DEC 29, 2017

GitHub / git – Eclipse overview and integration for FRC teams.

GitHub is an invaluable source control and collaboration tool for FRC teams. Below are some explanations on what GitHub and git are, why they should be used by all teams and how to use them. The how portion of this document will be broken in multiple topics each building on top of each other, starting with the basics.

What is git and Github?

- Git is a version control system created by Linus Torvolds along with other Linux kernel developers in 2005. It was created to provide an open source alternative to the proprietary system BitKeeper and to address issues with other open source version control systems. Key benefits of git, distributed workflow, corruption resilient, fast commits and self-contained repositories in every directory under git control.
- GitHub – A commercial provider of online git repositories. Free services are provided by GitHub that meet most non-commercial users' needs. Commercial version are also available if desired.

What is a version control system, AKA VCS?

- A system that provides services useful for the storage, distributed development and versioning of source code.
- Version control systems, track changes made to source code files and provide methods for storing those changes either locally, remotely or both. This tracking allows for changes to be backed out if needed, provide details on what the change contains or allow multiple people to change the same source files and have the system be aware of this and provide for automated ways to "merge" the changes or provide feedback to the users notifying them that issues need to be resolved manually. Along with tracking comes other useful features such as tagging, which allows all code at a certain point and time to be labeled and recalled later in a the state it was at the time of tagging. This is useful for "releasing" code to be used for general consumption. There are numerous other benefits to a version control system, more of which we will cover later in this document.
- Version control systems typically store code in what's called a "repository". Think of it as a safe deposit vault that includes accounting and ledgers for your source code.

Why use git?

- Git provides a well tested, well documented, very capable and widely used version control system.
- Although any version control system has a learning curve, git has many similarities with other version control systems and as such, if you are familiar with CVS, BitKeeper or SVN, learning git will be relatively easy.
- Change tracking and commenting. The primary reason for using a version control system is typically the ability of the system to track changes, along with comments made when those changes are added to the system. This tracking allows a user to revert, share and compare changes between branches, tags, forks, etc... The act of tracking changes occurs when a user "commits" his changes into the repository. Good commit practice also includes a high level overview of the changes.
- Multiple code branches. Branching code is a common practice that allows multiple independent changes to be made to a common code base without interfering with each other. The analogy for this is a tree, with a common code base making up the "trunk" with branches being made off the trunk which have a common ancestor in the trunk and so share that starting code base, but each branch has its own unique changes that are stored only in the branch and thus allow the trunk code to remain static while allowing branches to test out new fixes, features, optimizations, etc. Changes in a branch can be merged back into the trunk or the branch can be destroyed leaving the trunk in its original state. Development can continue in trunk after branching, but changes in the trunk do not automatically make it into the branches and vice versa, changes must be merged between the branches and trunk. Changes can be merged between branches as well.
- Tagging code releases. Tagging code is the practice of labelling a code base, typically in the trunk, with a label. This label can then be used to see all the code as it was at the time of tagging. Tagging is typically done when releases are made. But the practice of tagging can be done at any time on any branch for any reason.
- Forking code bases is a GitHub feature and is documented later, it's noted here for reference since it was mentioned above.

Why use GitHub?

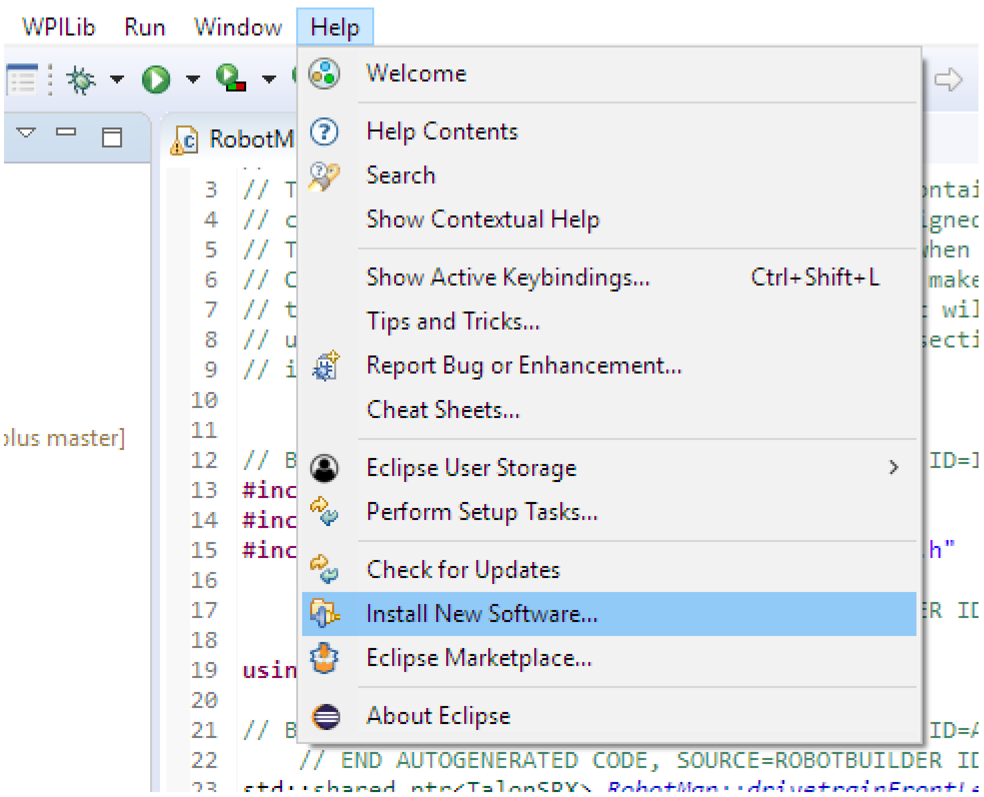
- Centralized git repository with a very capable web interface and access controls.
- GitHub repositories can be “forked” meaning one person can make a their own copy of any GitHub repository they have access to. This fork can then be modified, branched, tagged, etc… like any other git repository. Changes between forks are typically done via “Pull requests”. Though changes between forks can also be done via merges, just like changes between branches can be done. A pull request, aka PR, is a request from someone to have the changes they have made to their local fork of a repository, incorporated into another repository, typically the source repository for the initial fork. This is a very powerful feature of GitHub as it allows anyone to make a copy of a repository, make changes to it, and then have those changes pulled back into the source repository. This is the typical method of development for community open source projects. Eg; Linux kernel
- Teams are another way to have multiple individuals work on a common code base. A team in GitHub is a set of individuals with direct access to the source code repository. These individuals will “clone” the source repository to a local location. The individuals in a team can also directly modify the original GitHub repository.
- The main primary difference between Teams and forks are; access to the code, what code is modified when changes are committed and the act of getting their changes inserted into the original GitHub repository.

Useful terms not described above.

- Cloning is the process that copies the contents of the source repository to a local location. Changes are then made to the local code base and save to the local repository via a “commit”. The commit saves the changes to the repository directly responsible for the source, in this case that repository is the local repository. You can also make changes directly via GitHub and those commits would then be made to the GitHub repository. To have local changes become part of the GitHub repository they must be “pushed”. The act of pushing, moves all the change made in a local repository to the remote repository, in this case, GitHub.
- Pulling is the opposite of a push, it “pulls” all changes made in the remote repository, in this case GitHub, to the local repository.
- Commit, push, pull and PRs are all intelligent processes and will only act on deltas between the various repositories.

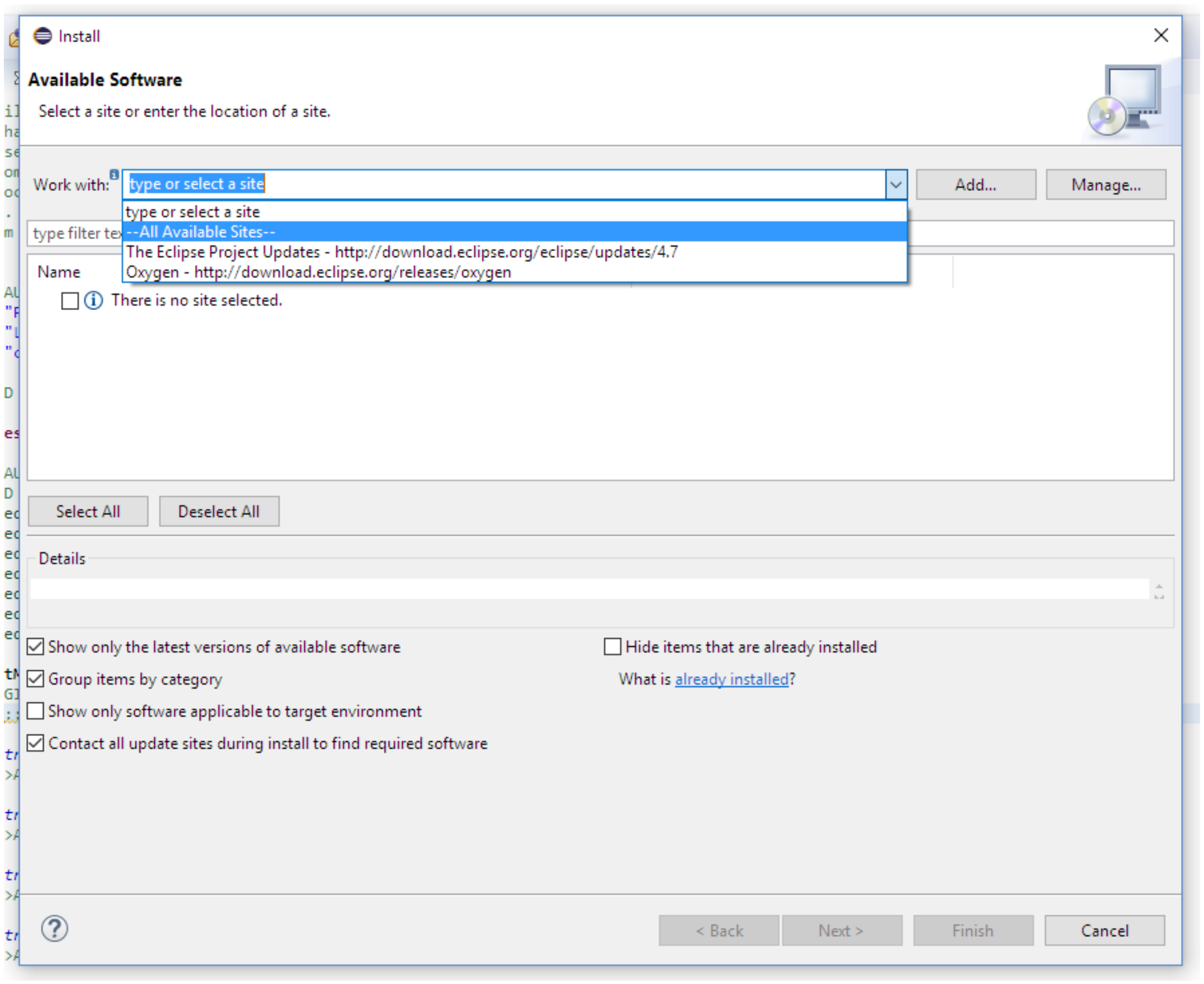
GitHub / git integration with Eclipse.

1. Open Eclipse
2. Help -> Install New Software



```
// END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID
```

3. Change Work with: to “--All Available Sites—”



Install

Available Software

Select a site or enter the location of a site.

Work with: type or select a site

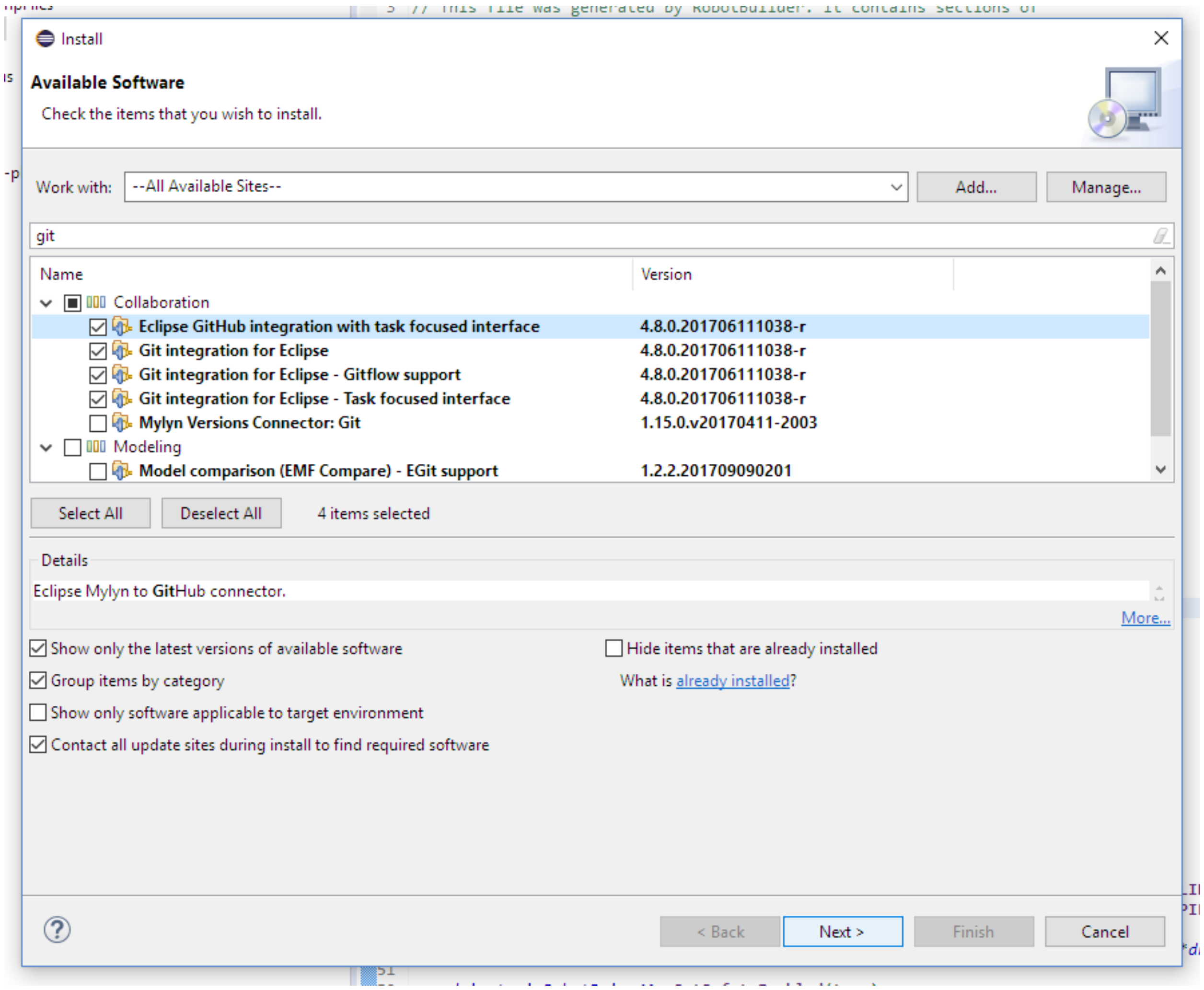
- type or select a site
- --All Available Sites--
- The Eclipse Project Updates - http://download.eclipse.org/eclipse/updates/4.7
- Oxygen - http://download.eclipse.org/releases/oxygen

There is no site selected.

[✓] Show only the latest versions of available software
[✓] Group items by category
[ ] Show only software applicable to target environment
[✓] Contact all update sites during install to find required software
[ ] Hide items that are already installed

What is already installed?

4. Enter “git” in search box directly below Work With: then hit Enter key.
    a. This will restrict the items displayed to those with “git” in the title.
5. Put check marks in the following software items.
    a. Eclipse GitHub integration with task focused interface.
    b. Git integration for Eclipse
    c. Git integration for Eclipse – Gitflow support
    d. Git integration for Eclipse – Task focused interface
6. Push “Next >” button at the bottom of the window.



Work with: --All Available Sites--

git

Name	Version
▣ Collaboration	
☑ Eclipse GitHub integration with task focused interface	4.8.0.201706111038-r
☑ Git integration for Eclipse	4.8.0.201706111038-r
☑ Git integration for Eclipse - Gitflow support	4.8.0.201706111038-r
☑ Git integration for Eclipse - Task focused interface	4.8.0.201706111038-r
☐ Mylyn Versions Connector: Git	1.15.0.v20170411-2003
☐ Modeling	
☐ Model comparison (EMF Compare) - EGit support	1.2.2.201709090201

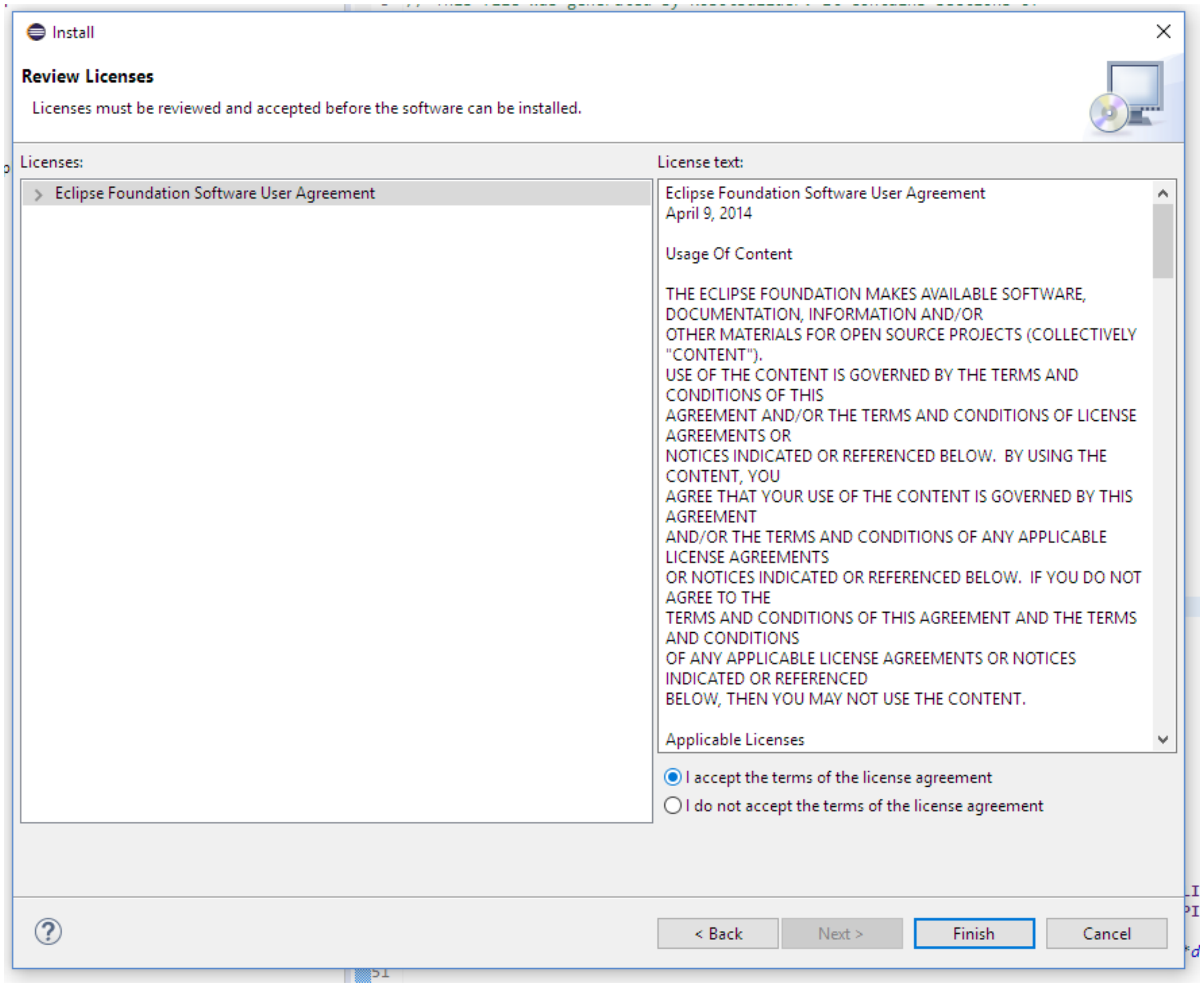
4 items selected

Details

Eclipse Mylyn to GitHub connector.

☑ Show only the latest versions of available software
☑ Group items by category
☐ Show only software applicable to target environment
☑ Contact all update sites during install to find required software
☐ Hide items that are already installed
What is already installed?

7. Push "Next >" button at the bottom of the window.
    a. Note: This step may notify you that some or all of the components are already installed, this is ok, push next.



Licenses must be reviewed and accepted before the software can be installed.

Eclipse Foundation Software User Agreement
April 9, 2014

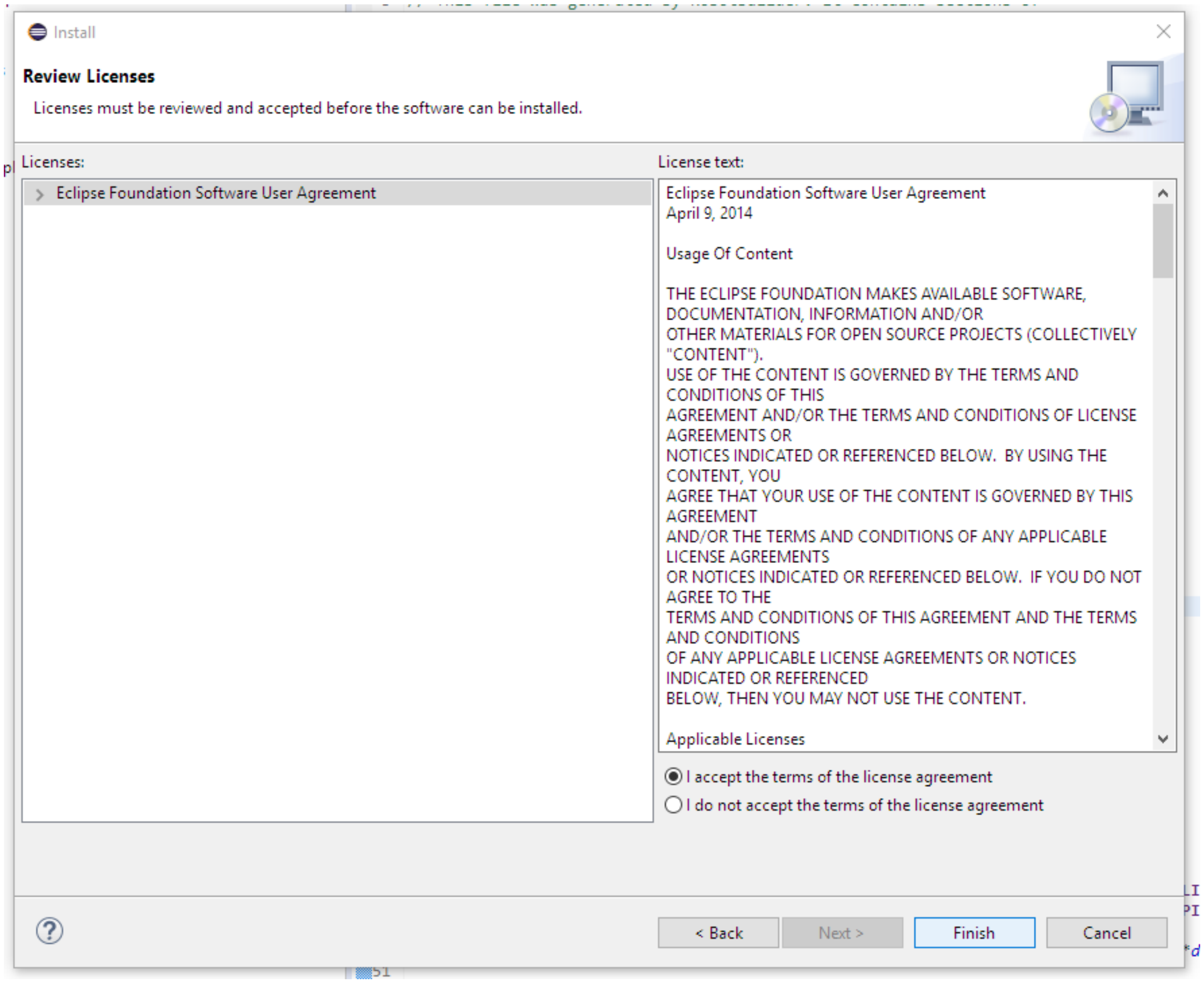
Usage Of Content

THE ECLIPSE FOUNDATION MAKES AVAILABLE SOFTWARE, DOCUMENTATION, INFORMATION AND/OR OTHER MATERIALS FOR OPEN SOURCE PROJECTS (COLLECTIVELY "CONTENT"). USE OF THE CONTENT IS GOVERNED BY THE TERMS AND CONDITIONS OF THIS AGREEMENT AND/OR THE TERMS AND CONDITIONS OF LICENSE AGREEMENTS OR NOTICES INDICATED OR REFERENCED BELOW. BY USING THE CONTENT, YOU AGREE THAT YOUR USE OF THE CONTENT IS GOVERNED BY THIS AGREEMENT AND/OR THE TERMS AND CONDITIONS OF ANY APPLICABLE LICENSE AGREEMENTS OR NOTICES INDICATED OR REFERENCED BELOW. IF YOU DO NOT AGREE TO THE TERMS AND CONDITIONS OF THIS AGREEMENT AND THE TERMS AND CONDITIONS OF ANY APPLICABLE LICENSE AGREEMENTS OR NOTICES INDICATED OR REFERENCED BELOW, THEN YOU MAY NOT USE THE CONTENT.

Applicable Licenses

[✓ I accept the terms of the license agreement] I do not accept the terms of the license agreement

8. Read license text and if you agree, put check mark in “I accept the terms of the license agreement”.
9. Push “Finish” button at the bottom of the window.


Install
Review Licenses
Licenses must be reviewed and accepted before the software can be installed.
Licenses:
Eclipse Foundation Software User Agreement
License text:
< Back
Next >
Finish
Cancel

Eclipse Foundation Software User Agreement
April 9, 2014

Usage Of Content

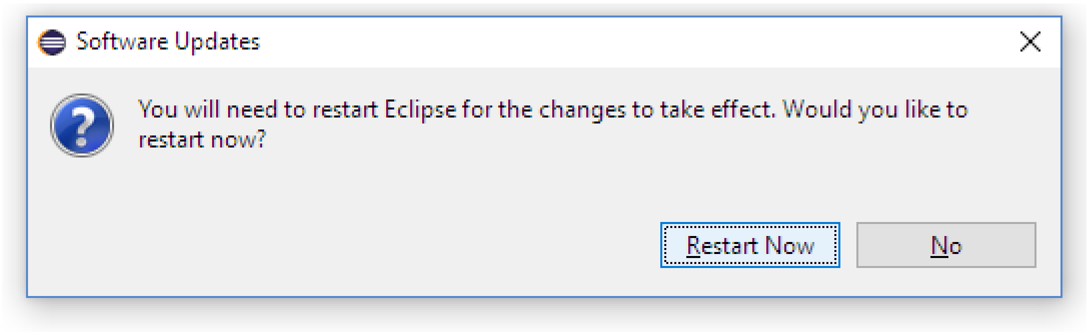
THE ECLIPSE FOUNDATION MAKES AVAILABLE SOFTWARE, DOCUMENTATION, INFORMATION AND/OR OTHER MATERIALS FOR OPEN SOURCE PROJECTS (COLLECTIVELY "CONTENT"). USE OF THE CONTENT IS GOVERNED BY THE TERMS AND CONDITIONS OF THIS AGREEMENT AND/OR THE TERMS AND CONDITIONS OF LICENSE AGREEMENTS OR NOTICES INDICATED OR REFERENCED BELOW. BY USING THE CONTENT, YOU AGREE THAT YOUR USE OF THE CONTENT IS GOVERNED BY THIS AGREEMENT AND/OR THE TERMS AND CONDITIONS OF ANY APPLICABLE LICENSE AGREEMENTS OR NOTICES INDICATED OR REFERENCED BELOW. IF YOU DO NOT AGREE TO THE TERMS AND CONDITIONS OF THIS AGREEMENT AND THE TERMS AND CONDITIONS OF ANY APPLICABLE LICENSE AGREEMENTS OR NOTICES INDICATED OR REFERENCED BELOW, THEN YOU MAY NOT USE THE CONTENT.

Applicable Licenses

[✓ I accept the terms of the license agreement]
I do not accept the terms of the license agreement

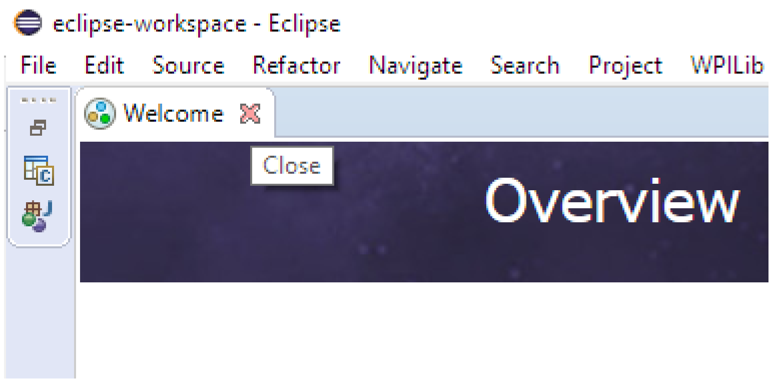
10. Eclipse will execute the changes you requested. You may monitor the status in the lower right-hand corner of the eclipse window.

11. Push “Restart Now” when window stating a restart of Eclipse is required for the changes to take effect.



You will need to restart Eclipse for the changes to take effect. Would you like to restart now?

12. If the Welcome / Overview window appears when Eclipse starts back up, click on X to close the window and return to task focused interface.

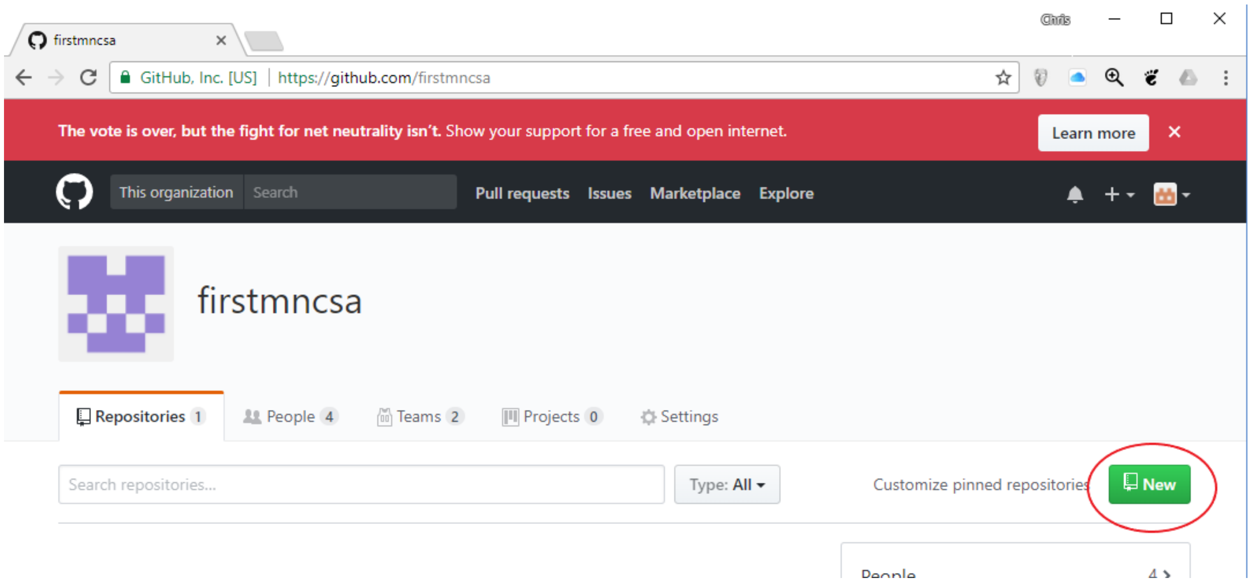


At this point your Eclipse now can now interact with GitHub and git repositories directly. Depending on your prior use of git and GitHub, the actions required to incorporate your source code into git, GitHub and Eclipse will vary. Please use the appropriate section below to complete the integration of your source code into git, GitHub and Eclipse.

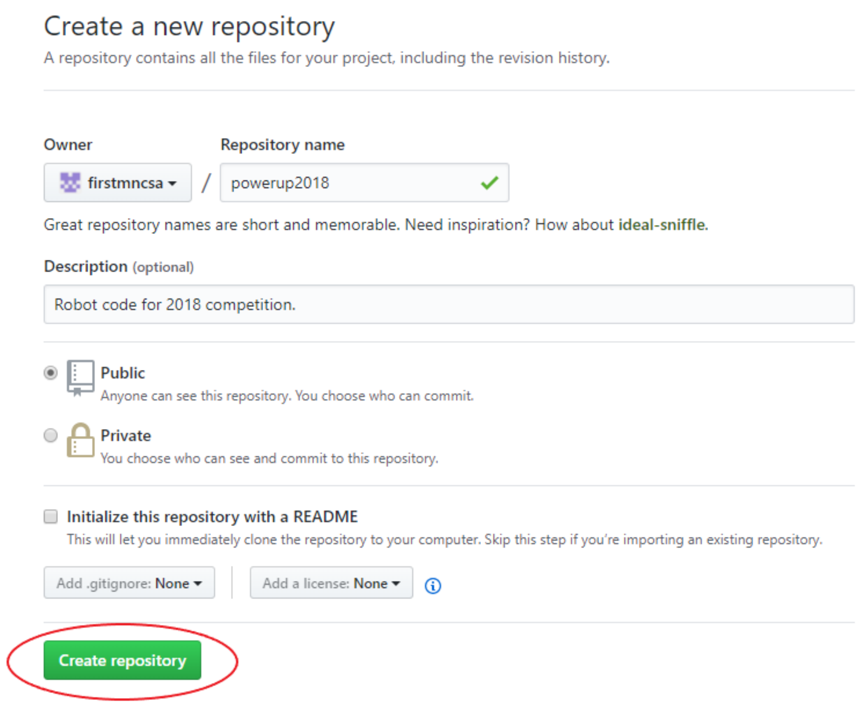
GitHub Repo Creation.

Before we can post code to GitHub, we need to create a repository. This section assumes you've already created your GitHub accounts and Team for your robotics team. Example shown is a new repo in our MN CSA GitHub team. Also this section assume your account has admin level rights to create repositories for your GitHub team.

1. Login to GitHub and navigate to your team's GitHub repository list.
   a. Eg; https://github/com/firstmncsa
   b. Replace firstmncsa, with your teams GitHub name.
2. Push "New" button on right hand side to create a new repo.


firstmncsa
GitHub, Inc. [US] | https://github.com/firstmncsa
The vote is over, but the fight for net neutrality isn't. Show your support for a free and open internet.
Learn more
This organization
Search
Pull requests
Issues
Marketplace
Explore
firstmncsa
Repositories 1
People 4
Teams 2
Projects 0
Settings
Search repositories...
Type: All
Customize pinned repositories
New
People 4

3. Enter the name for the new repository.
    a. If this repository is for a specifc year's robot, it's suggested to use the competition year in the name of the repo for easy identification.
    b. Eg; powerup_2018 or frc2018
    c. If the repository is for a generic code base, a year in the name may not be appropriate.
    d. Eg; team_manual or vision_processing
4. Enter a short description on what the repo will contain.
5. Select if you want the repo to be public.
    a. Please consider keeping your repo public and encouraging teams to fork your code in the spirit of Gracious Professionalism and they might find a bug in your code and fix it. Or you may even get enhancements from other teams in return!
6. You may want to initialize with a README if you want to write up some documentation for others to read regarding the code in the repo.
    a. How to use the code,
    b. Design guidelines.
    c. Key information needed for use.
    d. Etc…
7. Push "Create Repository" button when ready to create the repo.



Create a new repository

A repository contains all the files for your project, including the revision history.

Owner: firstmncsa / Repository name: powerup2018

Great repository names are short and memorable. Need inspiration? How about **ideal-sniffle**.

Description (optional): Robot code for 2018 competition.

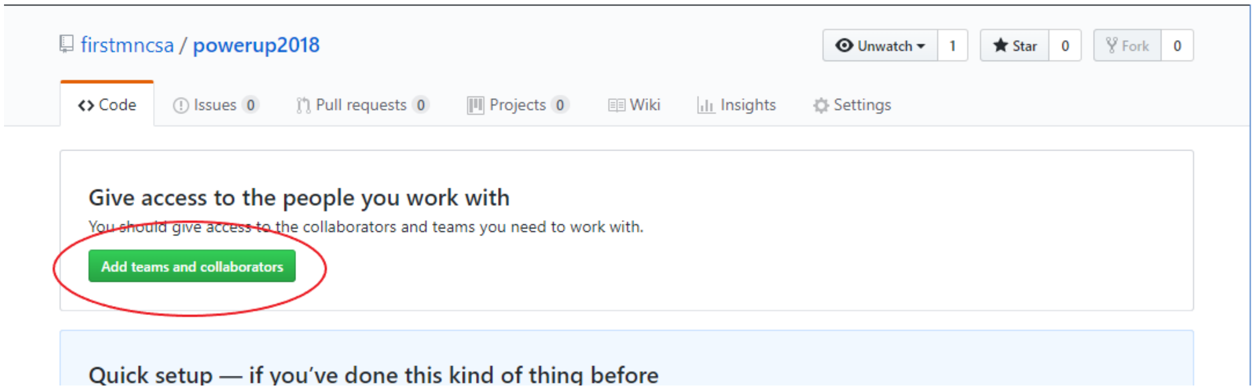
[✓ Public] Anyone can see this repository. You choose who can commit.

Private — You choose who can see and commit to this repository.

☐ Initialize this repository with a README
This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: None | Add a license: None

8. The next screen allows us to easily add GitHub Teams and Individual account holders to access the repo. Also you can see instructions and URLs for adding the new repo to various different git clients.
9. First up, let's address repo access.
    a. This is especially critical if you choose to make a private repository as no one will have access to it otherwise.
10. Push "Add teams and collaborators"

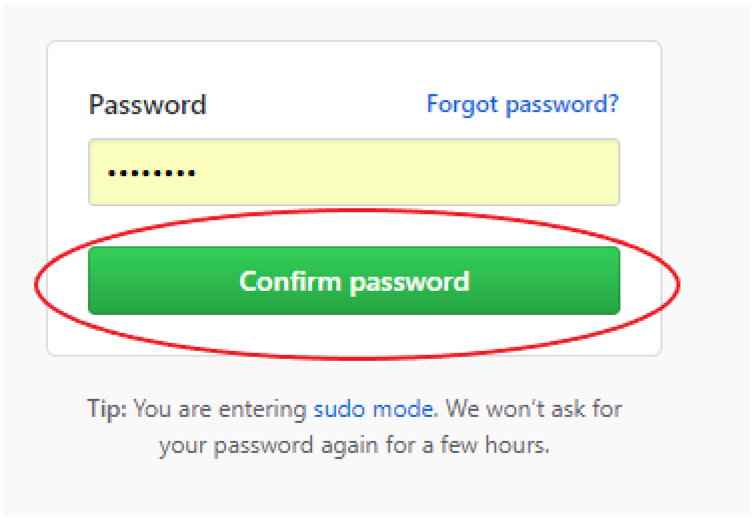


Give access to the people you work with

You should give access to the collaborators and teams you need to work with.

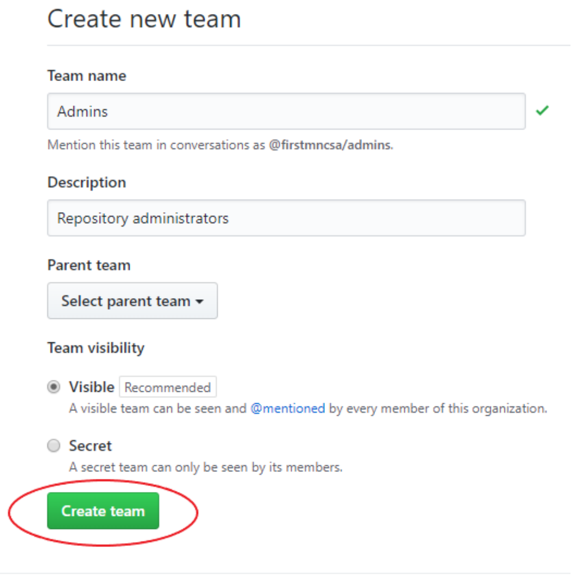
Quick setup — if you've done this kind of thing before

11. You will most likely be challenged to enter your password again, enter it and then push "Confirm Password"



Tip: You are entering sudo mode. We won't ask for your password again for a few hours.

12. Next we need to review, assign and create permissions, teams and individuals who will access and / or modify the repo and it's contents.
13. First up, review the default repository permissions.
    a. Typically that is going to be set to "write" which allows all individuals you designate on this page to have the ability to change the contents of a repo.
    b. The default "write" permissions are most likely what you want unless you want to have a single individual or limited individuals with the ability to change repo contents and a lot of folks to have read only access. Note this does not affect public repo which will be read-only to the public by default.
    c. You can change the default behavior by clicking on "member privileges page"
        i. Further info on custom access privileges is beyond the scope of this document.
14. Next up is Teams. Teams are groups of individuals that you wish you allocate access rights to as a group vs as and individual.
    a. Typically you will have a team of "Admins" and another team of "Developers". – this is the model this document will use.
    b. Individuals can be on either or both teams if that is appropriate.
    c. Option 1: Create a new team.
        i. Useful if you don't already have a GitHub team setup for your robotics team.
        ii. Click on "+ Create new team"
        iii. Enter the team name, eg; Admins
        iv. Enter a description for the team, eg; Repository Admins
        v. Select a parent team if this team should inherit permissions or members from another team. Typically, not used for Admins, but a Developers team might inherit the Admins team so the Admin members automatically become a member of the Developer team as well.
        vi. Select the team visibility. Recommended to be Visible so others can identify team members responsible for a specific role.
        vii. Push "Create team" when you are ready to create the new GitHub team.


Create new team

Team name: Admins ✓

Mention this team in conversations as @firstmncsa/admins.

Description: Repository administrators

Parent team: Select parent team ▾

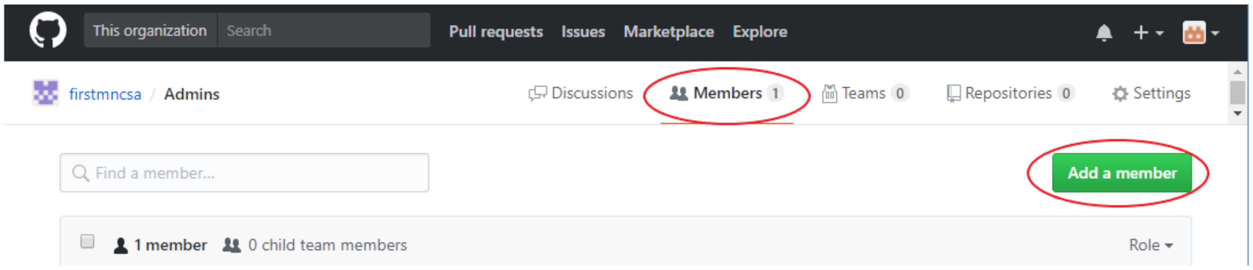
Team visibility

● Visible Recommended
A visible team can be seen and @mentioned by every member of this organization.

○ Secret
A secret team can only be seen by its members.

viii. After creating the new team, you will be sent to the discussion page for your new team. Here you may start discussion topics others in your team will be able to see and reply to.

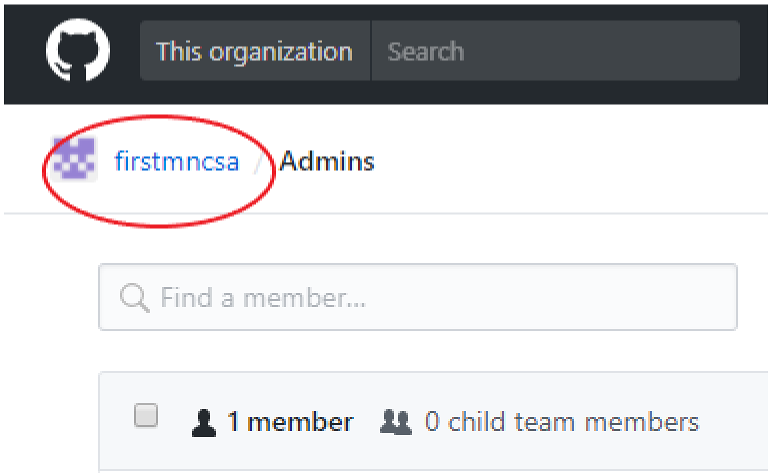
ix. Next you may want to add individuals to the new team, you can do that by click on the "Members" tab next to "Discussions" and then click on "Add a member"



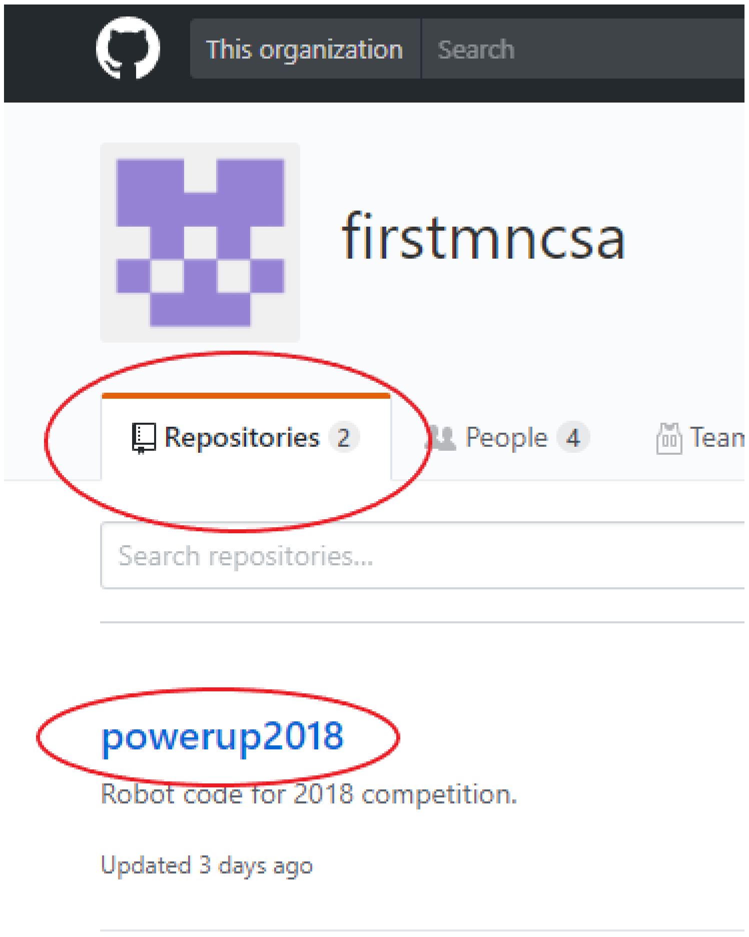
x. Enter the username, full name or email address of the individual you want to add.
   1. Enter a GitHub username if the individual has a GitHub account already.
   2. Enter a Full Name if the individual has a GitHub account already.
   3. You can enter any email address regardless if it was used for a GitHub account already. If there's no existing GitHub account associated with the email address, they will be invited via email to join and create one.

xi. The remaining tabs, "Teams" "Repositories" and "Settings" most likely don't need to be changed at this point. But here's a short synopsis of what they are for.
   1. Teams – This is teams within teams, you can further organize members of this team into sub-teams. Eg; drivetrain developers, vision developers, etc…
   2. Repositories – This is where you can create repositories this team will own and maintain. You might want to use this to fork the teams central repository and then use that forks for team member commits. Or you may want to use this to store code specific for the purposes of the team or sub-team. Eg. Vision code.
   3. Settings – Any of the settings you used when creating the team can be changed here. Eg; Team name, visibility. This is also where you can delete the team if desired. Just be ware that if you have a repository associated with the team, it will be deleted as well.

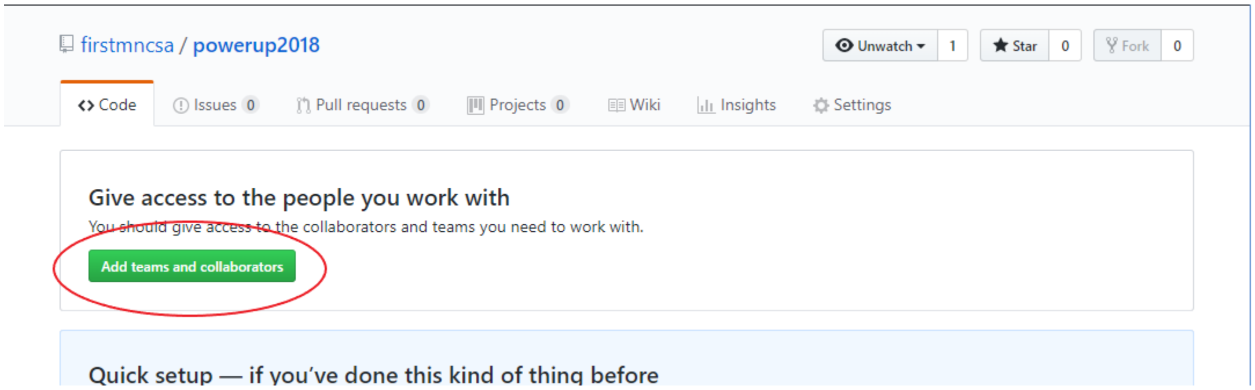
15. Now that we have our team(s) created, we need to assign them to the repository.
    a. Navigate back to the "Settings" for the repository by click on the name of the organization or GitHub account in the upper left.



b. Next click on “Repositories” tab and then click on the desired repository to add the team(s) to.



c. Next click on "Add Teams" again or click on the "Settings" tab and then "Collaborators & teams"

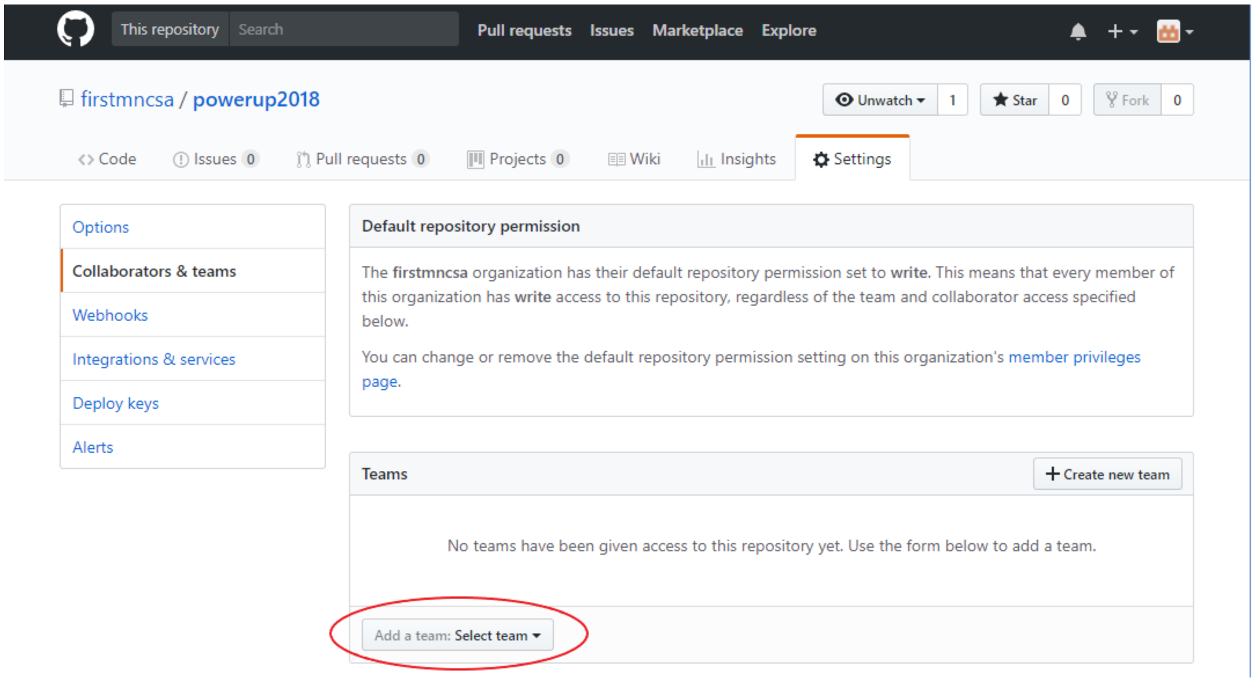


Give access to the people you work with

You should give access to the collaborators and teams you need to work with.

Quick setup — if you've done this kind of thing before

d. Next, we need to add the team to this repository. Due this by selecting the team(s) from the "Select team" pull down menu.



Default repository permission

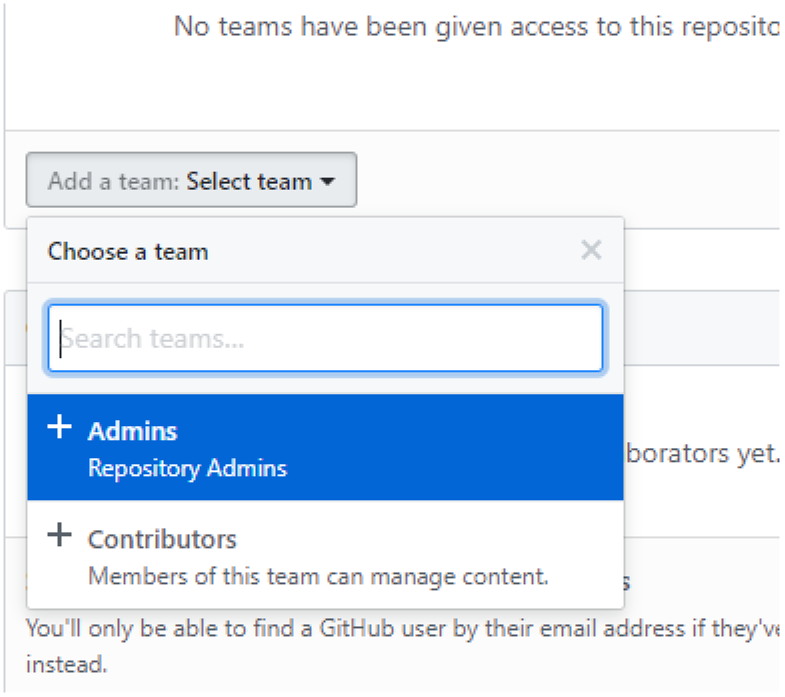
The firstmncsa organization has their default repository permission set to write. This means that every member of this organization has write access to this repository, regardless of the team and collaborator access specified below.

You can change or remove the default repository permission setting on this organization's member privileges page.

Teams

No teams have been given access to this repository yet. Use the form below to add a team.

e. Select the team you want to add.


No teams have been given access to this reposito

Add a team: Select team

Choose a team

Search teams...

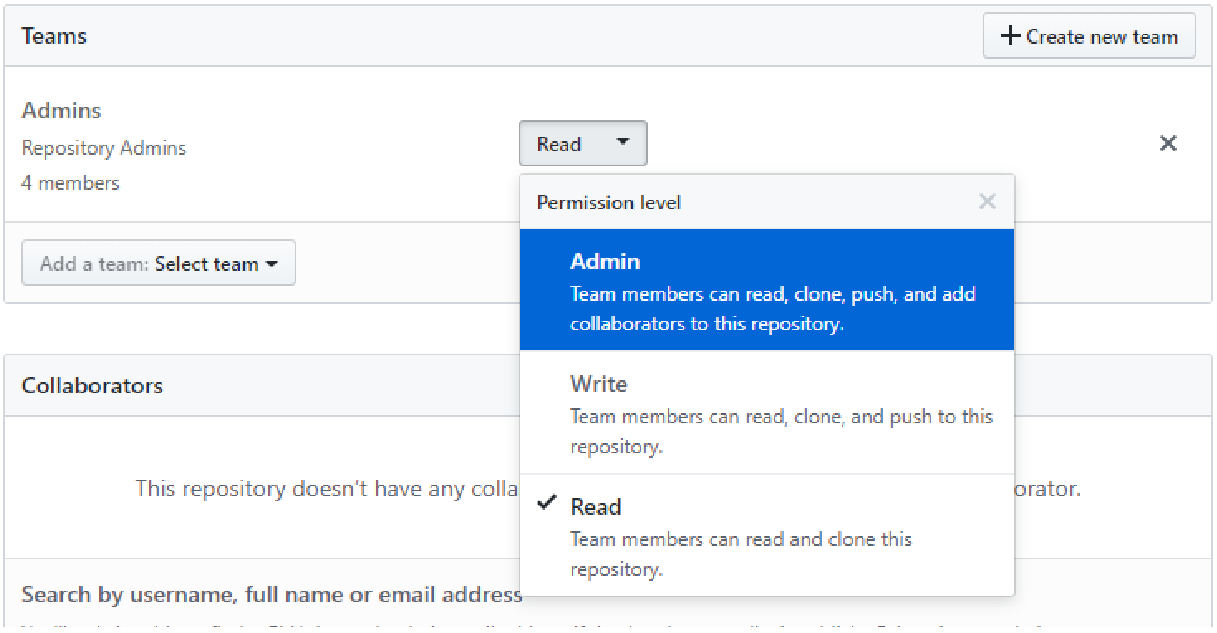
Admins
Repository Admins

Contributors
Members of this team can manage content.

borators yet.

You'll only be able to find a GitHub user by their email address if they've
instead.

f. Next select the level of permissions the team should have.


Teams

Create new team

Admins
Repository Admins
4 members

Read

Add a team: Select team

Permission level

Admin
Team members can read, clone, push, and add collaborators to this repository.

Write
Team members can read, clone, and push to this repository.

✓ Read
Team members can read and clone this repository.

Collaborators

This repository doesn't have any colla orator.

Search by username, full name or email address

g. Repeat the team selection and permission assignment for each team.

Year Round CSA Contact info: firstmn.csa@gmail.com Slack: firstmncsa.slack.com

First MN Website: http://mnfirst.org/first-community-resources/local-assistance/

First MN CSA Github: https://github.com/firstmncsa

First MN Google Drive: http://goo.gl/STtiAg