Joshua Camacho

Professor Raheja

CS 260

7/19/2017

Bourne Again Shell vs C shell vs Korn shell

For this writeup I will be comparing and contrasting three different Unix shell environments and the ease of use, features, and annoyances. For each shell I wrote an ATM-like script which allows the user to manage a banking account with features like checking balance, withdrawals, and transfers. Basic control structures like switch, while, and if statements were used, as well as arithmetic operations and comparison operations.

I'll start with a simple metric, line count. Bash and Korn shell rank in at 144 lines respectively and C shell comes in at 121 lines. Bash and Korn allow reusable bits of script called functions. Normally functions greatly reduce the amount of redundancy in your code, which works to reduce the amount of code lines in your script. The ATM script, however, was so simple and non-redundant already, and the functions ended up adding more lines of code because of the overhead of naming them, calling them, and the curly braces which define their scope. Looking at the line count metric, you would think that C shell would be my preferred shell, or at least it seems superior than Bash and Korn, but that is not the case. While Bash and Korn ended up with more lines for this script, I could have brought that line number down by not utilizing functions for this script. That aside, however, If I was going to choose a shell to write a large script in I would definitely choose Bash or Korn for them having the option to use functions and would make good use of functions.

Next I'll discuss syntax between the three shells. In many ways the shells have similar syntax. Bash and Korn are almost identical in this regard, so I'll lump their style together. C shell has nicer conditional syntax, only requiring single parenthesis () and having a switch statement with similar syntax to C. Switch statements in Bash/Korn shell were harder to remember with its beginning, "case $myvarible in", lacking the word "switch" and seemingly arbitrary break statements ";;" and default case "*)". C shell was much more intuitive with its break statements "breaksw" and the default case simply being "default". Bash/Korn also had some interesting ways to end control structures, with "if" ending as "fi" and "case" ending as "esac". I feel adding these backwards words decreases the readability of the scripting language, and takes away the discoverability of learning the language.

Arithmetic operations required special syntax in all the shells, which is something not found in traditional programming languages. The Bash/Korn way of handling arithmetic operations, double parenthesis(), was easier to remember than the C shell's @. The C shell does not support "+=", "-=", or "++", "--". This was a deal breaker for me with the C shell. Those operators are my favorite shortcuts in programming. Bash/Korn does support these, which makes the case even stronger for my continued use of these shells.

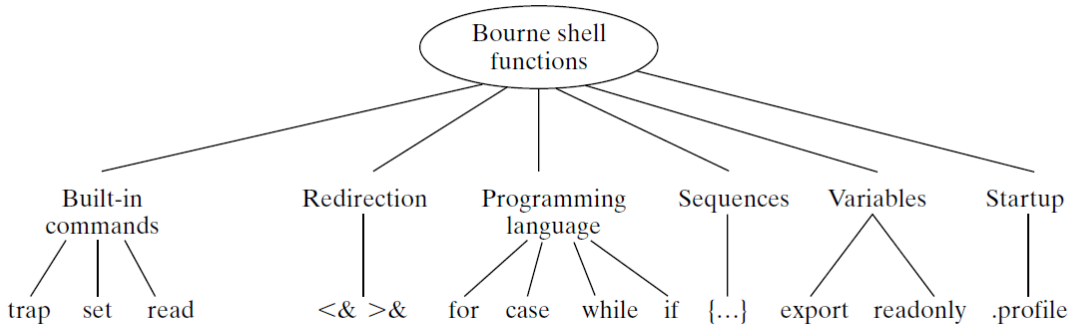
It's important to note that my script in the Bash shell was completely compatible with the Korn shell, no syntax change required. This makes a nice case for the use of Korn shell, especially since the Korn shell does add some nice extra features. Although I didn't utilize it, Korn comes with a nice feature for making quick user menus. The "select" command allows you to quickly and easily create a menu with the necessary switch and while loop statements already built into the control structure. This feature, combined with backwards compatibility makes Korn a strong contender for me.

Bourne Again Shell added some nice features over its predecessor, the Bourne Shell. Instead of having to use sub-command notations to do arithmetic operations, "myvar=`expr $1 + $2`", you could simply encase the operation in double parenthesis ((myvar = $1 + $2)). This syntax was much nicer on the eyes, especially when working with single or double quotes near the arithmetic operation, as it can be easy to confuse the tiny inverted quotes, "`". Bourne Again Shell also allows you to force your numerical variables to be integers, with the "declare" keyword. This allows you to take advantage of faster operations on your variables if you know they are going to be all integer values. Bourne again shell also supports file conditionals, allowing for you to check if a file exists, if it is readable, writable, and other file related conditions.

One annoyance that was true mostly for the Bourne Again Shell and the Korn shell was the necessity of spaces in arithmetic and conditional logic. "((myvar=3*5))" does not work in these shells, it requires you to do "((myvar = 3 * 5))". Having to account for tiny required amounts of white space is headache inducing, especially when the scripting languages lack any real form of debugging. Coming from programming languages which ignore whitespace, this gave me the most amount of issues, but I did get the hang of it after a while. Another huge annoyance related to spacing was declaring variables. "myvar = 0" does not work where "myvar=0" does work. This is backwards from the other issue, with the declaration requiring no whitespace or it breaks. This is pretty unintuitive but I suppose just another small quirk that requires some adjusting to get use to.

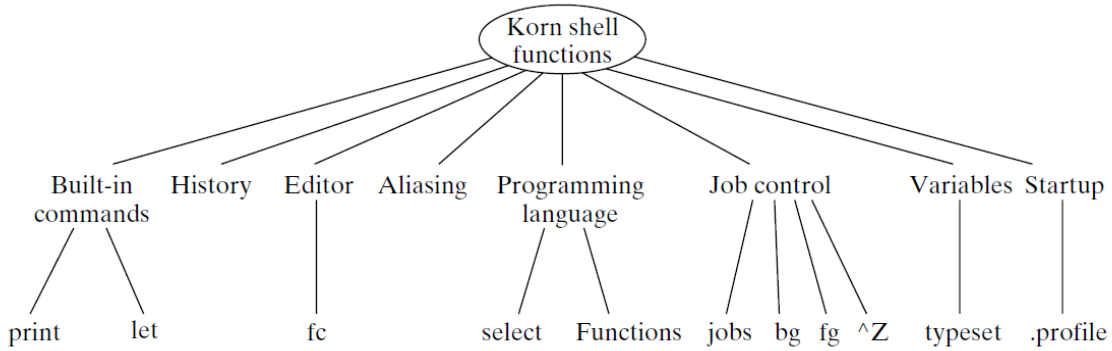
From my impression reading around online, Bourne Again Shell is hugely popular in the industry. That further supports its case for me adopting it. However, the Korn shell lets me run Bash scripts from the get go and do even more, but my scripts wouldn't always be compatible if somebody else was going to try to run it on Bash. Korn makes user menus super easy, so if I was going to write a user menu I would choose the Korn Shell. C shell might have easier syntax, but because it does not have any functions, surprising for a shell with the C language in mind, I would not use the C shell. In C shell dollar signs "$" cannot be escaped in double quotes. For example "set foo = "this is a \$money" would give you the message "money: Undefined variable". Little annoyances like that are probably the reason why "trusted" C shell was written. But because of my impression of C shell, and the lack of adoption in the industry, I would not even choose trusted C shell.

BASH



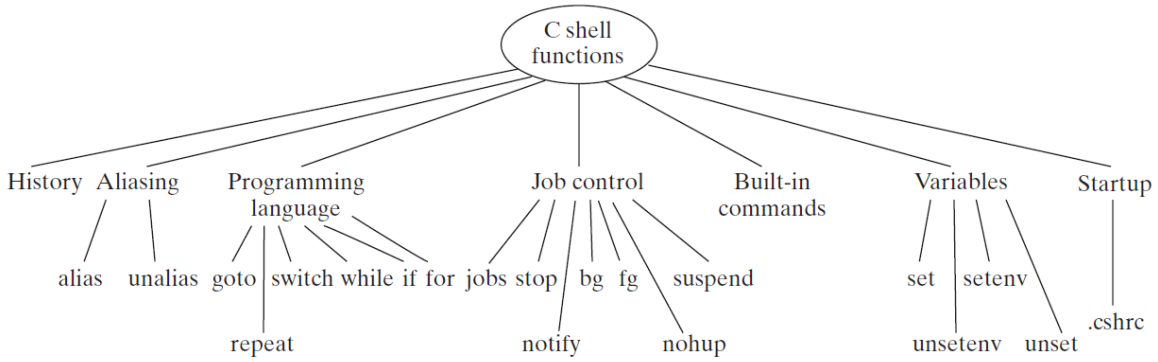
Arithmetic Operations
((var = $1 + $1)) or var=`expr $1 + $1`
Supported operations
The usual +, -, % plus ++, --, +=, -=, **
Command editing set -o vi
Mandatory integer variables with **declare** for faster operations
Switch
Case $var in
1)
;;
*)
;;
Esac
If then elif then else fi
File conditional statements ((-f filename))
While
While ((case))
do
//stuff
Done
Supports until
Functions
Read input

KORN



Supports all of Bash plus
Select for easy menus
Return values for functions
the ability to strip leading tabs off of "here" documents

C SHELL



No functions
Easier conditional syntax () single parenthesis
If endif
Switch ($myvar)
Case "1":
Breaksw
Default:
Endsw

@operator for arithmetic
Set input = $<
Comparison with wildcards =~