INTRODUCTION TO GIT/GITHUB

WORKSHOP AGENDA

WORKSHOP AGENDA

1. **Installing Git**
2. **Introduction to Version Control**
3. **Git Basics**

WORKSHOP AGENDA

1. Installing Git
2. Introduction to Version Control
3. Git Basics
4. **Creating a new local Git repository**
5. **Cloning a Git repository**
6. **Making use of Git commit history**
7. **Reverting files to previous states**

WORKSHOP AGENDA

1. Installing Git
2. Introduction to Version Control
3. Git Basics
4. Creating a new local Git repository
5. Cloning a Git repository
6. Making use of Git commit history
7. Reverting files to previous states
8. **Creating a Github Repository**
9. **Adding, Committing & Pushing changes**
10. **Branching**
11. **Merging Branches**
12. **Sending Pull Requests**
13. **Conflict Resolution**

and 3 Exercises…

INSTALLING GIT

Windows: git-scm.com/download/win

Mac: git-scm.com/mac

Linux: sudo apt-get install git (Ubuntu)
    or sudo yum install git (Fedora)

Once installed check **Git version:** git --version

FIRST TIME GIT SETUP

Your Identity

git config --global user.name "Your Name"

git config --global user.email example@ncsu.edu

Your editor

git config --global core.editor <editor like vim, emacs etc>

Enable color in git

git config --global color.ui auto

Checking your settings

git config --list

INTRODUCTION TO VERSION CONTROL

"Git is an example of Version Control"

Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later.

It allows you to:
- Revert files to previous state
- Revert entire project back to previous state
- Compare changes over time
- See who modified what? And much more…

It means if you screw things up or lose files, you can easily recover.

OTHER POPULAR VERSION CONTROL SYSTEMS

Subversion (SVN)

Concurrent Versions System (CVS)

Perforce

Mercurial

Bazaar

and the ones you know...

GIT: THE BIG PICTURE

Use Cases:

- Individual Development
- Collaborative Development
- Offline Usage

Why Git?

- Everything is local (full history tree available offline)
- Everything is fast
- Snapshots, not diffs
- It is distributed not centralized
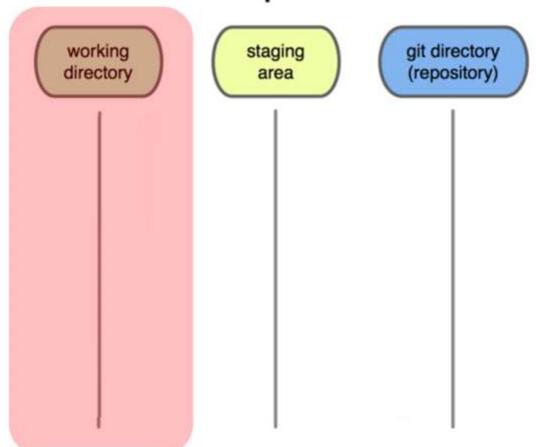- Great for those who hate CVS/SVN

GIT WORKFLOW: THE THREE STATES

"Something very important"

In a Git repository your file can reside in three main states:
- Modified
- Staged
- Committed
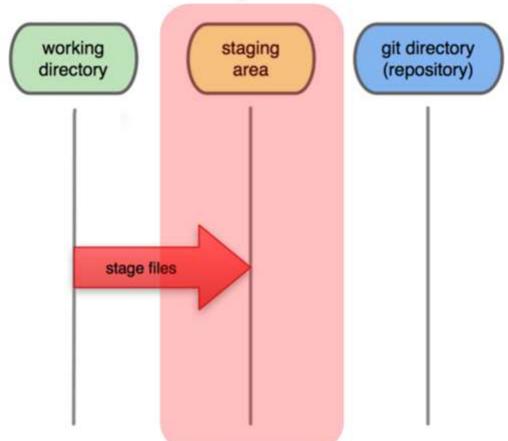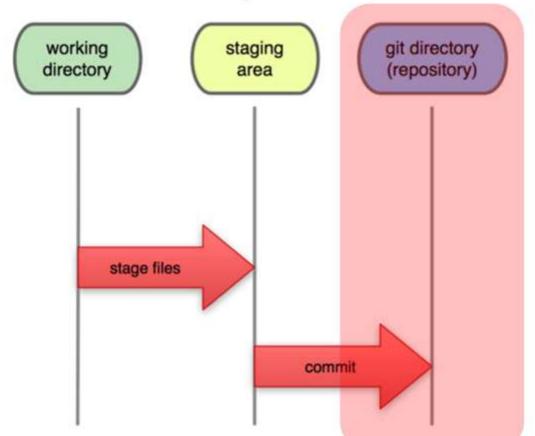
What does this mean?

GIT WORKFLOW: THE THREE STATES

Local Operations

working directory

staging area

git directory (repository)

You modify files in your working directory.

GIT WORKFLOW: THE THREE STATES

Local Operations

working directory

staging area

git directory (repository)

stage files

You stage the files, adding snapshots of them to your staging area.

GIT WORKFLOW: THE THREE STATES

Local Operations

working directory → **staging area** → **git directory (repository)**

stage files

commit

You do a commit that stores snapshot permanently to your Git directory.

GIT WORKFLOW: THE THREE STATES

Local Operations

working directory

staging area

git directory (repository)

checkout the project

stage files

commit

Then, you can checkout any existing version, make changes, stage them and commit.

ENOUGH OF THEORY

LETS GET STARTED...

CREATE A NEW REPOSITORY

Create a new directory, open it and perform a

`git init`

to create a new local git repository

CHECKOUT A REPOSITORY

In a new folder create a working copy of local repository

```
git clone /path/to/repository
```

```
git clone https://github.com/rprataps/Exer1.git
```

GET GIT COMMIT HISTORY

You can study repository history using

```
git log
```

you can see last 'p' (a number) commits

```
git log -n p
```

GET GIT COMMIT HISTORY

You can also see git history with affected files

```
git log --stat
```

to see a compressed log where each commit is one line

```
git log --pretty=oneline
```

COMPARE CHANGES OVER TIME

See changes in a commit

```
git show commitID
```

and compare two commits by

```
git diff commit1ID commit2ID
```

REVERT FILES TO PREVIOUS STATES

Revert to previous state

```
git checkout commitID
```

and return to current state

```
git checkout master
```

EXERCISE-1

1. Clone the repository (if you haven't):

```
git clone https://github.com/rprataps/Exer1.git
```

2. You have to find a bug in "intro_to_git1.html" (Hint: Open the web page in browser and select all text).

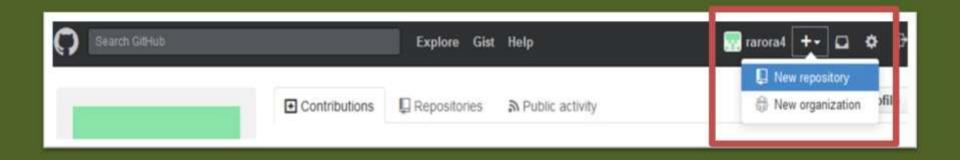3. After identifying the bug you have use "git diff" to identify the commit in which this bug was introduced.
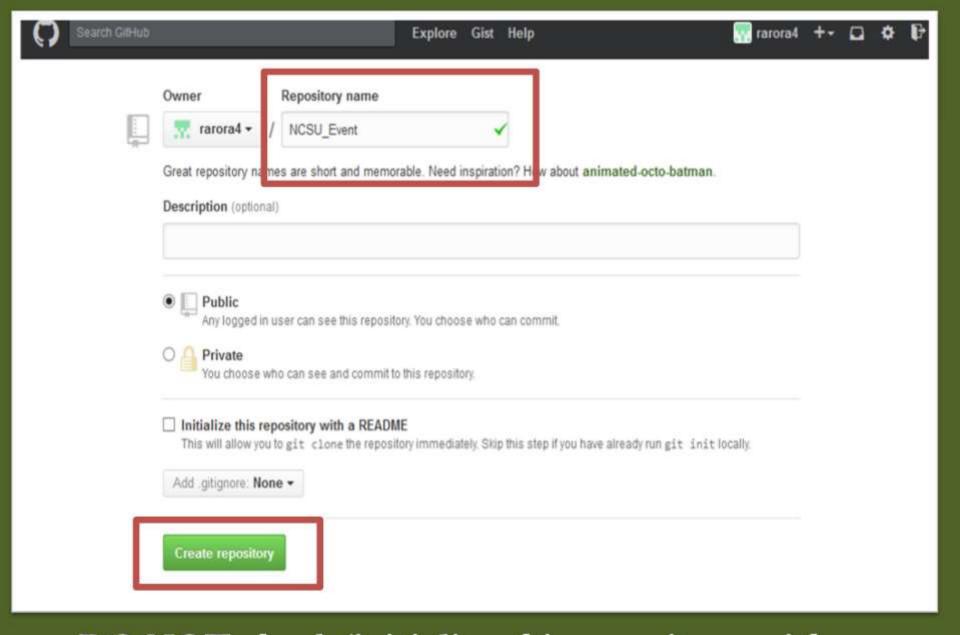
LOGIN NSCU GITHUB ACCOUNT

Go to https://github.ncsu.edu
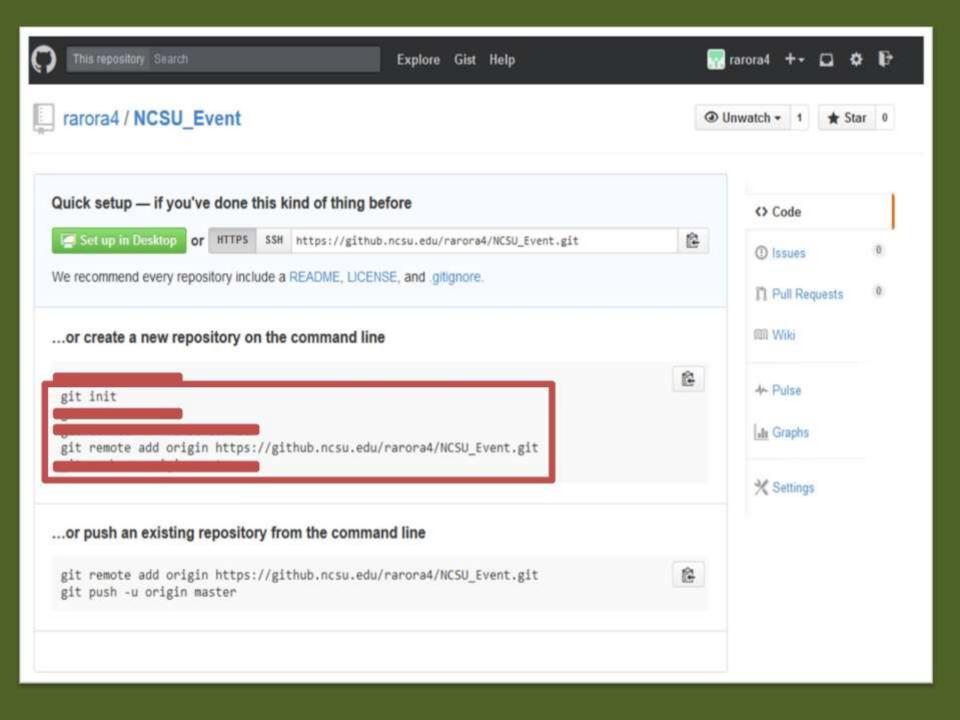
Login with your unity id and password

CREATE A GITHUB REPOSITORY

Click on '+' sign on top right corner of github homepage and select 'New Repository'

DO <u>NOT</u> check 'initialize this repository with a README.

📖 rarora4 / **NCSU_Event**

👁 Unwatch ▾ | 1 | ★ Star | 0

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

🖥 Set up in Desktop | or | HTTPS | SSH | https://github.ncsu.edu/rarora4/NCSU_Event.git | 📋

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

...or create a new repository on the command line

```
git init

git remote add origin https://github.ncsu.edu/rarora4/NCSU_Event.git
```

...or push an existing repository from the command line

```
git remote add origin https://github.ncsu.edu/rarora4/NCSU_Event.git
git push -u origin master
```

<> Code

ⓘ Issues | 0

🔀 Pull Requests | 0

📖 Wiki

✦ Pulse

📊 Graphs

✕ Settings

EXERCISE - 2

1. Create a new directory (folder)
2. Add a new file git_exercise.txt
3. Write any 3 git commands
4. Save this file.
5. Move this file to Staging Area
6. Commit this file
7. And, push this file to your NCSU Github repository.

ADD & COMMIT

Add new files to staging area

```
git add <filename>
```

```
git add *
```

then commit changes

```
git commit -m "Commit Message"
```

IT'S A STANDARD GIT PRACTICE TO ADD MESSAGE AS A COMMAND AND NOT IN PAST TENSE.

PUSHING CHANGES

To send these changes to your remote repository execute

```
git push origin master
```

you can also push to a branch other than 'master'

```
git push origin branch_name
```

BUT WAIT...

WHAT IS ORIGIN?

WHAT IS MASTER?

WHAT IS BRANCH?

ORIGIN

When you clone a repository for the first time

Origin is a default name given to the original remote repository that you clone, from where you want to pull and push changes.

```
git clone /path/to/repository
```

By saying **git** push **origin** branch name you're saying to push to the **origin** repository.

ORIGIN

When you initialize a new repository

By convention you name your new Github (remote) repository as **Origin.**

```
git init
git remote add origin /path/to/repository
```

There's no requirement to name the remote repository **origin** , and there can be multiple remote repositories.

BRANCHING

Branches are used to develop features isolated from each other. The *master* branch is the "default" branch when you create a repository. Use other branches for development and merge them back to the master branch upon completion.

BRANCHING

To create a new branch

```
git branch new_branch_name
```

switch to a different branch

```
git checkout other_branch_name
```

BRANCHING

To create a new branch and switch to that branch

```
git checkout -b new_branch_name
```

BRANCHING

To show all the branches

```
git branch
```

to delete a branch

```
git branch -d branch_name
```

BRANCHING

Remember, a branch is not available to others unless you push the branch to your remote repository

```
git push origin branch_name
```

MERGING BRANCHES

Checkout the branch you want to update

```
git checkout branch_name
```

Then merge branches by executing

```
git merge this_branch_name that_branch_name
```

MERGING BRANCHES

Git tries to auto-merge changes. Unfortunately, this is not always possible and results in *conflicts*. You are responsible to merge those *conflicts* manually by editing the files shown by git.
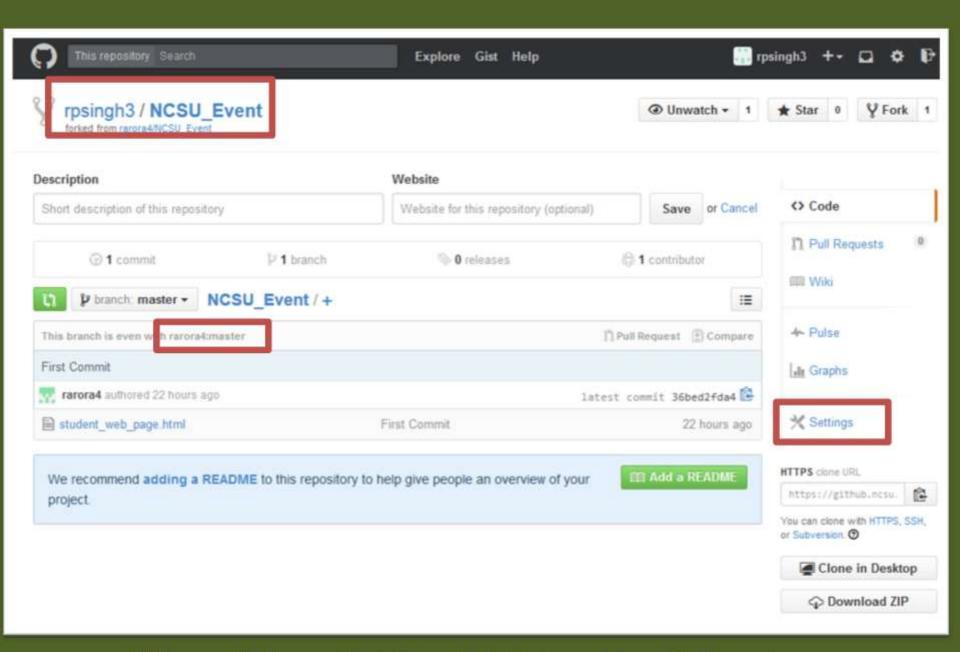
Useful tip: Before merging changes, preview changes
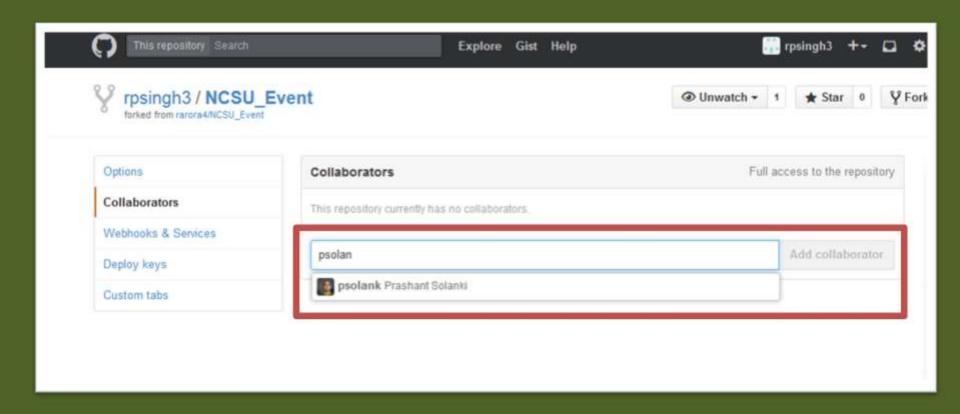
```
git diff this_branch_name that_branch_name
```

EXERCISE - 3

1. Form a team of 2
2. 1 Member will Fork this repository:
   https://github.ncsu.edu/rarora4/NCSU_Event
3. The same member will add other member as
   collaborator to this forked repository.

Go to Repository you want to fork and click on 'Fork' button

Then click on 'Settings' link to add collaborators.

Enter the Unity ID of the collaborators, then click 'Add Collaborator' button

EXERCISE - 3

1. Form a team of 2
2. 1 Member will Fork this repository: https://github.ncsu.edu/rarora4/NCSU_Event
3. The same member will add other member as collaborator to this forked repository
4. Both the members will clone this repository
5. 1 Member will work on master branch and the other will create a new branch
6. Open 'student_web_page.html', and add more student tasks ' this is a new task '

```
git clone
https://github.ncsu.edu/rarora4/NCSU_Event.git
```

Person working on branch:

```
git branch new_branch_name
```

```
git checkout new_branch_name
```

Person working on master

```
You work on Master Branch
```

```
14    <li> Complete all assignments on time </li>
15    <li> Participate in ACM/AITP Hack Nights </li>
16    <li> Enjoy Pig Picking Event </li>
17    <li> Eat Pizzas </li>
18  ⊟<li> Perform American Line Dance
19   └</ul>
20   ├</body>
```

EXERCISE-3

7. **Save changes, stage changes, commit them.**

```
git add *
```

```
git commit -m "Commit Message"
```

8. **Push changes to remote repository by executing:**

Person working on master:
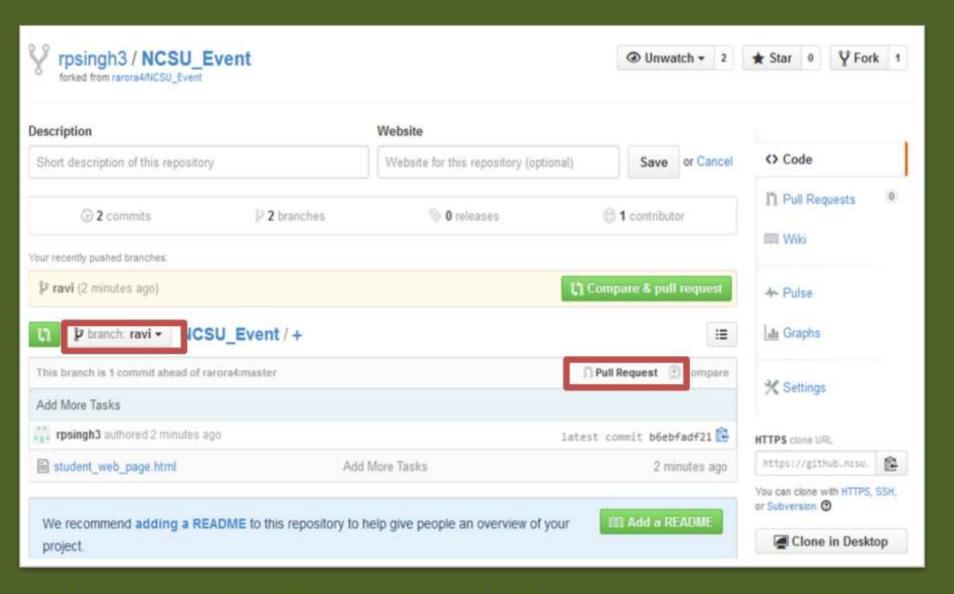```
git push origin master
```

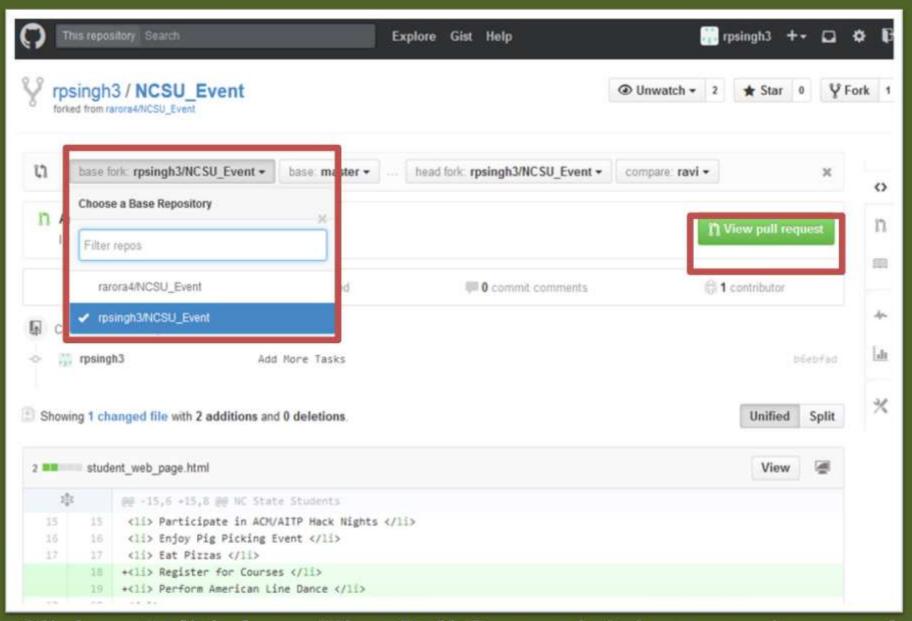**Person working on branch:**
```
git push origin branch_name
```

You can see all the branches by clicking on 'branch' dropdown.
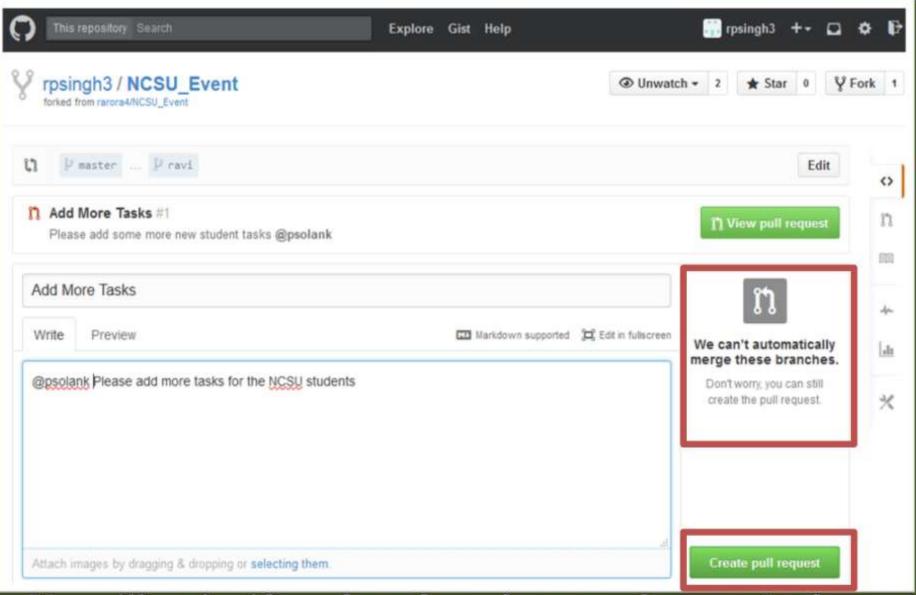To switch to a branch select that branch.

Now you are in different branch. Contents of a branch (if not merged) will be different then that of master branch.
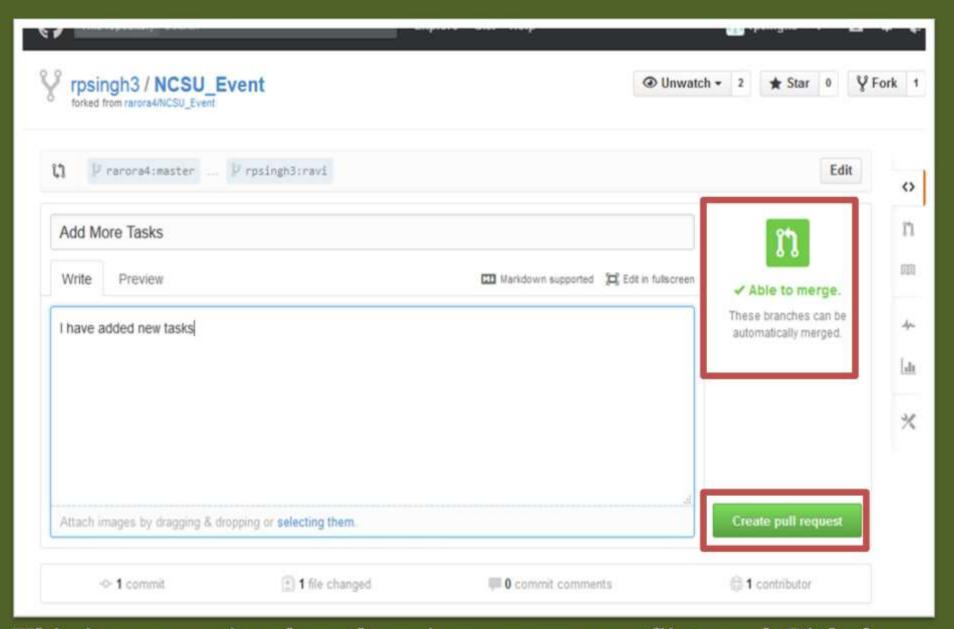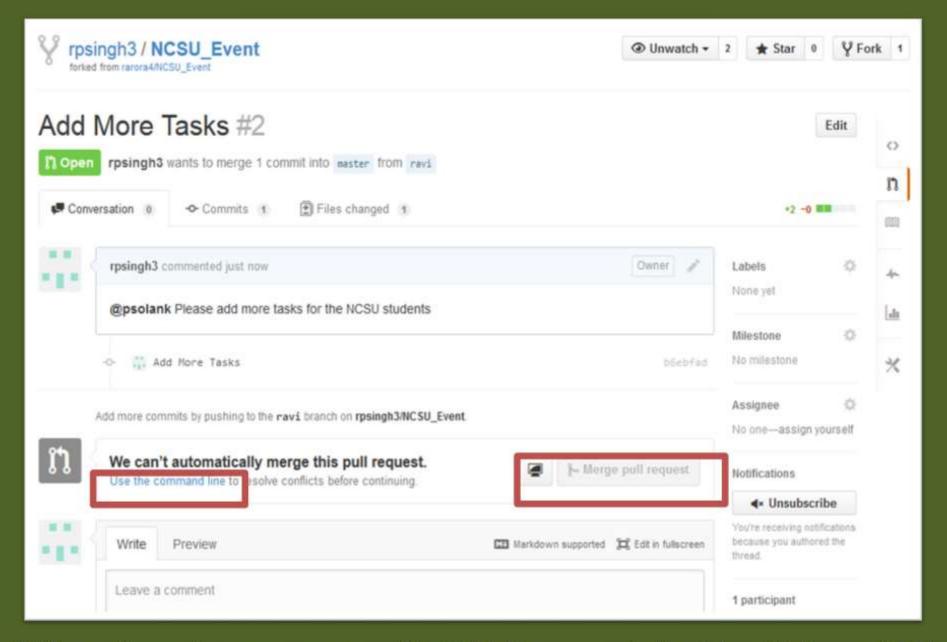To marge this branch with master, click on 'Pull Request'.

Click on 'Edit' above 'View Pull Request'. Select repository and name of the branch with which you want to merge your branch.

You will notice if you branch can be merged or not. In above scenario there is a merge conflict. Still you can create a pull request.

This is a scenario when there is no merge conflict, and Github can itself merge branches.

When there is a merge conflict. Pull request disables "Merge Pull Request" button and suggests you to use your machine to resolve it.

# EXERCISE-3

9. To resolve the conflict - collaborator working on the new branch has to switch to master branch, pull latest code from repository's master branch.

```
git checkout master
```

```
git pull origin master
```

10. Then switch back to the branch, merge master's new code with this branch's code.

```
git checkout new_branch_name
```

```
git merge new_branch_name master
```

EXERCISE - 3

11. You will receive a merge conflict message

```
RAVIPRATAP (ravi) NCSU_Event $ git merge ravi master
Auto-merging student_web_page.html
CONFLICT (content): Merge conflict in student_web_page.html
Automatic merge failed; fix conflicts and then commit the result.
RAVIPRATAP (ravi *|MERGING) NCSU Event $
```

12. Open 'student_web_page.html' to determine the conflicting portion and resolve them manually.

```
<li> Attend all classes </li>
<li> Complete all assignments on time </li>
<li> Participate in ACM/AITP Hack Nights </li>
<li> Enjoy Pig Picking Event </li>
<li> Eat Pizzas </li>
<<<<<<< HEAD
<li> Register for Courses </li>
<li> Perform American Line Dance </li>
=======
<li> Attend Git Workshop </li>
<li> Code </li>
>>>>>>> master
</ul>
```
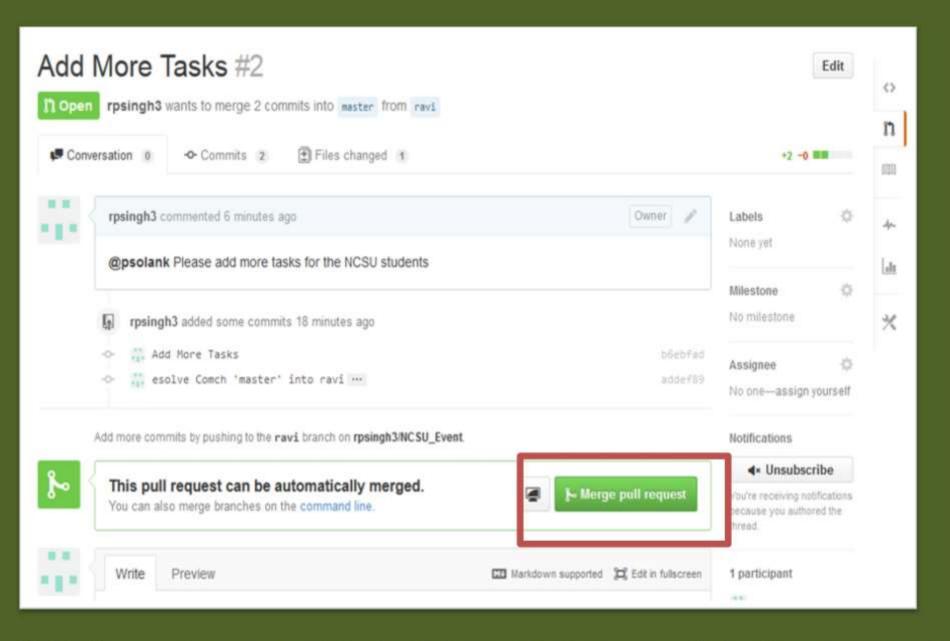
```
<li> Attend all classes </li>
<li> Complete all assignments on time </li>
<li> Participate in ACM/AITP Hack Nights </li>
<li> Enjoy Pig Picking Event </li>
<li> Eat Pizzas </li>
<li> Register for Courses </li>
<li> Perform American Line Dance </li>
<li> Attend Git Workshop </li>
<li> Code </li>
</ul>
```
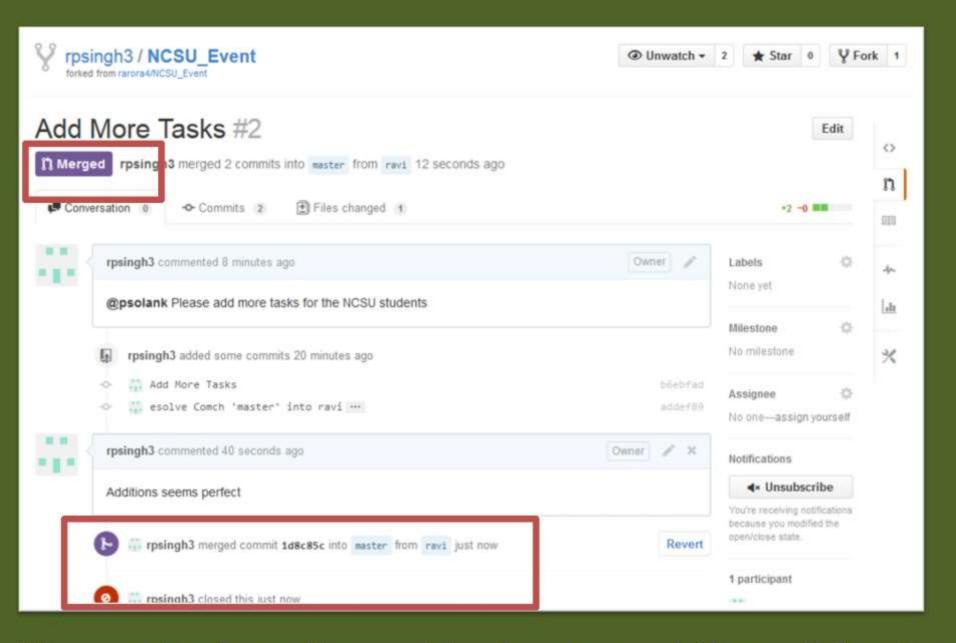
EXERCISE - 3

13. After resolving the conflict save changes, stage them, commit them and push code to the branch on Github.

```
git push origin branch_name
```

14. On Github check your pull request, it will automatically update, now you will see the message "This pull request can be automatically merged."

15. The collaborator working on master branch has to marge the pull request into master by clicking on "Merge Pull Request" button.

Pull Request page updated, "Merge Pull Request" button becomes active.

Message showing pull request has been merged. Now collaborator can also discard the extra branch created.

EXERCISE - 3

16. Now both the collaborators can pull the latest merged from Github on master branch.

```
git checkout master
```

```
git pull origin master
```

"When in doubt, do exactly the opposite of CVS"
                                        - Linus Torvalds

REFERENCES

- http://git-scm.com/book/en/v2/Getting-Started-Git-Basics
- http://rogerdudler.github.io/git-guide/
- https://www.udacity.com/wiki/ud775/lesson-1-notes#morsel-2-course-overview
- http://www.slideshare.net/glen_a_smith/git-one-day-training-notes
- https://www.f30.me/2013/05/some-graphics-on-git/