gitdemo SCRIPT v1
1/27/15

Introduction
This script walks you through, step by step, setting up an existing project for Git version control and merging changes made by 2 users on separate dev branches into the master branch. Things to do before starting this script are:
1. Download the accompanying gitdemo.zip for your starter project in one of your user's directories. It is a very simply Java non-Eclipse project. The included shell scripts are not needed for this demo.
2. You will need to create a remote repository on Github or some other host like my Gitlab server. If you don't know how to do this, please email me. Replace all instances of <your remote repo URL> with the URL that represents the remote repo you have created. For example, the URL for the remote repo used in this demo in class was git@github.com:UTSA-Software-Engineering/gitdemo.git

Please try to stick completely with the script for this first part of Assignment 2. Take any great ideas you get from this demo and use them in the 2nd part of Assignment 2. I have included screenshots below of my remote repo on Github as the script progresses. Your remote remote repo should look very similar (perhaps with different branch names).

Document Conventions
This demo requires two entities (users) to independently commit changes to a shared remote repo. The entities can be two separate team members, or for those working alone: two separate Eclipse workspaces or simply two separate folders. When the script refers to "User 1", it is referring to one of those entities, and "User 2" refers to the other entity, whether the entities are humans or folders.

User 1 is considered the owner of the master branch and is the only entity who is supposed to make changes to master.

Replace anything in angle brackets ("<" and ">") with information specific to you, your teammate, and/or your project.

Any text in the `Courier New` font should be typed directly in the command line.

Begin Ze Script

User 1 (prepare the project for Git)
1. unzip the gitdemo.zip archive and change (cd) to the gitdemo directory

2. create the local Git repo
```
git init
```

3. edit .git/config and add your user settings:

```
[user]
	name = <your name>
	email = <your email address>
```

4. create a .gitignore file, add the bin directory to it, and have it ignore all .DS_Store files

```
echo "bin" > .gitignore
echo ".DS_Store" >> .gitignore
```

5. add .gitignore, the shell scripts, and README.txt to the stage

```
git add ./README.txt
git add ./*.sh
git add ./.gitignore
```

6. add a second file to src and stage all new/changed files in the src folder

```
echo "This is file 2" > src/core/file2
git add ./src
```

7. commit the staged files (by default we are currently on the master branch)

```
git commit -m "First commit"
```

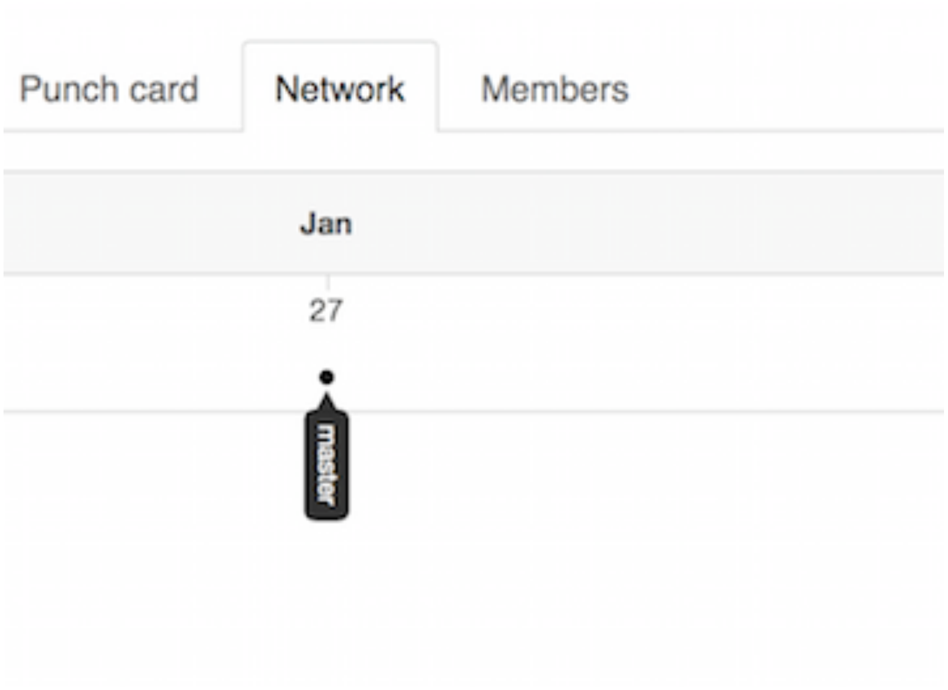
8. create a Git remote to the remote repository and call the remote “origin”

```
git remote add origin <your remote repo URL>
```

9. push the local repo changes on the master branch to the remote repo

```
git push origin master
```

*** admire the network graph of your remote repo



User 2 (download the project from the remote repo and contribute on a new branch)

10. download the project from the remote repo
```
git clone <your remote repo URL>
```

11. change into the gitdemo directory and edit .git/config and add the other entity's settings:
```
[user]
	name = <the other user's name>
	email = <the other user's email address>
```

12. create a dev branch and make it the current branch (i.e., it changes value of HEAD)
```
git checkout -b <your branch name for User 2>
```

13. create a new file in src
```
echo "this is User 2's new file" > ./src/core/file3
```

14. stage the untracked/modified files in src
```
git add ./src
```

15. commit the staged files
```
git commit -m "Added file3"
```

16. edit ./src/core/file2 and append a line similar to:
"User 2's change to file2"

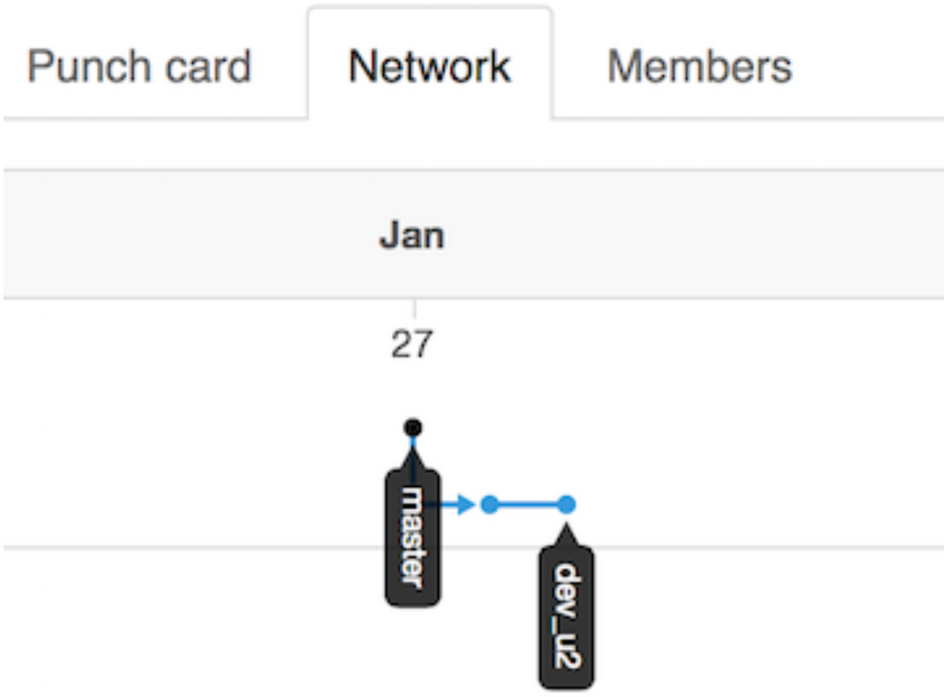
17. stage the untracked/modified files in src

```
git add ./src
```

18. commit the staged files
```
git commit -m "Added line 2 to file2"
```

19. push the local repo changes on User 2’s branch to the remote repo
```
git push origin <your branch name for User 2>
```

*** admire the network graph of your remote repo



User 1 (contribute on a new branch)
20. create a dev branch and make it the current branch
```
git checkout -b <your branch name for User 1>
```

21. create a new file in src
```
echo "this is User 1’s new file" > ./src/core/file4
```

22. stage the untracked/modified files in src
```
git add ./src
```

23. commit the staged files
```
git commit -m "Added file4"
```

24. edit ./src/core/file2 and append a line similar to:

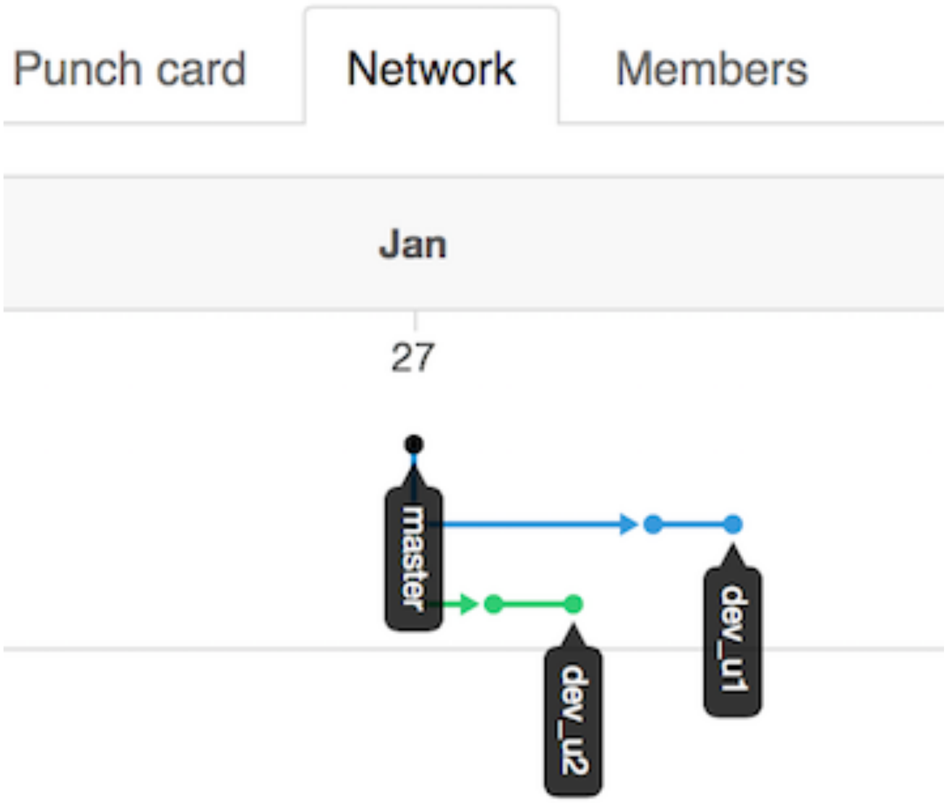
"User 1’s change to file2"

25. stage the untracked/modified files in src
```
git add ./src
```

26. commit the staged files
```
git commit -m "Added line 2 to file2"
```

27. push the local repo changes on User 1’s branch to the remote repo
```
git push origin <your branch name for User 1>
```

*** admire the network graph of your remote repo



User 1 (merge User 1’s branch into master)
28. switch to master in case we are not there already (best to be paranoid)
```
git checkout master
```

29. merge User 1’s branch into master (we force an additional merge commit with the --no-ff switch). Git will open a text editor for the merge commit message. You can use the default that Git provides. Save/quit the commit message editor when done and the commit will continue.
```
git merge --no-ff <your branch name for User 1>
```

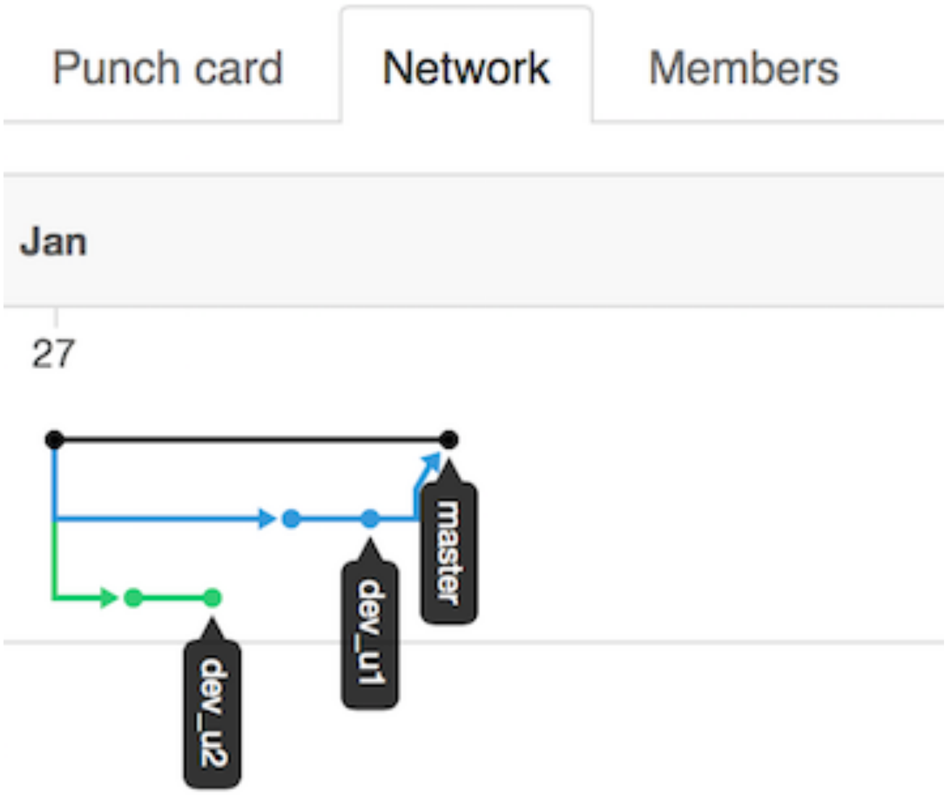
30. delete User 1’s branch since it is no longer needed (does not delete any commits, just a pointer to a commit that will no longer be used)

```
git branch -d <your branch name for User 1>
```

31. push the local changes to master to the remote repo

```
git push origin master
```

*** admire the network graph of your remote repo



User 1 (merge User 2’s branch into master)

32. fetch User 2’s changes in his/her dev branch

```
git fetch
```

33. switch to master in case we are not there already (best to be paranoid)

```
git checkout master
```

34. try to merge User 2’s branch (this will cause a merge conflict). Note that because this is a remote branch, you need to precede the branch name with “origin/” and no space after the forward slash.

```
git merge --no-ff origin/<your branch name for User 2>
```

35. use git status to see which files are conflicted (should just be ./src/core/file2)

```
git status
```

36. edit ./src/core/file2 and move User 2's line above User 1's line and remove all of the Git conflict indicators. Save and close the file editor when done.

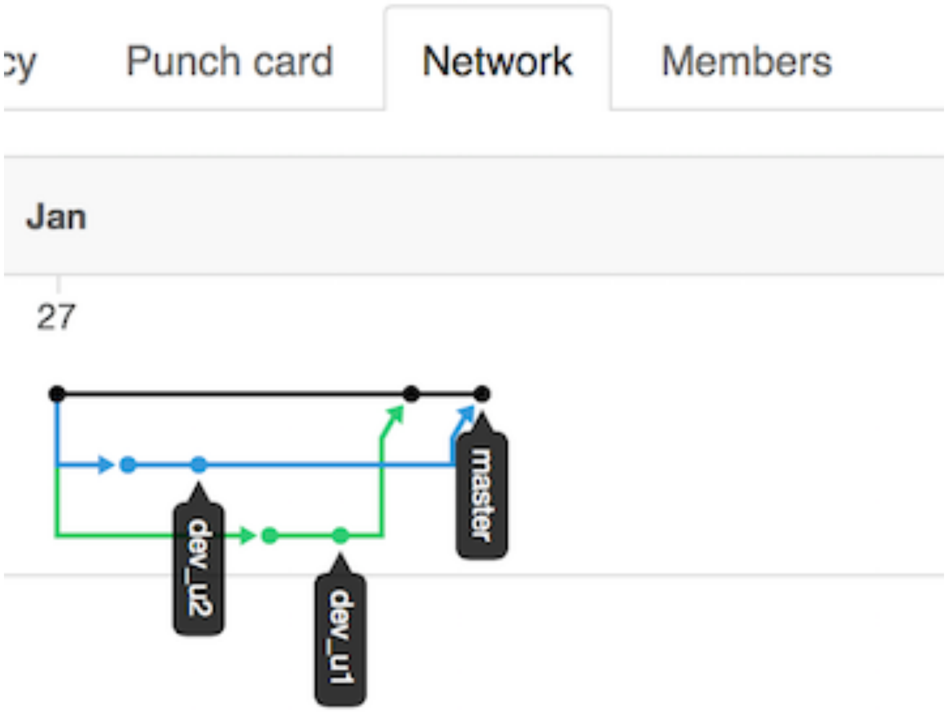
37. use git add to tell Git that the conflict has been resolved
```
git add ./src
```

38. use git status again to verify that no other files have conflicts
```
git status
```

39. finalize the merge commit. You will need to supply a commit message for the merge commit but you can use the default that Git provides. Save/quit the commit message editor when done and the commit will continue.
```
git commit
```

40. push the local changes to master to the remote repo
```
git push origin master
```

*** admire the network graph of your remote repo



User 2 (update local master branch to the latest version on the remote repo)
41. switch to the master branch
```
git checkout master
```

42. since we know that User 2 is not the owner of master, we also know that his master branch has not diverged from what is in the commit history of the remote's master branch. Therefore, we can simply fetch and merge the latest changes from the remote into User 2's local master branch

```
git pull
```

43. examine ./src/core/file2 to make sure the contents match what User 1 merged into master. User 2's contribution should be the 2nd line of file2 and User 1's contribution should be the 3rd line of the commit.

43. delete User 2's branch since it is no longer needed (User 1 told us that the merge was successful)

```
git branch -d <your branch name for User 2>
```