Perl Debugging

Professor Don Colton

Brigham Young University Hawaii

1 Introduction

Your Perl program looks right to you, but it does not work. What you (are supposed to) do next is called "debugging." Where do you start in finding the problem and correcting it?

For our examples, we will assume you have written a program called `myprog`.

2 Two Kinds of Errors

There are two major kinds of errors: syntax errors and run-time errors.

2.1 Syntax Errors

Syntax errors are flaws in the wording of your program. They include such things as forgetting to end a statement with a semi-colon, or spelling "print" with a capital P as in "Print". Generally computers are very stupid about figuring out what you mean. They are very good at doing what you say if it is within their power, but you must say it exactly the right way. If you say it wrong, the program will not be able to run.

Syntax errors are caught by the compiler or interpreter, usually before the program even starts to run. They are reported by identifying the line at which the compiler decided that it was hopelessly confused. The actual error is usually just before that point, but it could be many lines earlier if the mistake is a missing quote mark, for instance.

Many compilers will try to report as many syntax errors as they can find. This is a throw-back behavior more suited to an earlier time in the history of computing, when programs were submitted through a real window on a deck of punched cards and outputs were retrieved four hours later. It was desirable to see as many errors as possible all at once. But those days are gone for most of us.

Focus your attention on only the first error. Later errors could well be part of a domino effect much like taking the wrong turn with driving directions. Once you are off course, the other errors may simply reflect the fact that you are already off course, and may not be actual errors themselves. Focus your attention on only the first error (or other things you know are errors). Fix it and then try running your program again.

2.2 Run-Time Errors

Run-time errors, sometimes called semantic errors, are flaws in the logic of your program. They include things like forgetting to initialize a variable before adding to it, or doing things in the wrong order. The resulting program will still run, but will do the wrong thing.

Run-time errors cannot be reliably caught by the compiler. Probably the best and easiest way of figuring them out is by lacing your program with print statements to dump the values of various variables to show you how the program is acting. This effect can also be achieved by using a good debugger using features like single-step and breakpoint. Most languages have debuggers available. Learning to use a debugger can take time and effort. It is usually easier to rely on print statements and to graduate to debuggers after you have developed some skill.

Sometimes it is good to put `assert` statements into your program. In Perl, these are `if` statements that check for things you are sure must be true, and if they are not true, you want your program to stop so you can fix the error. For example:

```
if ( $a > 1 ) { print "problem with $a"; die }
```

3 Run Your Program

The first step in debugging is to try running your program. In fact, before you try running your program you may not know whether there is a bug or not. It is usually safe to assume there is at least one bug.

3.1 Microsoft Windows

Normally with Microsoft Windows, Perl programs are identified by having a filename extension of `.pl` (or sometimes `.perl`). Make sure your program ends with `.pl`. In our case, your program name should be `myprog.pl`.

When your program name ends with `.pl`, does the icon turn into a cute little gecko? If so, you probably have ActiveState Perl installed and you should be able to run your program. If not, you may need to install Perl before you can run any Perl programs.

To run your program in the Graphical User Interface (GUI) environment, double-click on your programs desktop icon and it should run. A text window should open up, and the inputs and outputs of your program should appear there.

If the text window opens up and then closes immediately, it means one of two things. Maybe your program failed to start due to a syntax error. Or maybe your program started and ran to completion, and then the window closed.

To keep the window open, add an input line to your program. Something like this may do the trick:

```
$wait = <STDIN>;
```

This will cause your program to wait for input. The variable name `$wait` is not special. That name was chosen to reflect its intended use. You could just as well use `$x`.

After adding a wait line, run your program again. If the window still opens and abruptly closes, you probably have a syntax error. You will have to run your program from the command line to see the error messages.

Windows CMD

To run from the Windows command line, do something like start / run / cmd. It should open up a command-line window.

In the command-line window, type `cd` followed by the directory name of the folder where your program is located. The `cd` by itself will ask Windows to tell you in which directory you are currently located. You may be able to get to your desktop by typing

```
cd Desktop
```

although this may be different on your system. Try it. If it fails, you can also use the

```
dir
```

command to look for likely alternatives.

Once you find your way to the directory where your program is stored you can run it by typing its name. At this point it should attempt to run and should report to you all the syntax error messages that are preventing it from actually starting.

3.2 Linux

First make sure you have Perl installed. It is almost 100% guaranteed to be there, but just to be sure you can type the `which` command:

```
which perl
```

This should report back to you something like `/usr/bin/perl` or `/usr/local/bin/perl`. Whatever it reports should be typed into the shebang line of your script. If nothing is reported, Linux cannot find Perl on your machine and you need to install it or get help.

Second, run your program explicitly using Perl.

```
perl myprog
```

Your program should run, or you should see some syntax errors. Fix them.

Shebang

The next step is to make your program run Perl implicitly.

With Linux, scripts including Perl programs are identified by having a shebang line as the first line of the script. For Perl it should look like one of these (matching whatever the `which` command returned):

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

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

Make sure this is the first line of your program. Blank lines or blank spaces are not allowed before the `#!` which must be the absolute first thing in your program file.

Set the permissions on your program to make it executible instead of merely readable and writable. You can do that by typing the following command or something similar:

```
chmod 700 myprog
```

Then you can run your program by typing this:

```
./myprog
```

If you set your `path` to include the current directory, you can run your program by just typing this:

```
myprog
```

Several things can go wrong in this process. Here are some common error messages and what you can do to fix them.

```
./myprog: permission denied
```

This means that your permissions are not set properly. Run the `chmod` command to fix the permissions.

```
./myprog: line 4: whatever: command not found
```

This can mean that your program is not being understood as a Perl program, but instead is being run as a normal `/bin/sh` shell program. Check your file to make sure the `#!` line is absolutely first.

```
./myprog: command not found
```

This can mean several things. Maybe your program is not present in the directory where you are running. Type this command to make sure:

```
ls -l myprog
```

The `ls` command lists the directory contents. You should see `myprog` listed. If not, you are in the wrong directory or your program is in the wrong directory. Fix that and try again. If you do see your program then it may mean the Perl interpreter was the command that was not found. Carefully check the spelling on your `#!` line. Make sure you put `/usr/` instead of `/user/` and `perl` instead of `pearl`. (Yes, these are common mistakes.)

If the program you are running was originally written on an MS Windows system, perhaps using Notepad, it may have transferred across with carriage returns between the lines. Under Linux and Unix, lines are normally separated by `\n` whereas with MS Windows, MS-DOS, and CP/M, lines are normally separated by `\r\n`. The `\r` represents a carriage return (CR). The `\n` represents a line feed (LF). Sometimes this is called the CRLF problem.

You can fix the CRLF problem in at least two ways. One way is to remove the CRs from your program. This is great if you know how to do it, but it is difficult to explain clearly The other way is to fix your `#!` line so the `\r` is not seen as part of the Perl command name. Add `--` to make your line look like this:

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