CS406: Compilers

Programming Assignments - Setup and Submission instructions

The instructions remain the same for all assignments. Please replace the assignment number in the examples with the number of the assignment you are submitting.

1 Setup

Create a Github account (if you do not already have one). This is the account you should use to create and submit all of your assignments this semester.

Fill the Google form sent earlier and inform the TA and Instructors with your GitHub username by Wednesday, 15/1/2020.

Your github account should be your name (or something very close to your name, if someone else has taken the account of your name). Do not use any funny account name.

Please understand why most companies assign fistname.lastname for their employees' accounts. Nobody has time to remember the relationships between account names and real names. The easiest solution is to use real names for account names.

The teaching staff reserves the right to reject your assignments if your github account does not reflect your name.

1. Create a git repository for the assignment.

   Log in to your Github account. Then log in to CS406's Moodle/Piazza account, and find the announcement for PA0 and click the link. This will create a repository on Github for the assignment (you will follow a similar procedure for all future assignments). Make sure that the repository is called 'IITDhCSE/PA0-<your GitHub username here>'.

2. Clone the repository to develop your assignment

   Cloning a repository creates a local copy. Change your directory to whichever directory you want to create your local copy in, and type:

   ```
   > git clone git@github.com:IITDhCSE/pa0-<your GitHub username here>.git PA0
   ```

   This will create a subdirectory called `PA0`, where you will work on your code.

   In this command: `git clone` copies a repository. `git@github.com:IITDhCSE/pa02-<your GitHub username here>.git` tells `git` where the server (remote copy) of your code is. `PA0` tells `git` to place the code in a local directory named `PA0`

   If you change to directory `PA0` and list the contents, you should see the files you will need for this assignment:

   ```
   > cd PA0
   ```

   ```
   > ls
   ```

   And you should see all of the files, including the file you will need to edit, `Makefile`

3. As you develop your code, you can commit a local version of your changes (just to make sure that you can back up if you break something) by typing:

   ```
   > git add <file name that you want to commit>
   ```

   ```
   > git commit -m ‘‘< describe your changes>’’
   ```

`git add <filename>` tells `git` to "stage" a file for committing. Staging files is useful if you want to make changes to several files at once and treat them as one logical change to your code. You need to call `git` add every time you want to commit a file that you have changed.

`git commit` tells `git` to commit a new version of your code including all the changes you staged with `git add`. Note that until you execute `git commit`, none of your changes will have a version associated with them. You can commit the changes many times. It is a good habit committing often. It is very reasonable if you commit every ten minutes (or more often).

Do not type `git add *` because you will likely add unnecessary files to the repository. When your repository has many unnecessary files, committing becomes slower. If the unnecessary files are large (such as executables or core files), committing can take several minutes and your assignments may be considered late.

4. To copy your changes back to Github (to make sure they are saved if your computer crashes, or if you want to continue developing your code from another machine), type

   ```
   > git push
   ```

If you do not push, the teaching staff cannot see your solutions.

2 Write a Makefile (PA0 task)

Edit the `Makefile` given to you in the main directory of your repository (we will be using the `Makefile` to drive the building and testing process of all of your code.

Create a target called `team` that first prints out your team name (the same team name you used to create Github repo), then prints out the following information for each member of the project:

1. Full name (first and last)
2. IITDh email ID

For example, if the instructor and a TA Pavan—this course has no TA for this semester—were in a team called `Instructors`, typing

```
make team
```

would print the following:

```
Team: Instructors

Nikhil Hegde
nikhilh@iitdh.ac.in

Pavan Patil
pavan@iitdh.ac.in
```

3 Submitting your code

You will use `git`'s "tagging" functionality to submit assignments. Rather than using any submission system, you will use `git` to *tag* which version of the code you want to grade. To tag the latest version of the code, type:

```
> git tag -a <tagname> -m ‘‘<describe the tag>’’
```

This will attach a tag with name <tagname> to your latest commit. Once you have a version of your program that you want to submit, run the following commands:

```
> git tag -a submission -m "Submission for PA0"
> git push --tags
```

This will create a tag named "submission" and push it to the remote server. The grading system will check out whichever version you have tagged "submission" and grade that. If you want to update your submission (and tell the grading system to ignore any previous submissions) type:

```
> git tag -a -f submission -m "Submission for PA0"
> git push -f --tags
```

`> git tag -a -f submission -m "Submission for PA0"` overwrites the tag on the local repository.

`git push -f -tags`, overwrites the tag on the local repository.

These commands will overwrite any other tag named submission with one for the current commit. Please be careful about the following rules:

1. For each assignment, you should tag only one version with "submission". It is your responsibility to tag the correct one. You CANNOT request regrading if the grading program retrieves the version that you do not want to submit.
2. After tagging a version "submission", any modifications you make to your program WILL NOT BE GRADED (unless you update the tag, as described above).
3. The grading program starts retrieving soon after the submission deadline of each assignment. If your repository has no version tagged "submission", it is considered that you are late.
4. The grading program checks every student's repository 120 hours after the submission deadline. If a version tagged "submission" is found, the grading program retrieves and grades that version.
5. The grading program uses only the version tagged "submission". It does NOT choose the higher score before and after the submission deadline. If a later version has the "submission" tag, this later version will be graded with the late discount. Thus, you should tag a late version with "submission" only if you are confident that the new score, with the late discount, is higher.
6. The time of submission is the time when you push the code to the repository, not the time when the grading program retrieves your code. If you push the code after the deadline, it is late. Even though you push before the grading program starts retrieving your program, it is still considered late.
7. You should push at least fifteen minutes before the deadline. Give yourself some time to accommodate unexpected situations (such as slow networks).
8. You are encouraged to tag partially working programs for submission early. In case anything occurs (for example, your computer is broken), you may receive some points. Please remember to tag improved version as you make progress.

Do not send your code for grading. The only acceptable way for grading is to tag your repository.

Under absolutely no circumstance will the teaching staff (instructors and teaching assistants) debug your programs without your presence. Such email is ALWAYS ignored. If you need help, go to office hours, or post on Piazza.