LAB 2 - Git Educated

A mini tutorial on vi, gcc, make, and git

John Dempsey
COMP-232: Programming Languages
California State University Channel Islands
https://comp232.com

git is a Version Control System (VCS). git keeps track of the changes you make to your source code or any file, like a Makefile. It has the ability to store your changes locally on your laptop in the Centralized VCS or on remote websites, like github.com or bitbucket.com, called Distributed VCS.

So let's git with the program and try out some git commands on your local system.

1. Open an Ubuntu terminal window (or Mac Terminal Window).

2. Let's fetch a list of available updates using:

   % **sudo apt update**

3. Let's install updates using:

   % **sudo apt upgrade**

4. Use script to record our work into the git.txt file.

   % script git.txt ← **NEEDED TO TURN IN YOUR WORK.**
   Ignore the git.txt file in the ls & git commands.

5. Let's see what version of Ubuntu we are using:

   % **lsb_release -a**

6. Check to see if git exists on your system by typing:

   % **git --version** ← There's two hypens being used.

   % **which git** ← Shows us the directory where the binary for git is located.

7. Check to see if gcc exists on your system by typing:

% **gcc --version**

If gcc doesn't exist, you'll see this message:

% **gcc --version**

Command 'gcc' not found, but can be installed with:

% **sudo apt install gcc**

If gcc is not found on your system, then run:

% **sudo apt install gcc**

8. Let's create a test directory LAB2 to play with git. You can name your directory anything. I like directory names in capital letters, like LAB2, but you can name your directory LAB2.d (.d for directory), lab2, or LAB2.

% **cd** ← Change to your home directory

% **mkdir LAB2**

% **cd LAB2**

% **ls** ← No files in LAB2 directory

9. Now using vim again, let's edit the file main.c. Here's what main.c should look like:

```
#include <stdio.h>

// Function prototypes
void hello();

void main()
{
   hello();
   hello();
   hello();
}
```

10. Write hello.c to disk by using:

:w

11. Now while still in vi editing main.c, let's edit hello.c by using:

:e hello.c

12. Using vim, let's edit hello.c to be:

```
#include <stdio.h>

void hello()
{
   printf("Hello World!\n");
}
```

13. Save file hello.c using **ZZ** or **:wq**

14. To compile the project, we need a Makefile. **But don't forget to use the TAB characters when indenting in a Makefile.** Using vim, create your Makefile to look like:

```
#
#  Makefile
#

SOURCE =\
      hello.c\
      main.c

CFLAGS  = -g

.c.o:
      gcc $(CFLAGS) -c $*.c;

hello: *.o
      gcc *.o -o $@;
```

15. We should have three files in LAB2, which are Makefile, hello.c, and main.c. Run:

% **ls -l**

16. Now let's compile main.c and hello.c by running:k

    % **make**

17. Assuming no errrors, run your program:

    % **hello**

18. Let's create a documentation file called hello.txt which contains:

    The hello program writes out "Hello World!" three times.

    This is the classic hello world program.

19. Let's see what files we currently have in our directory:

```
john@oho:~/COMP232_LABS/LAB2$ ls -l
total 40
-rw-r--r-- 1 john john   113 Sep  2 16:12 Makefile
-rwxr-xr-x 1 john john 17448 Sep  2 16:13 hello
-rw-r--r-- 1 john john    65 Sep  2 16:10 hello.c
-rw-r--r-- 1 john john  3472 Sep  2 16:13 hello.o
-rw-r--r-- 1 john john    99 Sep  2 16:14 hello.txt
-rw-r--r-- 1 john john   104 Sep  2 16:09 main.c
-rw-r--r-- 1 john john  3472 Sep  2 16:13 main.o
```

20. Now let's use git. To create your Centralize VCS repository, type:

    % **git init** ← Creates the git CVCS repository on your local laptop.

```
john@oho ~/LAB2
$ git init
hint: Using 'master' as the name for the initial branch. This default branch name
hint: is subject to change. To configure the initial branch name to use in all
hint: of your new repositories, which will suppress this warning, call:
hint:
hint:   git config --global init.defaultBranch <name>
hint:
hint: Names commonly chosen instead of 'master' are 'main', 'trunk' and
hint: 'development'. The just-created branch can be renamed via this command:
hint:
```

hint: git branch -m <name>
Initialized empty Git repository in /home/john/LAB2/.git/

21. Let's see what was created in the .git directory:

% **ls -la** ← To see the .git directory, add the -a (for all).

john@oho:~/COMP232_LABS/LAB2$ **ls -la**

```
total 40
drwxr-xr-x 1 john john  4096 Sep  2 16:16 .
drwxr-xr-x 1 john john  4096 Sep  2 16:06 ..
drwxr-xr-x 1 john john  4096 Sep  2 16:16 .git
-rw-r--r-- 1 john john   113 Sep  2 16:12 Makefile
-rwxr-xr-x 1 john john 17448 Sep  2 16:13 hello
-rw-r--r-- 1 john john    65 Sep  2 16:10 hello.c
-rw-r--r-- 1 john john  3472 Sep  2 16:13 hello.o
-rw-r--r-- 1 john john    99 Sep  2 16:14 hello.txt
-rw-r--r-- 1 john john   104 Sep  2 16:09 main.c
-rw-r--r-- 1 john john  3472 Sep  2 16:13 main.o
```

% **cd .git** ← Change into the .git directory

% **ls -l** ← Let's start by viewing just the top level directories.

% **ls -lR | more** ← Now list all files and directories recursively and in long format.

% **cd ..** ← Go up one directory which is /home/<username>.

john@oho:~/LAB2$ **ls**
Makefile hello hello.c hello.o hello.txt main.c main.o

john@oho:~/COMP232_LABS/LAB2$ **ls -l**
total 40
-rw-r--r-- 1 john john 113 Sep 2 16:12 Makefile
-rwxr-xr-x 1 john john 17448 Sep 2 16:13 hello
-rw-r--r-- 1 john john 65 Sep 2 16:10 hello.c
-rw-r--r-- 1 john john 3472 Sep 2 16:13 hello.o
-rw-r--r-- 1 john john 99 Sep 2 16:14 hello.txt
-rw-r--r-- 1 john john 104 Sep 2 16:09 main.c

-rw-r--r-- 1 john john 3472 Sep 2 16:13 main.o

22. In git, files can be in one of three areas:

    1. **Untracked Files** – These files exist in the current directory but have not been added to the Staging Area.

    2. **Staging Area** – These files can be under review by others before being added to the Repository.

    3. **Repository** – This is your local Repository for storing files on your local laptop.

23. Let's figure out where our current files are right now, by typing:

    john@oho:~/COMP232_LABS/LAB2$ **git status**
    On branch master

    No commits yet

    Untracked files:
      (use "git add <file>..." to include in what will be committed)
        Makefile
        hello
        hello.c
        hello.o
        hello.txt
        main.c
        main.o

    nothing added to commit but untracked files present (use "git add" to track)

24. Let's see what's in the Repository:

    % **git log**

    You'll see:

        fatal: your current branch 'master' does not have any commits yet

because we haven't added any files to the repository yet.

But before we check in all the files in our directory, we don't want to check the .o or the executable files. So edit the file **.gitignore** using **vi .gitignore** and add the following two lines:

***.o**
hello

This says to ignore the object and executable files and the git.txt file.

% **more .gitignore**
*.o
hello
git.txt

25. Let's check our status again. Note that *.o and hello files are not listed as Untracked files anymore.

john@oho:~/COMP232_LABS/LAB2$ **git status**
On branch master

No commits yet

Untracked files:
(use "git add <file>..." to include in what will be committed)
.gitignore
Makefile
hello.c
hello.txt
main.c

nothing added to commit but untracked files present (use "git add" to track)

26. Let's add the main.c file to the Staging area and check our status by typing:

john@oho ~/LAB2

$ **git add main.c**

27. Check status.

john@oho ~/LAB2
$ **git status**
On branch master

No commits yet

Changes to be committed:
 (use "git rm --cached <file>..." to unstage)
	new file: main.c

Untracked files:
 (use "git add <file>..." to include in what will be committed)
	.gitignore
	Makefile
	hello.c
	hello.exe
	hello.txt

Note that the main.c file is now in the staging area.

28. Let's add the rest of the untracked files into the staging area.

john@oho ~/LAB2
$ **git add .**

john@oho ~/LAB2
$ **git status**
On branch master

No commits yet

Changes to be committed:
 (use "git rm --cached <file>..." to unstage)
	new file: .gitignore
	new file: Makefile
	new file: hello.c
	new file: hello.exe
	new file: hello.txt
	new file: main.c

29. Let's see what's in the repository.

$ **git log**
fatal: your current branch 'master' does not have any commits yet

No change here. All of the files are still in the staging area.

30. Let's add the files into our local repository.

    $ **git commit -m "Baseline Code and Documentation"**
    Author identity unknown

    *** Please tell me who you are.

    Run

      git config --global user.email "you@example.com"
      git config --global user.name "Your Name"

    to set your account's default identity.
    Omit --global to set the identity only in this repository.

    fatal: unable to auto-detect email address (got 'john@oho.(none)')

31. What's this? What this? An error! Let's add your email and name to git.

    $ **git config --global user.email "john.dempsey@csuci.edu"**

    $ **git config --global user.name "John Dempsey"**

32. It's time to add the files in the staging area to the repository by typing:

    $ **git commit -m "Baseline Code and Documentation"**
    [master (root-commit) 7a82e82] Baseline Code and Documentation
     6 files changed, 35 insertions(+)
     create mode 100644 .gitignore
     create mode 100644 Makefile
     create mode 100644 hello.c
     create mode 100755 hello.exe
     create mode 100644 hello.txt
     create mode 100644 main.c

33. Let's check our status. No files are in the staging area.

    $ **git status**
    On branch master
    nothing to commit, working tree clean

34. Let's check what's in the repository, using: git log; git shortlog; and git ls-tree.

    john@oho ~/LAB2
    $ **git log**

commit 887f947aea1d64082738f3f04561cfb8411bfa1a
Author: John Dempsey <john.dempsey@csuci.edu>
Date: Tue Sep 2 16:27:03 2025 -0700

Baseline Code and Documentation

$ **git shortlog**
John Dempsey (1):
Baseline Code and Documentation

$ **git ls-tree --full-tree -r --name-only HEAD**
.gitignore
Makefile
hello.c
hello.txt
main.c

35. Our original files are still in the directory, as you can see by running:

$ **ls**
Makefile hello.c hello.exe hello.o hello.txt main.c main.o

36. Let's remove file hello.c.

$ **rm hello.c**

$ **ls**
Makefile hello.exe hello.o hello.txt main.c main.o

37. And now let's copy hello.c back from the repository.

$ **git restore hello.c**

$ **ls**
Makefile hello.c hello.exe hello.o hello.txt main.c main.o

38. Let's edit hello.c again and this time underline the text Hello World Underline!

$ **vim hello.c**

```
#include <stdio.h>

void hello()
{
        printf("Hello World Underlined!\n");
```

```
            printf("-----------------------\n");
}
```

$ **make**
gcc -g -c hello.c; ← Note that only hello.c was recompiled and not main.c.
gcc *.o -o hello; ← Link *.o object files to create hello executable.

$ **hello**
```
Hello World Underlined!
-----------------------
Hello World Underlined!
-----------------------
Hello World Underlined!
-----------------------
```

39. Let's check our status:

$ **git status**
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified: hello.c

no changes added to commit (use "git add" and/or "git commit -a")

$ **git log**
commit 887f947aea1d64082738f3f04561cfb8411bfa1a
Author: John Dempsey <john.dempsey@csuci.edu>
Date: Tue Sep 2 16:27:03 2025 -0700

    Baseline Code and Documentation

40. Let's add the updated hello.c to the staging area.

$ **git add .**

$ **git status**
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified: hello.c

41. Commit the files in the staging area into the repository.

$ **git commit -m "Updated hello.c to underline Hello World! "**
[master 38187fa] Updated hello.c to underline Hello World!
1 file changed, 1 insertion(+)

42. Check the status and repository.

$ **git status**
On branch master
nothing to commit, working tree clean

$ **git log**
john@oho:~/COMP232_LABS/LAB2$ git log
commit 38187fa7133561e795c2a53cae44f49571c972b5 (HEAD -> master)
Author: John Dempsey <john.dempsey@csuci.edu>
Date: Tue Sep 2 16:32:36 2025 -0700

Updated hello.c to underline Hello World!

commit 887f947aea1d64082738f3f04561cfb8411bfa1a
Author: John Dempsey <john.dempsey@csuci.edu>
Date: Tue Sep 2 16:27:03 2025 -0700

Baseline Code and Documentation

$ **git ls-tree --full-tree --name-only -r HEAD**
.gitignore
Makefile
hello.c
hello.txt
main.c

43. We're all done!

% **exit** ← Exits the script command.

% **ls -l git.txt**

% **head git.txt**

DURING LAB 3, I WILL LET YOU KNOW HOW YOU CAN TURN IN THE git.txt FILE FOR CREDIT.