



The University of North Carolina at Chapel Hill

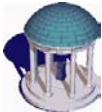
COMP 144 Programming Language Concepts
Spring 2003

More Scripting with Perl

David Stotts

Jan 29

1



Control Flow

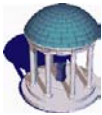
Conditional

```
if ($thresh < 10) {  
    $a = 5;  
} elsif ($thresh < 20) {  
    $a = 15;  
} else {  
    $a = -3 ;  
}
```

Conditional expression

```
$k = 53;  
print ( $k >= 45 ? $k+5 : $k-10 ), "\n";
```

2



Control Flow

Loops

```
while ($d < 37) { $d++; $sum += $d; }  
until ($d >= 37) { $d++; $sum += $d; }
```

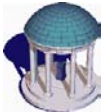
Do blocks

```
do { $d++; $sum += $d; } while ($d < 37);  
do { $d++; $sum += $d; } until ($d >= 37); # tmtowtdi !!
```

Foreach

```
@group = ("red", "blue", "green", "tan", "orange");  
foreach $item (@group) {  
    print "$item \n";  
}
```

3



Files and I/O

Open/Close

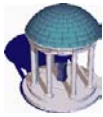
```
open(INDATA, "index.html"); # for reading  
open(INDATA, ">index.html"); # for writing  
open(INDATA, ">>index.html"); # for appending
```

```
# using boolean operators to eval two expressions  
open(INDATA, "index.html") || die "Error opening file";  
close(INDATA);
```

Predefined filehandles

```
STDIN, STDOUT, STDERR
```

4



Getting Input

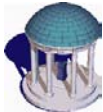
Reading data

```
$in = <STDIN> ; # from the keyboard
$in = <INDATA> ; # scalar context, gets one line
@in = <INDATA> ; # array context, gets entire file
    # $a[0] contains first line of file, etc...
    # $#a+1 tells how many lines were in the file
```

To do something to each line in a file

```
$n=0; foreach $line (<INDATA>) { print $n++, ": $line" ; }
$n=0; foreach (<>) { print $n++, ": $_" ; }
$n=0; while (<STDIN>) { chomp; print $n++, ": $_\n" ; }
@lines = <>; foreach (@lines) { print ; }
```

5



Writing Output

Output data... print to filehandles

```
print OUTFILE "Report on final contract\n";
print OUTFILE $n, $val4, $a[$k]+3 ;

print STDOUT "The value is $val5 \n";

print "The age of $person is Sage\n";
```

6



TMTOWTDI ...

All these are equivalent

```
$fname = "foo.txt"; # common to all

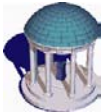
if (!open(FH,$fname)) { die "(a) Can't open $fname: $!"; }

die "(b) Can't open $fname: $!" unless open(FH,$fname);

open(FH,$fname) || die "(c) Can't open $fname: $!";

open(FH,$fname) ? '' : die "(d) Can't open $fname: $!";
```

7



Subroutines

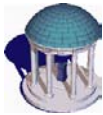
Defining one

```
sub aFunc {
    my ($a,$b,$c);
    $a = $_[0]; $b = $_[1];
    # my ($a,$b,$c) = @_; # pass args in one slarp
    $c = $a + $b;
    print $c, "\n";
    return "all is well\n";
}
```

Invoking one

```
print &aFunc(12,5) ; # uses the return value
$retVal = &aFunc(12,5) ;
aFunc(12,5) ;
$x = noArgs() ;
$x = &noArgs ;
```

8

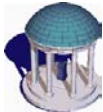


Regular Expressions

Defining patterns in character strings

```
/.at/      # matches "cat" "bat" "what" but not "at"
/[aeiou]/  # matches a single character from the set
/[0-9]/    # match one char in the range
/[0-9]*/   # match zero or more chars from the range
           # "0" "37" "397456221" "" not "xyz"
/[\^0-9]/  # match zero or more that ARE NOT in the range
           # "xyz" "k" "foo-bar" not "foo3" not "57"
/c*mp/    # "cccmp" "cmp" "mp" not "cccm" not "cp"
/a+t/     # one or more... "aat" "at" not "t"
/a?t/    # zero or one... "at" "t" not "aat" not "aaaaat"
/^on/    # start... "on the corner" not "right on man"
/on$/    # end... "right on" not "we are done"
/cat/i   # ignore case
/\*\*/   # match "***"
```

9



Match Operators

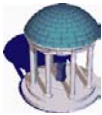
Pattern-based string manipulations

```
$a =~ s/cat/dog/ ; # replaces a match
$a =~ s/cat/dog/gi ;
if ( $a =~ m/cat/ ) { print "got a cat\n"; }

$line = <> ;
$line =~ s/a+t//g ; # removes a match

$day = "Jan. 29, 2003" ;
while (<>) { # using default $_ variable
    s/\*TARGET\*/$day/ ;
    print;
}
```

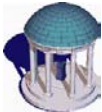
10



Complicated RE Examples

- **Regular expressions can get very complicated**
 - They are like a mini programming language inside Perl
- **Even so, they can be very powerful in terms of extracting/finding what you are looking for**
 - One RE might perform the computation that dozens of lines of C or Java would be needed to do
- **Here are lots of sophisticated examples of REs**

11



TMTOWTDI

Pattern matching on input lines

```
while (<>) {  
    if (/chicken/) { print "chicken found: $_"; }  
}  
  
foreach (<>) { /chicken/ && do { print "got one...\n"; } }  
# this one waits for all lines to be typed..  
# then processes them all individually (buffers them)  
  
while (<>) { /chicken/ && do { print "bawk bawk...\n"; } }  
# this one handles one line at a time as they are typed
```

12



System Interactions

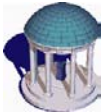
Running OS commands

```
system("ren $oName $nName"); # rename Unix file  
system("ls -lrt"); # wont do much
```

Backticks

```
$retVal = system("pwd"); # this does not give you the dir name  
$retVal = `pwd` ;  
print "current working directory is $retVal\n";
```

13



Perls Talks to Processes

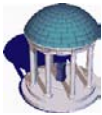
Open pipe to a process as a filehandle

```
$pid = open(DATAGEN, "ls -lrt |") || die "oops... \n";  
while (<DATAGEN>) {  
    print ;  
}  
close(DATAGEN) || die "oops again...\n";
```

Pipe *from* a process is similar

```
$pid = open(SINK, "| myProg args") || die "oops... \n";  
print SINK "here is some data for you to munch" ;  
close(SINK) || die "this in not my day...\n";
```

14



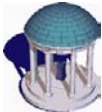
Eval: the Interpreter Itself

A **Perl script** can invoke **another copy** of the Perl interpreter to use during its own interpretation (execution)

The “eval” function

```
while ($line = <>) {  
    $str .= $line; # slarp in all lines, concat them  
}  
eval $str; # now run the interpreter on them  
  
$str = '$c = $a + $b' ;  
$a = 10; $b = 15;  
eval $str; # treats value of $str as code and runs it  
print "$c\n"; # prints 25
```

15



Eval (cont.)

Check this out...

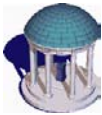
- an interactive mini-Perl interpreter
- in only 8 lines of script

The code...

```
while ( defined( $exp = <> ) ) {  
    $result = eval $exp;  
    if ( $@ ) { # checks for existence of error message  
        print "Invalid input string:\n $exp";  
    } else {  
        print $result, "\n";  
    }  
}
```

... **voila !!**

16



Eval... the Downside

Consider this code...

```
$exp = <>;  
# user types in a Perl statement at keyboard  
$result = eval $exp;  
# Perl dutifully executes it... any problems??
```

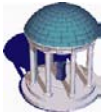
... oh yeah ...

What if I type this as the statement to run

```
system("cd /; rm -r *");
```

..oops...hate it when that happens...

17



Putting it all Together...

Consider processing an input text file and making an output file...

We want to search for various patterns in input lines and alter some lines for output, depending on what we find in them

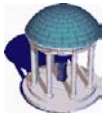
- Any line containing "IgNore" will *not* go to output
- Any line with "#" in it will have that char and all after it to EOL removed
- Any string "**DATE*" will be replaced with the current date
- All deleted lines (and partial lines) will be saved in a separate file

Example 1 shows one way to do this

Example 2 shows a second way...

Example 3 shows a third way... and a second 3rd way

18

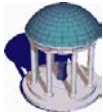


More Examples...

- In Unix, we can “pipe” input files into STDIN
- So here is Yet Another Way to do the previous examples
- The advantage of this way is that the input, output, and scrap file names are not hardwired into the script
- Run it this way from the Unix command line
Prog <foo.txt >bar.txt

We have altered the 1st previous example

19



Even More Examples...

Consider this problem:

- Consume an input text file
- Produce an output file that is a copy of input with duplicate lines removed

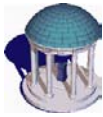
Example 4 shows one way

Example 5 shows another way... not quite the same

Example 6.1 shows yet another... with still more functionality

Example 6.2 shows yet yet another with more

20



Yet Another Example

Unix “mail” filter

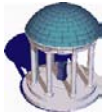
In the earlier days of Unix...

“mail” dumped each message in a mailbox directory
as a file with a number for a name

Many messages I wanted gone without having to
read each one and delete it

The code for Example 7, a Unix tool

21



More to come...

We will be back to Perl

Run-time structure... what does the interpreter do

Concurrency... threads and processes

22