CPE 334 SOFTWARE ENGINEERING,
SOFTWARE CONFIGURATION MANAGEMENT LAB SHEET

DUE ON FRIDAY 22 SEPTEMBER AT NOON

This is not a group assignment. It can be done either with one partner or by yourself. If you do this by yourself, you will need 2 github accounts.

Updated 15 September 2023

There are many different software configuration management tools. In this lab we will be using GitHub, which is a very popular online development environment based on the **git** software configuration management system. **git** is open source SCM software created by Linus Torvalds, the original developer of Linux, to help his world-wide team manage changes to the Linux kernel (the core modules of the Linux OS). Read about the history of **git** here: https://en.wikipedia.org/wiki/Git

Do not confuse **GitHub** with **git**! GitHub is an online environment that provides SCM services using **git** plus other kinds of services (such as continuous integration and deployment). It is a commercial company that was bought by Microsoft in 2018. You can read about the history of GitHub here: https://en.wikipedia.org/wiki/GitHub There are other online environments that compete with GitHub, for instance GitLab (https://gitlab.com).

The **git** SCM is fundamentally a command-line-based system. One thing that GitHub does is offer a more user-friendly graphical UI. It also provides a platform for working with other developers who can be anywhere in the world.

There are also local (non-web-based) GUIs available for **git**.

Unlike some SCM systems, which are client-server based, **git** was designed for distributed software development which is common in open source projects. The most important difference from many client-server SCM tools (such as CVS, the Concurrent Version System) is that **git** relies much more on branching and merging. Each developer has a full copy of the source code repository, which is treated typically treated as a branch from the main line of development. When a developer is finished with her work, she first commits her changes to her local repository, then as a second operation, attempts to merge the changes into the main line of development. The software relies heavily on automated processes for conflict identification or resolution. (It is possible for multiple developers to all be working in the main line – the "origin master", but this is not common in practice. However, the lab exercise uses this simpler scenario.)

Another difference between **git** and CVS is that CVS stores source code in text format if possible, only recording the deltas (differences) between one version and the next (except for binary files under SCM control, such as images). In contrast, **git** works by maintaining snapshots that are entire files, not just initial files plus changes, as binary objects signed with unique hash values. This prevents tampering but also means that problems due to file corruption can be difficult to fix (so backing up the **git** repository is really important!).

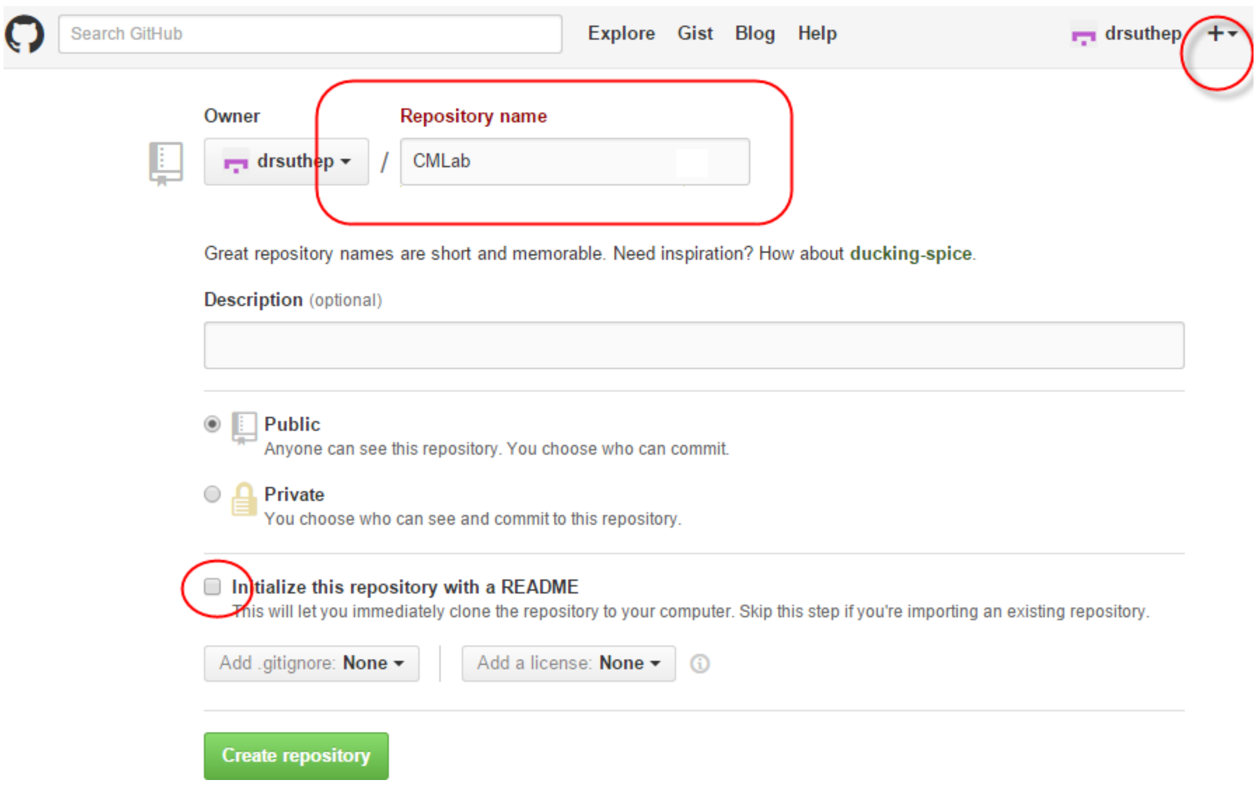
In this lab, you will practice using the basic capabilities of GitHub to manage some software. Follow the instructions below to complete this lab. Note that the screen shots in these instructions date from several years ago. You may find that the current user interface for both git and GitHub looks somewhat different. We decided not to update the screen shots, as a learning experience. In the real world, web applications change their organization frequently. You need to be able to adapt!

Installation and preparation

1. Sign up for a GitHub account https://github.com
2. If you are working on your own, create a second account with a different user name.
3. Download and install git software. GitHub won't work on your local computer if you don'tinstall git. Install Git for Windows, Mac or Linux depending on your computer's operating system. (See http://git-scm.com/downloads)
4. Run all git commands on the Git Bash shell (terminal). This provides a Linux-like command line environment if you are running the software on Windows or a Mac.

Example Workflow Steps

1. Create a blank repository in GitHub with no readme file. Print hub status.



Owner: drsuthep / Repository name: CMLab

Great repository names are short and memorable. Need inspiration? How about ducking-spice.

Description (optional)

● Public
Anyone can see this repository. You choose who can commit.

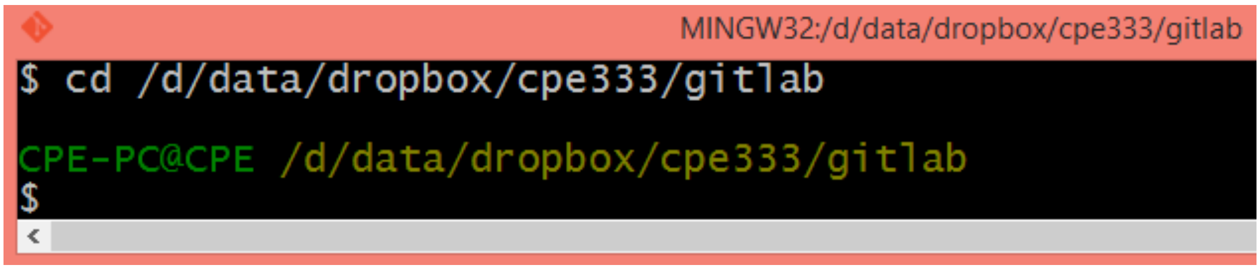
○ Private
You choose who can see and commit to this repository.

☐ Initialize this repository with a README
This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: None | Add a license: None

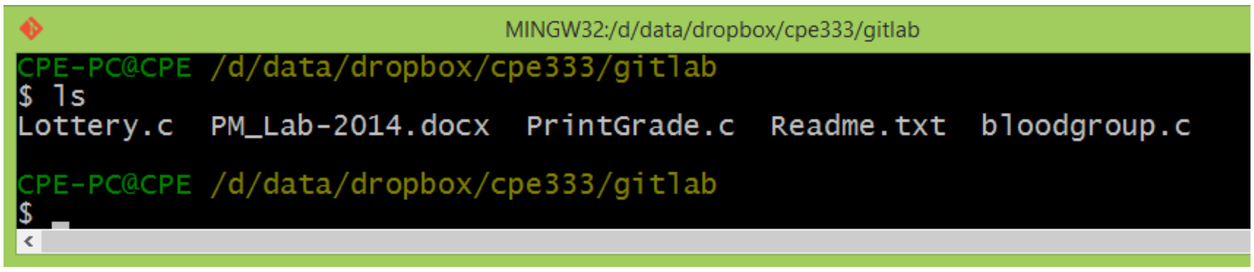
2. Unpack the zip file that includes the files you will be putting under git control, from the *src* directory in the zip. Move the src files into your work directory. Make a git repository in that work directory, then push it to the GitHub repository.

- In this example, the files are unpacked in ***d:/data/dropbox/cpe333/gitlab***. Go there first using 'cd' to change directory.



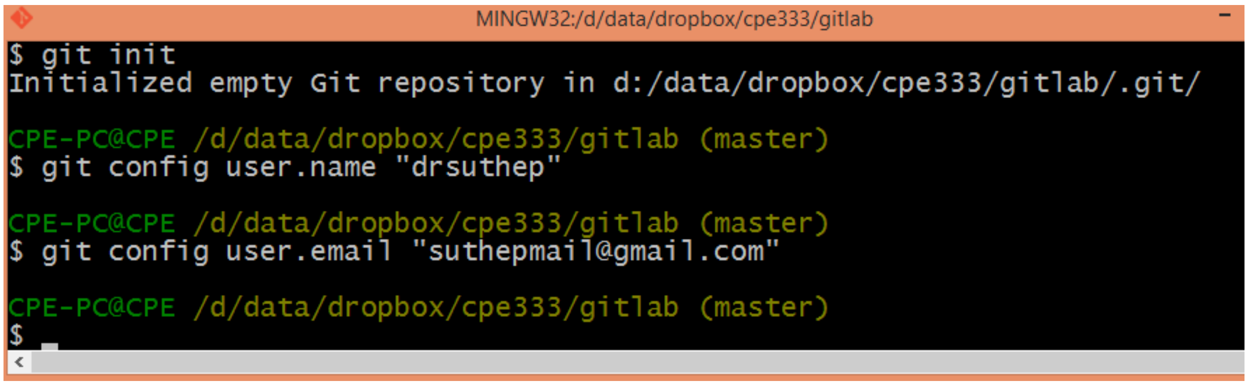
```
MINGW32:/d/data/dropbox/cpe333/gitlab
$ cd /d/data/dropbox/cpe333/gitlab

CPE-PC@CPE /d/data/dropbox/cpe333/gitlab
$
```

- List the contents using 'ls'



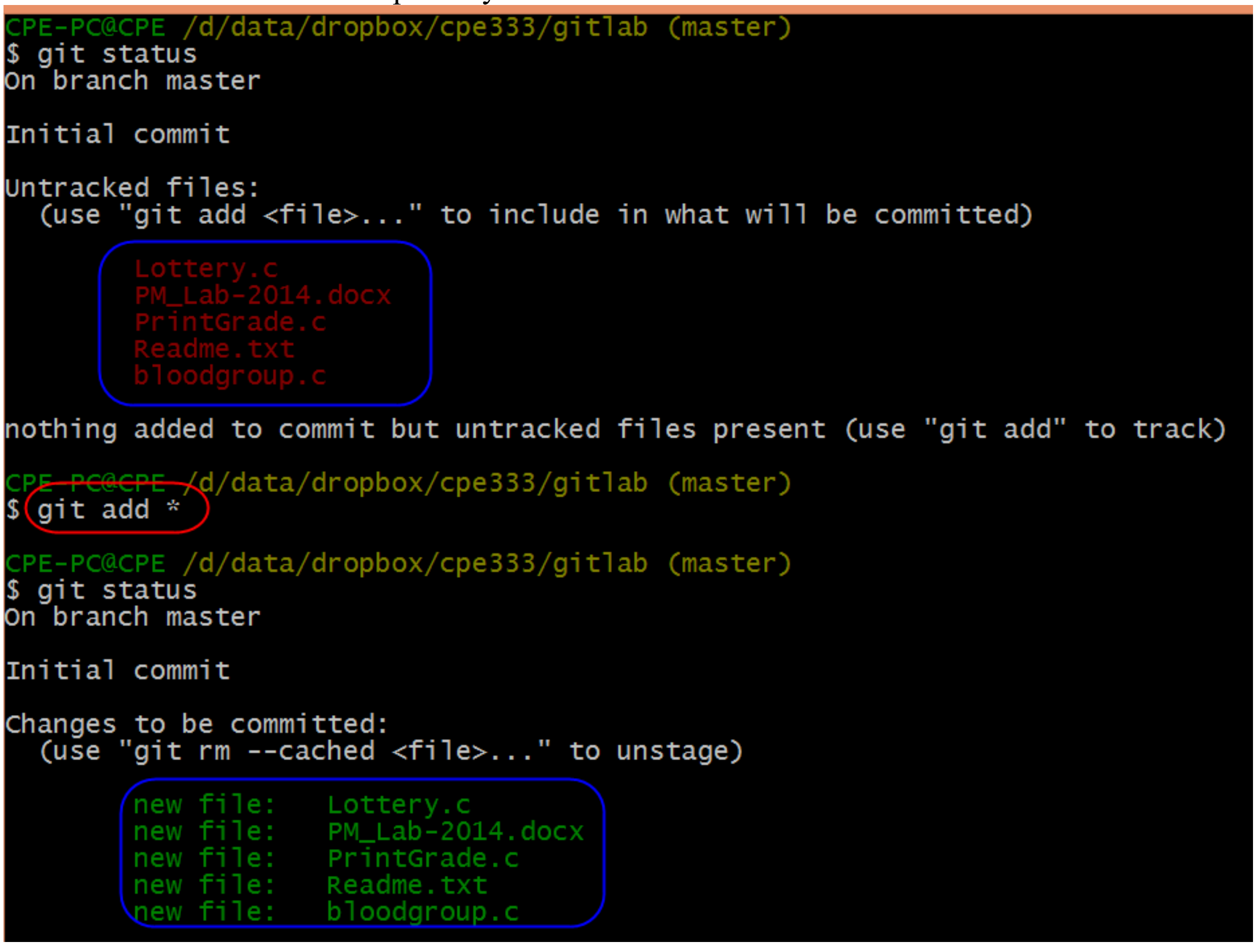
```
MINGW32:/d/data/dropbox/cpe333/gitlab
CPE-PC@CPE /d/data/dropbox/cpe333/gitlab
$ ls
Lottery.c  PM_Lab-2014.docx  PrintGrade.c  Readme.txt  bloodgroup.c

CPE-PC@CPE /d/data/dropbox/cpe333/gitlab
$ _
```

- Make this a .git repository using **git init**. This example sets the user name and email for this repository to"drsuthep" and "suthepmail@gmail.com"



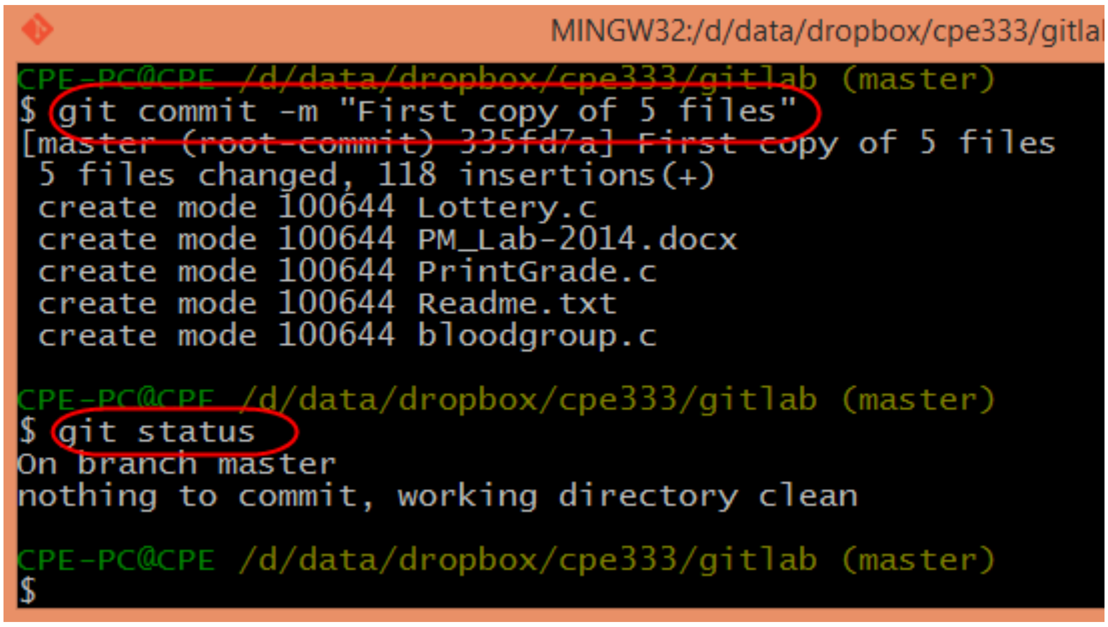
```
$ git init
Initialized empty Git repository in d:/data/dropbox/cpe333/gitlab/.git/

CPE-PC@CPE /d/data/dropbox/cpe333/gitlab (master)
$ git config user.name "drsuthep"

CPE-PC@CPE /d/data/dropbox/cpe333/gitlab (master)
$ git config user.email "suthepmail@gmail.com"

CPE-PC@CPE /d/data/dropbox/cpe333/gitlab (master)
$ _
```

- Add all files to the git repository using the **add** command. Then use the **commit** command to finalize the files to the local repository.



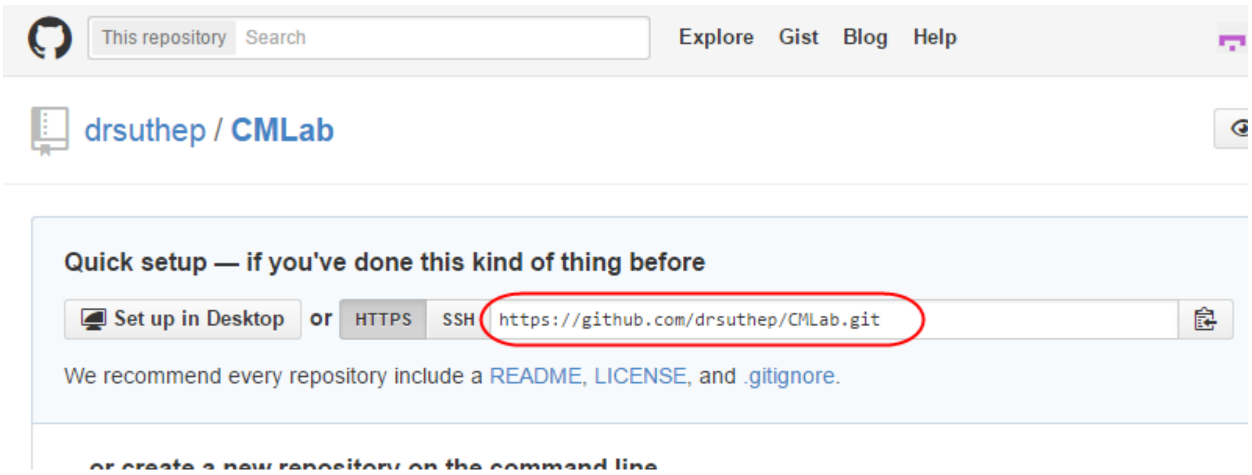
```
CPE-PC@CPE /d/data/dropbox/cpe333/gitlab (master)
$ git status
On branch master

Initial commit

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

        Lottery.c
        PM_Lab-2014.docx
        PrintGrade.c
        Readme.txt
        bloodgroup.c

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

CPE-PC@CPE /d/data/dropbox/cpe333/gitlab (master)
$ git add *

CPE-PC@CPE /d/data/dropbox/cpe333/gitlab (master)
$ git status
On branch master

Initial commit

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

        new file:   Lottery.c
        new file:   PM_Lab-2014.docx
        new file:   PrintGrade.c
        new file:   Readme.txt
        new file:   bloodgroup.c
```

- “Staged” means ready to be committed. Now we **commit** these changes with a comment as “First copy of 5 files”



```
CPE-PC@CPE /d/data/dropbox/cpe333/gitlab (master)
$ git commit -m "First copy of 5 files"
[master (root-commit) 335fd7a] First copy of 5 files
 5 files changed, 118 insertions(+)
 create mode 100644 Lottery.c
 create mode 100644 PM_Lab-2014.docx
 create mode 100644 PrintGrade.c
 create mode 100644 Readme.txt
 create mode 100644 bloodgroup.c

CPE-PC@CPE /d/data/dropbox/cpe333/gitlab (master)
$ git status
On branch master
nothing to commit, working directory clean

CPE-PC@CPE /d/data/dropbox/cpe333/gitlab (master)
$
```

- Log in to your GitHub account and create an online repository location for your repository.



Quick setup — if you've done this kind of thing before

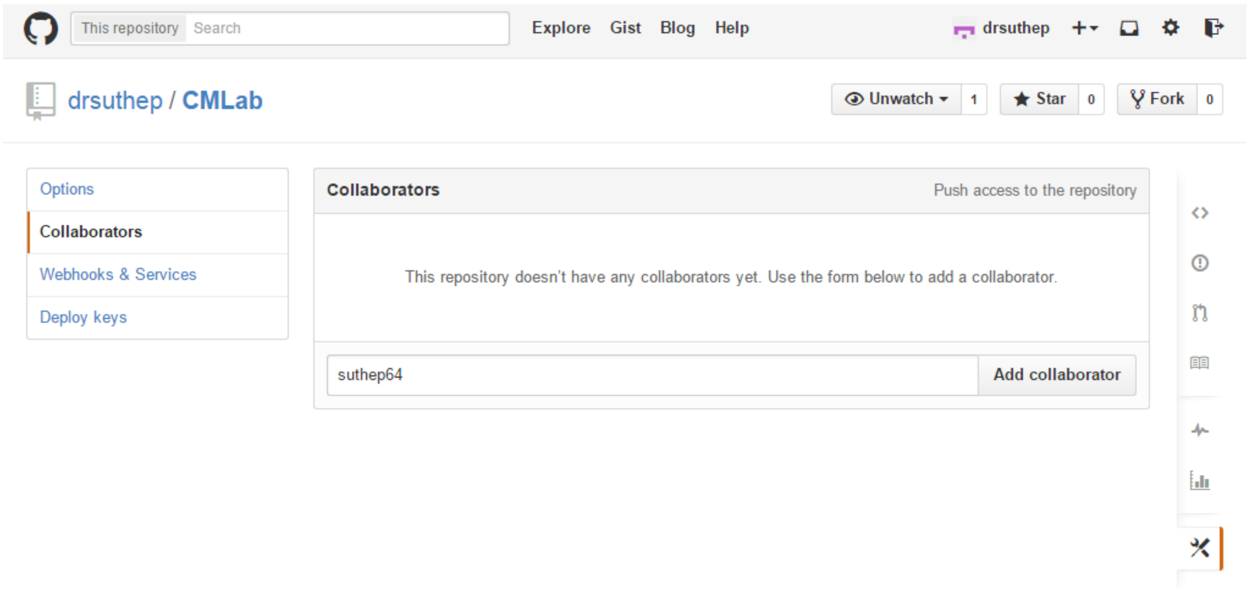
```
https://github.com/drsuthep/CMLab.git
```

We recommend every repository include a README, LICENSE, and .gitignore.

…or create a new repository on the command line

-

- Add a collaborator. If you are working in a pair, this will be your partner's username. Otherwise, use the username for the second GitHub account you created. We will call this collaborator "User B".



Collaborators

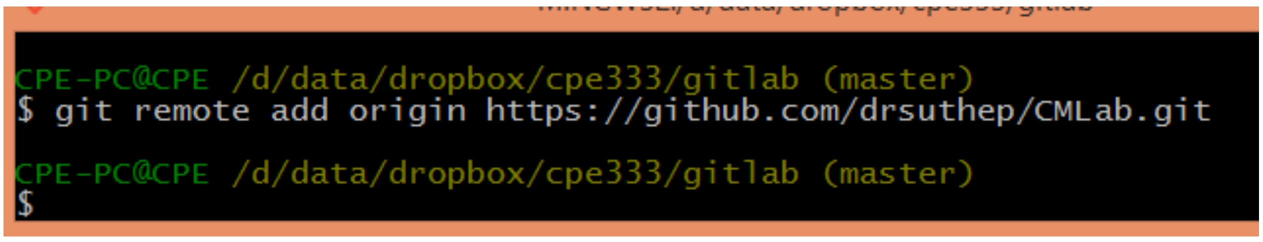
Push access to the repository

This repository doesn't have any collaborators yet. Use the form below to add a collaborator.

suthep64

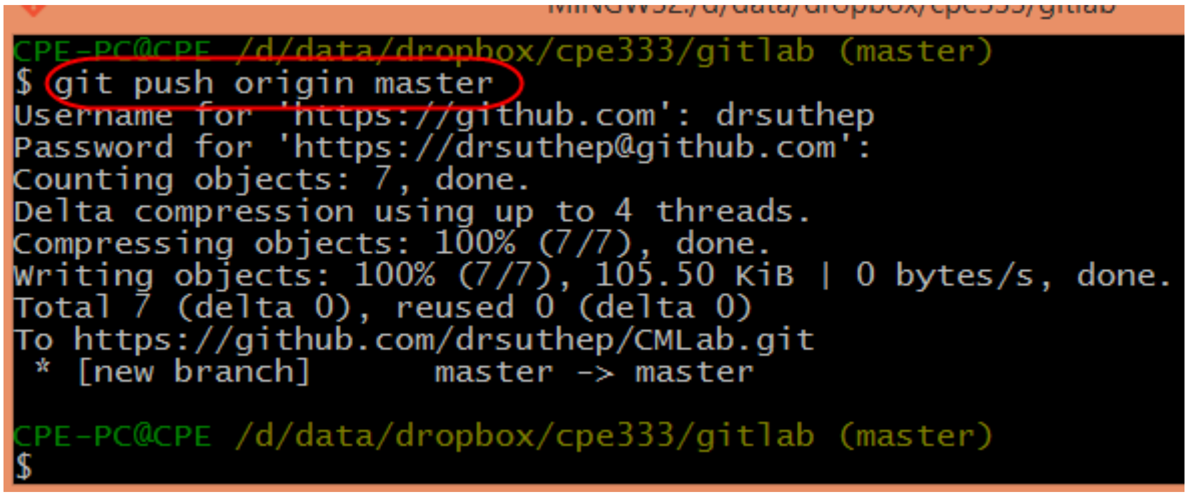
-

- Now link the remote repository to your local repository.



```
CPE-PC@CPE /d/data/dropbox/cpe333/gitlab (master)
$ git remote add origin https://github.com/drsuthep/CMLab.git

CPE-PC@CPE /d/data/dropbox/cpe333/gitlab (master)
$
```

- Next, push the local folder (called ***origin master***) to the GitHub repository using the **push** command.

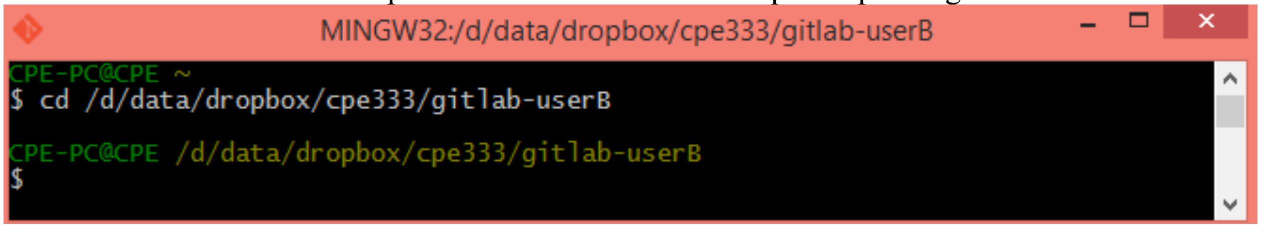


```
MINGW32:/d/data/dropbox/cpe333/gitlab
CPE-PC@CPE /d/data/dropbox/cpe333/gitlab (master)
$ git push origin master
Username for 'https://github.com': drsuthep
Password for 'https://drsuthep@github.com':
Counting objects: 7, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (7/7), done.
Writing objects: 100% (7/7), 105.50 KiB | 0 bytes/s, done.
Total 7 (delta 0), reused 0 (delta 0)
To https://github.com/drsuthep/CMLab.git
 * [new branch]      master -> master

CPE-PC@CPE /d/data/dropbox/cpe333/gitlab (master)
$
```

-

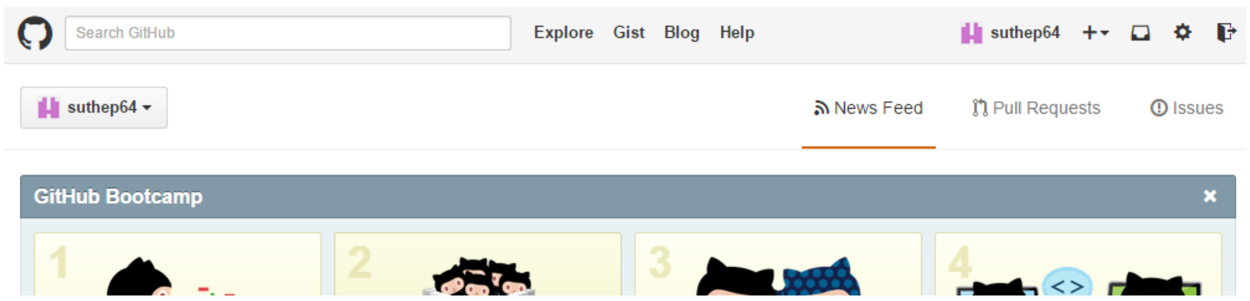
3. Now we will look at collaboration. User B will creates a copy of the online repository locally in another folder of his/her own computer, then print local statusUsers B.

- Let's assume on user B's computer it is in folder d:/data/dropbox/cpe333/gitlab-userB:

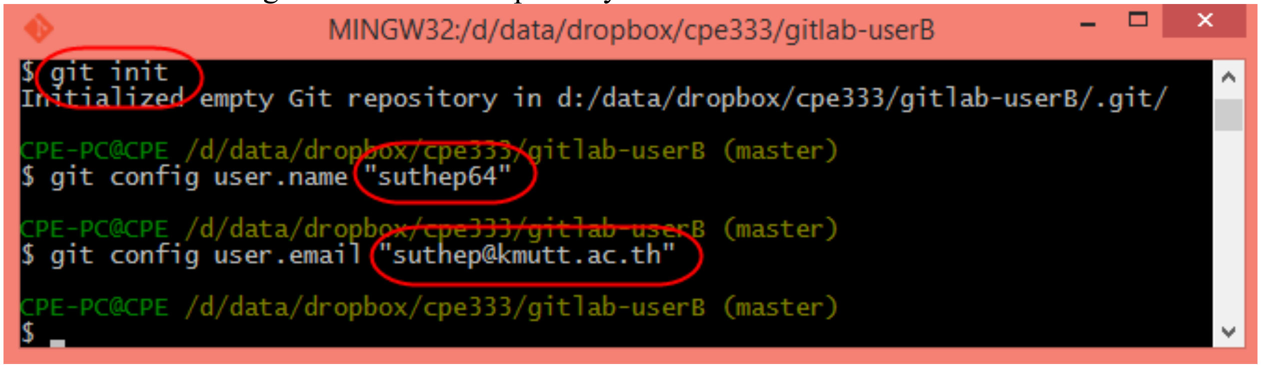


```
MINGW32:/d/data/dropbox/cpe333/gitlab-userB
CPE-PC@CPE ~
$ cd /d/data/dropbox/cpe333/gitlab-userB

CPE-PC@CPE /d/data/dropbox/cpe333/gitlab-userB
$
```

- In this example, the second user or user B is "suthep64" (You need 2 email accounts to simulate 2 users if you are doing the lab alone). Thisaccount has email of "suthep@kmutt.ac.th".

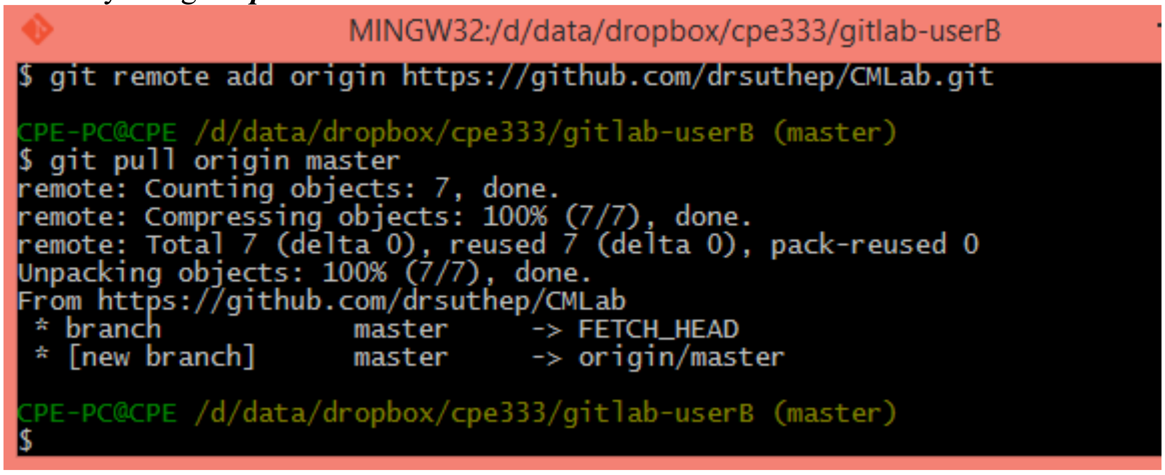


- Initialize the local repository at the current folder to create a .git repository and then set the user name and configurations for this repository:



```
MINGW32:/d/data/dropbox/cpe333/gitlab-userB
$ git init
Initialized empty Git repository in d:/data/dropbox/cpe333/gitlab-userB/.git/

CPE-PC@CPE /d/data/dropbox/cpe333/gitlab-userB (master)
$ git config user.name "suthep64"

CPE-PC@CPE /d/data/dropbox/cpe333/gitlab-userB (master)
$ git config user.email "suthep@kmutt.ac.th"

CPE-PC@CPE /d/data/dropbox/cpe333/gitlab-userB (master)
$
```

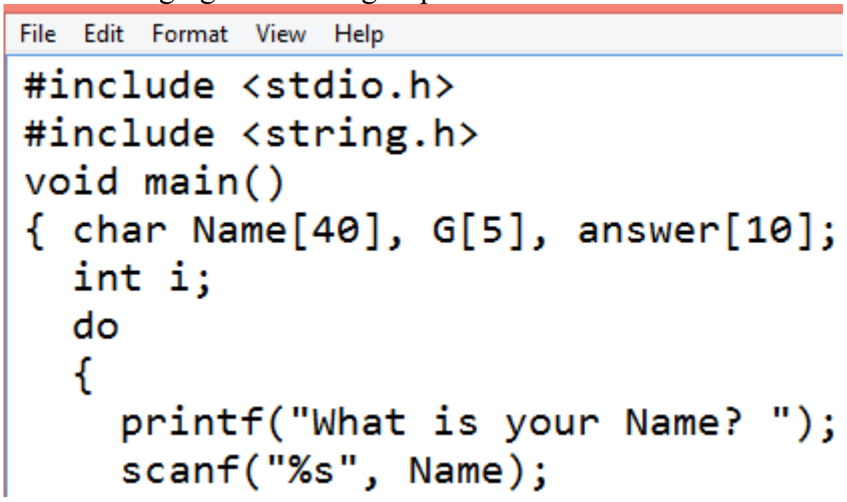
- Next, inform the local directory where the GitHub is located, and then pull the code into that directory using the ***pull*** command.



```
MINGW32:/d/data/dropbox/cpe333/gitlab-userB
$ git remote add origin https://github.com/drsuthep/CMLab.git

CPE-PC@CPE /d/data/dropbox/cpe333/gitlab-userB (master)
$ git pull origin master
remote: Counting objects: 7, done.
remote: Compressing objects: 100% (7/7), done.
remote: Total 7 (delta 0), reused 7 (delta 0), pack-reused 0
Unpacking objects: 100% (7/7), done.
From https://github.com/drsuthep/CMLab
 * branch            master     -> FETCH_HEAD
 * [new branch]      master     -> origin/master

CPE-PC@CPE /d/data/dropbox/cpe333/gitlab-userB (master)
$
```

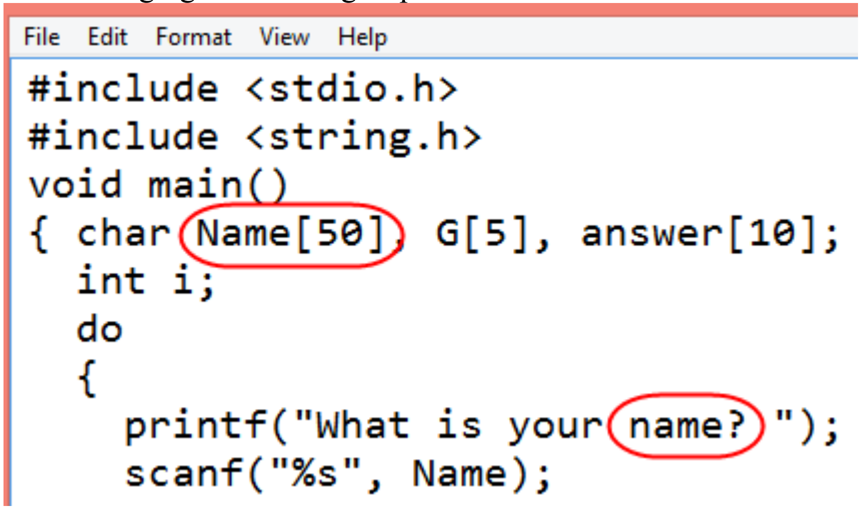
4. Go back to user A's directory. Change the code function in one source files. Print the local status of User A.

- Before changing file "bloodgroup.c":



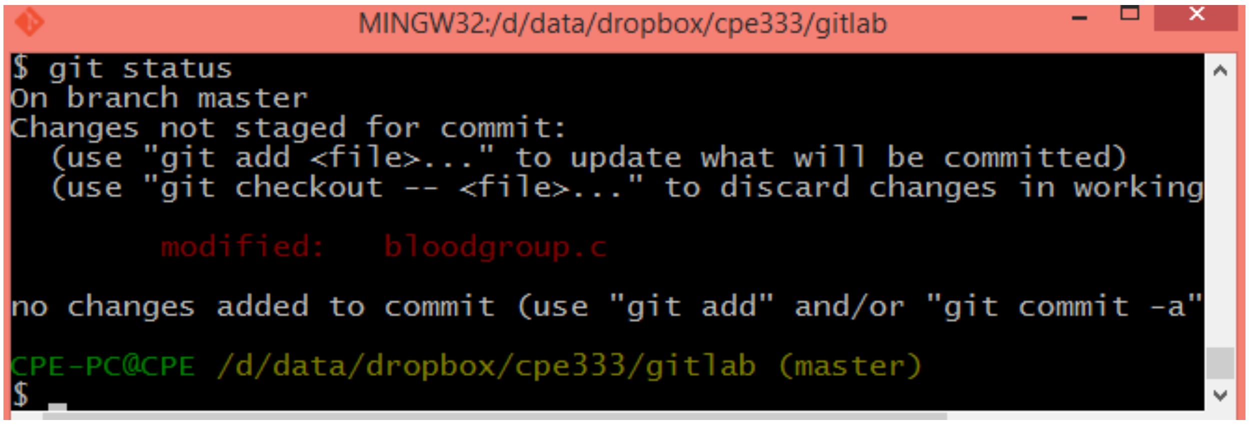
```
#include <stdio.h>
#include <string.h>
void main()
{ char Name[40], G[5], answer[10];
  int i;
  do
  {
    printf("What is your Name? ");
    scanf("%s", Name);
```

- After changing file "bloodgroup.c":



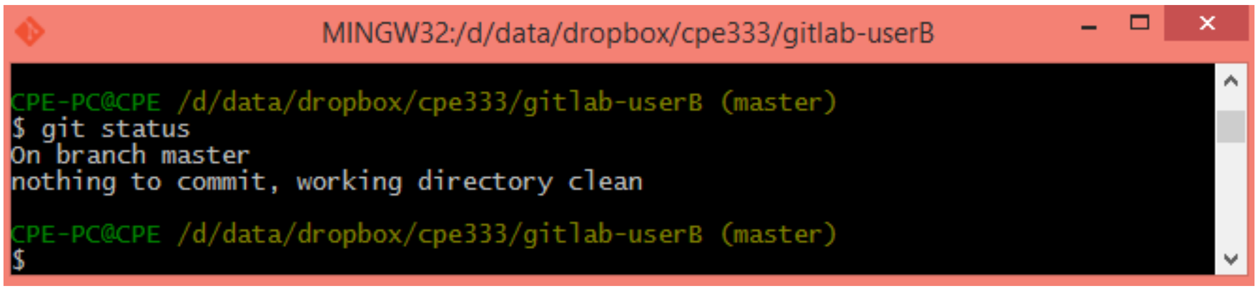
```
#include <stdio.h>
#include <string.h>
void main()
{ char Name[50], G[5], answer[10];
  int i;
  do
  {
    printf("What is your name? ");
    scanf("%s", Name);
```

- Print User A's status.



```
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working

        modified:   bloodgroup.c

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

CPE-PC@CPE /d/data/dropbox/cpe333/gitlab (master)
$
```

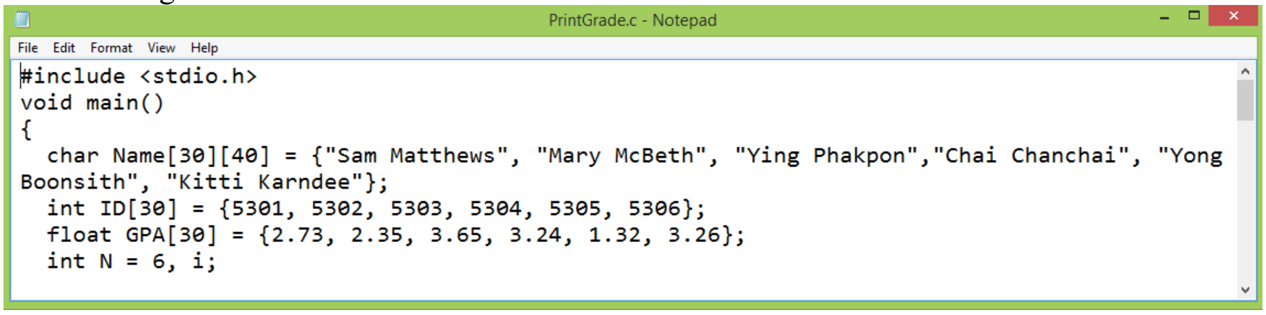
- Print User B's status. Of course there's no record of what User A has done.



```
CPE-PC@CPE /d/data/dropbox/cpe333/gitlab-userB (master)
$ git status
On branch master
nothing to commit, working directory clean

CPE-PC@CPE /d/data/dropbox/cpe333/gitlab-userB (master)
$
```

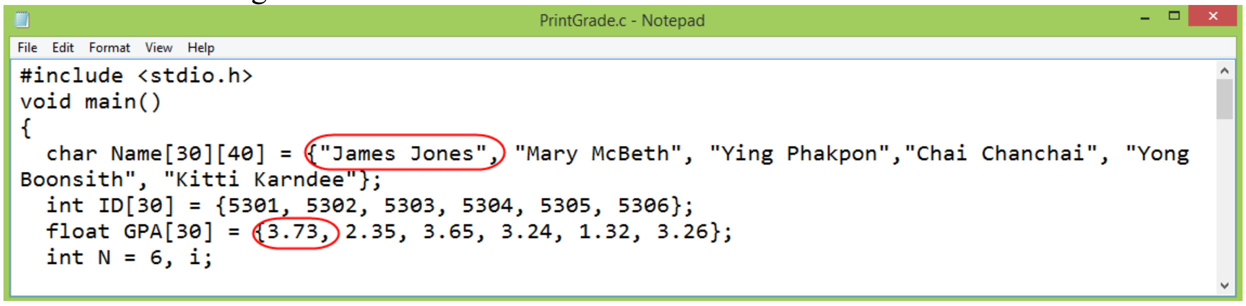
5. Now, as User B, change a function in another source file, then print local status for User B.

- Before change to "PrintGrade.c"



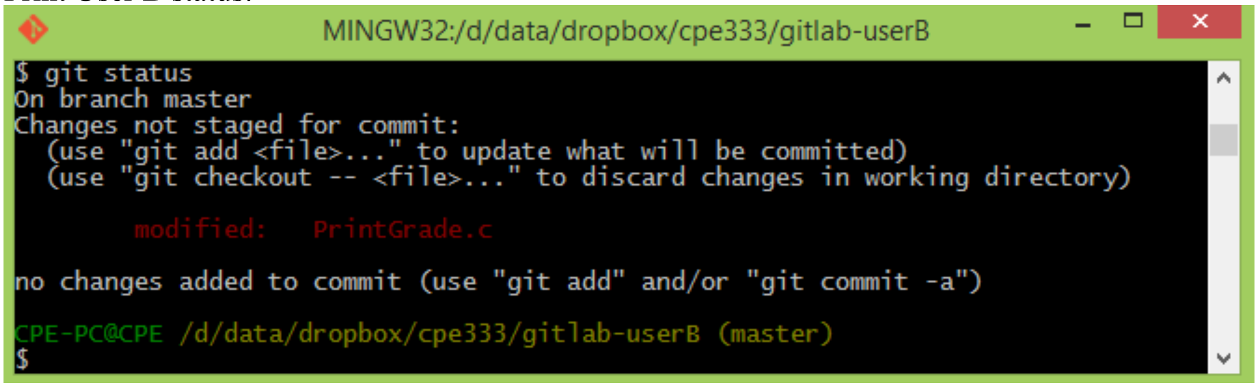
```
#include <stdio.h>
void main()
{
  char Name[30][40] = {"Sam Matthews", "Mary McBeth", "Ying Phakpon","Chai Chanchai", "Yong
Boonsith", "Kitti Karndee"};
  int ID[30] = {5301, 5302, 5303, 5304, 5305, 5306};
  float GPA[30] = {2.73, 2.35, 3.65, 3.24, 1.32, 3.26};
  int N = 6, i;
```

- User B makes change to "PrintGrade.c":



```
#include <stdio.h>
void main()
{
  char Name[30][40] = {"James Jones", "Mary McBeth", "Ying Phakpon","Chai Chanchai", "Yong
Boonsith", "Kitti Karndee"};
  int ID[30] = {5301, 5302, 5303, 5304, 5305, 5306};
  float GPA[30] = {3.73, 2.35, 3.65, 3.24, 1.32, 3.26};
  int N = 6, i;
```

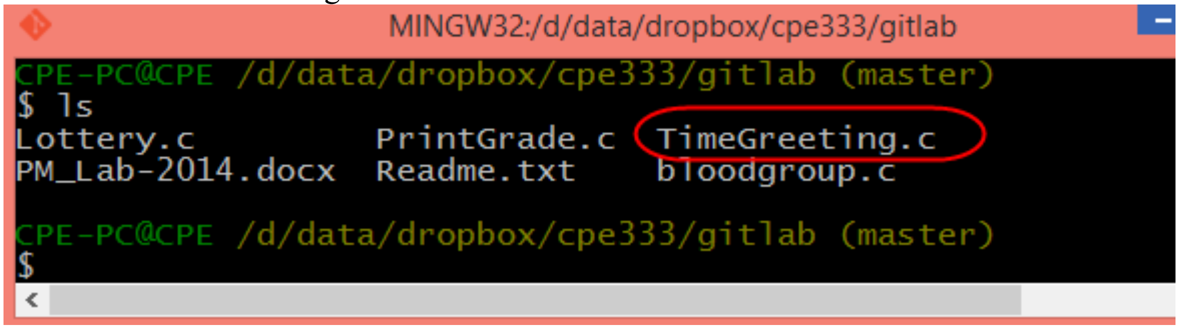
- Print User B status:



```
MINGW32:/d/data/dropbox/cpe333/gitlab-userB
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   PrintGrade.c

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

CPE-PC@CPE /d/data/dropbox/cpe333/gitlab-userB (master)
$
```

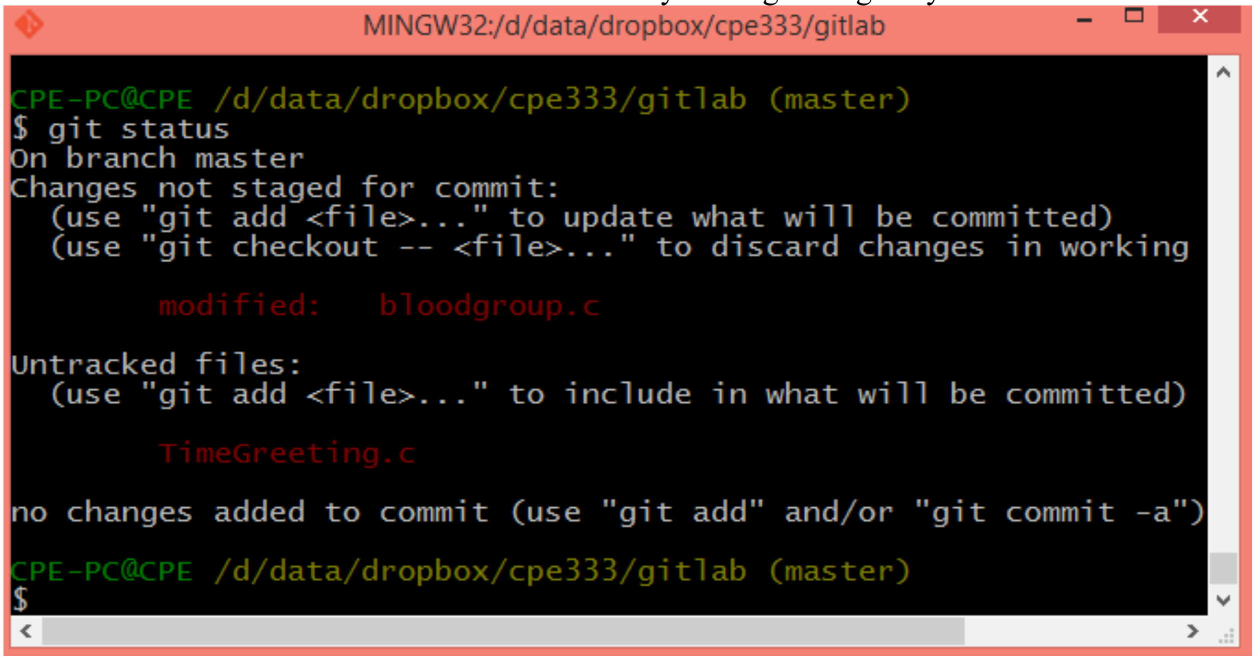
6. Go back to User A's directory. Create a new C program called TimeGreeting.c. (If you are following htis workflow, you can copy this from the *NEW* subdirectory created when you unzip the lab files.) Do not put into Git yet. Print local status User A.

- A new file "TimeGreeting.c" was added to user A's folder.



```
MINGW32:/d/data/dropbox/cpe333/gitlab
CPE-PC@CPE /d/data/dropbox/cpe333/gitlab (master)
$ ls
Lottery.c          PrintGrade.c  TimeGreeting.c
PM_Lab-2014.docx   Readme.txt    bloodgroup.c

CPE-PC@CPE /d/data/dropbox/cpe333/gitlab (master)
$
```

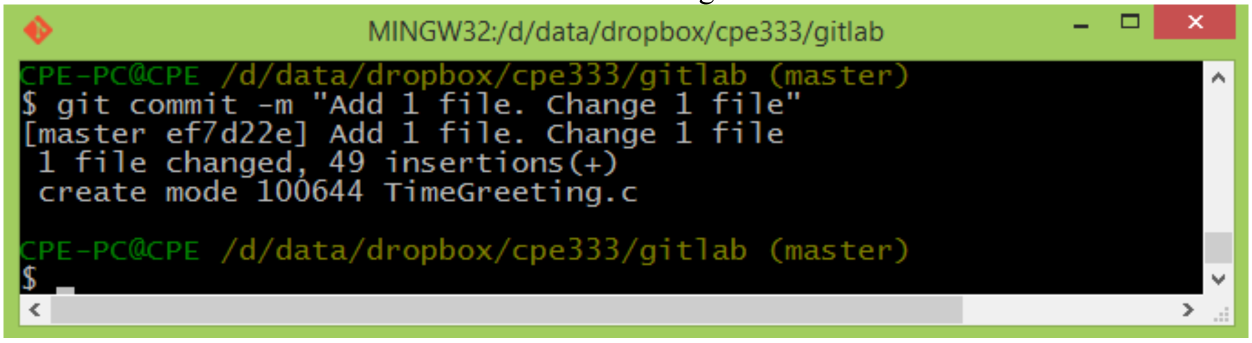
- The new status. Git knows there is a file that is not yet being managed by the SCM.



```
MINGW32:/d/data/dropbox/cpe333/gitlab

CPE-PC@CPE /d/data/dropbox/cpe333/gitlab (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working

        modified:   bloodgroup.c

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

        TimeGreeting.c

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

CPE-PC@CPE /d/data/dropbox/cpe333/gitlab (master)
$
```

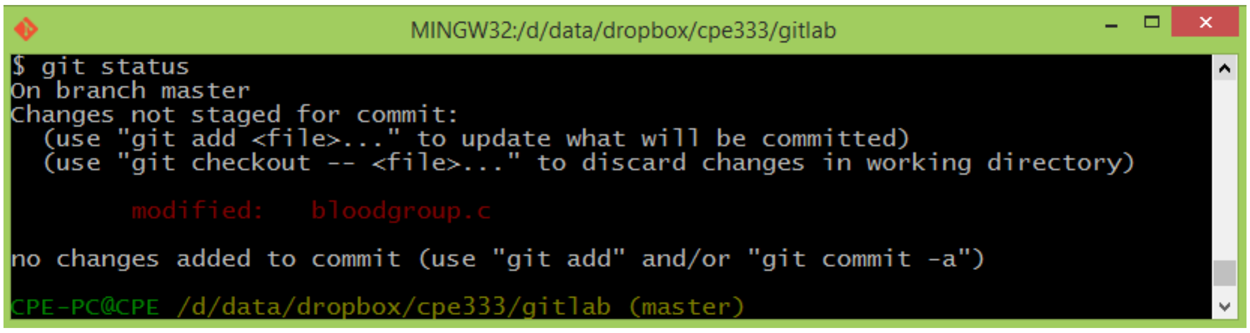
7. User A puts TimeGreeting.c under Git control using **git add**. Then User A commits changes (note that changed files are bloodgroup.c and TimeGreeting.c). Print local status for User A.

- Usesr A commits with the comment "Add 1 file. Change 1 file"

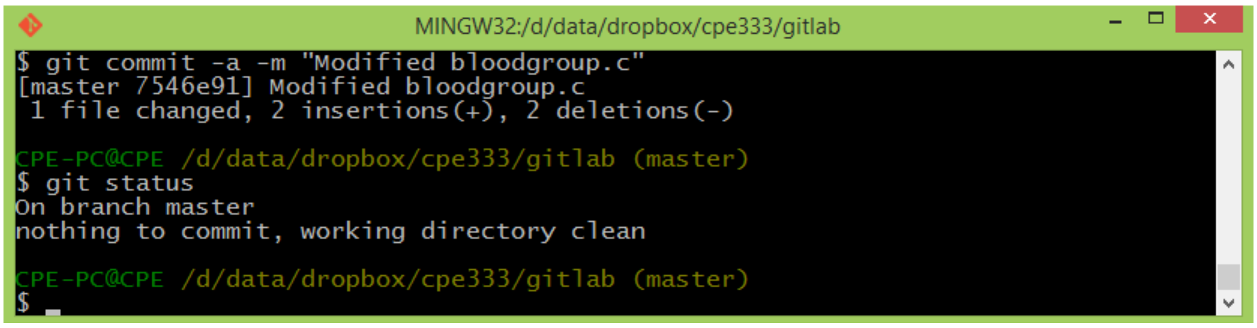


```
CPE-PC@CPE /d/data/dropbox/cpe333/gitlab (master)
$ git commit -m "Add 1 file. Change 1 file"
[master ef7d22e] Add 1 file. Change 1 file
 1 file changed, 49 insertions(+)
 create mode 100644 TimeGreeting.c

CPE-PC@CPE /d/data/dropbox/cpe333/gitlab (master)
$
```

- The git status shows only one file changed. This is because User A forgot to add bloodgroup.c. You can add and commit at once by using "git commit –a" or otherwise you do in two steps: 1) git add bloodgroup.c; 2) git commit –m "Modified bloodgroup.c".

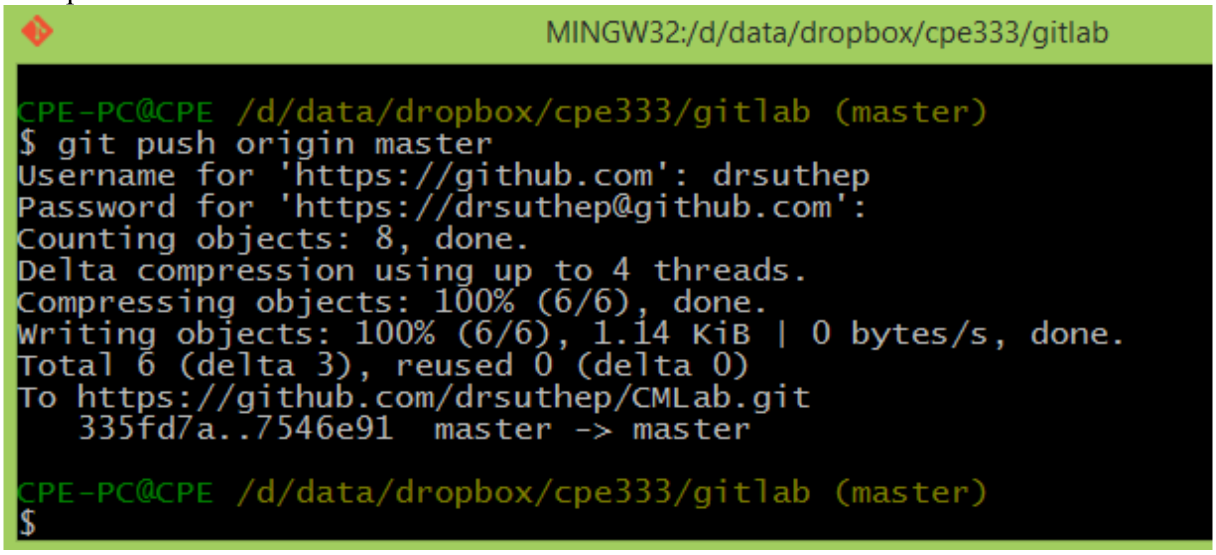


```
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   bloodgroup.c

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

CPE-PC@CPE /d/data/dropbox/cpe333/gitlab (master)
```



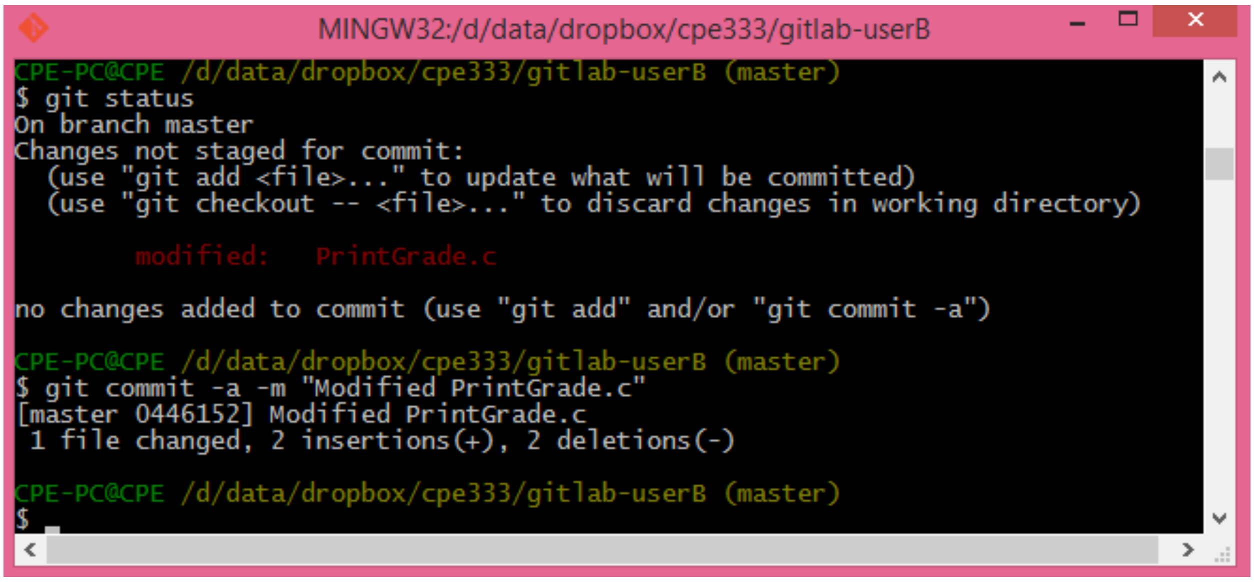
```
$ git commit -a -m "Modified bloodgroup.c"
[master 7546e91] Modified bloodgroup.c
 1 file changed, 2 insertions(+), 2 deletions(-)

CPE-PC@CPE /d/data/dropbox/cpe333/gitlab (master)
$ git status
On branch master
nothing to commit, working directory clean

CPE-PC@CPE /d/data/dropbox/cpe333/gitlab (master)
$
```

8. Having committed changes to the local repository, User A pushes changes to the GitHub repository, then prints User A's status.



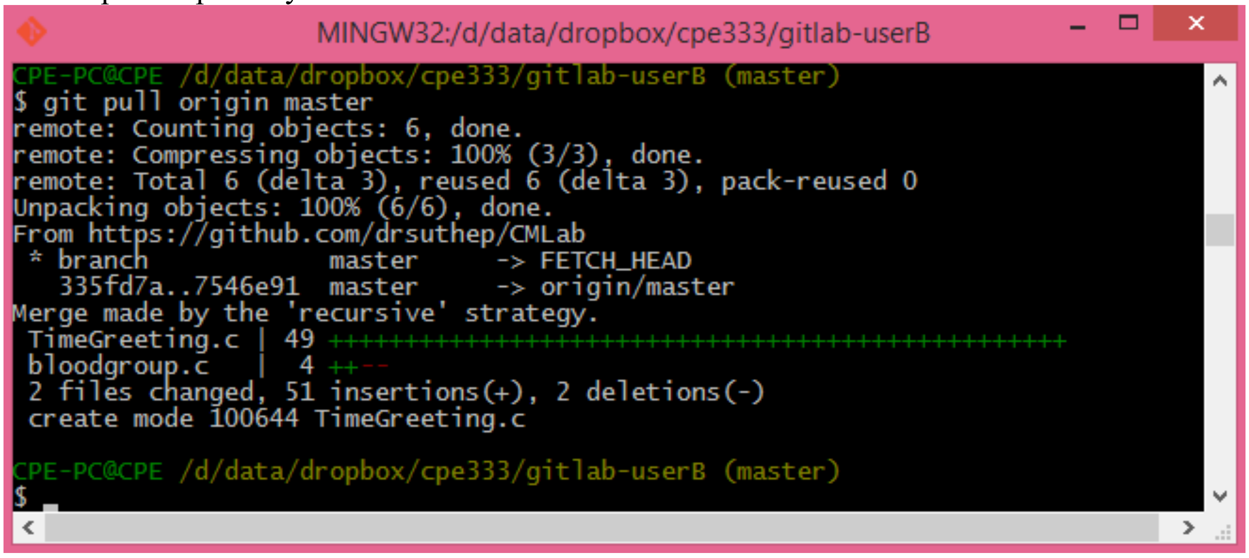
```
CPE-PC@CPE /d/data/dropbox/cpe333/gitlab (master)
$ git push origin master
Username for 'https://github.com': drsuthep
Password for 'https://drsuthep@github.com':
Counting objects: 8, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (6/6), done.
Writing objects: 100% (6/6), 1.14 KiB | 0 bytes/s, done.
Total 6 (delta 3), reused 0 (delta 0)
To https://github.com/drsuthep/CMLab.git
   335fd7a..7546e91  master -> master

CPE-PC@CPE /d/data/dropbox/cpe333/gitlab (master)
$
```

9. Going back to User B's environment, this user adds the changes to PrintGrade.c and commits in a single step, then prints local status; note that the changes have not yet been pushed to the central repository.



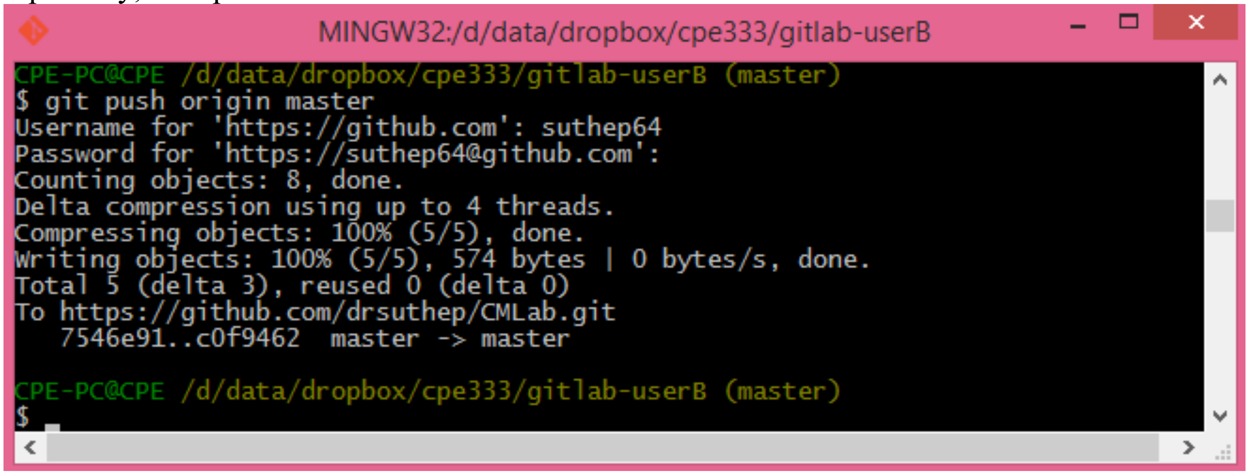
```
CPE-PC@CPE /d/data/dropbox/cpe333/gitlab-userB (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   PrintGrade.c

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

CPE-PC@CPE /d/data/dropbox/cpe333/gitlab-userB (master)
$ git commit -a -m "Modified PrintGrade.c"
[master 0446152] Modified PrintGrade.c
 1 file changed, 2 insertions(+), 2 deletions(-)

CPE-PC@CPE /d/data/dropbox/cpe333/gitlab-userB (master)
$
```

10. User B wonders what other collaborators have been doing. To get an up-to-date version of all code, User B pulls repository from GitHub.



```
CPE-PC@CPE /d/data/dropbox/cpe333/gitlab-userB (master)
$ git pull origin master
remote: Counting objects: 6, done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 6 (delta 3), reused 6 (delta 3), pack-reused 0
Unpacking objects: 100% (6/6), done.
From https://github.com/drsuthep/CMLab
 * branch            master     -> FETCH_HEAD
   335fd7a..7546e91  master     -> origin/master
Merge made by the 'recursive' strategy.
 TimeGreeting.c | 49 +++++++++++++++++++++++++++++++++++++++++++++++++
 bloodgroup.c   |  4 ++--
 2 files changed, 51 insertions(+), 2 deletions(-)
 create mode 100644 TimeGreeting.c

CPE-PC@CPE /d/data/dropbox/cpe333/gitlab-userB (master)
$
```

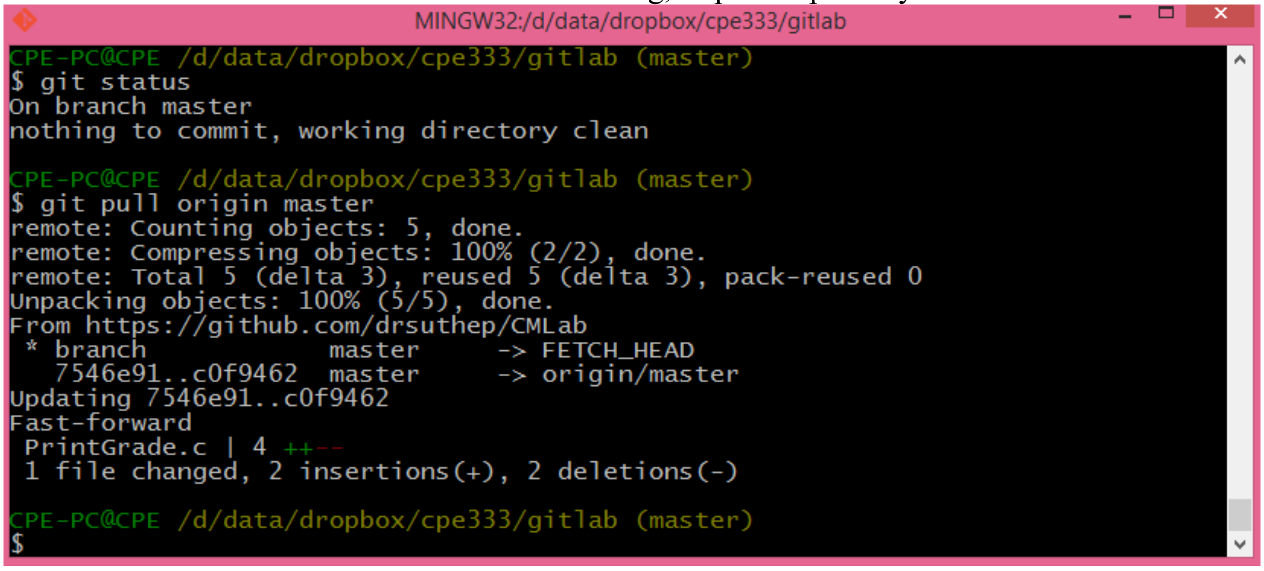
11. Now User B's local copy includes other people's changes. So User B pushes her changes to GitHub repository, then prints the status.



MINGW32:/d/data/dropbox/cpe333/gitlab-userB

```
CPE-PC@CPE /d/data/dropbox/cpe333/gitlab-userB (master)
$ git push origin master
Username for 'https://github.com': suthep64
Password for 'https://suthep64@github.com':
Counting objects: 8, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (5/5), done.
Writing objects: 100% (5/5), 574 bytes | 0 bytes/s, done.
Total 5 (delta 3), reused 0 (delta 0)
To https://github.com/drsuthep/CMLab.git
   7546e91..c0f9462  master -> master

CPE-PC@CPE /d/data/dropbox/cpe333/gitlab-userB (master)
$ _
```

12. User A wonders what collaborators have been doing, so pulls repository from GitHub.



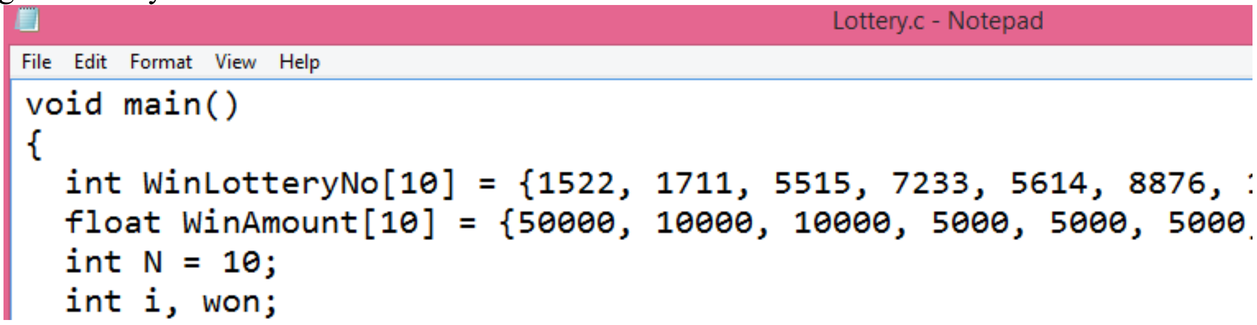
MINGW32:/d/data/dropbox/cpe333/gitlab

```
CPE-PC@CPE /d/data/dropbox/cpe333/gitlab (master)
$ git status
On branch master
nothing to commit, working directory clean

CPE-PC@CPE /d/data/dropbox/cpe333/gitlab (master)
$ git pull origin master
remote: Counting objects: 5, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 5 (delta 3), reused 5 (delta 3), pack-reused 0
Unpacking objects: 100% (5/5), done.
From https://github.com/drsuthep/CMLab
 * branch            master     -> FETCH_HEAD
   7546e91..c0f9462  master     -> origin/master
Updating 7546e91..c0f9462
Fast-forward
 PrintGrade.c | 4 ++--
 1 file changed, 2 insertions(+), 2 deletions(-)

CPE-PC@CPE /d/data/dropbox/cpe333/gitlab (master)
$
```

- **All 3 repositories have same copy now.**

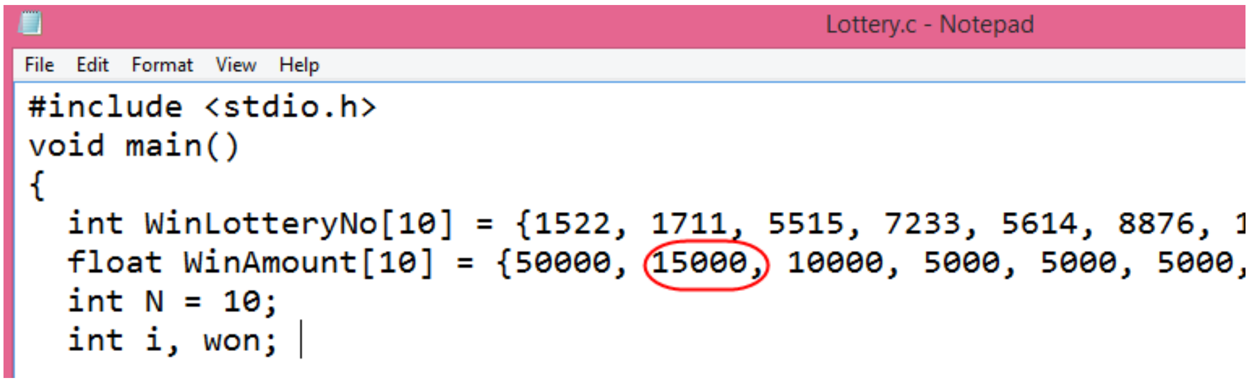
13. However, we have not yet looked at the common situation where two users make changes to the same source file. Suppose User A changes Lottery.c and User B also changes the same lines in Lottery.c, but with different content. Both add andcommit changes in one statement. Print status of User A and User B.

Original Lottery.c file:



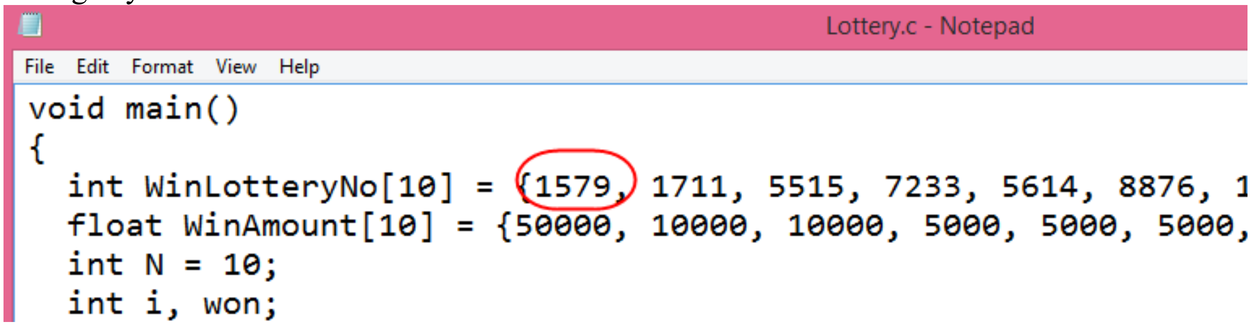
```
void main()
{
  int WinLotteryNo[10] = {1522, 1711, 5515, 7233, 5614, 8876, 
  float WinAmount[10] = {50000, 10000, 10000, 5000, 5000, 5000
  int N = 10;
  int i, won;
```

- Change by User A:



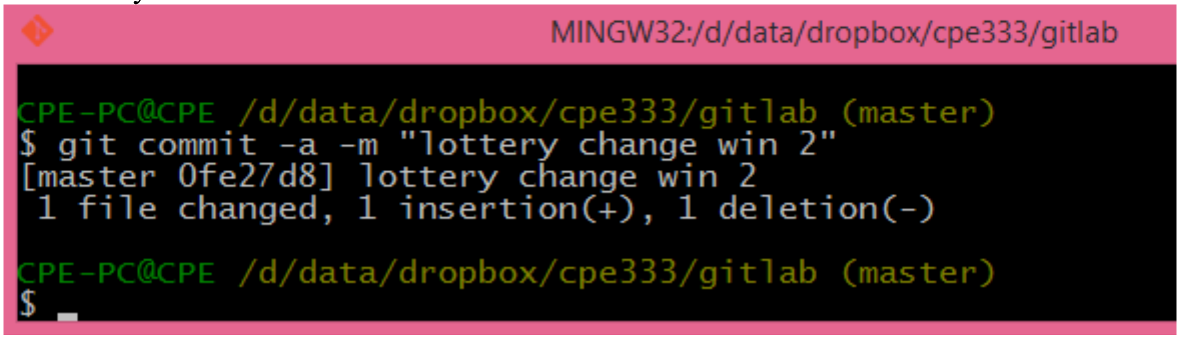
```
#include <stdio.h>
void main()
{
  int WinLotteryNo[10] = {1522, 1711, 5515, 7233, 5614, 8876, 1
  float WinAmount[10] = {50000, 15000, 10000, 5000, 5000, 5000,
  int N = 10;
  int i, won;
```

- Change by User B:



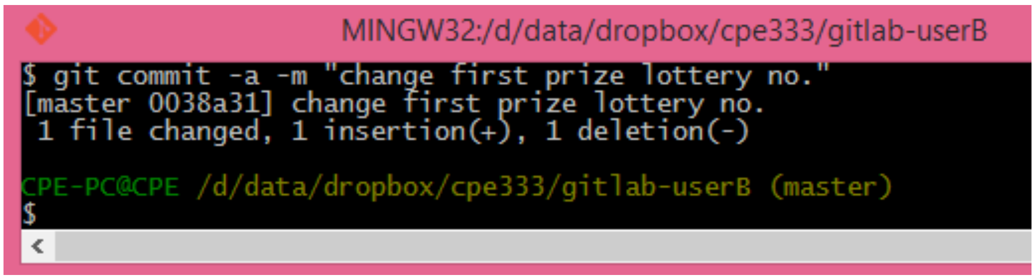
```
void main()
{
  int WinLotteryNo[10] = {1579, 1711, 5515, 7233, 5614, 8876, 1
  float WinAmount[10] = {50000, 10000, 10000, 5000, 5000, 5000,
  int N = 10;
  int i, won;
```

- Commit by User A:



```
CPE-PC@CPE /d/data/dropbox/cpe333/gitlab (master)
$ git commit -a -m "lottery change win 2"
[master 0fe27d8] lottery change win 2
 1 file changed, 1 insertion(+), 1 deletion(-)

CPE-PC@CPE /d/data/dropbox/cpe333/gitlab (master)
$ _
```

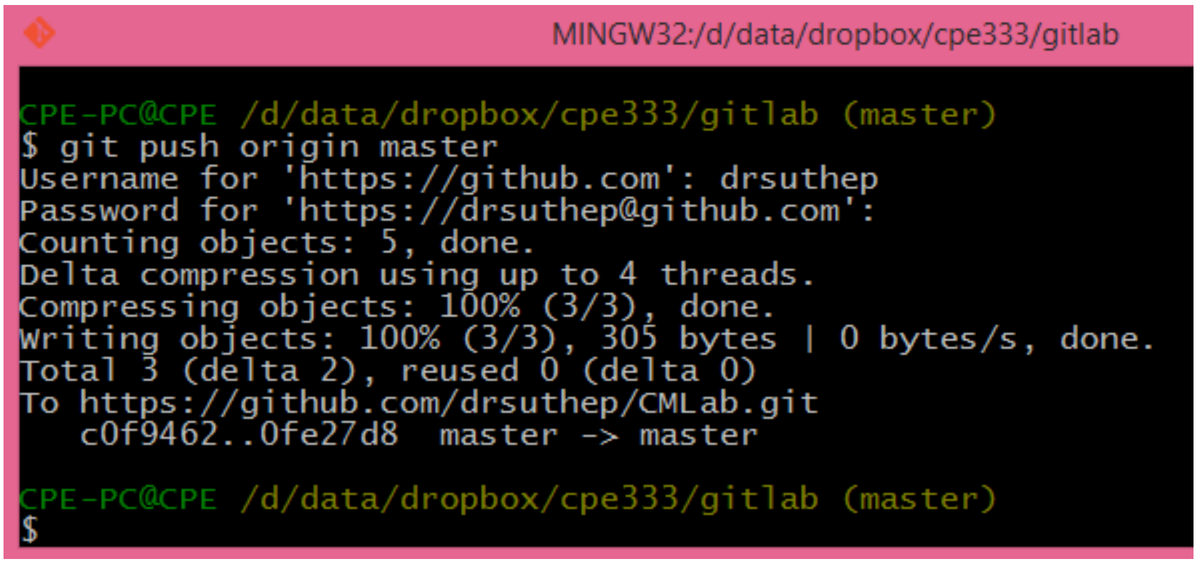
- Commit by User B:


MINGW32:/d/data/dropbox/cpe333/gitlab-userB

```
$ git commit -a -m "change first prize lottery no."
[master 0038a31] change first prize lottery no.
 1 file changed, 1 insertion(+), 1 deletion(-)

CPE-PC@CPE /d/data/dropbox/cpe333/gitlab-userB (master)
$
```

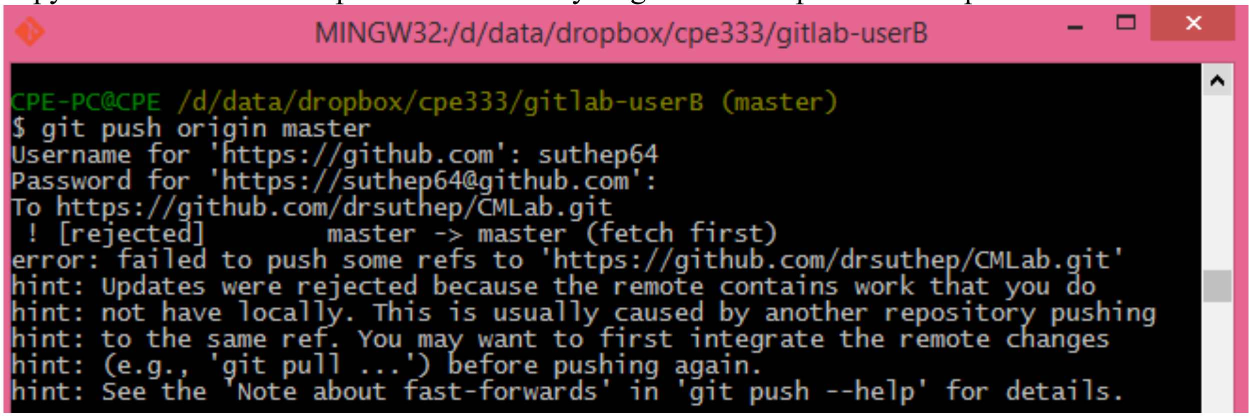
14. They both push changes to GitHub Repository.

- User A:


MINGW32:/d/data/dropbox/cpe333/gitlab

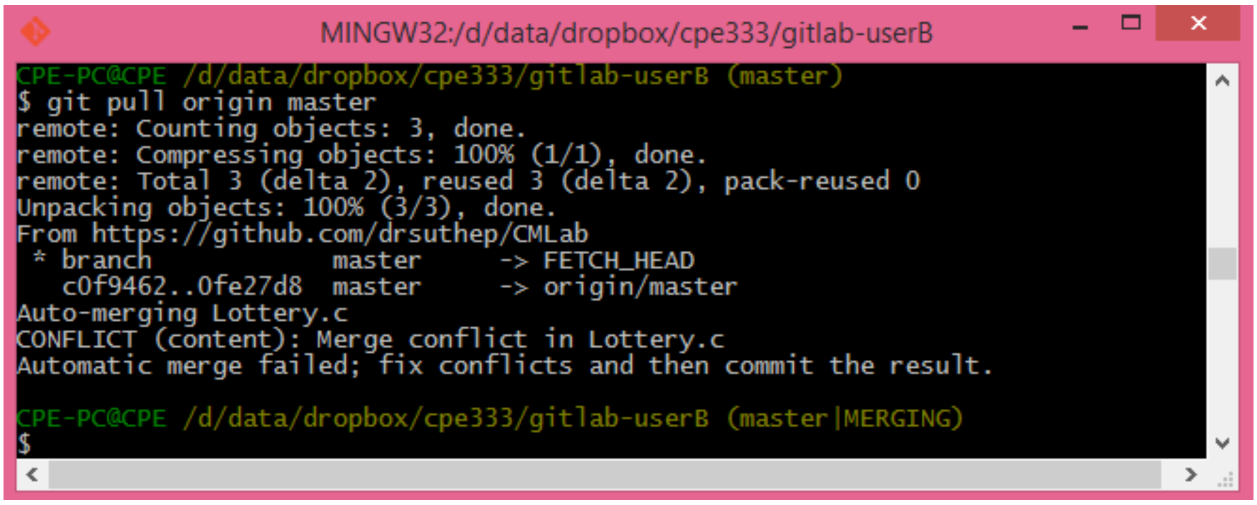
```
CPE-PC@CPE /d/data/dropbox/cpe333/gitlab (master)
$ git push origin master
Username for 'https://github.com': drsuthep
Password for 'https://drsuthep@github.com':
Counting objects: 5, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 305 bytes | 0 bytes/s, done.
Total 3 (delta 2), reused 0 (delta 0)
To https://github.com/drsuthep/CMLab.git
   c0f9462..0fe27d8  master -> master

CPE-PC@CPE /d/data/dropbox/cpe333/gitlab (master)
$
```

- User B cannot push because some changes exist in the master copy that are not in User B's local copy. Hence User B must pull first. It's always a good idea to pull andthen push.


MINGW32:/d/data/dropbox/cpe333/gitlab-userB

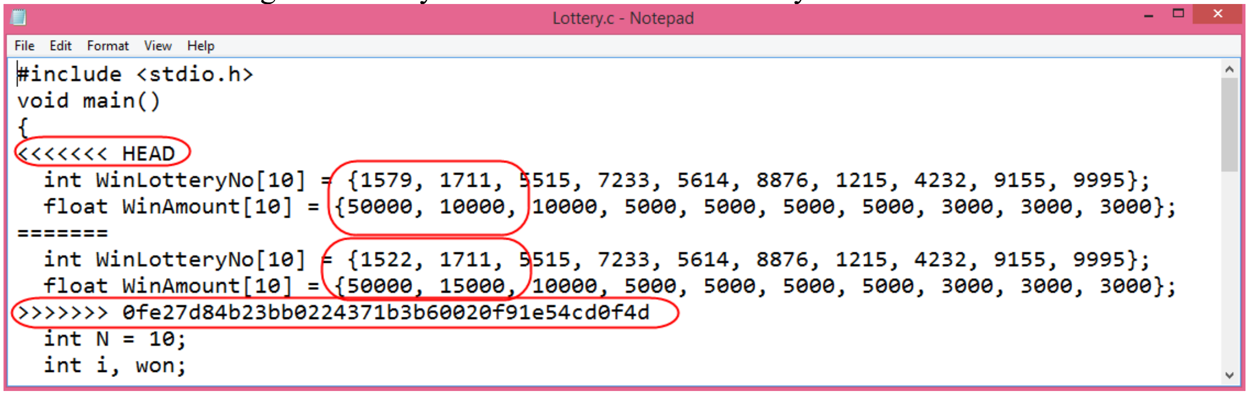
```
CPE-PC@CPE /d/data/dropbox/cpe333/gitlab-userB (master)
$ git push origin master
Username for 'https://github.com': suthep64
Password for 'https://suthep64@github.com':
To https://github.com/drsuthep/CMLab.git
 ! [rejected]        master -> master (fetch first)
error: failed to push some refs to 'https://github.com/drsuthep/CMLab.git'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

- When the pull executes, **git** discovers there is a conflict, that is, incompatible changes by two users in the same area of the same file.



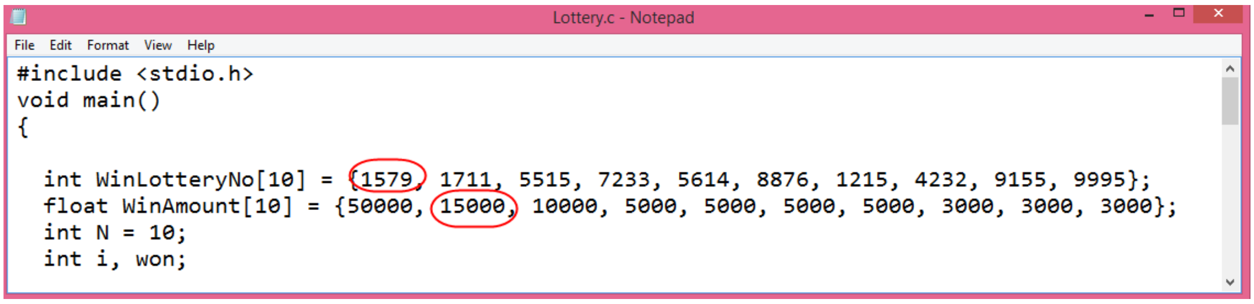
```
CPE-PC@CPE /d/data/dropbox/cpe333/gitlab-userB (master)
$ git pull origin master
remote: Counting objects: 3, done.
remote: Compressing objects: 100% (1/1), done.
remote: Total 3 (delta 2), reused 3 (delta 2), pack-reused 0
Unpacking objects: 100% (3/3), done.
From https://github.com/drsuthep/CMLab
 * branch            master     -> FETCH_HEAD
   c0f9462..0fe27d8  master     -> origin/master
Auto-merging Lottery.c
CONFLICT (content): Merge conflict in Lottery.c
Automatic merge failed; fix conflicts and then commit the result.

CPE-PC@CPE /d/data/dropbox/cpe333/gitlab-userB (master|MERGING)
$
```

- The two sets of changes in Lottery.c must be resolved manually:



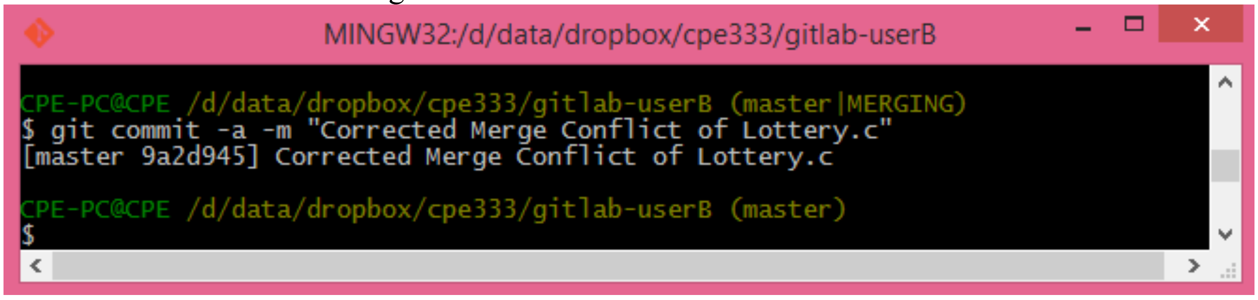
```
#include <stdio.h>
void main()
{
<<<<<<< HEAD
  int WinLotteryNo[10] = {1579, 1711, 5515, 7233, 5614, 8876, 1215, 4232, 9155, 9995};
  float WinAmount[10] = {50000, 10000, 10000, 5000, 5000, 5000, 5000, 3000, 3000, 3000};
=======
  int WinLotteryNo[10] = {1522, 1711, 5515, 7233, 5614, 8876, 1215, 4232, 9155, 9995};
  float WinAmount[10] = {50000, 15000, 10000, 5000, 5000, 5000, 5000, 3000, 3000, 3000};
>>>>>>> 0fe27d84b23bb0224371b3b60020f91e54cd0f4d
  int N = 10;
  int i, won;
```

- User B fixes the conflict as follows and then saves the file.



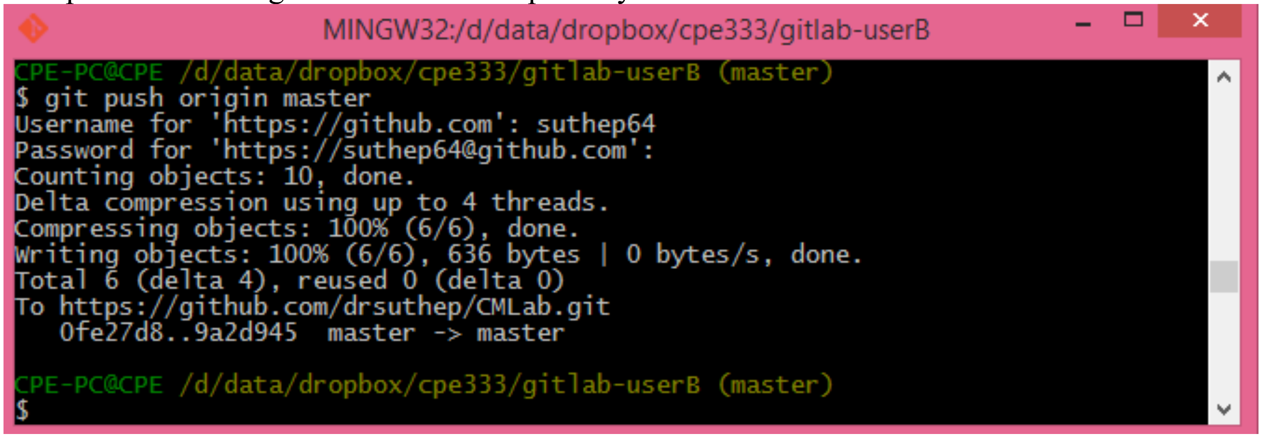
```
#include <stdio.h>
void main()
{

  int WinLotteryNo[10] = {1579, 1711, 5515, 7233, 5614, 8876, 1215, 4232, 9155, 9995};
  float WinAmount[10] = {50000, 15000, 10000, 5000, 5000, 5000, 5000, 3000, 3000, 3000};
  int N = 10;
  int i, won;
```

- User B then commits the change:



```
CPE-PC@CPE /d/data/dropbox/cpe333/gitlab-userB (master|MERGING)
$ git commit -a -m "Corrected Merge Conflict of Lottery.c"
[master 9a2d945] Corrected Merge Conflict of Lottery.c

CPE-PC@CPE /d/data/dropbox/cpe333/gitlab-userB (master)
$
```

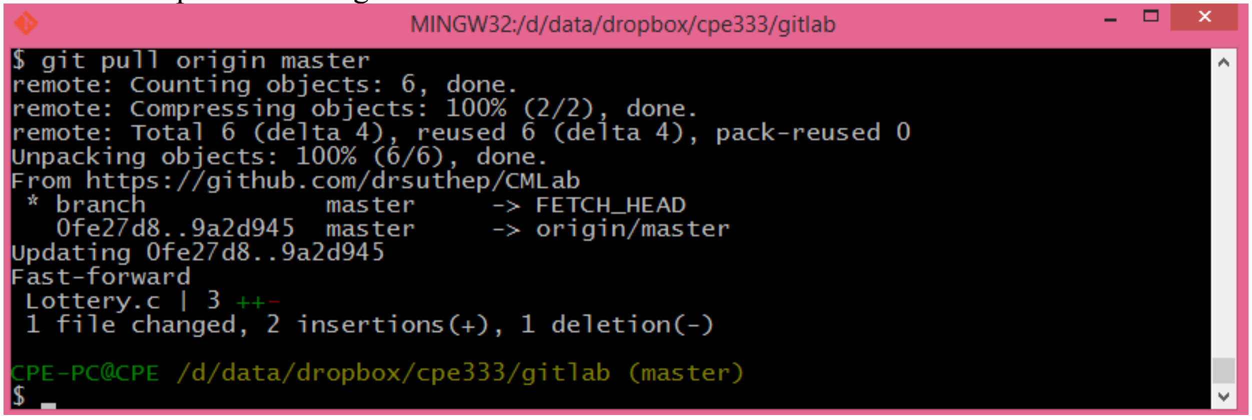
- And pushes the changes to the GitHub repository:



MINGW32:/d/data/dropbox/cpe333/gitlab-userB

```
CPE-PC@CPE /d/data/dropbox/cpe333/gitlab-userB (master)
$ git push origin master
Username for 'https://github.com': suthep64
Password for 'https://suthep64@github.com':
Counting objects: 10, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (6/6), done.
Writing objects: 100% (6/6), 636 bytes | 0 bytes/s, done.
Total 6 (delta 4), reused 0 (delta 0)
To https://github.com/drsuthep/CMLab.git
   0fe27d8..9a2d945  master -> master

CPE-PC@CPE /d/data/dropbox/cpe333/gitlab-userB (master)
$
```

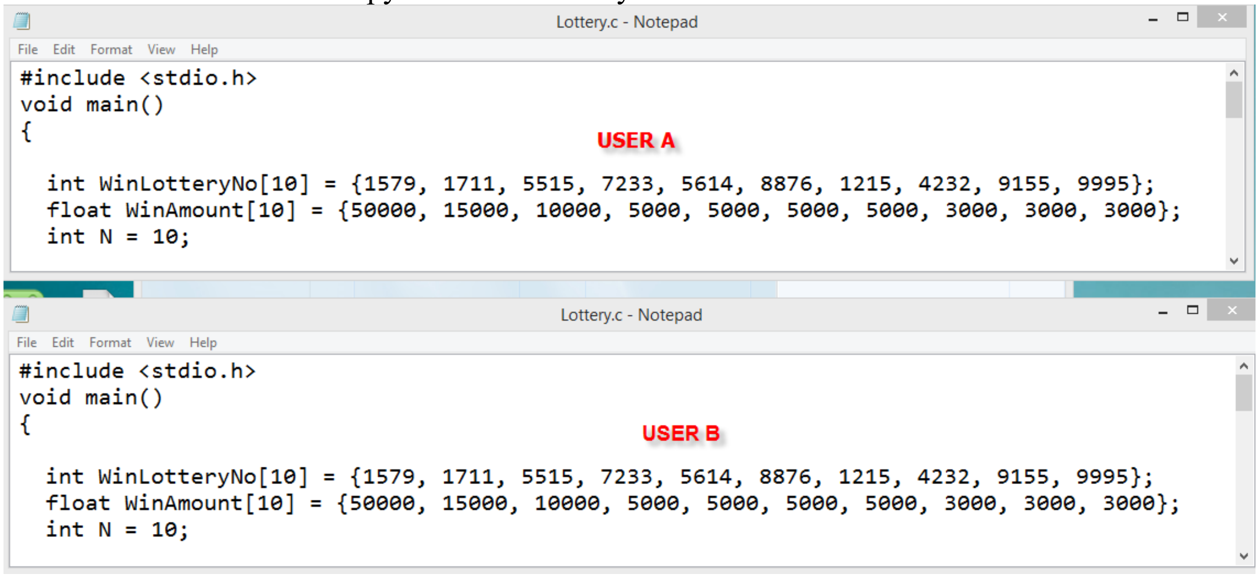
15. Then User A pulls the changes.



MINGW32:/d/data/dropbox/cpe333/gitlab

```
$ git pull origin master
remote: Counting objects: 6, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 6 (delta 4), reused 6 (delta 4), pack-reused 0
Unpacking objects: 100% (6/6), done.
From https://github.com/drsuthep/CMLab
 * branch            master     -> FETCH_HEAD
   0fe27d8..9a2d945  master     -> origin/master
Updating 0fe27d8..9a2d945
Fast-forward
 Lottery.c | 3 ++-
 1 file changed, 2 insertions(+), 1 deletion(-)

CPE-PC@CPE /d/data/dropbox/cpe333/gitlab (master)
$ _
```

- Now user A has the same copy of the file Lottery.c with conflict resolved as User B:



```
#include <stdio.h>
void main()
{

  int WinLotteryNo[10] = {1579, 1711, 5515, 7233, 5614, 8876, 1215, 4232, 9155, 9995};
  float WinAmount[10] = {50000, 15000, 10000, 5000, 5000, 5000, 5000, 3000, 3000, 3000};
  int N = 10;
```

```
#include <stdio.h>
void main()
{

  int WinLotteryNo[10] = {1579, 1711, 5515, 7233, 5614, 8876, 1215, 4232, 9155, 9995};
  float WinAmount[10] = {50000, 15000, 10000, 5000, 5000, 5000, 5000, 3000, 3000, 3000};
  int N = 10;
```

Laboratory Exercise

Run the following and submit a PDF dump similar to my above workflow dump for the following commands:

1. Unzip the lab zip file. The original source files can be found under *src*. The new file *TimeGreeting.c* is located in *NEW*.
2. Create a blank repository in GitHub under User A's account called "GITLAB" with 1 Readme file. Print hub status. Both users A and B will use this repository. Add User B as collaborator to this repository.
3. In some path create a folder called "CPE327-A" for User A. Make this a git repository and pull data fromGitHub. Show status.
4. In some path create a folder called "CPE327-B" for User B. Make this a git repository and pull data fromGitHub. Show status.
5. User A. Use file manager to put two C programs (*bloodgroup.c* and *PrintGrade.c*), and one MS Word document intoCPE327-A.
6. User A. Add all files to git. Then commit. Show Status.
7. User A. Push your local repository to the GitHub repository. (You should always pull before you push). Showscreen.
8. User B pulls data from GitHub. Show status.
9. User B change 1 function in *PrintGrade.c.* Print local status User B.
10. User B create a new C program *TimeGreeting.c*. Do not put into Git yet. Print local status User B.
11. User B adds *TimeGreeting.c* to Git control (git add). Print local status User B.
12. User B commits changes. Print local status User B.
13. User B pushes changes to the GitHub repository. (You should always pull before you push). Print hub status.Print User B status.
14. User A change something in *bloodgroup.c*. Print local status User A.
15. User A adds its changes and commits at once. Print local status User A.
16. User A pulls repository from GitHub.
17. User A pushes changes to GitHub repository. Print status User A.
18. User B pulls repository from GitHub.

- All 3 repositories have same copy now.

19. Then User A changes *bloodgroup.c* at two places and User B changes same *bloodgroup.c* at two places (make sure theychange same general area of the code or in the next step there may be no merge conflict visible □) . Both add and commit changes. Print status of User A and User B.
20. User A pulls from GitHub and then pushes to GitHub. User B then pulls and pushes to GitHub, but there's conflict. (There will be a conflict if user A and B both change *bloodgroup.c* at about the same lines). Resolve thismerge conflict for User B and commit change. Show each step.
21. Then User A pulls the changes. Display the contents of *bloodgroup.c* for both users to demonstrate that they are now the same.

If two users make *compatible changes* to the same file, then **git** may be able to resolve conflicts automatically. For instance, if User A added a line *#include <time.h>* at the top of the file, and User B included a brand new function at the end of the same source file, these two changes could be automatically combined. However, **git** would still inform you of the differing contents. You should always check the results of automatic merges, to be sure they make sense from a programming perspective. SCM systems, including **git**, don't know anything about program meaning; they can only detect conflicts or multiple changes based on the text content of the source files.

Note that this exercise does not provide any practice using branches. In fact, in many real world projects that use GitHub, developers do not make changes to the origin master directly. Instead they create a branch using the clone command, do their work, then merge changes back into the main line of development.