CS 251 Outlab 5 : Git, (C)Make and Latex

Source for all comics: xkcd

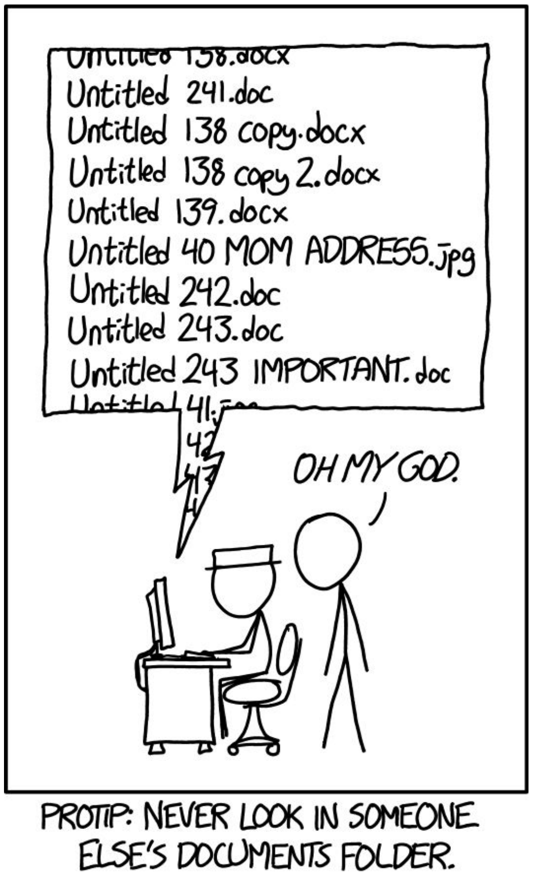
Please refer to the submission guidelines at the end of this document before submitting. If it hasn't been put up yet, it will be soon.

**** Added files required for q6 to the outlab5-resources tar file on Friday, 13 Sept, 3:15 pm. So please download it again from moodle !! ****

General Instructions

- Make sure you know what you write, you might be asked to explain your code at a later point in time.
- Parts of this lab will be graded automatically, so stick to the naming conventions strictly. We won't tolerate any kind of wrong submissions. You will get 0 for wrong submission.

  Please don't name your files like this (jk you guys are doing great)



- The deadline for this lab is **Tuesday, 24th September, 2019, at 11:55pm**

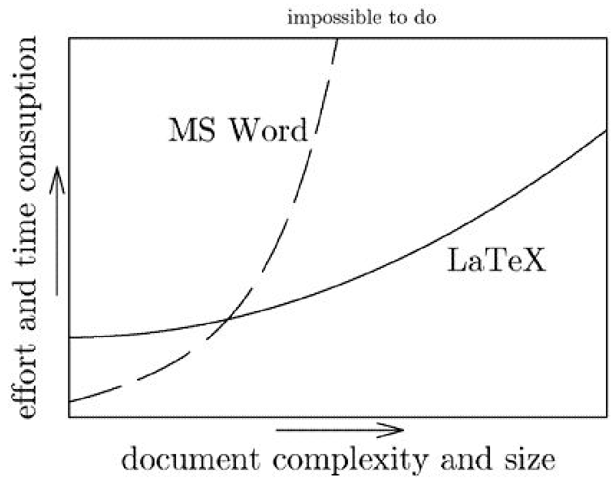
For each of these tasks you will work in groups (as usual)

---

q1 will take a lot of effort, timewise, to complete so plan accordingly

q1 - LaTeX (80 Marks + 10 Bonus Marks)

(1 out of 80 marks is for the title-page)
(It's tek not tekssss :P)



This is a PDF file that was created using LaTeX. You need to replicate the same document in PDF format using LaTeX and name it **outlab5-<team_name>.pdf**. Make sure you try to replicate all the styles, sizes, fonts, etc. used in the document provided. **Read through the PDF for better insight on how to replicate.**

You can change the text, for example, the team name in the title of the document, the names present anywhere else in the document, and/or anything which you would like to replace. But the format of the document, including the Table of Contents, Sections, Subsections, bold/italic/emphasized fonts, images, trees, etc. and so on should be replicated as it is **(Page numbering should not change)**.

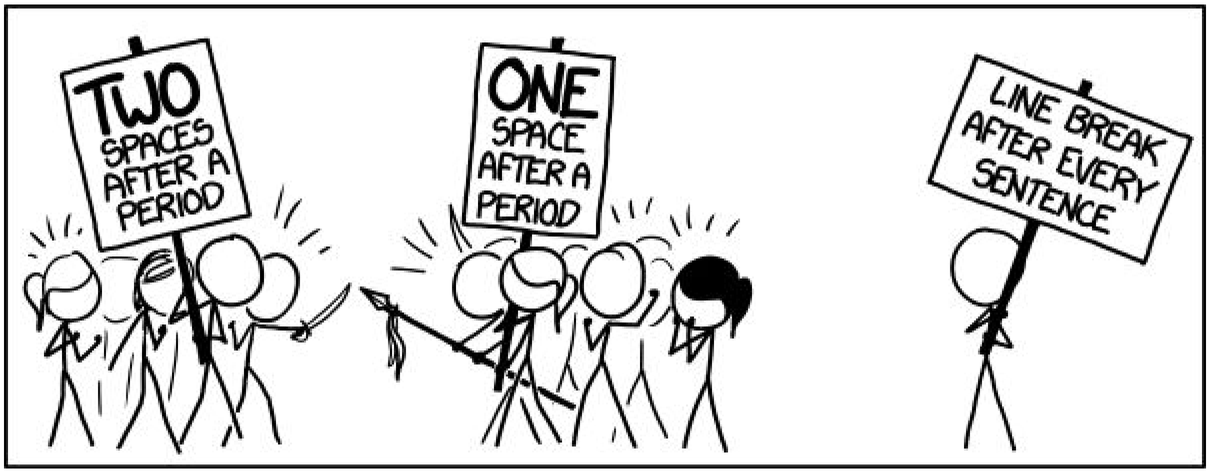
You can use online LaTeX editors like Overleaf or offline editors like Texmaker. In any case, do get accustomed to Texmaker or pdflatex because access to Overleaf might not be given during the Quiz so it'll help in practicing beforehand.

You need to submit the PDF named **outlab5-<team_name>.pdf** along with all the supporting latex files and images required for generating the PDF from the tex file in a folder named **q1**. Go through the submission guidelines for more detail.

Updates:

1. The font style used in the sample PDF is **bera** and the font size is **default**.
2. Marks division specified in the PDF file are just for an overview. Additional marks might be deducted if some significant things are not replicated. So try to replicate the PDF to as much extent possible.
3. You might need to edit the **inter-word spacing** as well to match the sample PDF exactly. You may use any package which you might find useful for that.
4. In case you might require, the divisions within the **references** is as follows:
   author, title, journal, year (though some of these divisions might be missing in some) Place in the text for these by yourselves accordingly.
5. Make sure that your LaTeX file is successfully getting **compiled** correctly with appropriate PDF replication using the **pdflatex** command. It'll be used for evaluation of this question so please make sure that your code is working with the specified command.

(Bonus marks will be given only if you score full (80/80) in the rest)



Let's get started with **git**

This task is inspired from [here]

q2 - Welcome to GitHub (15 Marks + 5 Bonus)

Updates on PIAZZA:

1. **“No other files must be present”** means no additional files apart from what was present during clone and the file you make. There might be even more files than instructors.json and README when you clone, depending on when you clone.
2. **REPEAT for every student if there are more than 3. That’s what the “...” represents in file format. In case your group had a 4th (or more) member and for some reason you haven’t included him/her, please send a new pull request with exactly 1 commit, with details of studentX (bonus details too if you have attempted it).**

BONUS - 5 marks - please look at the end of this task

student1 will be the student with the first roll number in ascending order sorted fashion, just like how you submit your Outlabs. student2 will be second, and so on.

What follows are a set of commands/actions (offline and online) that will ultimately constitute your basic knowledge of git, and GitHub.

`git fork`

“Forking” is creating a copy of someone else’s project. It does not mean you own the project now, its is a direct result of open source- to distribute software and your brain child to the community, in return receive contributions from the community. Fork allows you to experiment and contribute to others work, without affecting their own project until they approve of your changes. It could also be to use someone else’s code as a starting point for your amazing idea. Why reinvent the wheel?

1. student1 has to create a fork of this on his/her GitHub account. This can be public, as this task is not something we expect groups to sneak into each other’s code for.
2. student1 must now add student2 and student3 as collaborators on this repo.
3. student2 and student3 must accept collab invitations via GitHub.

`git clone REPO`

Each student downloads a “clone” of this newly forked repository on their computer. This can be done via https or ssh, doesn’t matter. Make sure you have cloned **your own** repository and not mine.

`git add <team_name>.json`

1. student2 makes a file called <team_name>.json (with your team name, e.g. PeekABoo.json). Inside it, these details of your own team should be filled in this exact format.

```
{
```

<roll_no2>: <github-id2>,

}

(the format may not be correctly displayed here, please visit the repository)

(<detail>2 refers to details of student2)
This is a json format file (like you have seen in Outlab 1) and all roll_numbers and github-ids must be strings. An example of how the file looks like is already present as instructors.json - replace names with roll numbers (ascending order) and empty strings with github-id (like mine!).

2. Learn how to use `git add`. **"git add"** this new file with and appropriate argument. Do not add or remove any other files. For our dear mac users, we have ensured that ".DS_Store" files are not accidentally added. Thank us later.

```
git status
```

Get the status of your repository. What files have been modified? Which are staged for commit? Which are just new? These questions will be answered, understand this, it will not be graded ofcourse.

```
git commit -m "<team_name>-<roll_no2>"
```

student2 continues must now commit his changes (must have added first) using the message given above, replacing appropriate details

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

student2 "pushes" his changes onto the group's repository. Yay! You've made your first contribution on GitHub.

```
git pull
```

student3 should now "git pull" on his computer on this repo.

REPEAT THIS PROCESS, make changes to the file <team_name>.json, commit with the right commit message, and push.
(REPEAT for every student if there are more than 3)
student1 should now repeat this process, pull, make changes, commit, and push. His roll number and id should be first

The file should look something like this

{

<roll_no1>: <github-id1>,

<roll_no2>: <github-id2>,

<roll_no3>: <github-id3>,

...

}

No other files must be present

`git pull request`

student3 must go to GitHub, and make a pull request to my repository. This can be done by visiting the original repository, making a new pull request, and selecting **compare across forks**. Write an appropriate message so we know which team is responsible for the request.

Your pull request will be accepted, sooner or later, and don't worry about merge conflicts, at least in this task

BONUS

After the above parts are done, do the following

Currently, every student has the same copy of the file (after they git pull) and so does the remote repository.

Each student must change his entry in the file as follows, such that now student1's copy of the file looks like

{

<roll_no1>: [

<github-id1>,

<name1>,

],

<roll_no2>: <github-id2>,

<roll_no3>: <github-id3>,

...

}

(all strings)

And student2's looks like

```
{
        <roll_no1>: <github-id1>,
        <roll_no2>: [
                <github-id2>,
                <name2>,
        ],
        <roll_no3>: <github-id3>,
        ...
}
```

Etc.

Now sequentially, student1 must push his changes, student2 must pull, resolve conflicts if, then push, and finally student3 must pull, resolve conflicts if any, then push. **All this must be done on the same branch**

The final file must look like this -

```
{
        <roll_no1>: [
                <github-id1>,
                <name1>,
        ],
        <roll_no2>: [
                <github-id2>,
                <name2>,
        ],
        <roll_no3>: [
                <github-id3>,
                <name3>,
        ],
        ...
}
```

After all changes have been pushed, send a second pull request.

A team should send no more than 2 pull requests, and no more than 1 if not attempting the bonus. Even if you attempt the bonus, you cannot fetch (git joke) additional marks in the main task with a second pull request. For every extra pull request you will be penalised by 10 %.

Note: For the next two tasks, submit the *.git* folder as well. This is a hidden folder (visible using `ls -a`) and could be missed out by you in copying. Please see submission format at the end of this document for more details.

q3 - Git Rebasing (15 Marks)

UPDATES ON PIAZZA:

1. you can add the pngs later. Don't commit them to any branch. Add the README whenever you make changes, and commit only when asked to.
2. For part 9 - Rebase from wherever necessary to match the image given as much as possible. If you use extra commands after rebase, mention the exact command in the README.

Git has excellent features called branching and rebasing. To show the power of branching, let us have an example (an example that is very common in open source projects). Suppose there is a main code base for a project, and you have come up with an amazing feature, but you do not know whether the feature will work or not. Now, you can commit your changes and come back to a previous snapshot of your code, but that gets very tricky if you mess up midway. To get all your features coded up "separately" and then "merge" your work with the main work once you are satisfied, you can use a git "branch" to diverge from the main code, develop your feature, and then merge the branch. Branching helps to clean up your work environment, and make changes permanent (by merging into the main branch) only when you are completely satisfied with the code (in terms of testing it, debugging, etc).

Rebasing is very useful when you want to write commits on top of the tip of the other branch instead of a normal merge. For example, if you have worked on a feature, and you want to replay the changes on top of the master branch, you can rebase instead of merge.

For this assignment, do the following.

Caution: Perform all the steps **sequentially**.

1. Create an init repository. You do not need to add a remote url as you will not be putting this up online. A master branch will be created automatically.
2. Add 2 commits. Add/remove files as you wish in the first commit. Create a file **demo.txt** and leave it blank in the second commit.
3. Create a new branch "feature1". Add some content to the file **demo.txt**. Commit the changes.

4. Go back to the "master" branch. Add some other content to the file **demo.txt.** Commit the changes.
5. Add some more files and create at least 3 commits (do not modify or delete the **demo.txt** in any of the commits).
6. Go back to the "feature1" branch. Add some more commits (do not modify or delete the **demo.txt** in any of the commits).
7. Take a screenshot of the graph and put it in the submission folder with the name step7.png, by printing the graph on the terminal (see *git log*). Write the command and output in your README file.
8. At this point. Your graph should be something like this (this is a conceptual view).

```
        A1---A2---A3---A4   feature1
       /
  O1---O2---B1---B2---B3   master
```

9. Now, go to the ``feature1'' branch and replay the commits on the master branch using the rebase command. Do you notice any conflicts? Why do you think they must have occurred?

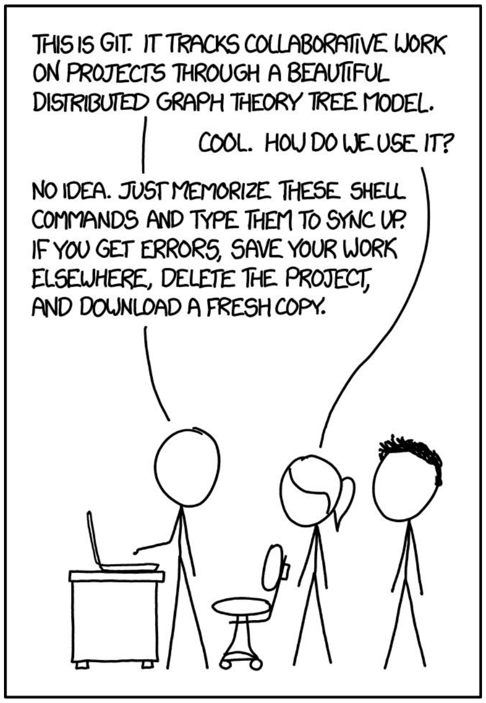
Also, do you notice any merge commit in the log? Why/Why not? Write the inferences in your README.

If you encounter any conflicts, skip them (also write the command used to skip them in README).

Note: Rebasing the wrong branch can be catastrophic if you maintain huge codebases using git. For example, in this case, after the rebasing, the resulting log of "master" branch should be (the order of commits can differ):

```
                       A1---A2---A3---A4   master
                      /
  O1---O2---B1---B2---B3
```

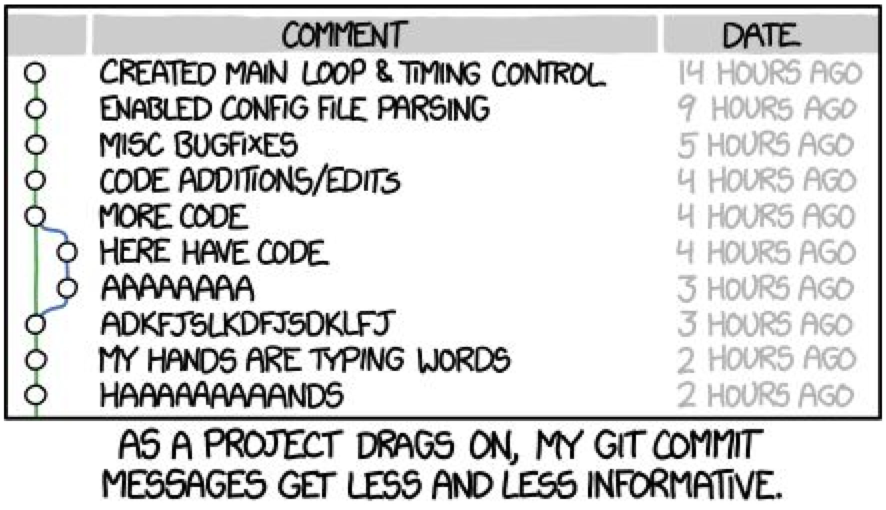
In other words, the master branch should contain all the commits (except the ones that were skipped).


THIS IS GIT. IT TRACKS COLLABORATIVE WORK ON PROJECTS THROUGH A BEAUTIFUL DISTRIBUTED GRAPH THEORY TREE MODEL.
COOL. HOW DO WE USE IT?
NO IDEA. JUST MEMORIZE THESE SHELL COMMANDS AND TYPE THEM TO SYNC UP. IF YOU GET ERRORS, SAVE YOUR WORK ELSEWHERE, DELETE THE PROJECT, AND DOWNLOAD A FRESH COPY.

q4 - Git Squashing (10 Marks)

UPDATES ON PIAZZA:

1. The image for the final graph appears to be incorrect. It may not be the final answer, please follow the problem statement and refer to the image only for reference on commit message format



COMMENT	DATE
CREATED MAIN LOOP & TIMING CONTROL	14 HOURS AGO
ENABLED CONFIG FILE PARSING	9 HOURS AGO
MISC BUGFIXES	5 HOURS AGO
CODE ADDITIONS/EDITS	4 HOURS AGO
MORE CODE	4 HOURS AGO
HERE HAVE CODE	4 HOURS AGO
AAAAAAAA	3 HOURS AGO
ADKFJSLKDFJSDKLFJ	3 HOURS AGO
MY HANDS ARE TYPING WORDS	2 HOURS AGO
HAAAAAAAANDS	2 HOURS AGO

AS A PROJECT DRAGS ON, MY GIT COMMIT MESSAGES GET LESS AND LESS INFORMATIVE.

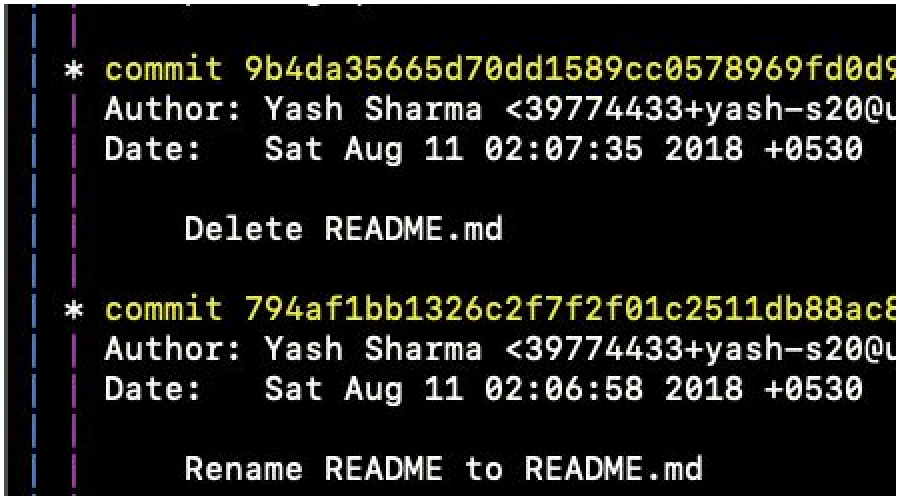
Git squashing is an excellent way to keep commit histories clean, organized, and formal. This is in organization/developers typically do at the end of a project to keep…… ah I'll just let you read this.

TASK

The task is simple. In the resources provided, there is a folder named q4. These are remnants of what was an outlab from last year. It's remote url has been detached.

`git log --all --graph`

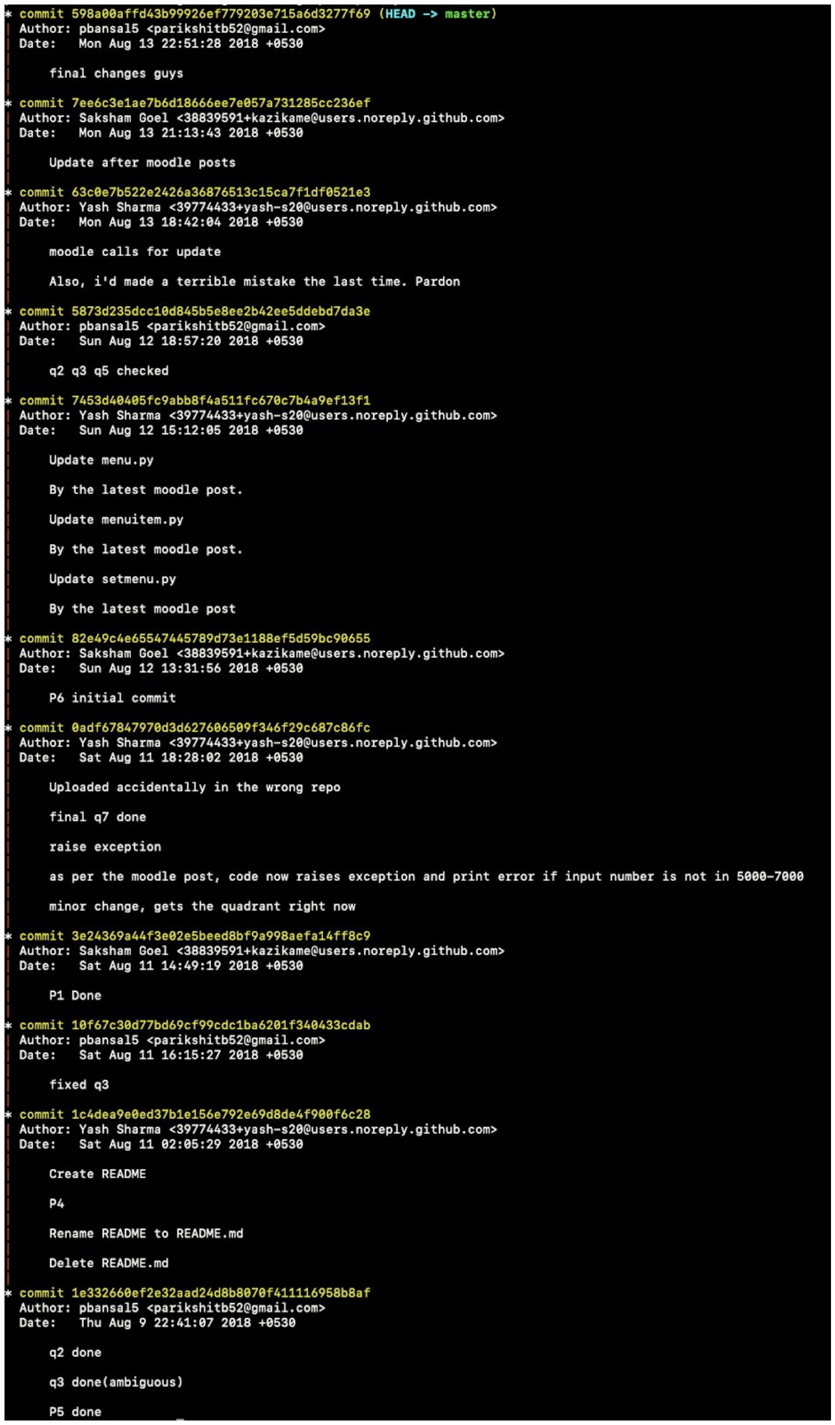
This command will give you the entire commit history of this project. It has 23 commits in total, from three different users, even on different bases that were later merged. Unfortunately, most of these is from a particularly irritating teammate who commits pretty much every small change.



```
| |
| * commit 9b4da35665d70dd1589cc0578969fd0d9
| | Author: Yash Sharma <39774433+yash-s20@u
| | Date:   Sat Aug 11 02:07:35 2018 +0530
| |
| |     Delete README.md
| |
| * commit 794af1bb1326c2f7f2f01c2511db88ac8
| | Author: Yash Sharma <39774433+yash-s20@u
| | Date:   Sat Aug 11 02:06:58 2018 +0530
| |
| |     Rename README to README.md
```

Your task is to squash all consecutive commits from a user into a single commit. Once you run the right commands, the output of `git log --all --graph` will look like this.

EDIT: I have fixed the commit graph below



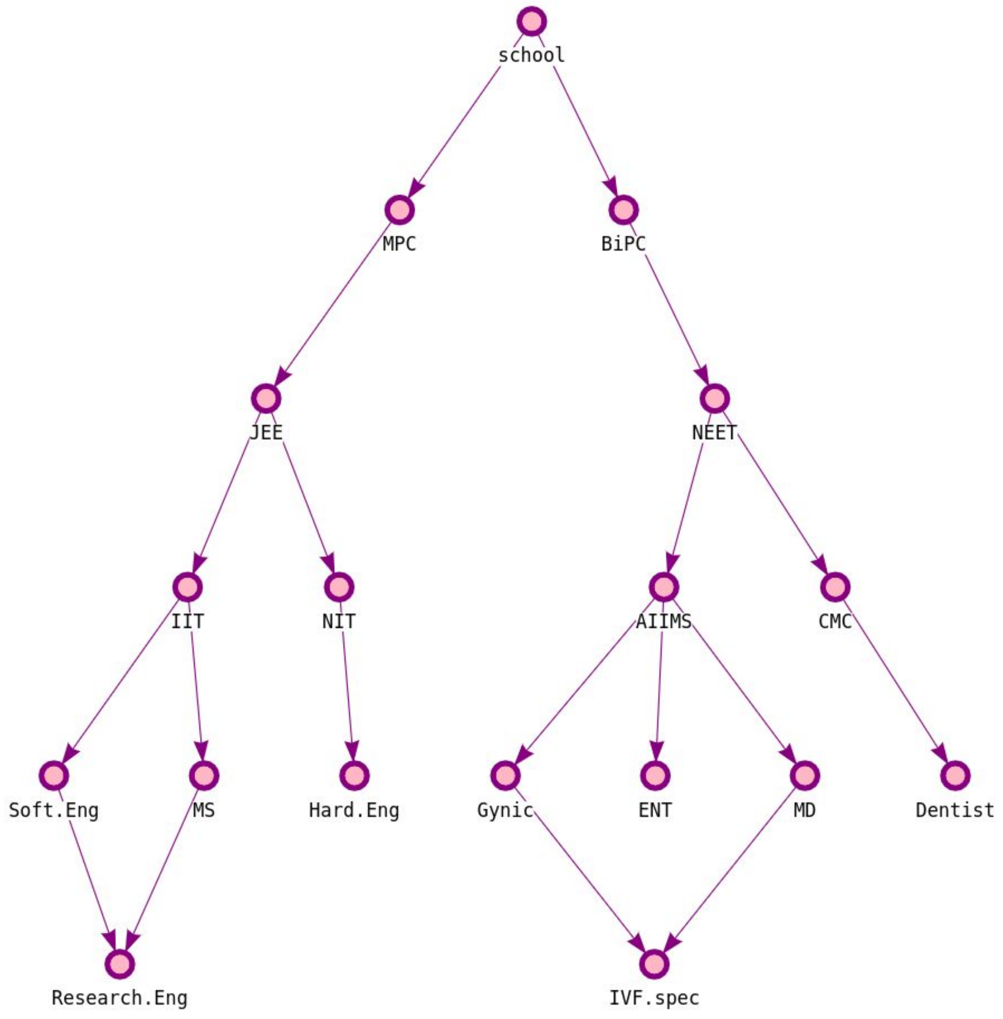
```
* commit 598a00affd43b99926ef779203e715a6d3277f69 (HEAD -> master)
| Author: pbansal5 <parikshitb52@gmail.com>
| Date:   Mon Aug 13 22:51:28 2018 +0530
|
|     final changes guys
|
* commit 7ee6c3e1ae7b6d18666ee7e057a731285cc236ef
| Author: Saksham Goel <38839591+kazikame@users.noreply.github.com>
| Date:   Mon Aug 13 21:13:43 2018 +0530
|
|     Update after moodle posts
|
* commit 63c0e7b522e2426a36876513c15ca7f1df0521e3
| Author: Yash Sharma <39774433+yash-s20@users.noreply.github.com>
| Date:   Mon Aug 13 18:42:04 2018 +0530
|
|     moodle calls for update
|
|     Also, i'd made a terrible mistake the last time. Pardon
|
* commit 5873d235dcc10d845b5e8ee2b42ee5ddebd7da3e
| Author: pbansal5 <parikshitb52@gmail.com>
| Date:   Sun Aug 12 18:57:20 2018 +0530
|
|     q2 q3 q5 checked
|
* commit 7453d40405fc9abb8f4a511fc670c7b4a9ef13f1
| Author: Yash Sharma <39774433+yash-s20@users.noreply.github.com>
| Date:   Sun Aug 12 15:12:05 2018 +0530
|
|     Update menu.py
|
|     By the latest moodle post.
|
|     Update menuitem.py
|
|     By the latest moodle post.
|
|     Update setmenu.py
|
|     By the latest moodle post
|
* commit 82e49c4e65547445789d73e1188ef5d59bc90655
| Author: Saksham Goel <38839591+kazikame@users.noreply.github.com>
| Date:   Sun Aug 12 13:31:56 2018 +0530
|
|     P6 initial commit
|
* commit 0adf67847970d3d627606509f346f29c687c86fc
| Author: Yash Sharma <39774433+yash-s20@users.noreply.github.com>
| Date:   Sat Aug 11 18:28:02 2018 +0530
|
|     Uploaded accidentally in the wrong repo
|
|     final q7 done
|
|     raise exception
|
|     as per the moodle post, code now raises exception and print error if input number is not in 5000-7000
|
|     minor change, gets the quadrant right now
|
* commit 3e24369a44f3e02e5beed8bf9a998aefa14ff8c9
| Author: Saksham Goel <38839591+kazikame@users.noreply.github.com>
| Date:   Sat Aug 11 14:49:19 2018 +0530
|
|     P1 Done
|
* commit 10f67c30d77bd69cf99cdc1ba6201f340433cdab
| Author: pbansal5 <parikshitb52@gmail.com>
| Date:   Sat Aug 11 16:15:27 2018 +0530
|
|     fixed q3
|
* commit 1c4dea9e0ed37b1e156e792e69d8de4f900f6c28
| Author: Yash Sharma <39774433+yash-s20@users.noreply.github.com>
| Date:   Sat Aug 11 02:05:29 2018 +0530
|
|     Create README
|
|     P4
|
|     Rename README to README.md
|
|     Delete README.md
|
* commit 1e332660ef2e32aad24d8b8070f411116958b8af
  Author: pbansal5 <parikshitb52@gmail.com>
  Date:   Thu Aug 9 22:41:07 2018 +0530

      q2 done

      q3 done(ambiguous)

      P5 done
```

q5 - Career Paths [10 Marks]



Previous figure lists a subset of [popular] career paths and the path one has to take for that, in the form of a directed acyclic graph. In this graph each node is annotated with a name and contains 0 or more parents.

In many senses this is the same as a Makefile. Assume that each node is a Makefile target with name, as node's name and any parent of a node is a dependency of target. With this specification your task is to write a makefile named [guess what...] **Makefile** which contains one rule for one node. The command of each target is simply echoing [only] the target's name. You can use @<bash-command> for this. **Be careful about the names of nodes as it will be autograded.**

For ex.
In green cells, the cell in the first row is command, corresponding cell in second row is expected result. Red cells represent wrong output.

$ make JEE	$ make AIIMS	$ make school	$ make school
school MPC JEE	school BiPC NEET AIIMS	school	echo school school

If there is more than one dependency resolve the leftmost dependency first.

$ make Research.Eng	$ make Research.Eng
school MPC JEE IIT Soft.Eng MS Research.Eng	school MPC JEE IIT MS Soft.Eng Research.Eng

Submit **Makefile**.

Note : **DO NOT HARDCODE THE ENTIRE DEPENDENCY GRAPH** (think how you use make to make this question easier !!) and we will **penalise** you if the number of 'echo' commands in your Makefile exceed **20**.

----

q6. C you again [30 Marks]

C was one of the first human readable programming languages designed for people in all domains (business and scientific community). C is an imperative procedural language created by Dennis Ritchie. Unix and Linux kernels are built in C (and assembly) language. Due to this C code runs faster than any other language. But still C was a very low level language, meaning that bigger **abstractions** were not readily available in it.

To deal with this Bjarne Stroustrup created C++ with objectives of speed and abstraction in mind. The core of his creation was something called a **class.** Thus C++ became one of the first object oriented languages.

C has a simple architecture and few paradigms which makes it simple to use. Remember the zen of python (*There should be one—and preferably only one—obvious way to do it.*) But due to the sheer flexibility of C++ (with several paradigms it posses) it became very difficult to maintain standards in bigger C++ projects.

One of the biggest misconceptions in this area is that C and C++ are the same language. C is not object oriented while C++ is. Acknowledge the difference.

Today we will learn about compiling & linking C/C++ code in various ways using raw commands, Makefile and CMake. This is one of the ways to gain insights into how exactly things work in C/C++ world.

Create a Makefile named **rawmake** (can be run using **make -f rawmake**) and fill it up as instructed in the below tasks. Source and Header files are provided in outlab5-resources/q6 directory. Assume that **rawmake** file will be placed directly under q6/ directory. The dir structure after creating **rawmake** file will be as follows

```
q6/
├── helloworld.cpp
├── myengine
│   ├── myengine.cpp
│   └── myengine.hpp
├── mygame
│   └── mygame.cpp
├── rawmake
└── usespthread.cpp
```

Task 1 [1 point]

Add a rule to compile **helloworld.cpp** to form **helloworld** executable.
Command run while checking: **make -f rawmake helloworld**

Task 2 **[2 points]**
Add a rule to compile **usespthread.cpp** to form **usespthread** executable. This file uses pthread library used to create and manage threads. You will be using this a lot in your Operating Systems course.
In order to properly compile this file you need to special use a flag for g++.
Command run while checking: **make -f rawmake usespthread**

Task 3 **[4 points]**
In this task we build a library using files **myengine.hpp** and **myengine.cpp.** There are actually two types of libraries categorized based on the way the library is linked to the main file. Dynamic and Static libraries. This and This are good reads on compiling static and dynamic libraries. This is a good read which distinguishes one from the other.

Use the given resources to add rules to make dynamic and static libraries. Dynamic library created should have name **libMyEngineDynamic.so** and similarly static **libMyEngineStatic.a**
Command run while checking:s **make -f libMyEngineDynamic.so [2 points]**
Command run while checking: **make -f rawmake libMyEngineStatic.a [2 points]**

Task 4 **[4 points]**
In this task we try to install the above built libraries into the system (requires sudo permission). Add a PHONY rule named **installdynamic** which installs dynamic version (.so file) to **/usr/local/lib/** and corresponding headers to **/usr/local/include/**

Do the same for static version (.a file) by creating a rule **installstatic**
Command run while checking: **make -f rawmake installstatic [2 points]**
Command run while checking: **make -f rawmake installdynamic [2 points]**
These rules should have previously created libraries as dependencies, so that they are built first (if not already built) and then installed.

Task 5 **[2 points]**
Now we use the installed libraries in **mygame.cpp**
Add a rule to compile **mygame.cpp** with the static library installed in the previous task and produce binary name **mygamestatic**.
Command run while checking: **make -f rawmake mygamestatic**

Task 6 **[2 points]**
Add a rule to compile **mygame.cpp** with the dynamic library installed in the previous task and produce binary name **mygamedynamic**.

Command run while checking: **make -f rawmake mygamedynamic**

Task 7 **[2 points]**

Also add a PHONY rule to **clean** all the generated intermediates and binaries (.o .a. .so and other binaries)
Command run while checking: **make -f rawmake clean**

Task 8 **[9 points]**

Now your job is to do the same tasks using CMake.
To do this you need to create a **CMakeLists.txt** file (directly under q6/ directory) which generated Makefile to build and install targets in the same manner as above tasks 1 - 4

1. **helloworld** binary from **helloworld.cpp [1 point]**
2. **usespthread** binary from **usespthread.cpp [2 points]**
3. **libMyEngineDynamic.so** and **libMyEngineStatic.a** libraries from **myengine.hpp** and **myengine.cpp [2 + 2 points]**
4. Instead of installing two types of libraries separately, it should now install both of the libraries and the corresponding header file **myengine.hpp** to **/usr/local/lib/** and corresponding headers to **/usr/local/include/** respectively. The command used will be **sudo make install [2 points]**
5. Observe that the makefile that **CMakeLists.txt** creates already contains PHONY rule clean which removes all the targets and intermediates generated

Command run while checking: (from q6 dir)

1. **mkdir build**
2. **cd build**
3. **cmake ..**
4. **make** (should create helloworld, usespthread, libMyEngineStatic.a, libMyEngineDynamic.so)
5. **sudo make install** (should install libMyEngineStatic.a, libMyEngineDynamic.so and header into specified paths appropriately)

Task 9 **[4 points]**

Now we want to do tasks 5 and 6 using CMake. To do this create a **CMakeLists.txt** file in **q6/mygame** directory. This file should compile **mygame.cpp** with the static library installed in the previous task and produce binary name **mygamestatic** and similarly compile **mygame.cpp** with the dynamic library installed in the previous task and produce binary name **mygamedynamic [2 + 2 points]**

Command run while checking: (from mygame dir)

1. **mkdir build**
2. **cd build**
3. **cmake ..**
4. **make** (should create **mygamestatic** and **mygamedynamic**)

The dir structure after creating **rawmake** file and two CMakeLists.txt files will be as follows

```
q6/
├── CMakeLists.txt                    (For task 8)
├── helloworld.cpp
├── myengine
│   ├── myengine.cpp
│   └── myengine.hpp
├── mygame
│   ├── CMakeLists.txt                (For task 9)
│   └── mygame.cpp
├── rawmake                           (For tasks 1-7)
└── usespthread.cpp
```

Make sure the directory structure like shown above and **submit the entire q6/ directory** during submission.

Submission Guidelines

DO NOT USE any other method of compressing your folder.

Please strictly use this command to tar gzip your submission as -

```
tar -zcvf outlab5-<team_name>.tar.gz outlab5-<team_name>/
```

We enforce this because it ensures our script can decompress your submission without hiccups.

Submissions that do not follow this will not be graded in this lab

```
outlab5-<team_name>
├── q1
│   ├── outlab5-<team_name>.pdf
│   ├── outlab5-<team_name>.tex
│   ├── bib file (any name)
│   ├── image files
│   ├── ....
│
├── q3
│   ├── README
│   ├── step7.png
│   ├── other screenshots (png files)
│   └── .git
│       ├── FETCH_HEAD
│       ├── HEAD
│       ├── branches
│       ├── ....
├── q4
│   ├── .git
│   │   ├── FETCH_HEAD
│   │   ├── HEAD
│   │   ├── branches
│   │   ├── ....
│   ├── P1
│   │   └── prime.py
│   ├── P2
│   │   └── choice.py
│   ├── ....
│
├── q5
│   └── Makefile
│
└── q6
    ├── CMakeLists.txt
    ├── helloworld.cpp
    ├── myengine
    │   ├── myengine.cpp
    │   └── myengine.hpp
    ├── mygame
    │   ├── CMakeLists.txt
    │   └── mygame.cpp
    ├── rawmake
    └── usespthread.cpp
```