**Lesson 11 – Bash Shell Programming**

Bash script – a file containing a series of commands and possibly flow-of-control (selection, interation, and functions).

– Note: if you have **single**, often used command, create an alias in your .bashrc file:

e.g., alias my='ls –la –color=auto'

– Google's Bash style guild: <https://google.github.io/styleguide/shell.xml>

Bash script file structure: cmds are separated by new-lines or semi-colons (;).

#!/path/to/bash

cmd(s)

File redirection:

* 0 is the file descriptor for stdin
* 1 is the file descriptor for stdout
* 2 is the file descriptor for stderr

|  |  |
| --- | --- |
| > filename | Send stdout to the specified file. (If the file already exists, the file is overridden!) |
| >> filename | Append stdout to the specified file. The file's original contents are retained. |
| 1> filename | Send stdout to the specified file. |
| 2> filename | Send stderr to the specified file. |
| 1> f1 2> f2 | Send stdout to file f1; send stderr to file f2. |
| 1> f1 2>&1 | Send stdout to file f1; send stderr to the same place as stdout. |

Variables:

* Names start with letter, contains letters a-z, digits, and underscore. Use ALL\_CAPS for global env variables.
* Variables are not declared. They come into existance on their first assignment.
* Variables are not typed. They can contain a number, a character, or a string.
  + Actually all variables are strings. If the string contains only digits, it can be used as a number.
* Use the local keyword to make variables local to a function.
* Pretend $ to access a variable's value. E.g., fred=37, echo $fred

Assignment statement:

* variableName=value
* No space(s) between the variable name and its assigned value
* If the value includes spaces, use quotes around the value

Calculations:

* Use let: a=3; b=4; let c=$a+$b
* Enclose the equation in $((equation)): c=$(($a + $b))

Conditional statements:

* **if** [[ *expression* ]]; **then** *statement(s);* **fi**
* **if** [[ *expression* ]]; **then** *statement(s);* **else** *statement(s);* **fi**
* **if** [[ *expression* ]]; **then** *statement(s);*  
  **else** **if** [[ *expression* ]]; **then** *statement(s);*

**else** *statement(s);* **fi**

expression: There are too many to list them all. ([See this list](https://www.gnu.org/software/bash/manual/html_node/Bash-Conditional-Expressions.html).) Here are some examples:

|  |  |
| --- | --- |
| -a file | True if the file exists |
| -f file | True if the file exists and is a regular file |
| str1 = str2 | True if string str1 is equal to string str2 |
| str1 != str2 | True if string str1 is not equal to string str2 |
| str1 < str2 | True if string str1 is less than string str2 |
| arg1 OP arg2 | Arithmetic binary operators; OP is one of -eq, -ne, -lt, -le, -gt, or -ge. Arg1 and arg2 may be positive or negative integers. |

Interation (looping) statements:

* **for** j in list; **do** statements(s); **done**
* **while** [ *expression];* **do** statements(s); **done**
* **until** [ *expression ];* **do** statements(s); **done**

Function definition:

* **name() {** statements(s) **}**
* within the function:
  + $0 is the function name.
  + $1, $2, $3, …, $N are the function arguments.
  + $\* represents all arguments sent to the function.
  + $# is the number of arguments sent to the function.

Function call: (use the function name, but no paratheses)

* **name**
* **name arg1 arg2 arg3 …**

Capture a command's output:

* Use the backtick (`) a=`ls –la` (depreciated style)
* Command substiution a=$(ls –la) (preferred style)

User input: (read command)

echo Please, enter your first and last name

read first last

echo "Hi $first $last!"