

Git Educated

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

John Dempsey

COMP-232: Programming Languages
California State University, Channel Islands

September 3, 2025

Hard Due Date: September 12, 2025

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. 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.
```

5. 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
```

6. Let's create a test directory LAB1 to play with git. You can name your directory anything. I like directory names in capital letters, like LAB1, but you can name your directory LAB1.d (.d for directory), lab1, or LAB1.

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

```
% mkdir LAB1
```

```
% cd LAB1
```

```
% ls                                      ← No files in LAB1 directory
```

7. 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();
```

```
}
```

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

```
#include <stdio.h>
```

```
void hello()
```

```
{
```

```
    printf("Hello World!\n");
```

```
}
```

9. 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 $@;
```

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

```
% make
```

11. Assuming no errors, run your program:

```
% hello
```

12. 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 message used by Dennis Richie in his classic book titled: The C Programming Language.

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

```
% ls
Makefile hello.c hello.exe hello.o hello.txt main.c main.o
```

14. To create your Centralize VCS repository, type:

```
% git init
```

← Creates the git CVCS repository on your local laptop.

john@oho ~/LAB1

\$ 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/LAB1/.git/

15. Let's see what was got created by changing into the .git directory.

% ls ← The directory starts with a dot so we do not see the .git directory.

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

% cd .git ← Change into the .git directory

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

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

% cd .. ← Go up one directory, which will be your /home/<username> again.

john@oho:~/LAB1\$ ls

1.txt 2.txt 3.txt 4.txt Makefile hello hello.c hello.o hello.txt main.c main.o

john@oho:~/LAB1\$ ls -l

total 40

-rw-r--r-- 1 john john 53 Aug 22 11:08 1.txt

-rw-r--r-- 1 john john 135 Aug 22 11:11 2.txt

-rw-r--r-- 1 john john 205 Aug 22 11:14 3.txt

-rw-r--r-- 1 john john 254 Aug 22 11:16 4.txt

```
-rw-r--r-- 1 john john 112 Aug 22 11:28 Makefile
-rwxr-xr-x 1 john john 17440 Aug 22 11:28 hello
-rw-r--r-- 1 john john 67 Aug 22 11:20 hello.c
-rw-r--r-- 1 john john 3448 Aug 22 11:28 hello.o
-rw-r--r-- 1 john john 167 Aug 22 11:30 hello.txt
-rw-r--r-- 1 john john 114 Aug 22 11:20 main.c
-rw-r--r-- 1 john john 3440 Aug 22 11:28 main.o
```

16. In a CVCS, 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.

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

```
% git status
```

```
john@oho ~/LAB1
```

```
$ git status
```

```
On branch master
```

```
No commits yet
```

```
Untracked files:
```

```
(use "git add <file>..." to include in what will be committed)
```

```
1.txt
```

```
2.txt
```

```
3.txt
```

```
4.txt
```

```
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)
```

18. 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** and add the following two lines:

```
*.o  
hello
```

This says to ignore the object and executable files.

```
john@oho ~/LAB1  
$ more .gitignore  
*.o  
hello
```

19. Let's check our status again. Note that *.o and hello files are not listed as Untracked files. (We'll ignore the 1-4.txt files.)

```
$ 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.exe  
hello.txt  
main.c
```

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

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

```
john@oho ~/LAB1
$ git add main.c
```

```
john@oho ~/LAB1
$ 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.

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

```
john@oho ~/LAB1
$ git add .
```

```
john@oho ~/LAB1
$ 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

21. 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.

22. 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)')
```

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

```
john@oho ~/LAB1
```

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

```
john@oho ~/LAB1
```

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

24. 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

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

```
$ git status
```

```
On branch master
```

```
nothing to commit, working tree clean
```

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

```
john@oho ~/LAB1
```

```
$ git log
```

```
commit 7a82e8265773149293c111bf8d30fc7aaa790437 (HEAD -> master)
```

```
Author: John Dempsey <john.dempsey@csuci.edu>
```

```
Date: Wed Aug 24 7:37:49 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.exe
```

```
hello.txt
```

```
main.c
```

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

```
john@oho ~/LAB1
```

```
$ ls
```

```
Makefile hello.c hello.exe hello.o hello.txt main.c main.o
```

28. Let's remove file hello.c.

```
john@oho ~/LAB1
```

```
$ rm hello.c
```

```
john@oho ~/LAB1
```

```
$ ls
```

```
Makefile hello.exe hello.o hello.txt main.c main.o
```

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

```
john@oho ~/LAB1
$ git restore hello.c
```

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

30. 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;
main.c.
gcc *.o -o hello;
```

← Note that only hello.c was recompiled and not

```
$ hello
```

```
Hello World Underlined!
-----
Hello World Underlined!
-----
Hello World Underlined!
-----
```

31. 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 7a82e8265773149293c111bf8d30fc7aaa790437 (HEAD -> master)

Author: John Dempsey <john.dempsey@csuci.edu>

Date: Wed Aug 24 17:37:49 2025 -0700

Baseline Code and Documentation

32. 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

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

\$ git commit -m "Updated hello.c to underline Hello World! output."

[master 543ffbf] Updated hello.c to underline Hello World! output.

2 files changed, 2 insertions(+), 1 deletion(-)

34. Check the status and repository.

\$ git status

On branch master

nothing to commit, working tree clean

\$ git log

commit 543ffbf224dc001c9923c14ca6b1d607401790 (HEAD -> master)

Author: John Dempsey <john.dempsey@csuci.edu>

Date: Wed Aug 24 7:48:38 2025 -0700

Updated hello.c to underline Hello World! output.

commit 7a82e8265773149293c111bf8d30fc7aaa790437

Author: John Dempsey <john.dempsey@csuci.edu>

Date: Wed Aug 24 7:37:49 2025-0700

Baseline Code and Documentation

\$ git ls-tree --full-tree --name-only -r HEAD

.gitignore

Makefile

hello.c
hello.exe
hello.txt
main.c

TO OBTAIN FULL CREDIT FOR THIS EXERCISE, PLEASE RUN THE FOLLOWING COMMANDS.

% script git.txt

% git status

% git log

% git ls-tree --full-tree --name-only -r HEAD

% exit

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