Byung Hun Lee

CS 260

Raheja

Assignment 6 Write Up

Compared to the previous assignment which required the same program to be written in different types of shell programs four times, this assignment which required the same program to be written in only Bash and Perl was much harder in my opinion. There were similarities between the two programming languages just like how Bash was like Korn shell and Z-shell; however, the differences between Perl and Bash was much more apparent than the differences between C-Shell and Bash in my opinion. Each language had its advantages and disadvantages but half way through the assignment I already had my preferred language.

To begin with let us look at the similarities between both languages. Both languages support control statements such as if/else, for loops, and while loops. Both languages have support for functions, or something similar as Perl calls it a *subprogram*. Both programs have basic commands you would expect to see in a scripting/programming language such as variables, print functions, etc. In short Perl and Bash could each do about 95% of what the other could do. The other 5% percent however, was either missing or much harder to implement compared to other.

With the similarities came similar difficulties. Writing regexes proved difficult in both scripts. Certain regexes had to be written in specific format to check to see if the input was formatted correctly. For example, when I needed the input to contain a certain number of letters or digits I needed to enter [[:digit:]] or [[:alpha:]], and certain characters needed to be escaped using the back slash (\) before I could use it to compare inputs. The number of brackets, curly braces, parentheses, and backslash required in some regex made it hard to check if the regex was formatted correctly. Or if an error popped up while compiling about mismatching number of brackets, it was hard to debug the code and place the closing bracket in the correct location at times. Other problems and difficulties arose depending on which script I was writing.

Now that the similarities have been laid out, it is time to look at the differences between Bash and Perl. To begin with, Bash did not need any specific keywords to be placed at the top like *use warnings* unlike Perl. Furthermore, Bash automatically adds new line character at the end of the print method and the input does not need to be *chomped* afterwards. Another plus for Bash was the additional control statement: *case* which Perl did not have. The *case* method allowed me to write simpler code to control where the program would be headed in the menu after getting the user input. The most important ability that Bash had over Perl in my opinion, was that Bash was easier to execute UNIX system commands such as *awk* over Perl. UNIX system commands could be activated without any extra steps to activate whereas in Perl, extra System commands had to be executed to use Unix commands such as *awk*. Unfortunately, despite the many advantages Bash had over Perl there were several syntaxes I had to follow in Bash that added extra difficulties. One of said difficulties was that certain control statements needed to be written using brackets while parentheses could be used but had to be used twice. For example to use if statements, the code had to be written like: if (( )) or if [ ]. Another problem was that Bash had strict rules on spacing. A missing space when declaring a variable or between control statements would cause an error. This forced me to pay careful attention to where spaces had to be placed. When declaring variables, I would often go back to remove the space to the left and right of the equal sign. Or when I was writing regex I would sometimes have to go back to the beginning of the regex to place a space lest the system throw an error during compilation. The third annoyance with Bash was that some control statements required extra keywords to start and end. If/else statements had to be ended using the *fi* keyword; while loops had to start with *do* and end with *done*; case statements required ;; to signal an end to each case and the entire case block had to end with *esac*. These extra keywords while easy to fix, delayed the finish as I constantly forgot to add the keywords and cause the compiler to throw an error.

I found Perl to be more difficult than Bash despite being closer to a programming language than a scripting language. First the pros of Perl. Perl used curly braces to signal a block of code instead of using keywords. If/else statements and while loops did not require extra keywords to signal the start and end, just braces. Since I was already used to using braces to signal a block of code from other programming languages, the braces were comfortable to use. Second pro was that while Bash had very strict rules on spacing, Perl was much more lenient with spacing. A missing space would not cause the script to crash. Now for the difficulties. One problem I kept running into while coding was having to use the *my $variableName = “”;* to declare variables. In bash *my* keyword was not needed when declaring variables and the switch was hard to settle into. Second annoyance was, ironically, having to end each statement with a semicolon (;). While the use of a semicolon to end a statement was commonplace in programming languages, I simply had a hard time remembering to end each statement with a semicolon in Perl as no other scripting languages used semicolons to end statements. Furthermore, there were two nuances that provided very minor inconveniences. One was that regexes had to begin and end with backslash when checking to see if inputs were in certain formats, the other was that Perl did not support case/switch statements unlike ever other scripting languages I learned this quarter. The lack of case/switch statements could be easily solved using lots of *if* and *elsif* statements, but this made the code longer to finish in my opinion. The hardest difficulty I had with Perl was that system commands such as *awk* had to be executed by storing the command in a *@variable* and executing said variable using *System(@variable)*. To begin with, storing the necessary commands as a variable took several tries to perform correctly and it was only near the end of the assignment that I found a way to store several commands in a single variable and execute it once.

In conclusion, I found both scripts to have its pros and cons. While UNIX commands were, in my opinion, much easier in Bash, the strict rules on spacing added some stress and difficulties writing Bash. Perl on the other hand, was the reverse; Perl did not care about spacing but executing system commands such as *awk* was several times more difficult, or at least that’s what it felt like. Syntax wise Bash needed extra keywords to execute control statements, but Perl had to start the script with *use* keyword and *my* keyword for variables. Thankfully however, despite the differences, the scripts were similar enough that one could be used as a basis for the other. Personally, I preferred Bash over Perl. For one, I had more experience with Bash and its counterparts; and two, UNIX system commands were much easier to execute because Bash was built to execute shell commands. Considering many of the commands the script had to execute needed system commands to perform, the ease of Bash made it my preferred language long before I was finished with Perl.