

CS251 – Git Assignment

Submission Deadline: 10:00AM March 5th, 2018

1. Create a new Git project in <https://git.cse.iitk.ac.in/> or in <https://github.com/>
2. (Optional but recommended) Add your computer's public SSH key by following the instructions at <https://git.cse.iitk.ac.in/help/ssh/README>
3. Now that you've successfully setup a central git repository, clone it once locally in your PC. (using git clone at a folder named "user_1")
4. To simulate a multi-user usage of your newly created central git repo, clone the repository again in another folder (a folder named "user_2").
5. Let us think of activities in the first clone as user_1's activity and the activities in second clone as user_2's activity. Re-enact the activities of user_1 and user_2 in the following order.

<p>As User_1 (i.e. first clone),</p> <p>6. Add a simple hello.c program that prints "HelloWorld" in main() and compile it.</p> <pre>hello.c #include<stdio.h> void main(){ printf("Helloworld!\n"); }</pre> <p>a.out</p> <p>7. Stage and Commit the code and the *.out binary. (using git add, git commit)</p> <p>8. Push these committed changes to the origin master (using "git push" or "git push -u" to set the upstream tracking reference)</p>	
	<p>As User_2 (i.e. second clone),</p> <p>9. Pull upstream changes into your local repository. (using git pull)</p>
<p>As User_1,</p> <p>10. You're in the middle of a feature addition, so add a new line into your main().</p> <pre>hello.c #include<stdio.h> void main(){ printf("Helloworld!\n"); printf("This must be a monolithic design\n"); }</pre>	

<p>11. Stage and Commit but do not push yet as you went to grab a cup of coffee.</p>	
	<p>As User_2, 12. You're simultaneously working on the same piece of code and have added a different feature.</p> <pre>hello.c #include<stdio.h> void microkernel_sendmsg(char *); void main(){ printf("Helloworld!\n"); microkernel_sendmsg("is more portable"); } void microkernel_sendmsg(char *a){ printf("microkernel: %s\n", a); }</pre> <p>13. Stage, Commit and Push.</p>
<p>As User_1, 14. Try pushing to remote. You can now no longer push to the origin/master as your code is not up-to-date, git push will fail.</p> <p>15. Pull and Resolve the merge conflicts (in hello.c and a.out) so that all of the newly added features (the print messages and function calls) work. (using git pull, git diff)</p> <p>Manual merge conflict resolution is required in hello.c to get the following</p> <pre>#include<stdio.h> void microkernel_sendmsg(char *); void main(){ printf("Helloworld!\n"); printf("This must be a monolithic design\n"); microkernel_sendmsg("is more portable"); } void microkernel_sendmsg(char *a){ printf("microkernel: %s\n", a); }</pre>	

<p>16. The binary (a.out) might also require merge conflict resolution. But as we do not need to track binaries, delete it (a.out), add the pattern "*.out" to a .gitignore file so that the binaries won't be tracked anymore, stage, commit and push all these changes (removed a.out, updated hello.c and added .gitignore).</p>	
	<p>As User_2,</p> <p>17. Pull the updates. It is not good practise to work directly in the master branch. Create a new branch called "feature_addition_getmsg" (using git branch and git checkout)</p> <p>18. Make changes so that the hello.c now looks like this,</p> <pre>#include<stdio.h> void microkernel_sendmsg(char *); void microkernel_getmsg(char *); void main(){ printf("Helloworld!\n"); printf("This must be a monolithic design\n"); microkernel_sendmsg("is more portable"); } void microkernel_sendmsg(char *a){ printf("microkernel: %s\n", a); } void microkernel_getmsg(char *b){ //TODO: getmsg feature }</pre> <p>19. Commit and Push (since new branch you may have to use git push –set-upstream).</p>
<p>As User_1,</p> <p>20. Similarly User_1 has also created a branch called "feature_removal_sendmsg" from master with an intention to locally experiment on the master code without the sendmsg feature, thus changing hello.c to the following,</p> <pre>#include<stdio.h> void main(){ printf("Helloworld!\n"); printf("This must be a monolithic design\n"); }</pre>	

}

21. As this is a local experiment, no need to push these changes, simply add and commit.

22. Pull from remote to get new branches from User_2. The feature_addition_getmsg branch of User_2 must be now downloaded and locally accessible to you. (using git pull and git branch -a)

23. As the new "feature_addition_getmsg" from user_2 looks intriguing, you would like to test these changes with your own experiment work "feature_removal_sendmsg".

24. So create a new branch called "testing_no_sendmsg_with_getmsg" from the "feature_removal_sendmsg" branch and merge the "feature_addition_getmsg" branch into it. (using git branch, git checkout, git merge)

Resolve the merge conflict resolutions so that hello.c in "testing_no_sendmsg_with_getmsg" looks like

```
#include<stdio.h>

void main(){
    printf("Helloworld!\n");
    printf("This must be a monolithic
design\n");
}

void microkernel_getmsg(char *b){
    //TODO: getmsg feature
}
```

25. Add, Commit and Push the branch upstream.

As User_1,

26. Checkout to master branch and edit hello.c so that you introduce an error or warning.

```
#include<stdio.h>

void main(){
    printf("Helloworld!\n");
    printf("This must be a monolithic
design\n");
    microkernel_sendmsg("is more
portable");
}
```

<pre> } void microkernel_sendmsg(char *a){ printf("microkernel: %s\n", a); } </pre> <p>27. Commit and push this erroneous code.</p> <p>28. Since now you've tainted the master branch, you need to revert these changes before the boss gets to know that you've broken the production code.</p> <p>29. Use git revert to revert the master branch to the previous commit, effectively undoing the last change, and push it back asap!</p> <p>Note that this doesn't hide the tainted commit on the history of commits but only reverts it. So it is still possible to identify the culprit!</p>	
	<p>30. As User_2, make some changes to the hello.c in master branch and communicate these changes to User_1's hello.c in master branch using patch files. (using "git format-patch" on sender side to generate a ".patch" file and "git am" on receiver side to apply this patch).</p>

Submission Guidelines: Make a tarball of the "user_1", "user_2" folders containing the respective local repositories and the patch used for Step 30.