Perl Workshop

C. David Sherrill

Center for Computational Molecular Science & Technology

Georgia Institute of Technology

References

- These notes follow the progression given by the introductory book, “PERL in easy steps,” by Mike McGrath (Computer Step, Warwickshire, UK, 2004)
- Another good book is “Learning PERL,” by Randal L. Schwartz, Tom Phoenix, and Brian D. Foy (O’Reilly, 2005)
- See also www.perl.org and www.perl.com

Perl at a Glance

- High-level language
- Popular
- Easy to use for processing outputs
- Good for web CGI scripts
- Interpreted language --- not high-performance
- Remember to make your scripts executable (e.g., chmod u+x [scriptname])

Part 1: Variables and Printing

Printing in Perl

```
#!/usr/bin/perl

print “this is a test\n”;
# slash will escape quotes
print “I said \”hello!\” \n”;
print << “DOC”;
Any stuff between here & DOC will be printed
DOC
```

Scalar variables

- Perl doesn’t have strong typing like C/C++ for Fortran
- Perl tries to be smart about how to handle the type of a variable depending on context
- Can have scalar floating point numbers, integers, strings (in C, a string is not a fundamental scalar type)
- Scalars are designated by the $ symbol, e.g., $x

Scalar variable example

```
#!/usr/bin/perl

# initialize a string
$greeting = “hello”;

# initialize an integer
$number = 5;

# initialize a floating point number
$energy = -10.823;

print “Let me say $greeting\n”;
print “There are $number problems on the test\n”;
print “The energy is $energy\n”;
```

Formatted output

- It is also possible to print according to a specified format, like the printf() function in C

```
#!/usr/bin/perl
$pi = 3.1415926;
printf “%6.3f\n”, $pi;
# prints pi in a field 6 characters long with
# 3 digits after the decimal, rounding up
#  3.142
```

Array variables

- Unlike C or Fortran, an array in Perl can contain a mixture of any kinds of scalars
- Assigning an array to a scalar makes the scalar equal the *length* of the array (example of Perl trying to be smart)
- Arrays are designated by the @ symbol, e.g., @a

Array example

```
#!/usr/bin/perl

# set up an array
@array = (“hi”, 42, “hello”, 99.9);

# print the whole array
print “The array contains: @array\n”;

# access the 2nd element --- counting starts from 0
# note also we use scalar syntax ($) for a particular element
# because a single element is a scalar
print “The second element is $array[1]\n”;
# this prints 42 not “hi”

$length = @array;
print “There are $length elements in the array\n”;
```

Hash variables

- These contain key/value pairs and start with the % symbol, e.g., %h

```
#!/usr/bin/perl
%h = (“name”, “David”, “height”, 6.1, “degree”, “Ph.D.”);

# Note that each element of %h when accessed is a scalar, so
# use $ syntax to access an element, not %

print << “DOC”;
Name: $h{“name”}
Height: $h{“height”}
Degree: $h{“degree”}
DOC
```

Part 2: Operators

Arithmetic operators

- + : Addition
- - : Subtraction
- * : Multiplication
- ** : Exponential
- / : Division
- % : Modulus (remainder)
- ++: Increment
- -- : Decrement

Arithmetic operators example

```
#!/usr/bin/perl

$x = 3;
$y = 5;

$z = $x + $y;
print "$x + $y = $z\n";
# 3 + 5 = 8

$z = ++$x + $y;
print "$x + $y = $z\n";
# 4 + 5 = 9

$x = 3;

# watch out for this one
$z = $x++ + $y;
print "$x + $y = $z\n";
# 4 + 5 = 8
```

Assignment operators

Operator	Example	Same as
=	a = b	a = b
+=	a += b	a = a + b
-=	a -= b	a = a – b
*=	a *= b	a = a * b
/=	a /= b	a = a / b
%=	a %= b	a = a % b

Logical operators

Operator	Does
&&	Logical AND
||	Logical OR
!	Logical NOT

- These logical operators are very similar to those in C
- Used with operands that have boolean values TRUE
- and FALSE, or which can be converted to these values; typically 1 means TRUE and 0 means FALSE
- Unlike in C, FALSE is not always evaluated as 0. In the case of ! for NOT, !1 evaluates as a blank

Example of logical operators

```
#!/usr/bin/perl

$x = 1; $y = 0;

# example of AND
$z = $x && $y;
print "$x && $y = $z\n";
# prints 1 && 0 = 0

# example of OR
$z = $x || $y;
print "$x || $y = $z\n";
# prints 1 || 0 = 1

# example of NOT
$z = !$y;
print "!$y = $z\n";
# prints !0 = 1

# example of NOT
$z = !$x;
print "!$x = $z\n";
# prints !1 = 0 ? No, actually it leaves $z as a blank!
```

Numerical comparison

Operator	Comparison
==	Is equal?
!=	Not equal?
< = >	Left-to-right comp
>	Greater?
<	Less than?
>=	Greater or equal?
<=	Less than or equal?

- < = > returns -1, 0, or 1 if the left side is less than, equal to, or greater than the right side
- Other operators return TRUE if the comparison is true, otherwise it will be blank!

Numerical comparison example

```
#!/usr/bin/perl

$z = (2 != 3);
print "(2 != 3) = $z\n";
# prints (2 != 3) = 1

$z = (2 == 3);
print "(2 == 3) = $z\n";
# prints (2 == 3) =
```

String comparison

Operator	Comparison/Action
eq	is equal?
ne	not equal?
gt	greater than?
Lt	less than?
cmp	-1, 0, or 1, depending
.	concatenation
x	repeat
uc(string)	convert to upper case
lc(string)	convert to lower case
chr(num)	get char for ASCII num
ord(char)	get ASCII num of char

- Every individual character, like “A”, has a numerical code equivalent given by the ASCII table

String comparison example

```
#!/usr/bin/perl

$a = "hi";
$b = "hello";

$equal = $a eq $b;
print "$a eq $b = $equal\n";

$equal = $a eq $a;
print "$a eq $a = $equal\n";

$equal = $a ne $b;
print "$a ne $b = $equal\n";

$compare = $a cmp $b;
print "$a cmp $b = $compare\n";

$compare = $b cmp $a;
print "$b cmp $a = $compare\n";
```

String operators example

```
#!/usr/bin/perl

$a = "hi";
$b = "hello";

$c = $a . $b;
print "c = $c\n";
# prints "c = hihello"

$c = uc($a);
print "uc($a) = $c\n";
# prints "uc(hi) = HI"

$c = $a x 5;
print "$a x 5 = $c\n";
# prints "hi x 5 = hihihihihi"
```

The range operator

- The range operator, .., fills in a range of values in between the endpoints
- @numbers = (1..10) gives @numbers = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
- @letters = (“a”..”z”) gives an array with all letters “a” through “z”
- A “for” statement can also use a range operator to loop through a range, e.g.,
  “for (1..10) { print “hi” };” would print “hi” 10 times

Math functions

- PERL has several built-in mathematical functions

Function	Operation
abs(x)	return absolute value of x
sin(x)	return sine of x
cos(x)	return cosine of x
hex(string)	decimal value of hexadecimal string
oct(string)	decimal value of octal string
sqrt(x)	return square root of x

Part 3: Loops and Conditions

IF statements

- If the test expression is true, then execute the statement(s) following

```
#!/usr/bin/perl
$major = “chemistry”;

if ($major eq “chemistry”) {
  print “Welcome, chemistry student!\n”;
}
if ($major ne “chemistry”) {
  print “You’re not a chemistry student.\n”;
  print “Why not?\n”;
}
# note: need the curly braces
```

IF/ELSE statements

- Sometimes more convenient than just “IF” statements

```
#!/usr/bin/perl

$major = "chemistry";

if ($major eq "chemistry") {
  print "Welcome, chemistry student!\n";
}
else {
  print "You're not a chemistry student.\n";
  print "Why not?\n";
}
# note: need the curly braces
```

ELSIF statements

- “elsif” is read as “else if”. It’s an “else” that has an “if” condition attached to it; useful in picking one possibility out of a list of several

```
#!/usr/bin/perl

$grade = "F";

if ($grade eq "A") {
  print "Excellent!\n";
}
elsif ($grade eq "B") {
  print "Good work.\n";
}
elsif ($grade eq "C") {
  print "Needs improvement.\n";
}
else {
  print "I suggest you start coming to office hours.\n";
}
```

FOR loop

- Loop (repeatedly execute a statement block) until a given condition is met
- for (initializer, test, increment/decrement) {statement block}

```
for ($i=0; $i<3; $i++) {
  print "i = $i\n";
}
# prints the following:
# i = 0
# i = 1
# i = 2
```

WHILE loops

- Execute the statement block while a certain condition holds; watch out to avoid infinite loops!

```
# important to initialize variable before loop!
$i=0;

while ($i<3) {
  print "i = $i\n";
  $i++;              # need this line to avoid infinite loop!
}
# prints the following:
# i = 0
# i = 1
# i = 2
```

DO/WHILE loops

- Like “WHILE” but always executes at least once; test is made at end not beginning of statement block
- There is a related “DO/UNTIL” loop

```
# important to initialize variable before loop!
$i=0;

do {
  print "i = $i\n";
  $i++;              # need this line to avoid infinite loop!
}
while ($i < 3);
# prints the following:
# i = 0
# i = 1
# i = 2
```

NEXT statement

- Skip to next iteration of a loop
- Equivalent to C’s “continue” statement

```
for ($i=0; $i<3; $i++)
{
  if ($i == 1) { next }
  print "i = $i\n";
}
# prints the following:
# i = 0
# i = 2
```

LAST statement

- Skip out of loop and exit it completely
- Equivalent to C’s “break” statement

```
for ($i=0; $i<3; $i++)
{
  if ($i == 1) { last }
  print "i = $i\n";
}
# prints the following:
# i = 0
```

Part 4: Arrays

Working with arrays

- Elements are accessed by number, starting from 0; can use -1 to access the last element in the array
- A particular element of an array is accessed using $ syntax not @ (because each element is a scalar, not an array)
- To make an array of strings, the function qw() is a shortcut to put a list of items in quotes

Array example

```
#!/usr/bin/perl

@names1 = ("David", "Daniel", "Justin");
@names2 = qw(Mutasem Micah Arteum);     # avoid annoying quotes

print "@names1\n";
# prints David Daniel Justin

print "@names2\n";
# prints Mutasem Micah Arteum

print "$names1[1]\n";
# prints Daniel, *not* David!

print “$names1[-1]\n”;
# prints last element, Justin
```

Converting scalars to arrays

- Can take a scalar (like a text string) and split it into components (like individual words) and place them in an array
- Most frequently split using spaces or commas
- Use the split() function

Scalars to arrays example

```
#!/usr/bin/perl

$string = "We are learning PERL";
@words = split(/ /,$string);

print "@words\n";
# prints "We are learning PERL"

print "$words[1]\n";
# prints "are"

$prime_list = "1,3,5,7,11";
@primes = split(/,/,$prime_list);

print "@primes\n";
# prints 1 3 5 7 11
```

Going through all elements

- “foreach” statement creates a loop that goes through all the elements in an array

```
#!/usr/bin/perl

@tasks = qw(plan simulation analysis);

$i=0;
foreach $task(@tasks) {
  print "Task $i: $task\n";
  $i++;
}

# prints the following:
# Task 0: plan
# Task 1: simulation
# Task 2: analysis
```

Copying parts of arrays

```
#!/usr/bin/perl

@tasks = qw(plan simulation analysis);
@priorities = @tasks[0,1];

print "Tasks are: @tasks\n";
print "Priorities are: @priorities\n";

# prints the following:
# Tasks are: plan simulation analysis
# Priorities are: plan simulation

$tasks[1] = "computation";  #changes @tasks not @priorities
print "Tasks are: @tasks\n";
print "Priorities are: @priorities\n";

# prints the following:
# Tasks are: plan computation analysis
# Priorities are: plan simulation
```

shift/unshift and push/pop functions

- shift() deletes the *first* element of the array and returns that value
- unshift() adds a new element or elements to the *beginning* array
- pop() deletes the *last* element of the array and returns that value
- push() adds an element or elements to the *end* of the array

Example of shift/unshift

```
#!/usr/bin/perl

@grades = (100, 90, 89);
print "Grades are: @grades\n";
# Grades are: 100, 90, 89

unshift(@grades,54);
print "Grades are: @grades\n";
# Grades are: 54, 100, 90, 89

$deleted = shift(@grades);
print "Deleted the grade $deleted\n";
print "Grades are now: @grades\n";
# Deleted the grade 54
# Grades are now: 100, 90, 89
```

Other array tricks

- Combine two arrays like
  @new = (@arr1, @arr2);
- Replace an individual element like
  $arr[0] = 42;
- Get the length of an array like
  $len = @array;
- Take a “slice” (subset) of an array
  @subset = @arr[0,5];
- Get the reverse of an array
  @rev = reverse(@arr);

Sorting

- Can sort the elements of an array alphabetically; will not change the original array but can assign result to a new array. $a and $b are temp strings.

```
@students = qw(Robert Amanda Chris Jan);
print "students are: @students\n";
# students are: Robert Amanda Chris Jan

@students1 = sort{$a cmp $b}@students;
@students2 = sort{$b cmp $a}@students;

print "students1 : @students1\n";
# students1 : Amanda Chris Jan Robert
print "students2 : @students2\n";
# students2 : Robert Jan Chris Amanda
```

- Could do similar thing with numbers but using {$a ⇔ $b} for comparison

Part 5: Hashes

Hashes

- Key-value pairs; hash variables start with % symbol
- Very useful for keeping data from HTML forms
- Access a value by giving its associated key in curly brackets; the accessed value is a scalar, not a hash, so use $ in front

```
%hash = qw(first David last Sherrill);
# need slash below to distinguish the inner quotes
# in the hash lookup
# from the outer quotes of the print statement
print "first name: $hash{\"first\"}\n";
# first name: David
```

Slice of a hash

- Can take a slice (subset) of hash values, similar to taking a slice of an array. The result is an array of hash values.
- Specify the key names of the desired elements, in quotes, separated by commas. Taking an array, use array syntax.

```
%hash = qw(first David last Sherrill job Professor);

@names = @hash{"first","last"};
print "names: @names\n";
# names: David Sherrill
```

Getting all keys or all values

- Can get a list of all keys or all values in a hash using the keys() and values() functions, which take the name of the hash as the argument
- Warning: the order of the keys/values is not necessarily the same as the original ordering

```
%hash = qw(first David last Sherrill job Professor);

@karr = keys(%hash);
print "keys: @karr\n";
# keys: first last job

@varr = values(%hash);
print "values: @varr\n";
# values: David Sherrill Professor
```

Looping through hash elements

- Can loop through the elements of a hash using the “foreach” statement; like a “for” loop but goes through an array of elements
- Similar to “foreach” in shells like tcsh
- %hash = qw(first David last Sherrill job Professor);

```
foreach $i (keys(%hash))
{
  # note: below we do $hash not %hash
  print "The key is $i and the value is $hash{$i}\n";
}

# The key is first and the value is David
# The key is last and the value is Sherrill
# The key is job and the value is Professor
```

Deleting key/value pairs

- Can delete a pair using the “delete” statement followed by the *value* (a scalar) to delete

```
%hash = qw(first David last Sherrill job Professor);

delete $hash{"job"};

foreach $i (keys(%hash))
{
  # note: below we do $hash not %hash
  print "The key is $i and the value is $hash{$i}\n";
}

# The key is first and the value is David
# The key is last and the value is Sherrill
```

Does a key exist?

- Can check if a key exists in a hash using the “exist” keyword; returns 1 if exists, “blank” if not (can be converted to 0 when necessary)

```
%hash = qw(first David last Sherrill);

$check_first = exists $hash{"first"};
$check_age = exists $hash{"age"};

# "false" doesn't show up as a 0 unless "forced"
$num = ( $check_age == 0 ) ? 0 : 1;

print "Does first exist? $check_first\n";
# Does first exist? 1

print "Does age exist? $check_age\n";
# Does age exist?

print "variable num = $num\n";
# variable num = 0
```

Part 6: Text Files

Reading a text file

- Use “open” and “close” functions
- Need a “file handle” to represent the file
- Use equality operator to read a line or an array of (all) lines

```
# Note: file random.txt must be in same directory, or else
# must specify an absolute path

open(TXT, "<random.txt");    # open the file for reading
$line = <TXT>;               # get the first line (note scalar)
close(TXT);                  # close file again

print "The first line of the file is: $line\n";
```

Reading the whole file

- To get all the lines, simply assign <filehandle> to an array variable

```
open(TXT, "<random.txt");    # open the file for reading
@lines = <TXT>;              # get all the lines
close(TXT);                  # close file again

print "The file contains:\n";
print @lines;
```

Writing to a text file

- Use the > symbol in front of the filename to write, instead of < to read

```
open(TXT, ">written.txt");      # open the file for writing
print TXT "hello, testing!\n"; # write a line
print TXT "end of test.\n";     # write another line
close(TXT);                     # close file again
```

Appending to a text file

- To append (add to the end of an existing file), use the >> symbol before the filename instead of >

```
open(TXT, ">>written.txt"); # open the file for writing
print TXT "Add a line!\n";   # write an additional line
close(TXT);                  # close file again
```

Exclusive access

- Errors or unexpected behavior might result if two programs tried to write to the same file at the same time
- Can prevent this by putting a “lock” on the file, preventing other programs from accessing the file until the first program has completed the essential operation

File locking example

```
#!/usr/bin/perl

# Note: file testfile.txt must be in same directory, or else
# must specify an absolute path

open(FP, ">testfile.txt"); # open the file for writing
# note - not all platforms support flock()
flock(FP, 2);              # lock the file
print FP "Hello!\n";       # write a line
flock(FP, 8);              # release the file
close(FP);                 # close file again
```

Detecting read/write errors

- If a file operation has an error, it typically returns an error message to the $! variable
- This example previews subroutines

```
open(FP, "<junk.txt") || &pr_error($!);
@lines = <FP>;
close(FP);

foreach $line(@lines)
{
  print "$line";
}

sub pr_error
{
   print "Received error on opening file.\n";
   print "Error message: $_[0]\n";
   exit;
}
```

Renaming and deleting files

- To rename a file

  rename(“old_filename”, “new_filename”);
- To delete a file
  (don’t use unless you’re sure!)

  unlink(“file_to_delete”);

File status checks

Operator	Operation
-e	Does file exist?
-d	Is the “file” a directory?
-r	Is the file readable?
-w	Is the file writable?
-x	Is the file executable?

Status check example

```
$file = "crazy_file.txt";

# Another example of TRUE=1, FALSE=blank
# Will print blank if file doesn't exist
$e = (-e $file);
print "Variable \$e = $e\n";

# The following ? : logic still works though
print "The file $file ";
print $e ? "exists\n" : "does not exist\n";
```

Files in a directory

- Can get all the files in a given directory using the opendir() function

```
opendir(CDIR, ".");                  # . gives current directory
@filenames = readdir(CDIR);          # get all the filenames
@filenames = sort(@filenames); # sort them!
closedir(CDIR);

foreach $filename(@filenames)
{
  print "$filename\n";
}
```

Selecting certain filenames

- Can use the grep() function, in conjunction with a “regular expression” (see later), to select only certain filenames

```
opendir(CDIR, ".");            # . gives current directory
# get only filenames ending in .txt; escape the . character
@filenames = grep( /\.txt/, readdir(CDIR));
@filenames = sort(@filenames); # sort them!
closedir(CDIR);

foreach $filename(@filenames)
{
   print "$filename\n";
}
```

Setting permissions

- Can set the file permissions on a file or directory using the chmod() function which works like the UNIX command

Permissions	Owner	Group	Others
0777	rwx	rwx	rwx
0755	rwx	r-x	r-x
0644	rw-	r--	r--

chmod() example

```
if (-e "chmodtest")
{
  chmod(0755, "chmodtest") || &pr_error($!);
}
else
{
  print "Can't find file chmodtest\n";
}

sub pr_error
{
  print "Error: $_[0]\n"; exit;
}
```

Making and deleting directories

- Make a directory (needs UNIX permissions code)

  mkdir(“subdir”, 0755);
- Delete a directory

  rmdir(“subdir”);
- Best to check for errors, e.g.,

  rmdir(“subdir”) || &pr_error($!);

Changing working directory

- The script usually assumes it is working in the same directory it resides in
- This means files in other locations need to be addressed with full or relative paths
- Instead, can tell PERL to use a different “working” directory and then use “local” filenames
- chdir(“../docs”); # go back up to the “docs” directory and do all subsequent work in there