Programming Assignment 1

Computer Organization
CSI404
Spring 2015

Finish and PUSH YOUR WORK: Due Thursday, Jan. 29, 11:59PM.

Learning Goals

How to achieve the goals is detailed after we state them.

1. Review very basic C programming and the mechanics of doing it in a command line Unix environment. You will program calls to the `printf` and `scanf` functions to output and input data.
2. Get started by cloning a GIT repository containing a separate subdirectory for the DataDemo project. Cloning gives you your own local repository that starts out being a copy of an existing one.
3. Practice incrementally developing and testing admittedly very simple software so you are introduced to how professional developers work on big projects and how you will work on the much more challenging projects that will come after this assignment.
4. Practice preserving (like professionals do) a copy of each incremental version so that you will know how to do this during much more complex projects. Learn that a GIT "commit" is effectively a copy of your entire project made at the time you decided to preserve what you have.
5. Use the following git commands and understand basically what each does and is for: `git status`, `git log`, `git commit -a`, `git add` and `git push`.
6. Learn to use the Unix `script` command to capture a typescript of all user input and program output into a file for future reference and analysis.
7. Follow our instructions to "push" your own local repository to our `git404.cs.albany.edu` server so we can grade and credit your work. This should enable you and the course staff to detect and fix an problems you might have with our new way of turning in course project work BEFORE problems cause your work to become late or unacceptable.

OK, here is what to do:

Command:

```
git clone git404@git404.cs.albany.edu:COProjects
```

This will NOT WORK UNLESS you have completed Programming Assignment 0. (That is, you have made the `id_rsa.pub` file and sent it with `mailx` to Prof. Chaiken in time for him to give you access to the server.)

Once you execute this *git clone* command correctly you should NEVER HAVE TO GIVE IT AGAIN IN THIS COURSE. (We may ask you to clone other repositories besides `COProjects` and we will ask you to *pull* this repository again and again to get new assignments.)

You should see something like:

```
unix1% git clone git404@git404.cs.albany.edu:COProjects
Cloning into 'COProjects'...
remote: Counting objects: 4, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 4 (delta 0), reused 0 (delta 0)
Receiving objects: 100% (4/4), done.
Checking connectivity... done.
unix1%
```

See what you got. Command: `ls -l`
and note that the `git clone` command created a directory (or folder) named `COProjects` for you. [If for some reason you already have a `COProjects` directory, perhaps from taking the course previously, just rename it with the command:`mv COProjects OLDCOProjects`] Now give the commands AND NOTE WHAT'S PRINTED EACH TIME:

```
ls -l COProjects
cd  COProjects
ls -l DataDemo
ls -l DataDemo/DataDemo.c
cd DataDemo
ls
```

Look at the program you got with the `less` pager. MEMORIZE: The `q` key gets you out of the pager. You must use it after the command:

```
less DataDemo.c
```

VERY IMPORTANT: MAKE SURE YOU CAN GET IT TO COMPILE AND RUN! How?
To compile (and link): `gcc DataDemo.c -o DataDemo`
To run: `./DataDemo`
(note: the `-o DataDemo` option makes `gcc` name the executable file that was output by `gcc` be `DataDemo` instead of the default name `a.out`.

1 Version to Print Your Own Name

First, make an improvement that admittedly is very simple. Of course, you'll have to review and actually use a text editor as you did in CSI333 to edit and save the file named `DataDemo.c`

1. Modify the program so Prof. Chaiken's name is replaced by your own name.
2. MAKE SURE IT COMPILES AND RUNS, showing YOUR NAME.

Second, learn how GIT will detect that you modified one of the files it is tracking.

1. Command: `git status`
2. Answer to yourself: Which file is GIT tracking?
3. Answer to yourself: Which file(s) is GIT not tracking?

Third, Learn or write in a notebook the GIT command which SAVES THE SNAPSHOT of the current version as a COMMIT, storing that snapshot (a new commit) in YOUR LOCAL REPOSTORY:

```
git commit -a -m "Profs name replaced by mine."
```

(The `-a` option means automatically add any changes of files that have been in the previous version.)

The option `-m "`*One line to describe this commit*`"` satisfies GIT's requirement that you MUST give a description of every commit. (If you omit this option, the `git commit` command will run a text editor and you will have to make the editor save the file and exit before doing the commit. So don't omit the `-m` and description string.)

1. Command: `git status`
   (AFTER having done your first GIT commit.)
2. Answer to yourself: How is the report from the `git status` commanded BEFORE you committed (with `-a`) the change to `DataDemo.c` COMPARE to the status report commanded AFTER that commit?

2 Version to Print C int Values in Decimal

1. Recall or find out what this the line of code does and why:
   `printf("Output in decimal:%d\n", 2015);`
   You will be using similar code. Please note (1) It is a function call with two parameters. (2) The first parameter is the "format string". (3) The `printf` function converts the value two thousand fifteen to the ASCII string `2015` and prints the format string with `%d` replaced with `2015` because `%d` is the decimal conversion format code.
2. Add the above line to your program so it prints its line after the "`Data demonstrator ...` " greeting line.
3. Code another line that outputs a different integer in the same way, preceeded by
   `Output in decimal:`

NOW, at this point practice compiling AND TESTING this admittedly simple increment AND PRESERVE IT by making another commit.

1. Command: `git status`
2. Answer to yourself: How is that status report different after adding the two output operations to the C program?
3. Command: `git log`
4. Answer to yourself: What does `git log` report?
5. Answer to yourself: What kind of data does GIT use for the names of the commits it had created? (Look up SHA-1 if you're curious.)
6. Now, give your second
   `git commit -a -m "`*A one line informative message abt the commit.*`"`
7. Command: `git log`
8. Answer to yourself: What's different?
9. Command: `git status`
10. Answer to yourself: What's different?

3 Version to Input ints in Decimal

Now that YOU FINISHED TESTING **outputting integers**, we want you to practice inputting them AND USE AN AVAILABLE way to TEST that the INPUT code is correct.

Revise your program so that it does what is SPECIFIED HERE in the numbered order:

1. It prints the greeting with your name the same as before.
2. It prints `Input a decimal integer:`
   There should be NO NEWLINE after the colon.
3. It inputs a decimal integer somebody types, and stores it in some variable (Please assume the person will make no mistakes.)
4. It outputs with `Output in decimal:` in front the same integer that somebody inputted. (This TESTS the inputting operation!)
5. It repeats the previous three operations (once). (Just repeat the lines of code. Putting them into a function will be required in Programming Homework 2.)

Notes:

- Use the function call:
  `scanf("%d",` *POINTER TO SOME int VARIABLE* `);`
  It inputs characters, tries to convert them to an `int` value, and if successful, *uses the POINTER to locate where in memory to copy the converted value.*

- You must declare that `int` variable, say with:
  `int input_var;`
  It is best to make it a local variable belonging to the function that uses it. Remember that you must put the declaration INSIDE that function `main()` to make it local.

- You must REMEMBER that whenever say `input_var` is an C variable, the expression `&input_var` expresses *the pointer to that variable* which is *the memory address where that variable is located in memory.*

Now COMPILE AND TEST, possibly edit, RECOMPILE and RETEST several times until the tests indicate it is right.

AFTER finishing perfecting and testing your work, you must, for full credit, put in your project a file that proves you really tested it. The Unix `script` program will capture user input and program output. Instructions:

1. Command: `script`
2. You will see the message: `Script started, file is typescript`
3. Give the command to run your program: `./DataDemo`
4. Type your first test input. See the test output.
5. Type your second test input. See the test output.
6. Notice your program should exit and you will see the shell prompt again.
7. **IMPORTANT:** Either command `exit` or type control-d (Hold ctrl key down and press `d`) Your choice.
8. You will see the message:
   ```
   exit
   Script done, file is typescript
   ```
9. Command: `cat typescript`
   (This shows you what's inside the file named `typescript`.)

The `script` command created a new file named `typescript`. It is NOT YET being tracked by GIT. You have to MANUALLY ADD IT. Do this:

1. Command: `git status`
   Note to yourself (1) Which file(s) is(are) tracked and modified? (2) Which files are tracked and not staged for commit? (3) Which files are not tracked?
2. Command: `git add typescript`
3. Command: `git status`
   Note to yourself how this differs from the `git status` before the `git add`
4. Now give the `git -a -m "`*Informative Message*`"` command.
   You don't have to `git add DataDemo.c` because you used the `-a` option AND `DataDemo.c` was already tracked. Now `typescript` is being tracked too.

4 You MUST Push to Get Credit!

Whenever you finish a programming assignment, it will NOT BE GRADED unless you successfully git push your repository to our server. So write the command to do this in your notebook so you'll give it correctly now and after you finish future assignments:

`git push git404@git404.cs.albany.edu:submissions/`*YourNetID*`/COProjects master`

DO NOT OMIT `master` at the end, and the space between it and the `4`!

DO NOT write *YourNetID* literally! Substitute your personal UA Net ID between the two `/` characters. Fortuanately, the push will give an error message if you make *that* mistake.

Frequently Asked Questions

4.1 What if I made mistakes and committed wrong work?

Don't worry! DON'T WASTE TIME STARTING ALL OVER! Fix the mistakes one at a time. Just make a new commit with a message that says "Fixed the *(specific short description)* mistake" Simply repeat fixing the mistakes until your work gets back on track. Then make a commit of the particular assigned work step. The TA's will tell from the commit message what commits to grade. NO CREDIT IS LOST (and much learning is gained) by committing mistakes and later committing corrections!

4.2 How can I see what I turned in?

That's easy. What we do and recommend is to (1) go to a temporary directory (or folder), (2) `git clone` a local copy of the repository holding your submission from `git404.cs.albany.edu`. (3) Examine it, including its log, to see what we got from you. (4) DELETE IT SO you don't confuse this temporary copy with the stuff in your WORKING `COProjects` directory and the local repository you are using for real.

1. Command: `cd /tmp`
2. Command: `mkdir` *YourNetID*
3. Command: `cd` *YourNetID* The above two commands will keep other student's stuff out of your way because the `/tmp` directory is shared.
4. Command:
   `git clone git404@git404.cs.albany.edu:submissions/`*YourNetID*`/COProjects`
5. Command: `cd COProjects`
6. Look around.
7. Use the commands `cd` and then `cd COProjects`, or learn about TILDE for your home directory and just do `cd ~/COProjects` to go back to the place where you are working for real.
   Use the command `/tmp/`*YourNetID*`/COProjects` to go back to looking at what you submitted.
8. When you done, VERY CAREFULLY remove all the temporary stuff with:
   `rm -rf /tmp/`*YourNetID*
9. Command `cd ~/COProjects` to go back to where you're working for real.

4.3 If I omit the -m in `git commit`, how can I get out of the text editor?

By default, the `vi` editor is started. Use it to edit in the message then save and exit if you know how.

Here is how to just exit that `vi` editor. Type ESC`:q` (that's the ESCAPE character (upper left-most key on the whole keyboard, left of the F1 function key), colon and then a `q`) and press ENTER.

In case you had typed something else and seem to be stuck, press the ESC key (to change `vi` from text entry mode to command mode) and type `:q!` (exclamation point after the q) and ENTER to exit without saving. Don't worry, `git` will not commit with an empty message so you get another chance.