:stop:strand',...);`
- `@dsort = sort {  
my ($a_sp, $a_start, $a_end) = split(/:/,$a);  
my ($b_sp, $b_start, $b_end) = split(/:/,$b);  
$a_sp cmp $b_sp || $a_start <=> $b_start; } @data;`
- `@dsort = map { $_->[0] }  
sort { $a->[1] cmp $b->[1] || $a->[2] <=> $b->[2] }  
map { [$_, split(/:/,$_)] } @data;`

64

Jason Stajich, Duke University 2003

## Get a Unique list

---

- ```
@ar = (1,2,1,17,1,37);  
my %seen;  
@unique = grep { !  
$seen{$_}++ }@ar;
```

65

Jason Stajich, Duke University 2003

Quick lookups

- Hashes are quick lookup tools ($O(1)$)
- If have a list of valid items, want to later test a new list for the valid items.
- ```
@valid = qw(gene1 gene2 gene3);
%lookup = map { $_ => 1 }
@valid;
foreach my $test (@testlist) {
 if($lookup{$test}) {
 # a valid item
 }
}
```

66

Jason Stajich, Duke University 2003