Software Engineering (C-T-303-HUGB), Week 07 Fall Semester 2018

Version Control Exercise

Main purpose of this exercise is that students strengthen their knowledge of using a Source/Version Control system, in this case Git. Make sure you have git installed, run this command in Git Bash (Windows), or in you terminal/shell:

```
git --version
```

You should see something like:

```
git version 2.19.0
```

If you don´t get similar output; you must install Git on the machine. There is a great chapter on that subject in the Pro Git Book.

Version control

Each team needs to have a version control server, we recommend that you use the free service from GitHub. Note: Github is not synonymous with Git, but simply a web based hosting service for source control using Git.

Github allows students to have free Private Repositories if you authenticate using your RU email address.

Exercise

The team will create a shared list of their favorite movies and try out the basic source control commands while doing it.

Setting up a repository

First, each team member must sign-up for a Github account (if they don't already have one).

Your team should create an Organization for the repositories. This organization is a hypothetical company where you and your team members are employees working on some cool projects. Each repository under this organization are products that your company is producing. This can be done from the organization page.

Next step is to invite the team members into that organization, which can be done from https://github.com/orgs/*organization-name*/people. We recommend you create a *team*, https://github.com/orgs/*organization-name*/teams and invite all the team members there.

With that ready, one of the team member can create a new repository for this exercise. This can be done from https://github.com/organizations/*organization-name*/repositories/new. Name the repository with some descriptive name, such as *GitExerciseInClass*.

The give the team members (or the entire team) rights to contribute to the repository. See: https://github.com/*organization-name*/*name-of-repo*/settings/collaboration

Adding files

One member of the team does the following:

1. Clone the newly created repository to your local drive. On the repository page on Github you can find a clone link that you can copy and use. Note; use "Clone with SSH"
   - In order to do this, you must set up a SSH keypair locally and connect that to your Github account. Follow the help from Github SSH help page.
2. Open up your terminal, change to some good location, such as `/home/username` or `C:\Users\username\hugb\week07` and type in the following lines:git clone git@github.com:Organization-name/repository-name.git
3. Then create a file called *movies.txt* and add one item to the list e.g. "Lilo & Stitch". Add the file under version control using `git add` as follows.git add movies.txt
4. Commit the changes with `git commit`. Add 2-3 items to the list and commit file after addition of each item. **Remember to add descriptive comments** with your changes.
5. Use **`git log`** and **`git status`** to monitor the changes that your are doing. These are the commands you will most often be using.
6. When you are happy with the changes that you have added, then push the index to Github (using `git push`) so other group members can pull the changes (using `git pull`) to their local clones.

Work on the project

Each member of the team does all the following steps:

1. Get a working clone to your computer
   - For this and the following steps, we assume you will be using SSH. In order to do this, you must set up a SSH keypair locally and connect that to your Github account. Follow the help from Github SSH help page.
2. Find and change the file put in your own items in the list, communicate within the group so you don´t do it all in the same line
3. Commit your change (with comments) Note: You´ll have to do an pull before committing if someone beat you to committing
4. Examine what happened to the file. What happened?

Conflict

Each member of the team does the following:

1. Choose one item on the list and everybody change to something new
2. Commit and update as needed. Someone at least should get a conflict.
3. Resolve the conflict.

There are multiple ways and various tools that you can use when resolving conflicts.

If you don't have any luxury of third party GUI merging tools (this is often the case when you are working on a terminal remotely) you can always resolve conflicts manually by hand. We recommend you resolve the conflict by hand in this exercise.

This is done by editing the file that is in a conflict state, remove the conflict lines within the document, add the file to index and commit the changes. The Pro Git book has a short chapter on resolving merge conflicts that is good to read when solving this part of the exercise.

Branching

Create a branch for yourself. F.ex. *master-username*. That is, you have a branch named after you. This can be done as follows.

```
git checkout -b master-username
```

`git status` or `git branch` shows you what branch your are currently working on. Change the file in your branch. Add a comment. When you push your local branch to Github up you have to do it as follows.

```
git push -u origin master-username
```

Note that you only have to do this the first time you "publish" your branch. After that you can do commit and push to the branch.

To see what branch are available or what branch is currently selected, use `git branch`. To change to a branch that has already been created, we use the git checkout command as follows.

```
git checkout branch-name
```

Merging

Merge the changes that you have added to your user branch into the master branch. If you do `git status` you should see that you are currently on your special branch. The goal is to merge the changes that you have in this branch to the master branch. Move over to the master branch, do a `git pull` and merge the branch using `git merge` as follows.

```
git merge master-username
```

That is, you are merging the master-username branch into the *master* branch. Note that *master* here is just a name, it could be *develop*, or *feature-x*, or *puppies*.

Reverting

Do a random change in your branch. -Oh no, this was not a wise change. This happens when you are coding and not concentrating.

In Git it depends how far you have gone with your mistakes how you revert them. You have the following possibilities.

1. You have done some changes locally and not added the changes to the next commit index.
2. You have done some changes that are locally and added the changes to the index.
3. You have done some changes and committed them locally
4. You have pushed your changes to the remote (Github)

For case 1 and 2 you can simply "checkout" the file again by issuing the command `git checkout filename`. Do some changes, revert the changes by checking the file out again. Note that if you have added the changes to the index area and you want to move these changes back to unindexed area, you can do `git reset filename`.

Now let us recover from case 3. Make some random changes again and commit them - don´t push to Github. If you do `git log` you can see that the commits you - have created have a unique identifier.

One possible way of removing the unwanted commits is to alter the head of your local branch. That is, move the head to a previous commit and discard commits that are in front of that commit id.

If you want to discard un-pushed commits we can use the command `git reset` as follows.

```
git reset --hard HEAD~1
```

This line moves the head of our repo to the commit that is before the current head, that is it moves the head to the previous commit. The number states how many comments we should move from. To learn more about git reset go through the man page for git-reset.

```
man git-reset
```

Do some changes and commit them and remove them using the technique described above.

For case 4. If you have pushed the changes to Github and others have pulled them to their local close - then we have to add revert commits that other developer will pull from Github. We can add a revert commit for selected commits as follows.

```
git revert commit-id.
```

This adds a new revert commit that you need to push to Github. Note when you have to make such changes, you'll need to do a "force push". Every time you find yourself in that position, be very careful to check that is what you need to do. Force pushing should be used almost exclusively on branches you have complete control over and no-one else is working on.

Try all these possibilities and try to win your way out of troubling and emberising commits and pushes.

Tagging

1. All team members agree on a final version of the movie list.
2. Create a tag/label for this version and call it "Version 1"

The book has a great chapter on Tagging see it here

To solve this part, skim though the pages and apply your knowledge from the book!

The results

The emphasis should be on doing each thing well, and understanding what you are doing. Git has a learning curve. It is not steep, but it might be long. Try to experiment as much as you can. Read the book and don't be afraid to experiment with comments. You should always be able to revert back from your mistakes.

Additional references

Pro Git Book

The book is titled Pro Git book, written by Scott Chacon and published by Apress. We recommend that students skim through the first chapters, and when you have any Git troubles, this book has the answers. The first two chapters and the next 30 pages or so should get you going.