

Introduction to Perl File Handles

Contributed by Mr-Oss
Saturday, 05 December 2009
Last Updated Saturday, 05 December 2009

Introduction to Perl File Handles

Perl is an extremely powerful and handy tool for any system administrator. Here we will look at different ways that perl can interact with files on your system.

Perl is a language that is very useful for it's reporting features among other things. Reading files, writing new files, and appending to existing files are quite easy tasks to accomplish. In the following example we will open two different files, combine them into one large file, and then write that file out to a new file.

```
#!/usr/bin/perl
```

^^ every perl script should begin with the sha-bang which points to your local perl binary

```
# OPEN A FILEHANDLE NAMED IN FOR READING  
open(IN,"<","/etc/passwd") or die "UNABLE TO OPEN FILE FOR READING\n";
```

^^ here we are creating a filehandle named IN. The "<" represents a read in method. the third option is the file we are going to be reading into our program and assigning to a file handle named IN. open(FILEHANDLENAME,"mode","path to file") . The statement or die "UNABLE TO OPEN ..." basically allows us to print out the error of UNABLE TO OPEN... if for some reason we cannot access the file defined in the filehandle

```
# CREATE AN ARRAY OUT OF THE FILE CONTENTS  
@passwd=<IN>;
```

^^ Here we are populating an array named password with the conents of the filehandle IN. When referencing a filehandle you need to use the name of the file handle inside of < >. so here we are referring to our previously created filehandle named IN and assigning it to an array named password. The @ sign is the variable prefix for an array so @password is now an array containing everything inside of /etc/passwd

```
# CLOSE THE FILE HANDLE BECAUSE WE ARE DONE WITH IT  
close(IN);
```

^^ Here we are closing the filehandle IN. we call the close function and inside the ()'s we put our filehandle name. When closing the filehandle you do not need to use the < > tags. So far we have opened /etc/passwd for reading and created an array named @password which contains the information inside of /etc/passwd. Now that we have populated our array we can safely close the filehandle completely.

```
#####  
# OPEN ANOTHER FILE HANDLE USING A VARIABLE FOR FILE PATH  
#####  
# SETUP THE VARIABLE  
$groupfile='/etc/group';
```

^^ here we are setting up a variable named \$groupfile which contains the full path to the /etc/group file. The variable is defined inside of single tics which will not get special treatment by perl unlike the double tics ". when using single tics what you see is what you get. if you use quotes instead " " that will most likely work but be careful when using double tics because you have to create escape sequences for special characters at times.

```
# OPEN ANOTHER FILE HANDLE NAMED IN2 FOR READING  
open(IN2,"<","$groupfile") or die "UNABLE TO OPEN $groupfile FOR READING\n";
```

^^ here we are defining a new filehandle named IN2, with a read in mode <. The difference in this example is that we are now using the variable name of \$groupfile which points to /etc/group. Notice that we are using quotes around the variable name. If you try to use single tics ' ' what you see is what you get. The script wil attempt to open a file named \$groupfile

and will not treat \$groupfile as a variable which points to /etc/group. We want to use the value of the variable here so we need to use quotes " "

```
# CREATE AN ARRAY OUT OF THE FILE CONTENTS
@group=<IN2>;
```

^^ Once again we are reading the contents of the filehandle IN2 and assigning it to an array named @group. This is the same thing that we did earlier when we created the @password array. This time our array will contain the contents of /etc/group

```
# CLOSE THE IN2 FILE HANDLE NOW WE ARE DONE WITH IT
close(IN2);
```

^^ now we once again are closing the open filehandle. This time we pass the file handle name of IN2 to the close function. We no longer need the filehandle because we have the file contents inside of @group

```
#####
# OPEN A FILE HANDLE TO WRITE SOME OUTPUT TO
#####
open(OUT,">","/tmp/test.txt") or die "UNABLE TO OPEN OUTPUT FILE\n";
```

^^ This command is different than the previous 2 examples. This time we are opening a filehandle named OUT but our mode has now changed from < to > . The > mode will write out to a file this time. Be aware that > is a destructive output. The > mode will open the output file /tmp/test.txt and overwrite any contents that may reside within the file. If the system does not have a file named /tmp/test.txt then it will be created. If the system does have a file named /tmp/test.txt it will be overwritten. Be aware of the possibility of clobbering data when using the > file handle mode. The or die "unable..." is the same as the previous examples. it generates an error if for some reason it cannot provision this file for writing.

```
# LISTS COLLAPSE TOGETHER WHEN STRUNG TOGETHER SO WE CREATE
# ONE LARGE NEW ARRAY OUT OF OUR 2 EXISTING ARRAYS
```

```
@fullpull=(@passwd,@group);
```

^ here we are creating a new array named fullpull which contains first the information in the password array followed by the information in the group array. Arrays are simply lists, so by defining a new array named @fullpull and assigning it to @password,@group the two arrays will collapse into one long list. The two arrays both still exist but they have been used to create our new array @fullpull

```
# NOW LETS PRINT THE CONTENTS OF BOTH FILES TO OUR OUT FILEHANDLE
# ALL WE HAVE TO DO IS PUT THE FILE HANDLE NAME AFTER PRINT TO
# WRITE INTO THE OUTPUT FILE
```

```
foreach(@fullpull) {
print OUT $_;
}
```

^^ The foreach loop will step through the entire array @fullpull. Foreach will perform commands located within the curly braces {} on each line inside of our array named @fullpull. So in this example we are saying for every line found in the @fullpull array perform the command print OUT \$_. The print command inside of the curly braces is followed by OUT. OUT is our file handle name for our output file. So that command will print the information contained in the current variable \$_ into the output file we had defined earlier. All that this foreach loop will do is step through the entire array fullpull and print it out line by line to our filehandle OUT which was defined to be /tmp/test.txt.

```
# CREATE SOME OUTPUT BREAKS FOR READABILITY
print OUT "\n";
print OUT '#x50;
print OUT "\n','# END OF WRITE AND BEGIN APPEND','\n';
print OUT '#x50;
print OUT "\n";
```

^^ This block of code is simply for readability inside of our output filehandle named OUT. Here we see the print command followed by our output filehandle name and various information. the \n will put a newline in the file. '#x50 will print out 50 # signs. then we have another newline entry followed by some text and another newline. Then we see another 50 # signs and a newline character. All of this information has been printed to our output filehandle named OUT. If you do not specify the filehandle name following the print command, you will print to STDOUT which will be your screen. When printing to an output filehandle you will not see anything appear on your screen. A common mistake I find myself falling into has to do with not including the filehandle name after the print command. Also be aware that the filehandle name is not contained inside of any type of brackets such as <> or ().

```
# CLOSE THE OUTPUT FILE HANDLE NOW THAT WE ARE FINISHED
close(OUT);
```

^^ now that we have finished writing our file out to our filehandle named OUT which is the /tmp/test.txt file, we will close the file. Closing the output files once you are finished is always a good idea. I typically try to close filehandles as soon as I am finished with them. Here we have closed /tmp/test.txt and can no longer write to the filehandle OUT.

```
#####
# OPEN A FILE HANDLE AND APPEND SOME INFORMATION TO IT
#####
$outfile='/tmp/test.txt';
open(OUT2,">>",$outfile) or die "UNABLE TO OPEN OUTPUT FILE $outfile\n";
```

^^ This time we are setting up a variable named \$outfile which contains the full path to /tmp/test.txt. Next, we call the open command again and define a filehandle named OUT2. The mode on this filehandle is different than the previous example. This time our mode is set to >>. This mode will open the file /tmp/test.txt for writing but instead of overwriting, we will be appending to the end of this file. The >> mode represents a non destructive append command. This will also create a file named /tmp/test.txt if it does not already exist. Finally we pass the variable \$outfile as the filepath and wrap it inside of quotes " " so that the variable is interpolated. Again the or die "error" message has been used to stop program execution and tell us if we cannot open the file for writing.

```
# NOW LETS PRINT THE CONTENTS OF BOTH FILES TO OUR OUT FILEHANDLE
foreach(@fullpull) {
print OUT2 $_;
}
```

^^ here we see our foreach loop once again. This time it will loop through the @fullpull array line by line and print the contents out to the OUT2 filehandle. This time we will be appending to the /tmp/test.txt file line by line.

```
# PUT SOMETHING IN THE END OF THE FILE FOR NO REASON OTHER THAN WE CAN
print OUT2 "\n\nTHAT IS PRETTY PIMP DONT YOU THINK\n";
```

^^ Here we are once again putting in some information for our own reference. Printing to OUT2 a few new line feeds and a string of some information followed by another new line character. Please note that the \n which is being printed will always be found inside of quotes " " and not inside of single ticks ' '. If we were to put the \n inside of single ticks we would end up printing out \n instead of a new line return. Remember ' ' = what you see is what you get.

```
# CLOSE OUT2 OUTPUT FILE HANDLE
close(OUT2);
```

^^ now we close our open filehandle named OUT2 because we are now finished with it.

This article covered 3 different file handle modes for opening files. We read a file in using < . We wrote a file out using > destructively overwriting anything that was previously inside of the file. Finally we opened another file handle for output using the >> operator mode which allows us to append information without losing whats already found within the file.

```
#####
# FILE HANDLE OPEN MODES
# < = read in
# > = write out
# >> = append out
#####
```

^^ The file handle mode operators above will be the most commonly used. Remember the syntax for the open command. open(NAME,"mode","file"). Using these 3 file handle opening modes you should be well on your way to taking advantage of perl's file handle capabilities.

Here is the example script in its entirety without all my long winded comments.

```
#!/usr/bin/perl
#####
# Perl file handle tutorial

# OPEN A FILEHANDLE NAMED IN FOR READING
open(IN,"<","/etc/passwd") or die "UNABLE TO OPEN FILE FOR READING\n";

# CREATE AN ARRAY OUT OF THE FILE CONTENTS
@passwd=<IN>;

# CLOSE THE FILE HANDLE BECAUSE WE ARE DONE WITH IT
close(IN);

#####
# OPEN ANOTHER FILE HANDLE USING A VARIABLE FOR FILE PATH
#####
# SETUP THE VARIABLE
$groupfile='/etc/group';

# OPEN ANOTHER FILE HANDLE NAMED IN2 FOR READING
open(IN2,"<",$groupfile) or die "UNABLE TO OPEN $groupfile FOR READING\n";

# CREATE AN ARRAY OUT OF THE FILE CONTENTS
@group=<IN2>;

# CLOSE THE IN2 FILE HANDLE NOW WE ARE DONE WITH IT
close(IN2);

#####
# OPEN A FILE HANDLE TO WRITE SOME OUTPUT TO
#####
open(OUT,">","/tmp/test.txt") or die "UNABLE TO OPEN OUTPUT FILE\n";

# LISTS COLLAPSE TOGETHER WHEN STRUNG TOGETHER SO WE CREATE
# ONE LARGE NEW ARRAY OUT OF OUR 2 EXISTING ARRAYS

@fullpull=(@passwd,@group);

# NOW LETS PRINT THE CONTENTS OF BOTH FILES TO OUR OUT FILEHANDLE
# ALL WE HAVE TO DO IS PUT THE FILE HANDLE NAME AFTER PRINT TO
# WRITE INTO THE OUTPUT FILE

foreach(@fullpull) {
print OUT $_;
}

# CREATE SOME OUTPUT BREAKS FOR READABILITY
```

```
print OUT "\n";
print OUT '#x50;
print OUT "\n", '# END OF WRITE AND BEGIN APPEND', "\n";
print OUT '#x50;
print OUT "\n";

# CLOSE THE OUTPUT FILE HANDLE NOW THAT WE ARE FINISHED
close(OUT);

#####
# OPEN A FILE HANDLE AND APPEND SOME INFORMATION TO IT
#####
$outfile='/tmp/test.txt';
open(OUT2,">>",$outfile) or die "UNABLE TO OPEN OUTPUT FILE $outfile\n";

# NOW LETS PRINT THE CONTENTS OF BOTH FILES TO OUR OUT FILEHANDLE
foreach(@fullpull) {
print OUT2 $_;
}

# PUT SOMETHING IN THE END OF THE FILE FOR NO REASON OTHER THAN WE CAN
print OUT2 "\n\nTHAT IS PRETTY PIMP DONT YOU THINK\n";

# CLOSE OUT2 OUTPUT FILE HANDLE
close(OUT2);

#####
# FILE HANDLE OPEN MODES
# < = read in
# > = write out
# >> = append out
#####
```

I hope you enjoyed this introduction to perl filehandles. Check back for future articles on perl.

Thanks for visiting,

Mr-Oss