CS246-Assign01 V1.1: Winter 2013
http://plg.uwaterloo.ca/~holt/cs/246/2013/asgn01/asgn01.htm

Submit. Instructions to submit assignments (using Marmoset):
https://marmoset.student.cs.uwaterloo.ca/

Instructions to submit to Marmoset can be found here:
http://www.student.cs.uwaterloo.ca/~cs145/marmoset.shtml

Bash. All shell scripts you write should work on the **bash** shell.
Linux. Do assignments on UW CS Linux machines.
ASCII. Turn in the written part of this assignment as an ASCII file.
ASCII editor. You will need to use an ASCII editor such as vi or emacs to do this assignment.

Asgn 1a: Getting started with the shell. For each part of this question, give one Linux command, on a single line. All of your commands should be in the file `1a.answer`. Create your answers using the UW Linux server.

Log on to Linux. Enter command to:

1a1) In your home directory, create a file named greetings that contains the words:
Hello there
by typing
echo Hello there > greetings
1a2) List the names of the files in your home directory by typing
ls
1a3) Clear your screen by typing
clear
1a4) Using the mkdir command, create a directory named action
1a5) Inside the action directory, create a file named things.to.do which contains the sentence:
Remember to say greetings
1a6) Move the things.to.do file into your home directory
1a7) Rename the things.to.do file to do-it-now
1a8) In your home directory, list the files and directories in the long format, by giving the command:
ls –l # List directory, with "long" flag -l

Submit your answers to 1a1) to 1a8) to Marmoset in a file named 1a.answer. The file should contain a total of 8 lines, all in ASCII.

Asgn 1b Debugging. It is recommended that when you develop a shell script that you should do it a bit at a time, using many debugging statements (which are usually echo statements). Here is an example shell script named tryeval containing debugging statements:

```
#!/bin/bash
echo tryeval START   # Check that script starts
```

```
echo NUMBER OF ARGS IS $#
echo ARG1 IS $1      # Look at arg1
echo NOW eval OF '$1'
eval $1
echo tryeval DONE    # Check that script finishes
```

Write a file called tryeval that contains this script. Make it executable using the chmod command. Test the script by typing:

```
./tryeval whoami
./tryeval date
./tryeval garbage1 garbage2
./tryeval
```

Submit the tryeval script to Marmoset

Asgn 1c: Creating directories. Create the following directory structure in your Linux home directory. Each leaf in the structure is a file (not a directory)

```
cs246
     A1
          hello
          goodbye
          junk.h
          source-code
               greeting.h
               greetingCPP.cpp
               ugly
```

Create the contents of leaf files as follows:

hello: `echo Hi`
goodbye: `echo Bye`
junk.h: `// Nothing interesting here`
greeting.h: `const string greetVariable = "Happy New Year";`

Note: Simple files can be created as follows:

```
echo content of file > fileName
```

Make the hello and goodbye files executable by using the following commands:

```
chmod u+x hello
chmod u+x goodbye
```

Try executing them by typing

```
./hello
./goodbye
```

The greetingCPP.cpp file should contain:

```
#include <iostream>
#include <string>
```

```
using namespace std;
#include "greeting.h"
int main ()
{
     cout << greetVariable << "\n";    // Print message
}
```

Compile greetingCPP.cpp by typing the following:

```
g++ greetingCPP.cpp
```

Execute the program by typing

```
./a.out
```

Create the contents of the *ugly* file to be:

```
// This is a snippet from a C++ program
     for (int counter = 0; counter < 5; counter ++;);
     {  // Body of for loop (supposedly)
          cout << "Counter is: " << endl;
     }
```

Now you will need to submit a1.zip to Marmoset. To do so navigate to the cs246 directory and run the following command:

```
zip -r a1.zip A1
```

Submit the newly created a1.zip file to Marmoset.

Asgn 1d: Aliases and bin commands. Create an alias for the *more* command called *p* (for **p**rint). The p command should work just like the *more* command. The result of you typing

```
p file1 file2 …
```

should be each of the listed files printed (output to) your screen.. The p command should work just like the *more* command. In your bin directory, create a command called *clr* which does the same thing as the Linux *clear* command. Make your *clr* command executable (using *chmod*). The first line of your *clr* command should specify that it uses bash (use #!/bin/bash).

Try typing the command

```
echo $PATH
```

and check that your bin directory is in your search path for finding commands.

Now consider the three commands named clear, clr and p. Inspect their types by entering these commands.

```
type clear
type clr
type p
```

Now consider the two commands p and clr. Start up a new subshell (by typing *bash*) and determine which of these two still work in the subshell. Explain why. (You can return to your original shell by typing *exit*). Log out and log back in again, and determine which of these two still work. Explain why.

Submit your answers to these questions to Marmoset in a file named 1d_answers.txt.

Asgn 1e: Ugly script. Write a shell script named uglyfor that takes no arguments. ~~It detects "ugly" for loops, as illustrated here.~~ It detects "ugly" for loops (as illustrated below) within files, and recursively within all files found in directories, from where uglyfor was called.

```
for (int i = 0; i < 10; i += 1);  // Ugly final semicolon
{    // This part is the loop body
    cout << i << endl;
}
```

This for loop is *ugly* because the final semicolon (inadvertently) causes the loop body to be ignored during the looping. You are to write a shell script uglyfor that prints out lines recursively that contain any characters, then the letters 'for', then any characters, then the characters ')' and ';'. As a result, ugly for loops are detected (along with some lines that look a lot like ugly *for* loops).

To complete this question you should be using a command called "egrep." You can read the description of egrep by typing

```
man egrep   # Manual for egrep
```

Submit your uglyfor file to Marmoset.

Asgn 1f: Grindx script. Write a shell script named grindx which takes a single argument, as in

./grindx stuff

As a result, the command recursively lists the name of every file whose name or contents contains the string *stuff*. Implement grindx by having it call in order two other commands, named grepx and findx, which will now be described. The grepx command, as in

./grepx stuff

takes a single argument, illustrated here as stuff, and recursively lists all files that contain the string *stuff*. Implement grepx using the Linux egrep command. The findx command, as in

./findx stuff

takes a single argument, illustrated here as stuff, and recursively lists all files whose names contain the string stuff. Implement findx using the Linux find command. To keep things simple you can assume that these commands always are given exactly one argument, and that argument contains no white space or magic characters such as * or ?.

Asgn 1g: Mark readme. Write a shell script named markinfo. This command accepts an optional "verbal" flag –v followed by one or more arguments a1, a2, etc. These arguments are the names of directories or files, as in

markinfo [-v] a1 a2 …

Each file (and each file recursively within the directories) having one of the 4 names

readme, read.me, README or READ.ME

is to be "marked" by the command. This is done by appending .INFO to the file's name. For example, suppose that this command is given:
$ markinfo READ.ME
Assuming READ.ME is a file (not a directory), then its name is marked so it becomes READ.ME.INFO
If the -v flag is given (right after markinfo), then the command writes a marking message of the form:

MARKING READ.ME

Here is another example.
$ markinfo a1
If a1 is a directory, then all files recursively within a1, having one of the above 4 names, is similarly marked (by appending .INFO to the end of the file name).
In this example

$ markinfo -v file1 dir2 dir3

Any files (file1 or files within dir2 or dir3) will be marked if they have one of the 4 names.

If an argument ai is not actually a file or directory, the command should print an error message, return 1(one) and halt.
If the command is given no argument (-v doesn't count as an argument), the command should print an error message, return 2 and halt.
Any error message must begin with the letters ERROR followed by a brief message fitting on a single line.

Hints: Research the "find" command to locate particular file names. Use a "for" loop to act on each argument (after skipping -v if it is present). Shell scripts such as markinfo can be very difficult to debug, so it is recommended that you build and test your markinfo command a bit at a time. You may want to study the cleanup example in Buhr's slides.

Submission. Submit the following files to Marmoset:

Asgn	**File to submit**	**Points**
1a	1a.answer	4
1b	tryeval	3
1c	a1.zip	6
1d	1d_answers.txt	5
1e	uglyfor	6
1f	grindx	8
1g	markinfo	8
	STYLE	10
	TOTAL	50