CS 35L Fall 19 Section 3 Notes W2 Mon
Zhaowei Tan

Shell basics:

1. We don't need to compile a shell script before we run it, unlike C or C++.

2. To declare the interpreter for the script, we start the file with #!/bin/sh (which can be a symbolic link to bash shell script), so that when we execute it, the system knows it is a shell script, and runs it with correct interpreter accordingly.

3. Comment in shell scripting: #

4. All the commands can be executed inside a bash script, just like when we execute them inside a terminal.

5. To assign a variable: var=5, no need to declare it beforehand. To print it, echo $var. Pay attention to when we need the dollar sign. Also pay attention to the space.

6. For arithmetic calculation, use $(()). Example: i=$(($i + 1))

7. In bash, in order to assign the output of commands to variables, we can wrap our command inside `. For example, a=`ls /usr/bin`. Alternatively, we could use a=$(ls /usr/bin)

8. String could be wrapped inside " or ', there's a major difference though. if we have lan=English
   echo "Language is $lan" and echo 'Language is $lan'
   have different outputs. You need the "" for the white space.

9. Some built-in variables:
   $0: the first command line argument (i.e. our script name)
   $1: the second command line argument (of script or function, depending on where you call it)
   $2: similar to $1
   …
   $#: the number of arguments (does not count $0)
   $*: all the input arguments
   $$: process ID

10. For loops, if, or function, check the script I upload.

11. If statement: (you need the space)

```
if [ condition ]
then
  commands
elif [ condition ]
then
```

```
 commands
else
 commands
fi
```

12. Some useful conditions:
    `string1 == string2` # if two strings are identical
    `integer_a –eq integer_b` # if a equals to b
    Similarly, we have `–gt, -ge, -lt, -le` for greater than, greater than or equals to, less than, less than or equals to, respectively.

13. Extra conditions used in `if` statement:
    ```
    -e file   # file exists
    -f file   # regular file exists
    -d file   # directory exists
    ```

14. Use `[ ! expression ]` returns true if expression is not true.

15. While loops:
    ```
    while [ condition ]
    do
     commands
    done
    ```

16. For loops:
    ```
    for i in list
    do
     commands
    done
    ```
    Here the list could be a string with items separated with space
    So this makes sense: `for i in `ls .``

17. Bash function:
    ```
    function function_name {
     commands
    }
    ```

18. Pass arguments to functions:
    `function_name argument1 argument2`
    Get the arguments inside the function: `$1, $2 …`
    Same as passing parameters from command lines

19. Get return value from the function:
    Inside the function, use `return` keyword for exit status.
    After calling the function, get return value: `$?`
    Also note that the variables you define inside the functions are usually global, which can be accessed outside the scope of the function

20. Integer list: `{start..end}`

e.g. for i in {1..10}

21. Get the length of the string
    ${#var}

22. Some other extras: Arrays, case, … In the script I upload!