ENR 259 Spring 2017
Team Github Account

Notify your coach of your team name and a repository will be assigned to your team.

Purpose

The purpose of a team github account is for keeping track of changes and allowing multiple members to work on the same code and files without interfering with others. Github is a version control system which essentially saves an instance in memory every time you "commit". If you ever make a mistake and need to view a file or code from a time in the past you can with github. In addition, this allows coaches to have visibility of your files and code at all times so if you ever need help they can simply view your work online.

Github Online

When setting up an account, using your student account will give you added benefits.

At any time, you can view files or changes and download files by visiting the repository website. To view your work online simply visit the following address, replacing ***repository-name*** with the name of your repository.

https://github.com/mcc-robotics/***repository-name***

Setting up github on your PC

This may be already done but refer to this section if installation is necessary.

1. Download the installation file at https://git-for-windows.github.io/
2. Leave all settings as they are and click the appropriate button to continue through the installation.

Cloning a repository

A repository is a designated directory for all of your project files. One will already be created and assigned to you once you have given your team name to your coach.

1. Navigate to a directory on the computer where you would like to "download" your repository
2. Right click and select "Git Bash Here"
3. Once git bash opens, use the following command (replacing ***repository-name*** with the name of your repository).

```
git clone https://github.com/mcc-robotics/ repository-name.git
```

General Github workflow

It's important to note the typical workflow when using github. The key benefit of github is its ability to allow multiple members to work on the same files without interfering with each other but this relies on the fact that you "commit" early and often.

With github you have files on your local computer that you can alter however other members will not see these changes until you "commit" and "push" those changes to the network or "remote" repository. In general you should not "push" anything that is in progress unless necessary but this is something that should be left up to the team to decide.

The general workflow is that you update your local computer with any changes other members may have made by "pulling" from the remote repository. After pulling, you make any changes you may need, you would then "commit" those changes which is essentially like making a save point. After "committing" to your local repository you then "push" your changes to the remote repository.

Pulling from remote

Pulling will update the files on your computer with any changes anyone has made in the remote repository. To pull from remote you simply use the following command.

```
git pull
```

Committing local changes

After you have finished making your changes to the file(s) you must commit them and leave a message stating what changes you made. In general you should commit often and only make small changes to as few files as necessary. To commit, you must perform a few steps. First you must add which tells github that you've made changes, added or removed files and you'd like to make those changes part of this commit. After adding, you can commit your changes and leave a message stating what changes you have made.

To add changes to files or any new/deleted files use the following command.

```
git add --all
```

To commit your changes, use the following command (changing italics to your own message)

```
git commit -m "A short message about what you changed such as 'Updated
chassis motor mounts to use new motors'"
```

Pushing local changes

The final step in making changes is to update the remote repository with the changes you have made so that other team members can see them and update their files when the proceed to make changes of their own.

To push your local changes to the remote repository use the following command

```
git push
```

Additional Information

Branches

Branches are very helpful for trying out new ideas that you aren't sure are going to work. You can essentially make a copy of all of your files, make whatever changes you want and you won't affect the original files you copied from.

If you choose to use branches you should always use master as a branch that will always contain the last fully working version of code. This way, you can create a branch from the files on master, make your changes and if the new changes work out you can apply the changes to master but if they don't, you can either keep the branch and continue to work alongside master or you can delete the branch and forget about it.

The one thing to note is that if you follow this practice and someone needs to get access to working code for the robot all they have to do is refer to the master branch because master will always have working code.

Creating a branch

A branch is created by "copying" the files from the branch you are already on. Github never copies anything, what github is doing is creating a snapshot and tracking the differences between branches.

A branch is created with the following command and you would name the branch something logical like maybe you are testing new motors so you can name your branch ***new-motors***

```
git checkout -b name-of-new-branch
```

Applying changes to another branch

Going along with the previous example, if you've tested your new motors and they work, you now want to apply your working code on the ***new-motors*** branch to the master branch. Sometimes you want to overwrite other branches but the command is the same, simply change the name of the destination branch.

To apply changes from one branch to another use the following command replacing the ***branch-names***

```
git merge source-branch destination-branch
```

Cheat sheet

This cheat sheet provides a very good quick reference to the most commonly used commands in github. Many of these you may never use but feel free to look into them or ask about them to find out more features and how github can help with your project even more.

Git Cheat Sheet

For more awesome cheat sheets visit rebellabs.org!

REBELLABS by ZEROTURNAROUND

Create a Repository

From scratch -- Create a new local repository
```
$ git init [project name]
```

Download from an existing repository
```
$ git clone my_url
```

Observe your Repository

List new or modified files not yet committed
```
$ git status
```

Show the changes to files not yet staged
```
$ git diff
```

Show the changes to staged files
```
$ git diff --cached
```

Show all staged and unstaged file changes
```
$ git diff HEAD
```

Show the changes between two commit ids
```
$ git diff commit1 commit2
```

List the change dates and authors for a file
```
$ git blame [file]
```

Show the file changes for a commit id and/or file
```
$ git show [commit]:[file]
```

Show full change history
```
$ git log
```

Show change history for file/directory including diffs
```
$ git log -p [file/directory]
```

Working with Branches

List all local branches
```
$ git branch
```

List all branches, local and remote
```
$ git branch -av
```

Switch to a branch, my_branch, and update working directory
```
$ git checkout my_branch
```

Create a new branch called new_branch
```
$ git branch new_branch
```

Delete the branch called my_branch
```
$ git branch -d my_branch
```

Merge branch_a into branch_b
```
$ git checkout branch_b
$ git merge branch_a
```

Tag the current commit
```
$ git tag my_tag
```

Make a change

Stages the file, ready for commit
```
$ git add [file]
```

Stage all changed files, ready for commit
```
$ git add .
```

Commit all staged files to versioned history
```
$ git commit -m "commit message"
```

Commit all your tracked files to versioned history
```
$git commit -am "commit message"
```

Unstages file, keeping the file changes
```
$ git reset [file]
```

Revert everything to the last commit
```
$ git reset --hard
```

Synchronize

Get the latest changes from origin (no merge)
```
$ git fetch
```

Fetch the latest changes from origin and merge
```
$ git pull
```

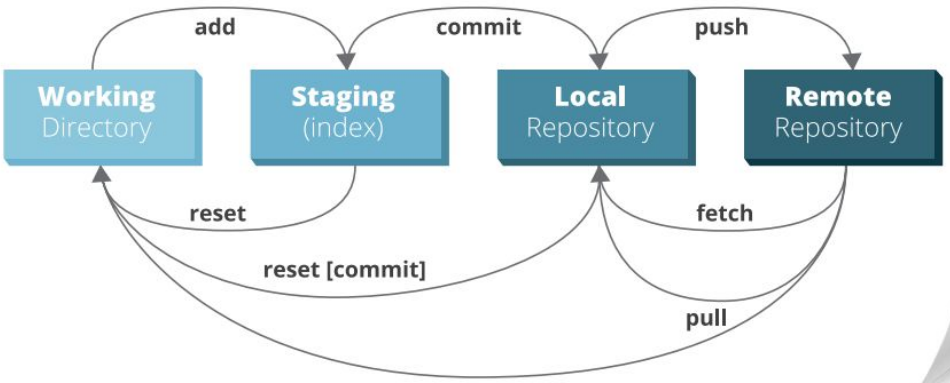
Fetch the latest changes from origin and rebase
```
$ git pull --rebase
```

Push local changes to the origin
```
$ git push
```

Finally!

When in doubt, use git help
```
$ git command --help
```

Or visit https://training.github.com/ for official GitHub training.



BROUGHT TO YOU BY JRebel