HW04 - Using Git (Single Developer)

6K:183 Systems Analysis and Design
Due February 12, 2013

What you will need to complete this assignment:

1. A free Github account

2. Two computers
   - 2.1. Both with Git installed and configured with your Github account. See "Configuring Git On A New PC" for help with global settings and ssh keys.

   - 2.2. Both with a text editor (Notepad++, Notepad, TextEdit)

   - 2.3. Decide which computer is “Computer1” and which is “Computer2”

     NOTE: I would recommend going to the MIS lab and either logging in to two computers (side by side) or your laptop and one MIS lab computer. If you are using two computers, side by side, you will only need to configure Git once. The settings will be stored on your h:\ drive and accessible on any lab PC.

3. Patience.

Ways we can use Git in our class and projects:

1. Serial, by yourself.

2. Serial, by yourself, with branches (common for individual projects)

3. Serial, in a group (inefficient)

4. Parallel, in a group

5. Parallel, in a group, with branches (most common scenario)

This assignment will introduce you to Git and how to use it for projects where you are the only developer.

> *NOTE: When prompted to name a file %hawkid%.html, do not literally name it %hawkid%. This is meant to be a variable that means “your hawk ID”. For example, mine will be colbert.html.*

Serial, by yourself

1. Log on to Computer1. If necessary, connect git on the local computer to your Github account. See "Configuring Git On A New PC" for help with global settings, ssh keys, and testing connectivity.

2. Create a local git repository

   2.1. Open git bash on your computer

   2.2. Create a directory to hold your local repository (follow either 2.2.1 or 2.2.2 depending on which computers you are using.

      2.2.1. If you are using a lab PC (MIS lab or C321)

      `pwd` (print working directory : shows you where you are. You are probably in your H drive(/h/)

      `cd /c/` (change to the C:\ drive)

      `cd Users` (change directory - into the Users folder)

      `cd %hawkid%` (change directory - into *your* folder)

      `cd Desktop` (change directory - into the Desktop subfolder)

      NOTE: You should be able to see the folder paths by opening “My Computer” clicking on the C:\ drive, clicking on the Users folder, then clicking on the folder with the same name as your HawkID, then clicking on Desktop.)

      `mkdir HW4` (make directory - create a folder named HW4 on your desktop. You should see the folder appear.)

      `cd HW4` (change into the directory you just created)

      2.2.2. If you are using your PC

      `pwd` (print working directory - shows you where you are. You are probably in your profile folder (c:\Users\%ProfileName%\))

      `cd Desktop` (change directory - into the Desktop subfolder)

      NOTE: You should be able to see the folder paths by opening “My Computer” clicking on the C:\ drive, clicking on the Users folder, then clicking on the folder with the same name as your HawkID, then clicking on Desktop.)

      `mkdir HW4` (make directory - create a folder named HW4 on your desktop. You should see the folder appear.)

      `cd HW4` (change into the directory you just created)

2.3. Initialize the directory as a Git repository

```
git init
```

2.4. See the history of the project.

`git log` fatal: bad default revision ‘HEAD’ --> no HEAD because there are no commits yet

2.5. View the state of the project. Are the files, tracked, modified, staged, etc.

```
git status
```

3. Minimize the Git Bash window. Go into the HW4 folder on your desktop and create a file named %hawkid%.html. Open the file with Notepad++ and add the following very basic HTML tags and info. Save the file. Open it in a browser to ensure you can view it correctly.

```
<html>
<head><title> Git Test Page </title></head>
<body>
Hello world!
</body>
</html>
```

4. Go back to Git bash. If you accidentally closed it, you will need to change back onto the c:\ drive and into the Users\Profile\Desktop\HW4 folders (2.2.1) or simply change into the Desktop and HW4 folders (2.2.2) to get back into the correct repo folder. **`pwd`** will show you where you are.

   After you are in your repo folder:

   `ls` (list: to see the contents of the folder. You should see your .html file. If you do not, use pwd and cd to move to the correct folder.)

   `git status` --> You should now see your .html file in red under “Untracked files:”. Git sees the file and knows it is in you repo folder, but it currently is not being tracked by git. It will not be saved in any snapshots.

5. Track the file.

```
git add <<filename>>
```

   or

`git add .` (the dot recursively adds all the files)

6. View the project status again.

```
git status
```

--> You should now see the file in green. It is being tracked and has been staged for a snapshot during the next commit.

7. Commit the file to the local repo.

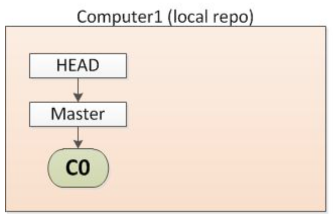
```
git commit -m 'Initial commit'
```

```
git log
```

--> You should now see information about the commit including the SHA value, author, date, and commit comments.

```
git status
```

--> Nothing should appear since no files have been added or modified since the last commit a few seconds ago.



Initial commit.

8. Push your master branch (master) to the github master (origin/master)

   8.1. From a web browser, go to www.github.com and log in to your account.

   8.2. Create a new repository named HW4

   8.3. Copy the URL that is created

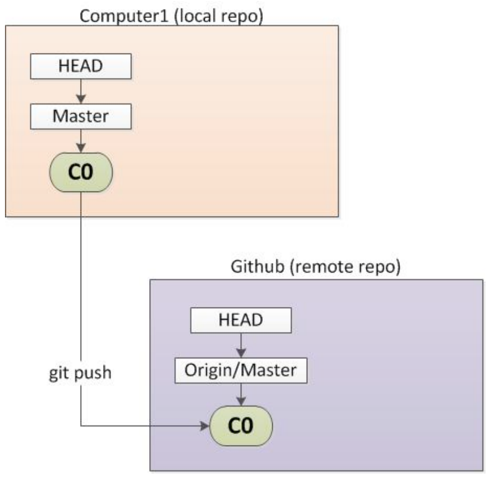
   8.4. In Git Bash, push your local repo (the PC you are working on) to the remote repo (Github)

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

```
git push origin master
```

(Push the master branch (local) to origin (remote))

Github (remote repository) is now in sync with Computer1 (local repository)



8.5. Look at your account on Github from a web browser. You should see your commit.

9. Using Notepad++, edit %hawkid%.html and add your full name to the body. Save the file.

```
git status
```
(should see it is modified, but not committed)

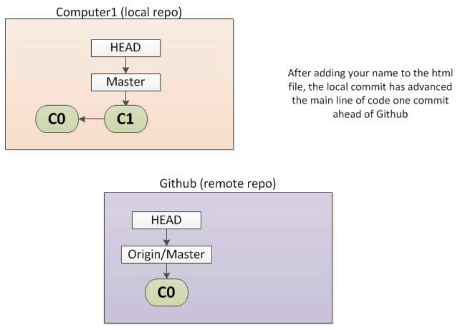
10. Stage the file for commit.

```
git add <<filename>>
```

11. Commit the file.

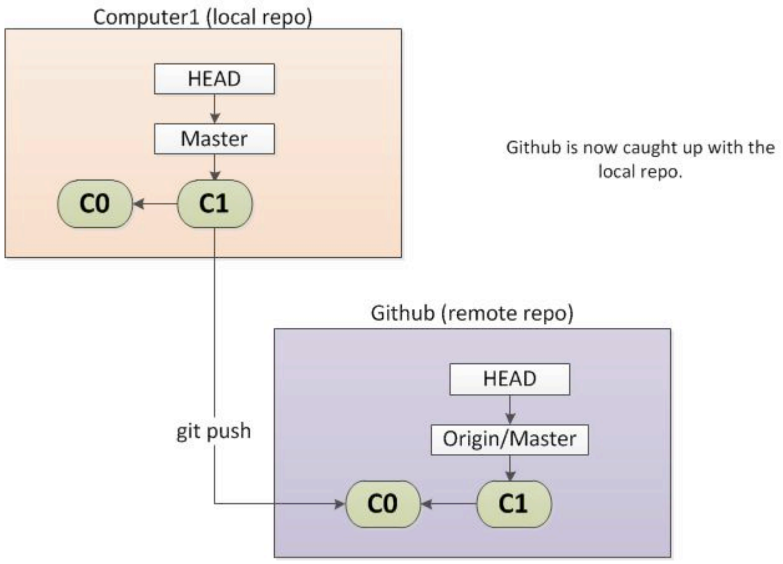
```
git commit -m 'Added my name to the file.'
git status
```
(should see it is committed)

Computer1 is one commit ahead of Github, because you have done a local commit, but not pushed it up to Github yet.


Computer1 (local repo)
HEAD
Master
C0
C1
After adding your name to the html file, the local commit has advanced the main line of code one commit ahead of Github
Github (remote repo)
HEAD
Origin/Master
C0

12. Push your local master branch (master) to the Github master (origin/master)

```
git push origin master
```



13. View your account on github. It will have updated and you should see the second commit and the new changes (your name).

Serial, by yourself on two different computers

14. Log on to Computer2. Connect git on the local computer to your Github account. See "Configuring Git On A New PC" for help with global settings, ssh keys, and testing connectivity. Remember, if you are using lab computers, the settings are already saved in your h:\ drive and you are ready to move on to #15.

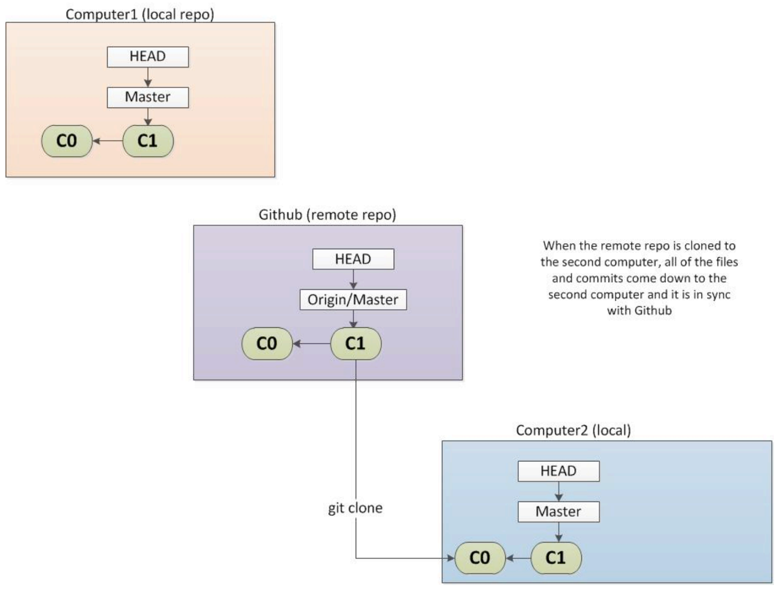
15. Open Git Bash. Use pwd to figure out where you are and **`cd`** to move to the \Desktop folder.

    For help moving to the \Desktop folder see 2.2.1 or 2.2.2

16. Pull your HW4 repo from Github to your local repo on Computer2.

```
git clone <<URL>>

      ie. git clone git@github.com:mikecolbert/HW4.git
```


Computer1 (local repo)
HEAD
Master
C0
C1
Github (remote repo)
HEAD
Origin/Master
C0
C1
When the remote repo is cloned to the second computer, all of the files and commits come down to the second computer and it is in sync with Github
Computer2 (local)
HEAD
Master
C0
C1
git clone

17. Change directory into the folder

```
cd HW4
```

18. Minimize the Git Bash shell.

19. Browse to the HW4 directory on Computer2 and open %hawkid%.html. It should look like you left it above, basic HTML with your name in the body.

20. Edit your HTML file to add a line about how much you are enjoying this git assignment, this class, the professor, or optimally - all three.

21. Save the file.

22. View the state of the project files.

    `git status`   (.html file should show red (modified))

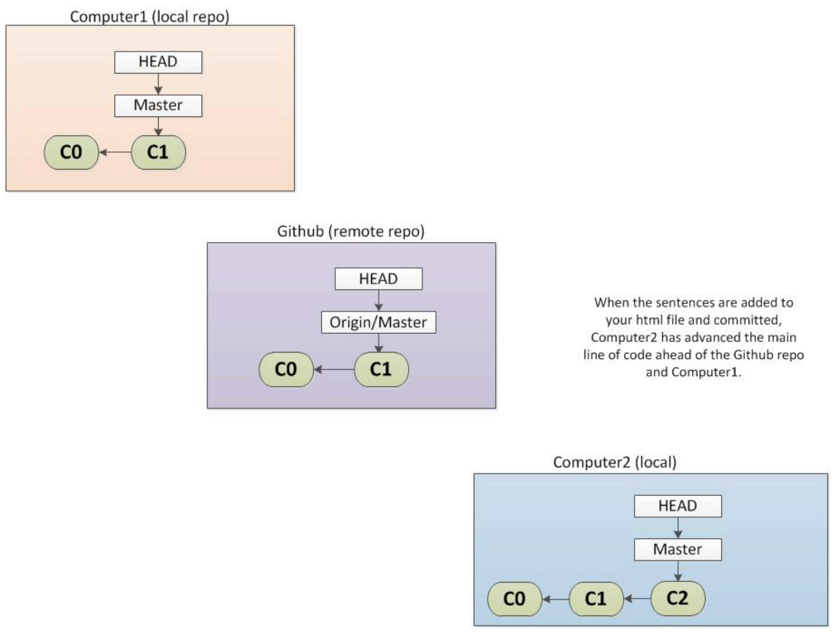
23. Stage and commit the file from Computer2.

24. View the state of the project files. It should be clear since no changes have been made since the last commit a few seconds ago.

    `git status`

25. View the history of the project. You should see a few commits, their SHA value, author, and commit messages.

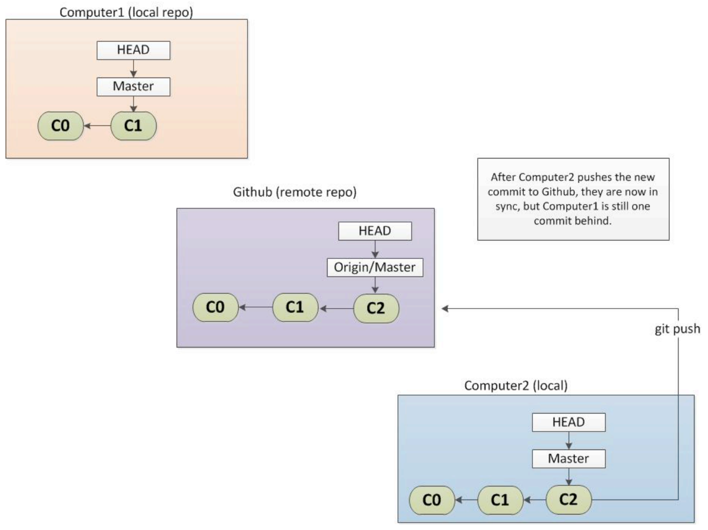
    `git log`



When the sentences are added to your html file and committed, Computer2 has advanced the main line of code ahead of the Github repo and Computer1.

26. Push the local repository (master) from Computer2 to the remote repository (origin/master) on Github.

```
git push origin master
```

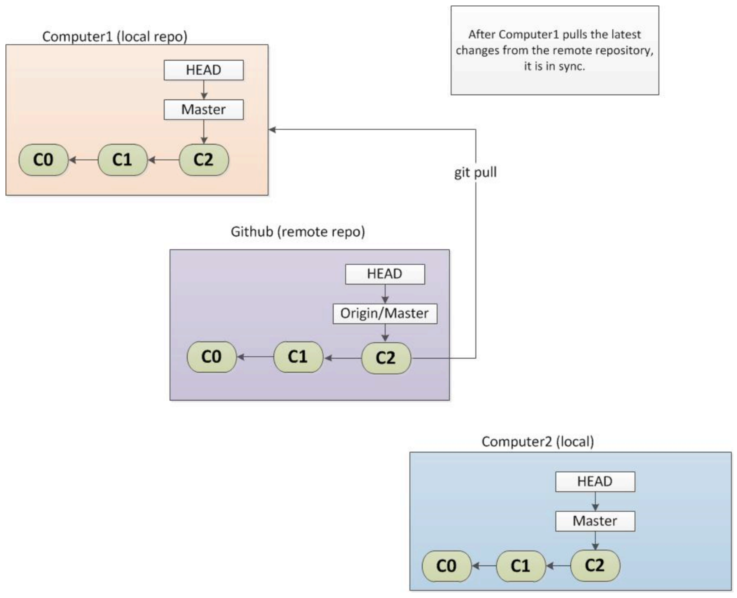


27. From a browser, view your github account. You should see this third commit and the changes you just made.

28. Open %hawkid%.html on both computers and compare. They should be different because Computer2 advanced the main line of code. To get Computer1 in sync, you need to sync Computer1's local repository with the remote repository (Github).

29. From computer1, pull the repo.

```
git pull <<URL>>
```



30. Compare again. Computers1 & 2 should now match.

31. On Computer2:

    31.1. Add a second file in the HW4 folder named README.txt. Open it and type your name. Save the file.

    31.2. Track the file

    31.3. Commit the file

32. On Computer2:

    32.1. Edit README.txt to include your favorite baseball team.

    32.2. Create a sub-directory in HW4 named “%your first name%” (ie. Mike).

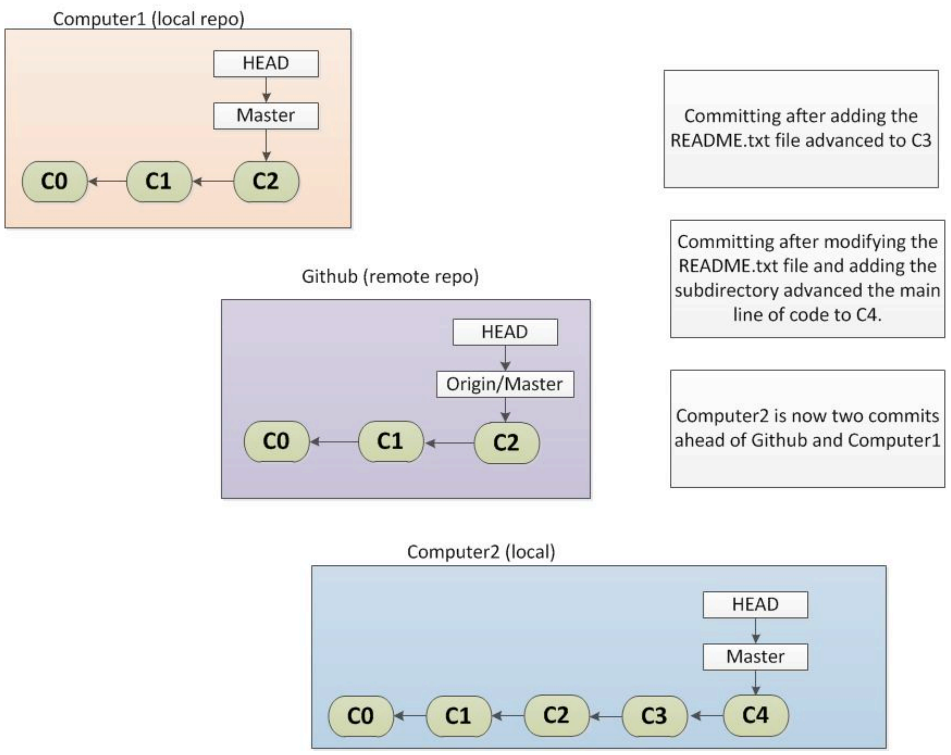
    32.3. Inside of the sub-folder, create a text file named TOWN.txt. Open it and type your hometown. Save the file.

    32.4. Track the new files

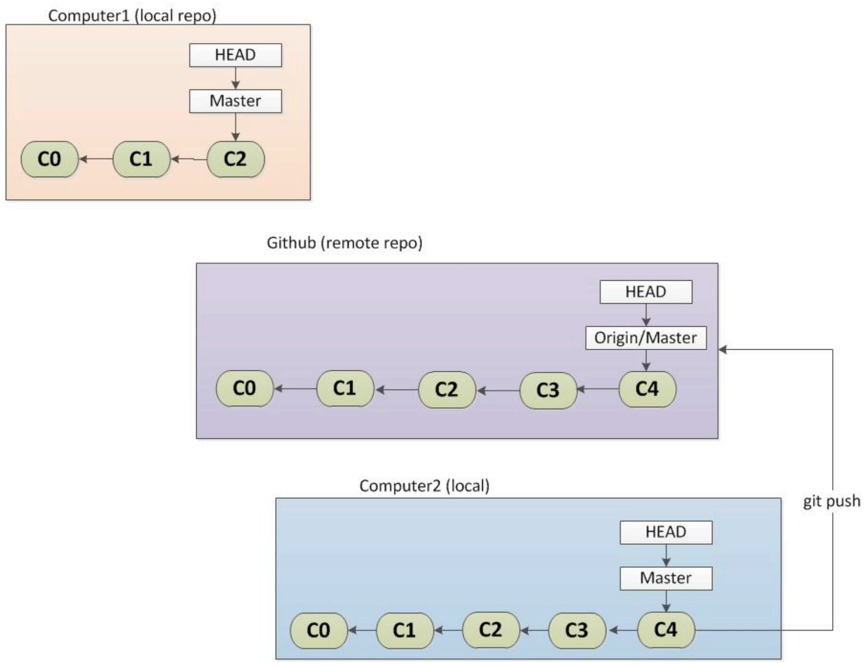
    32.5. Stage all the modified files

    32.6. Commit the files

Computer2 is now 2 commits ahead of Github and Computer1


Computer1 (local repo)
HEAD
Master
C0
C1
C2
Committing after adding the README.txt file advanced to C3
Committing after modifying the README.txt file and adding the subdirectory advanced the main line of code to C4.
Github (remote repo)
HEAD
Origin/Master
C0
C1
C2
Computer2 is now two commits ahead of Github and Computer1
Computer2 (local)
HEAD
Master
C0
C1
C2
C3
C4

32.7. Push the local repo to Github



33. View your account on Github. You should see the latest commit with the following files:

    33.1. hawkid.html with Hello World!, your name, and a comment about class.

    33.2. README.txt with your name and favorite baseball team.

    33.3. \FirstName\TOWN.txt with your hometown.

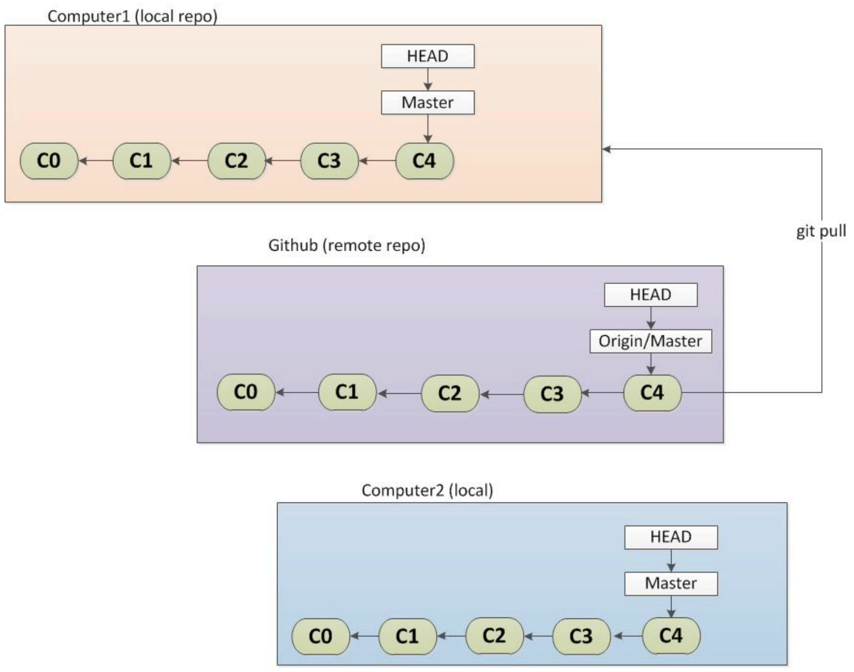
34. If you have everything is present - Congratulations!! If not, go back and troubleshoot.

35. Work with this, adding more files and making modifications, until you are comfortable with using local and remote repositories, on a single and multiple computers.

Serial, by yourself with branches

36. On Computer1, pull your remote repository to ensure your local repository is up to date and you are working with the most current version.

```
git pull <<URL>>
```

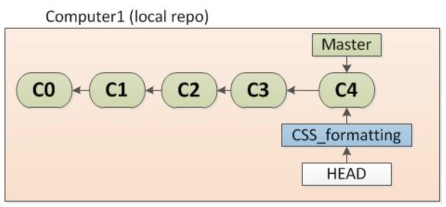


37. Create a branch named "CSS_formatting"

> **`git branch`** ***`branch_name`***
>
> **`ie. git branch CSS_formatting`**

38. Switch to a different branch (CSS_formatting)

> **`git checkout`** ***`branch_name`*** (Any changes you make now are going into this branch. Notice HEAD is now on the branch.)
>
> **`ie.   git checkout CSS_formatting`**



39. Open %hawkid%.html and add a link to a CSS file in the <head>

```
<html>
      <head>
            <title> Test Page </title>

            <link rel="stylesheet" type="text/css" href="style.css" />

      </head>
```

40. Using Git status, you should see the .html file is modified.

41. Go to the HW4 directory and create a new CSS file with the name you used above (style.css). Set the background color of your page to pink and save the file.

```
      body { background: pink; }
```

```
git status
```
-->you should now see a modified file (.html) and a new untracked file (.css)

42. Track the .css file

```
git add style.css
```

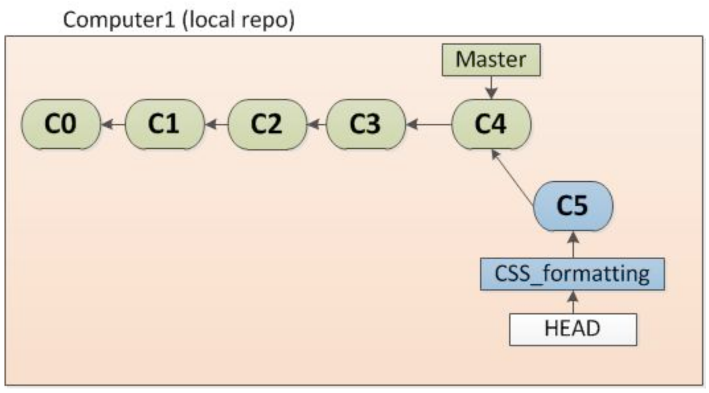
43. Stage your .html and .css files.

```
git add .
```

```
git status
```
--> you should now see two files (.html & .css) staged for commit

44. Commit them.

```
git commit -m 'Set the background color to pink.'
```



The "CSS_formatting" branch advances one commit.

45. Edit the .css file and set the font to Comic Sans. Edit the .html file to add a hyperlink to Google.

```
body { background: pink; font-family: "Comic Sans MS"; }

<a href=”http://www.google.com”> Google </a>
```

46. Stage and commit all modified files in one command (-a “adds” the files)

```
git status

git commit -a -m ‘Background color and hyperlink added‘
```

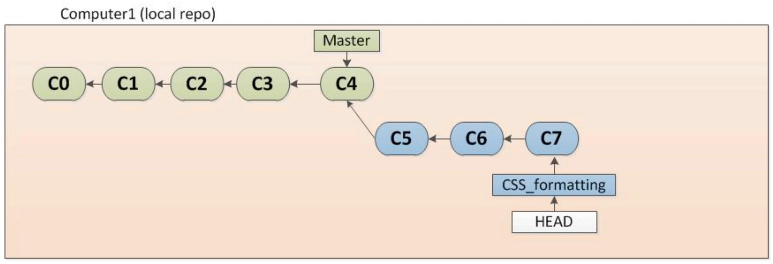
47. Edit the .css file to make <h1> text centered, bold and italic. Add an <h1> header to your html file.

```
h1 { text-align:center; font-weight:bold; font-style:italic; }

<h1> 6K:183 </h1>
```

```
git status

git commit -a -m ‘<h1> headings added‘
```



The local repo on Computer1 advances two commits .

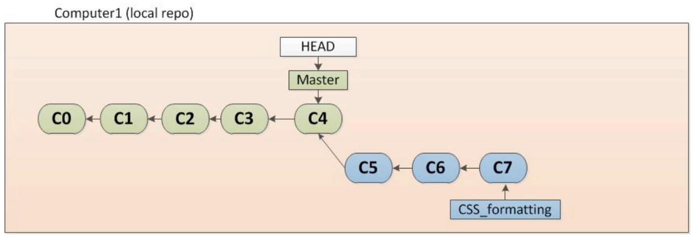
48. Check to see what branch you are on

```
git branch
```

You should see and * next to CSS_formatting indicating you are currently in the CSS_formatting branch.

49. Switch from the CSS_formatting branch to the master branch

```
git checkout master
```



50. Open %hawkid%.html. You should not see any of the CSS changes in this file. The css changes were not made on the main line of code (master branch). They were made on a topical branch (CSS_formatting).

51. Switch to the "CSS_formatting" branch. You should see the .css file in the repository. When you open the .html file you will see the link to .css in the <head>.

```
git checkout CSS_formatting
```

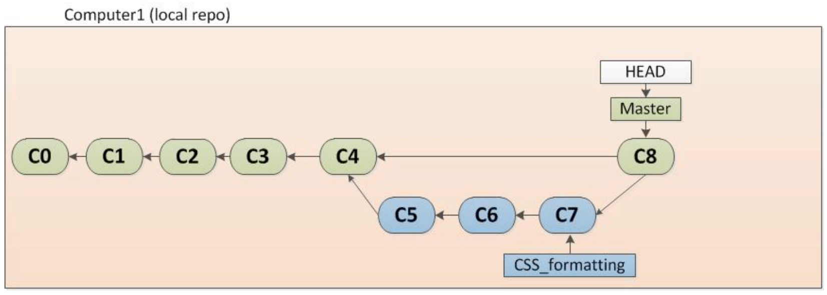
52. Now that the CSS formatting feature has been implemented and our settings made, we are done with this topical branch (CSS_formatting). We need to merge this working feature back into the main line of code.

53. Switch to the main line of code (master)

```
git checkout master
```

54. Merge the CSS_formatting branch into the main line of code (master)

```
git merge CSS_formatting
```



55. View the history of the project

```
git log
```

56. Push this local repo to the remote Github repo.

```
git push origin master
```

57. Delete the CSS_formatting topical branch from the local repo since it is not needed any longer.

```
git branch -d CSS_formatting
```

58. Clean out unneeded files from your local repository.

    58.1. Open the HW4 folder on your desktop and delete README.txt and the subdirectory named with your first name.

    58.2. From Git Bash

    ```
    git status
    ```

    The deleted files and folder now appear as deleted in red.

    58.3. Remove the files from git tracking

    ```
    git rm -r  firstnamedirectory
    ```

    The -r tells it to remove recursively (all the files within the folder)

    ```
    git rm README.txt
    ```

    58.4. Commit

    58.5. Push

59. Tag a commit as Software version 1.

    git tag -a v1.0 -m ‘My version 1.0’

    59.1. Commit

    59.2. Push

60. Work with this until you are comfortable with creating and merging branches.

Deliverables:

Zip your HW4 folder and upload to ICON Dropbox. I will also review your Github HW4 repository.

Appendix A

Git commit editing is Linux Vi

i for insert mode - to type a message

<esc> to get out of insert mode

:w to write your message (save)

:q! to quit the editor