IT 240
Shell Scripting for Administrators

Chapter 5
Input & Output

Stan J. Senesy
IT Program/CCS
New Jersey Institute of Technology

Reading from Standard Input

- As we saw earlier, capturing keyboard input is a relatively simple process:

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

- A more elaborate implementation to read multiple values:

```
while (defined($line = <STDIN>)){
print “I saw $line”;
}
```

More Input

- We could simplify the previous even further:

```
while (<STDIN>) {
print “I saw $_”;
}
```

- Or

```
foreach (<STDIN>) {
print “I saw $_”;
}
```

Even More Input

- Suppose our arguments are files and not just scalar values?
- We can use the diamond operator (<>) in place of <STDIN>

```
while (<>) {
chomp;
print “It was $_ that I saw!\n”;
}
```

Invocation Arguments

- We can populate the argument list manually by manipulating the @ARGV array

  @ARGV = qw# larry moe curly #;

- The diamond operator will work just as if *larry moe curly* had been passed as command line arguments

Producing Output

- We used print earlier to output the values of various things to standard output (usually the terminal)
- Printing an array and interpolating an array are different however:

  print @array; #onetwothree

  print "@array"; #one two three

- This can cause problems if you haven't used chomp!

More Output

- If we have a list of strings to print, we can use the <> operator, just like with input
- Parentheses can be a source of problems; in perl they're optional as long as their omission doesn't change the meaning of the statement
- Be careful when using the parentheses if they make a value expression look like a function call! (page 75)

Pretty Printing

- Just as in shell scripting, *printf* allows us to send formatted output to standard out
- Perl uses conversions *(%s, %d, etc)* to specify output format, just as we saw earlier in the course
- Output may be aligned by using a character width with the output conversion

Printing Arrays

- Arrays are not normally passed to printf since they old multiple values
- It is possible however to store the values in a single variable as an intermediary step to print:

```
my @items = qw( wilma dino pebbles)
my $format = “the items are:\n” . (“%s\n” x @items);
printf $format, @items
```

Files

- Perl does not live by standard input/output alone; we need files to keep us going
- Files are identified by a file handle, which is simply a connection from inside the program to the outside world
- Perl uses some reserved file handle names that you must avoid: *STDIN*, *STDOUT*, *STDERR*, *DATA*, *ARGV* and *ARGVOUT*

More Files

- File handles may be assigned using the open operator:

```
open CONFIG, “dino”;
open CONFIG, “<dino”;      #input only!
open BEDROCK, “>fred”;     #output only!
open LOG, “`>>logfile”;    #appending
open CONFIG, “<“, “dino”;  #new perl version
```

- Always test for success (or failure) when dealing with files!

And Finally

- Make sure you *close* every file that you open!
- If errors occur, use the *die* operator to exit normally yet still give you an indication of where the problem lies
- The *warn* function works similarly, but does not exit when an error occurs