CS231-01
Fall 2020
Bash Lab #5
Due: 12-13-2020, 11:59 PM Anywhere on Earth time (AoE)
Submission: None

---

1 Bash Lab #5

This lab is part of Assignment #8, and has the same deadline as Assignment #8. This lab introduces more commands of the `bash` (AKA "Bourne Again SHell") shell system. Use Pulse Secure and `ssh` to connect to gremlin.smith.edu to perform the lab activities. This lab has no associated Moodle quiz to submit when you have completed it, but you will be tested on the material in your final exam.

In this lab, you will explore new ways of redirecting standard out (`stdout`), and you will learn the powerful `for` -loop in `bash` .

1.1 A Different Way of Redirecting stdout

If you use `>` at the end of a command, and add the name of a file at the end, the output of your command will be stored in the file. If the file already existed, it will be erased first, and overwritten.

Example:

```
du /etc | sort -n |  tail -10  > largest.txt
cat largest.txt
```

Notice that this generates the list of the largest files found etc and stores this list in `largest.txt` . Try this now:

```
du /usr/games/* | sort -n | tail -10  > largest.txt
cat largest.txt
```

See that the file `largest.txt` has been overwritten.

Something fun: you will notice that Linux has a game directory! This is typical of Linux systems. There are always a few games hidden somewhere on a Linux computer. Ours has only a few. Some linux distributions come with a larger number of them. Let's spend 2 minutes playing: try this:

```
cowsay Hello there
cowsay csc231 is not for the faint of heart
```

Now try this command (a `for` loop which you will learn in this lab):

```
for i in hello there cs231a Student ; do cowsay $i ; done
```

OK, back to serious stuff. What if instead of erasing the contents of `largest.txt` every time we wanted to keep adding to its end, i.e. appending text to it? The redirection symbol for that is `>>` .

```
du /etc | sort -n |  tail -10  > largest.txt
du /usr/games/* | sort -n | tail -10  >> largest.txt
cat largest.txt
```

The first `du` command will create a new `largest.txt` and the second `du` command will append to it.

In general you want to use `>` once, for the first command, and `>>` several times for all the information you need to append to the output file.

1.2 `bash` Loops

`bash` supports loops! It is a powerful way of repeating commands. Try this simple example at the `bash` prompt:

```
for i in 1 2 3 4 5 6 7 8 9 10 ; do
     echo "i = $i"
done
```

Explanations:

- `i` is used as the index of the loop. It is declared by putting it right after the `for` keyword.
- `i` will take all the values in whatever list is given after the `in` keyword.
- The semi colon, `;` , ends the declaration, and is followed by `do`
- The next lines will be repeated for every value `i` will take. In our case, just the `echo` command, which is `bash` 's print command.
- We close the body of the `for` loop with `done`

Note that when you want to use the loop variable inside the body of the loop, you write `$i` . Variables are declared without `$` but are prefixed with `$` when used.

1.3 Several Different Kinds of Loops

Try these different variations of loops. They contain new features and constructs, and you will figure out how they work by playing with them.

1.3.1 Loop 2

```
for i in `seq 1 10` ; do
    echo "i is $i"
done
```

Note: you need to use backquotes (`` ` ``) around the `seq 1 10` command. With `bash` , putting backquotes around a string means that it is a command that should be executed, and the output of this command becomes a list of lines through which the `for` command will iterate.

1.3.2 Loop 3

```
for i in a b c d e f g h i j ; do
    echo "i = $i"
done

for j in {a..j} ; do
    echo "j = $j"
done
```

1.3.3 Loop 4

```
for i in `seq 1 10` ; do echo "i is $i" ;  done
```

Loops can be written on a single line if desired. The semicolon marks the ends of commands, and parts of the loop. Make sure you put spaces around the semi-colons!

1.3.4 Challenge #1:

Use the `man` page for `seq` and figure out a way to make your loop print all the numbers from 20 down to 1.

1.3.5 Loop 5 – Nested `for` Loops!

```
for i in a b c d e f ; do
    for j in 0 1 2 3 4 5 ; do
       echo -n "$i$j "
    done
    echo ""
done
```

Notes:

- The `-n` flag/switch for the first `echo` indicates that the cursor should not go to the next line after printing the string.
- The `echo ""` is a way to print a linefeed/carriage return to go to the next line.

1.3.6 Challenge #2:

Use a nested `for` loop to print this pattern of 10 lines with 1 to 10 stars:

```
*
**
***
****
*****
******
*******
********
*********
**********
```

1.3.7 Challenge #3:

Use a nested `for` loop to print all the `cs231a-xx` account usernames in this range:

```
cs231a-aa
cs231a-ab
cs231a-ac
...
cs231a-az
cs231a-ba
cs231a-bb
cs231a-bc
...
cs231a-bz
```

Note: this is the kind of loop that could be used to create the CS231 accounts. Instead of printing the string `cs231a-xx` , a Linux admin could use the `adduser` command, which adds a new user account to the system.

1.3.8 Loop 6 – Looping through files

You can loop through files by using `*` , or `*.asm` , or some combination of a string and the `*` character(s) to generate a list of files and iterate through them. Try the following examples:

```
for i in *.asm ; do
    echo "file = $i"
done

for i in *hello* ; do
    echo "file = $i"
done

for i in *.c ; do
    echo "file = $i"
done
```

1.3.9 More Complex Looping

Assume that you want to create a text file that contains all your assembly programs, with the name of the program between two dashed lines, followed by the contents of the program. This is how you could do it:

First let's make the output of the command go to the display, so that we can see the output:

```
for i in *.asm ; do
         echo "----------------------------------------"
         echo "$i"
         echo "----------------------------------------"
         cat $i
         echo ""
done
```

Try it!

Now let's redirect the output of all the commands to a file we'll call `allAsmFiles.txt`

```
for i in *.asm ; do
         echo "----------------------------------------" >> allAsmFiles.txt
         echo "$i" >> allAsmFiles.txt
         echo "----------------------------------------" >> allAsmFiles.txt
         cat $i >> allAsmFiles.txt
         echo "" >> allAsmFiles.txt
done
```

Try this as well, and look at `allAsmFiles.txt` with `emacs` or `less` . See how quickly you created this? This can be used to generate quick Jupyter or RStudio notebooks, with the correct header information, for those of you using these packages.

1.3.10 Additional Information on `bash` Loops

In case you want to learn more about loops, there are many interesting tutorials on the Web. Here are a few of them:

- ryanstutorials.net
- thegeekstuff.com

1.4 Challenge Solutions

1.4.1 Challenge 1

```
for i in `seq 10 -1 1` ; do echo "i is $i"; done
```

1.4.2 Challenge 2

```
for i in `seq 10` ; do
for j in `seq $i` ; do
echo -n "*"
done
echo ""
done
```

1.4.3 Challenge 3

```
for i in a b ; do
for j in a b c d e f g h i j k l m n o p q r s t u v w x y z ; do
echo "231b-$i$j"
done
done
```

```
 ____________________
< That's all folks! >
 --------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```