ECOL 553L

Command Line Args, Perl Docs, File IO

Command line arguments

- Perl scripts can accept arguments that are specified on the command line when the script is run:

```
perl  top_hits.pl  5  "bln"
```

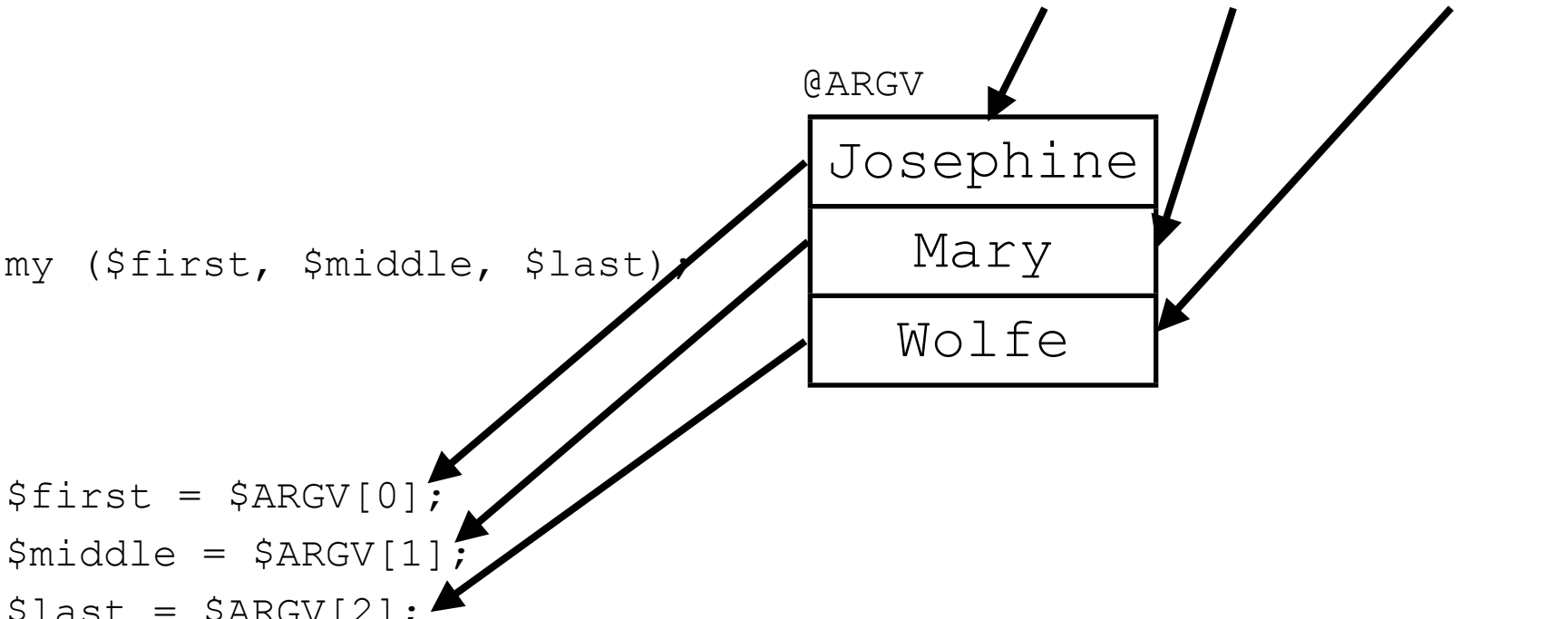
- Perl provides a special array named `@ARGV` that holds any command line arguments. Inside the script you can use elements of the `@ARGV` array just as you would with any array. Here is an example of using two command line arguments:

```
my $nhits = $ARGV[0];
my  $pat = $ARGV[1];
my  @files = glob("*.$pat");
…
```

More about Command line arguments

- Scripts can accept multiple arguments and Perl automatically places the arguments into `@ARGV`:

```
~$> perl read3names.pl "Josephine" "Mary" "Wolfe"
```




```
my ($first, $middle, $last);

$first = $ARGV[0];
$middle = $ARGV[1];
$last = $ARGV[2];
print  "$first $middle $last you are in trouble!!\n";
```

Testing for Command line arguments

- When your script requires command line arguments, you need to test to make sure the user provided them. The defined function can be used to do this test. There's also a shortcut for assigning all argument variables at once:

```
~$> perl read3names.pl "Josephine" "Mary" "Wolfe"

   my ($first, $middle, $last) = @ARGV;

   if (defined $first  &&  defined $middle  &&  defined $last) {
      print  "$first $middle $last you are in trouble!!\n";
   } else {
      print "You did not provide first, middle, and last name\n";
      print "Now you are REALLY in trouble!!!!\n";
   }
```

Perl documentation

- You can find Perl documentation online, and when Perl is installed, you should also be able to run the **`perldoc`** program. Try these:
  - `perldoc perlintro`
  - `perldoc perldoc`
- For functions, use **`perldoc -f functionname`**
  - `perldoc -f push`
  - `perldoc -f pop`
  - `perldoc -f chomp`
  - `perldoc -f glob`
  - (Sometimes Google search results are easier to understand than perldoc is!)

Last thing about command line args

- What if we wanted to give the program a list of files to do something with?
  - think like

```
mv file1.pl file2.pl file3.pl newFolder/
```

- We can pass it a list of undetermined length
  - `./test.pl Dan Sergei Mike Noah Susan Nirav ...`
- We can use the functions we already know (`sort`, `scalar`) on a copy of this list

```
my @names = @ARGV;
print "you gave me ".sca
print "In sorted order they are:\n";
foreach my $name ( sort(@names) ){
    print "\t$name\n";
}
```

This is not NECESSARY, but its a good idea.

More Perl functions

- Perl has many useful functions. We've seen some already: `chomp`, `push`, `pop`, `shift`, `unshift`. Can you think of any others?
- Functions can take arguments or parameters, perform some operations on them, and return results.
- Examples:
  - `$last_item = pop( @arr );`
  - `@sorted = sort( @arr );`
  - `if ( defined( $var ) ) { … }`
  - `chomp( $line );`
  - `@keys = keys( %id_hash );`

Function arguments and results

- When working with functions, it is important to know what arguments or parameters they expect as inputs, and what type of result(s) they return
- You can find this out with:

```
perldoc -f functionname
```

- The length function is easy: it takes a string argument and returns an integer
  - Example:
    - `$dna = "ATGCGTCAGTCGTAGTCA";`
    - `$dna_len = length($dna);`

File Input and File Handles

- We've used <STDIN> to read keyboard input. To read from a file in Perl, we need to set up a file handle with the open function. Standard practice is to make file handle names all upper case:

```
my $blastfile = "ovine_blast.bln_m9";

 open (BFILE, $blastfile);
 my $nlines = 0;
  while (my $line = <BFILE>) {
      $nlines++;
 }
 print "File $blastfile contains $nmatch matches\n";
 close BFILE;
```

- When you open a file, make sure you call the close function when finished with it.

The die function

- The die function prints a message to the screen and exits the script:

```
$min_hits = 2;
@files = ("ovine_blast.bln_m9", "bovine_blast.bln_m9");
foreach $file  (@files) {
   open (BFIL, $file) or die "Cannot open $file \n";
   print "Top $min_hits Hits from $file:\n";
   $hits_output = 0;
   while ($hits_output < $min_hits) {
      $lin = <BFIL>;
      print  "\t $lin";
      $hits_output++;
   }
   next;
}
```

File Output

- Writing output to a file from Perl also requires setting up a file handle with the open function and in case of failure to open, calling the die function. Notice the ">" prefix to the filename. With output it is important to remember to close the file or you may lose some data!

```
my $blastfile = "ovine_blast.bln_m9";

open (BFILE, $blastfile)  or  die  "Cannot open $blastfile \n";
open (RFILE, ">$blastfile.summary") or  die
                                "Cannot write $blastfile.summary \n";
$nmatch = 0;
while ($line = <BFILE>) {
   $nmatch++;
}
print  RFILE "File $blastfile contains $nmatch matches\n";
close  RFILE;   # Make sure output gets flushed to disk!
```

The Filename Matching function glob

- The `glob` function matches a collection of files based on a filename pattern (like * wildcards in Unix). Notice that `glob` returns an array result:

```
$max_hits = 2;
@files = glob("*.bln_m9");
if (scalar(@files) == 0) {
   die "No files match *.bln_m9 \n";
}

foreach $fil  (@files) {
   open (BFIL, $fil) or die "Cannot open $fil\n";
   print "Top $max_hits Hits from $fil:\n";
   $hits_seen = 0;
   while (++$hits_seen < $max_hits) {
      $lin = <BFIL>;
      print  "\t $lin";
   }
   next;
}
```

Better scripting practices…

- Using fixed strings for filenames, numbers, or patterns inside a script is known as hard-coding, and it is not very flexible! In the code below if we want to change `nhits`, we need to edit the script. To work on all files having names ending in `".bln"`, we need to change the script in two places:

```
$nhits = 2;
@files = glob("*.bln_m9 ");
if (scalar(@files) == 0) {
  die "No files match *.bln_m9 \n";
}
foreach $fil  (@files) {
  …
}
```

- We could use a variable to keep the file extension change to one line, but we still have to edit the script each time we want to use a different pattern.

Perl File Tests

- Perl can test file attributes: Does a file exist? Is it a directory? Is it readable? Writable? Empty?

```
if (-e $filename) {
   print  "$filename exists \n";
}
if (-d $filename) {
   print  "$filename is a directory \n";
}
if (-w $filename) {
   print  "$filename is writable \n";
}
if (-z $filename) {   # Zero size
   print  "$filename is empty \n";
}
```

Tips for Testing/Debugging Perl Scripts

- If your script does not work as expected:
  - Be sure that you are running the script that you have saved from the editor, and not another version of it.
  - Resolve all errors reported by the Perl interpreter

```
use warnings; use strict;
```

  - Write and test small segments of code at a time
  - Print the contents of variables, and check for non-printable characters within strings, e.g.:

```
print "str is !$str!\n";
```

  - Check for the existence of array or hash elements, e.g.:

```
if (exists $arr[$i] && exists $hash{'anopheles'}) {
  print "population size: $arr[$i]";
  print "hazard: $hash{'anopheles'}";
}
```

Homework

- Get started on the assignment handed out in class! It is more challenging than earlier assignments have been and you should not wait until the last minute to work on it. After today's class you should be able to complete Question 1, part a.

- Optional: Read or Skim "Beginning Perl", chapter 9, Running and Debugging Perl

- Due to the midterm, there will be **No Quiz next week!!** (*yay*)