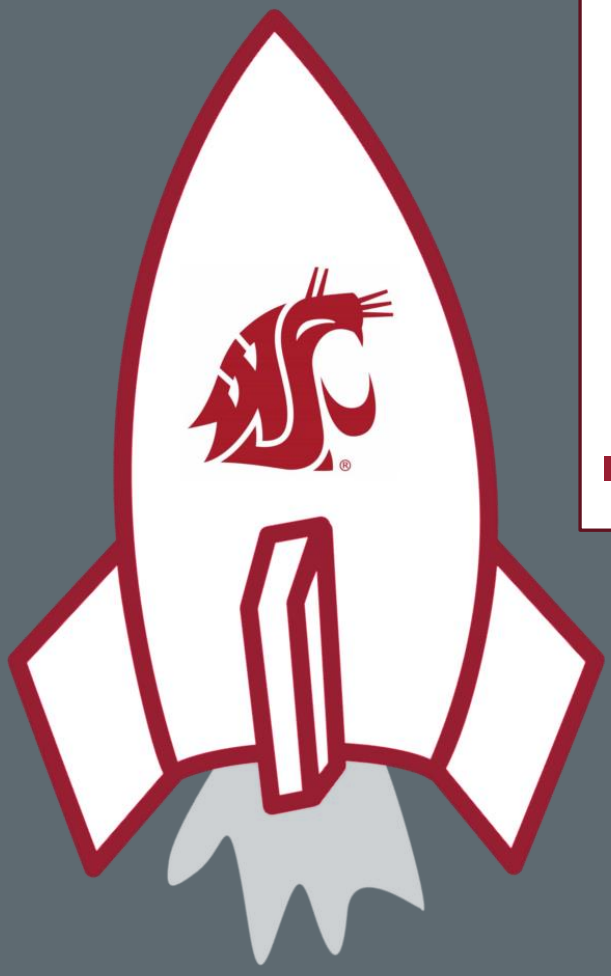
This document explains what Git is, how Cougs in Space uses it, and standardizes the operation of GitHub

# SOP: GitHub

Proper use and conventions

Revision: 1.0.0



Bradley Davis

# Table of Contents

# 1 Git Overview

## 1.1 Version Control

Version control is the management of changes to files. Tracking changes is invaluable in large and/or complex projects as every version of a file uploaded to the system is preserved and can be access in case of a future version causing errors.

Cougs in Space uses the system called Git. Git is a graph structure where users branch off trunks, develop their branch, and reconnect their branch to the trunk. GitHub is a company that provides online storage through a Git interface.

## 1.2 Git Workflow

The Git workflow defines a strict branching model for version control based around a graph structure.

### 1.2.1 Repositories

A Git **repository** contains a set of **commit objects** and a set of references to the commit objects, called **heads**. It is where the history of work is stored. Each repository can be described as a graph structure which is useful to see the flow of work being done, see figure 1.


Figure 1: An example Git Repository

### 1.2.2 Branches

A **branch** is a movable pointer to a commit object. To branch is to take the files from a parent branch and apply diverting changes to it. Once the branch's changes are complete, they are **merged** back into the parent. In figure 1, each color represents a branch. The orange branch is branched off the green branch. After several **commits**, the orange branch is merged back into the green branch. It does use the yellow intermediate branch to merge orange into green.

### 1.2.3 Operations

Various Git operations can be applied to repositories and branches. In figure 1, they are represented by the grey arrows.

#### 1.2.3.1 Branch

**Branch** is to duplicate an existing branch into another branch. In figure 1, it is represented by an arrow connecting two different colors together

#### 1.2.3.2 Commit

**Commit** is to create a snapshot of the current changes to specified files within a branch. It creates a new commit object and moves the branch's head to the latest commit object. In figure 1, a commit is a horizontal arrow between similar colors.

Each commit has three parts: a list of files, summary, and description. The files can be an addition, a revision, or a deletion. The summary describes the commit. The description is a more detailed explanatory text, if needed. Every commit must have a summary as it allows collaborators to easily follow changes. The summary and description should answer three questions:

1. **Why is it necessary?** Bug fix, feature add, improvement on performance, reliability, stability, or just a change for the sake of correctness
2. **How does it address the issue?** For short obvious patches this part can be omitted, but it should be a high-level description of what the approach was
3. **What effects does the patch have?** (In addition to the obvious ones, this may include benchmarks, side effects, etc.)

Summary Guidelines:

- Short (50 characters or less)
- Write in the imperative mode (commands, e.g. "Fix" not "Fixed")
- Use title case and don't end with a period
- If summarizing a commit is difficult, consider logically splitting up commits

Description Guidelines:

- Use if more than 1 object is being committed (e.g. corresponding source and header files are 1 object)
- Sort objects by type: Add, Change, Delete
- Use corresponding bullets (+, ~, -) to list objects (e.g. ~Github.docx: Added part about commits)
- Leave spaces between sections

#### 1.2.3.3 Push

**Push** is to upload a copy of the local repository or branch to the server repository or branch. A push requires the local repository or branch to have unsynchronized commits.

#### 1.2.3.4 Pull

**Pull** is to download a copy of the server repository or branch to the local repository or branch. A pull requires the server repository or branch to have unsynchronized commits.

#### 1.2.3.5 Merge

**Merge** is to take the changes from one branch and applies them to another. If a conflict arises, both files are edited, the merge will not continue. The conflict needs to be fixed before merging can occur. In figure 1, it is represented by two arrows going into one circle.

#### 1.2.3.6 Pull Request / Merge Request

**Pull Request / Merge Request** are proposed changes to a repository submitted by a user and accepted or rejected by a repository's collaborator. This is often done for the master branch and users cannot push to the master without going through a pull request and a review of the changes.

After a pull request is submitted wait for an admin to review it, approve it, and delete the branch before continuing working on the same branch. If you need to continue working immediately, create a new branch based of the parent, usually master. If you work off the branch you put a pull request in for, your commits will be added to the pull request. If you don't wait for the pull request branch to be deleted, the branch you are working on will be based off the branch you put a pull request in for, not the parent. You should not submit to the branch you put a pull request in for until the server tells you that you need to add the branch.

## 1.3 GitHub Projects

GitHub adds project management utilities to the standard Git system. Projects organize tasks and have features to show the progress of completing the project and subprojects.

### 1.3.1 Cards

**Cards** are tasks that are placed on the project page. They can be converted into **issues** and sorted into **columns**. The standard columns are "To Do", "In Progress", and "Done". Any card in the "In Progress" and "Done" columns needs to have an associated issue and assignees.

### 1.3.2 Issues

**Issues** are items in a task tracker of the same name. They are used to track ideas, enhancements, tasks, or bugs. Issues can be associated with projects and **milestones**.

#### 1.3.2.1 Assignees

Issues can be assigned to individuals or teams for them to work on.

#### 1.3.2.1 Labels

Labels are used to categorize issues for easy management. Table 1 lists the labels used in Cougs in Space.

Table 1

| Task type | Category | Skill Level |
|---|---|---|
| Design | Tests | Beginner |
| Docs | Refractoring | Intermediate |
| Code | New Feature | Advanced |
| CAD | Enhancement | |
| | Bugfix | |

Any task can also have Critical added to denote high priority.

### 1.3.3 Milestones

**Milestones** are a set of issues with a completion date. Each project should have at least one milestone at a time. Each milestone is celebration worthy with cake or brownies.

## 1.4 GitHub Wiki

GitHub adds documentation features to the standard Git system. The wiki is used to document the features. Each feature of a project should have an associated wiki page. Each project should also have a page describing the goals.

# 2 Cougs in Space GitHub

Cougs in Space GitHub is hosted through EECS. An EECS account is required to access the server.

## 2.1 Repositories

Cougs in Space has several repositories.

- References – Documentation not related to a specific craft
  - Here is information about how to build a satellite
- CougSat1-Hardware – Our first craft's mechanical and electrical systems
- CougSat1-Software – Our first craft's software.

## 2.2 Projects

Each repository has a GitHub project setup with corresponding top-level folders. For example, in CougSat1-Software, there is a GitHub project setup for the IHU processor and there is a top-level folder of the same name. This folder hosts the necessary files to program the IHU.

The software repository has GitHub projects for each individual processor we are programming. The hardware repository has GitHub projects for each unique PCB we are creating and a project for the mechanical structure.

# 3 File Conventions

File conventions are a set of rules for the names of files and their location in folders. Consistent file practices improve readability and efficiency of accessing resources.

## 3.1 File and Folder Names

The following are a list of rules to follow when naming files and folders:

- Names are short yet descriptive of the contents
- Names are unique
- Names are searchable
- Names of native files do not contain version info
- Names of distributable files contain version info
- Names do not contain the name of their parent folder

Distributable files require version information in the file name, it is as follows:

- Names must take the form: [Native Filename].[Major].[Minor].[Patch]
- Increment [Major] version when incompatible changes occur
- Increment [Minor] version when backwards-compatible changes occur
- Increment [Patch] version when backwards-compatible bug fixes occur
- All versions begin at 1.0.0
- After a distributable file is created, the native file is takes the next applicable version number
- Once a distributable file is created, the contents must not be modified. Any modification must be released as a new version

## 3.2 Folder Structure

The following are a list of rules to follow when arranging folders:

- There exists a single top-level folder for each project
- Folders are not too broad that they encapsulate hundreds of files and folders
- Folders are not too narrow that they encapsulate single files
- Distributable files are placed above a native folder contain its native source

## 3.3 Native versus Distributable Files

Native files are the file that can be edited. Examples include C++ source code, Inventor CAD models, and Microsoft Word documents. Distributable files are built or exported from native files that cannot be edited. Examples include program binaries, STEP CAD models, and PDFs. These types of files usually do not need proprietary software to view the contents.

# 4 Git Clients

Git clients are software that allows you to interface with the Git repositories.

## 4.1 GitHub Online

Using this client is only recommended for viewing the server repository. It is required to view GitHub projects, issues, milestones, and wikis.

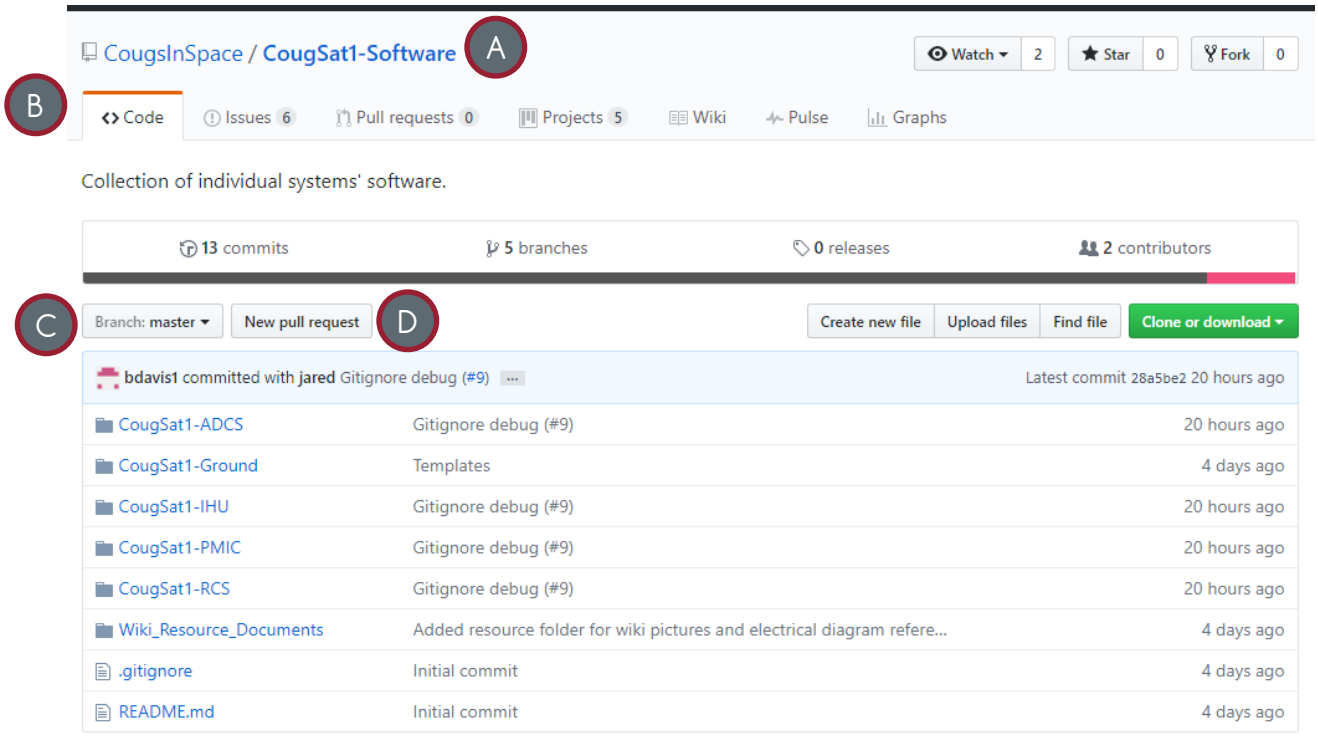



Figure 2: GitHub Online

Figure 2 shows the interface to GitHub Online.

(A) is the current repository.

(B) are the following tabs: Code, Issues, Pull requests, Projects, Wiki, Pulse, Graphs. Code is for viewing the repository's files. Pulse is to view a summary of repository's activity of the last week. Graphs is to visualize various metrics of the repository. The other tabs contain what they are names.

Whilst in the Code tab, (C) is the current branch being shown.

(D) is to create a pull request. The other buttons to create a new file, upload files, and cloning should not be used. Use a software client to access these features.

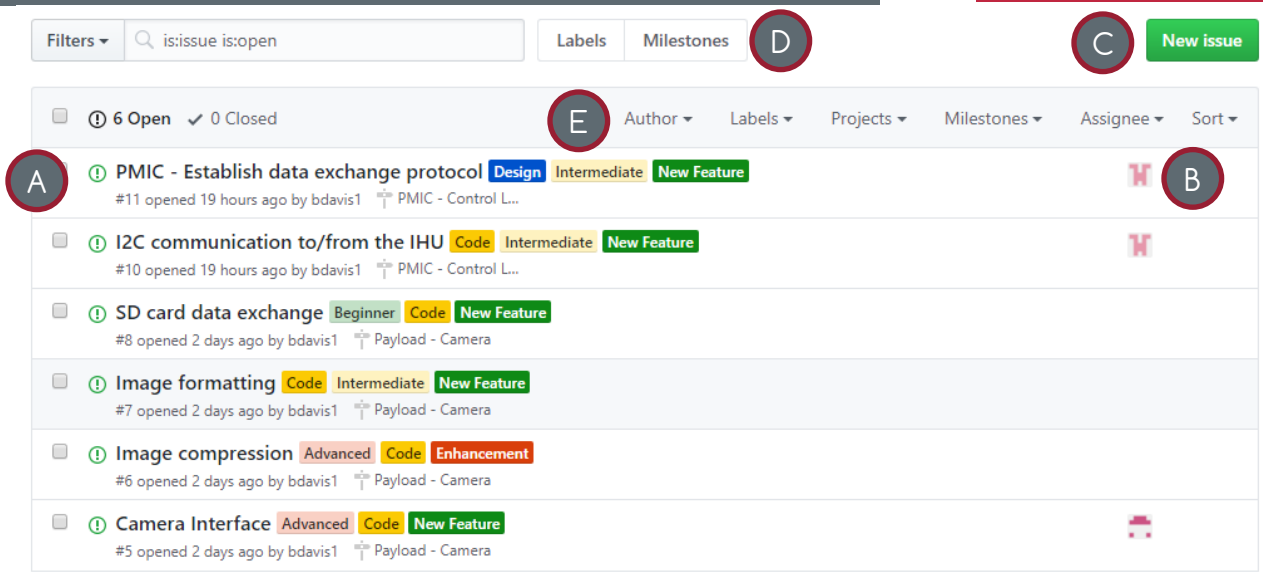



Figure 3

Figure 3 shows the issues tab.

(A) is an issue. It has a name, labels, associated label, and assignee (B).

(C) opens the interface to create a new issue. If the issue describes a new feature, enhancement, or critical issue, a card in the associated project should be created and then the issue off that. The other types of issues, such as bug fixes, are okay to not have an associated project card.

(D) will open the list of milestones, see Figure 4.

(E) are various ways to sort issues to easily find an issue you are wanting to work on.

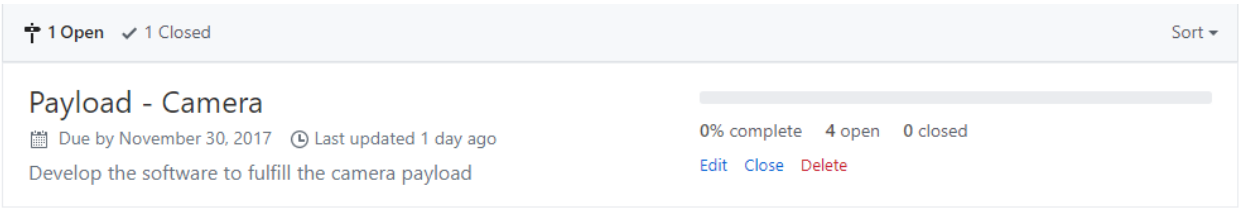



Figure 4

Figure 4 shows the lists of milestones. Each milestone has a name, a due date, description, and a progress bar. The progress is based on the completion of the associated issues. Milestones should be created by a team, so everyone is aware of the next milestone in completing the project. Once a milestone is completed, the team shall celebrate with cake and then set up the next milestone.

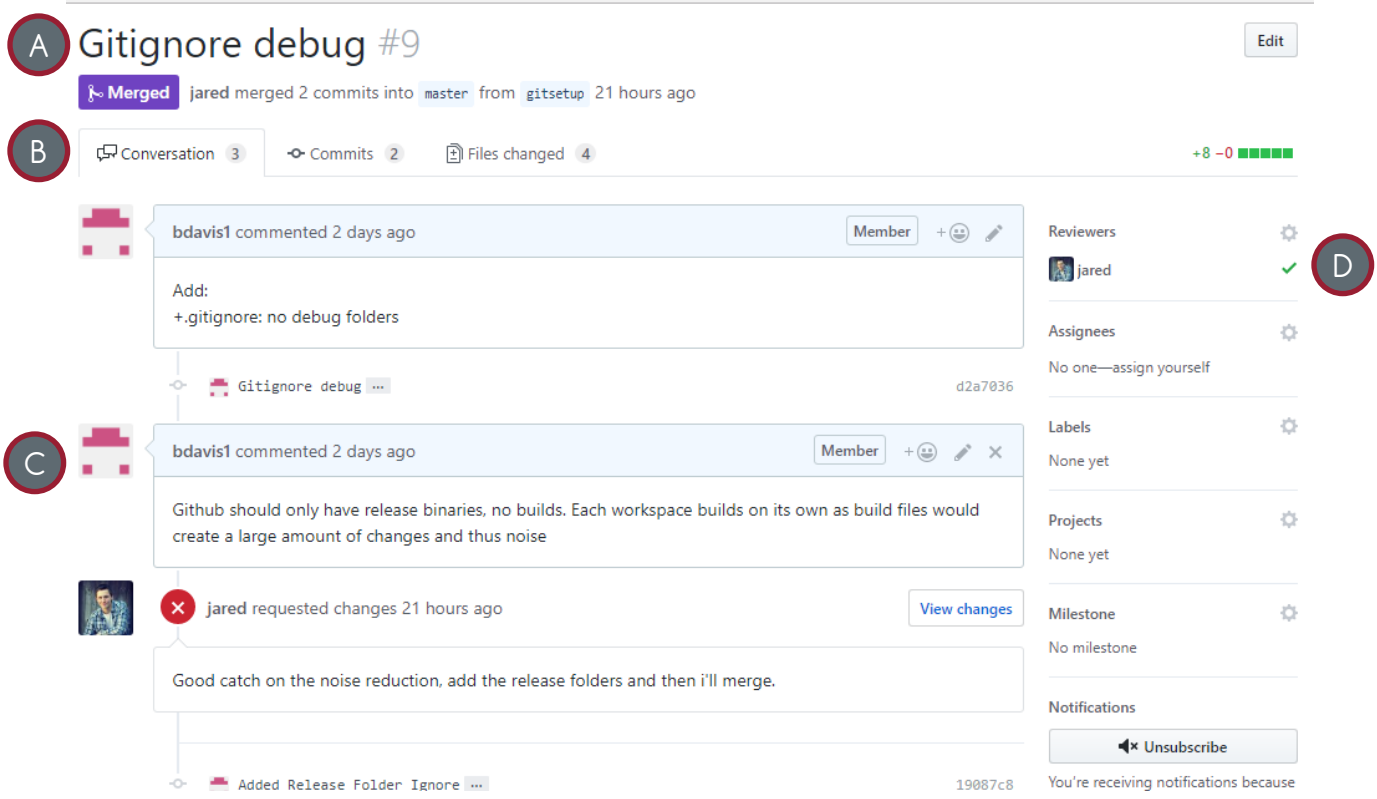



Figure 5

Figure 5 shows a pull request, the list of pull requests is under the tab of the same name.

(A) is the name of the pull request and the status. This pull request was approved by Jared and resulted in the "gitsetup" branch being merged into the "master" branch.

(B) are the pages of a pull request: Conversation, Commits, and File changed.

(C) Conversation allows the people involved with the pull request (the requestee, and the reviewers) to discuss the pull request.

(D) Are the reviewers requested to review and approve pull request. Request your team lead as a reviewer.

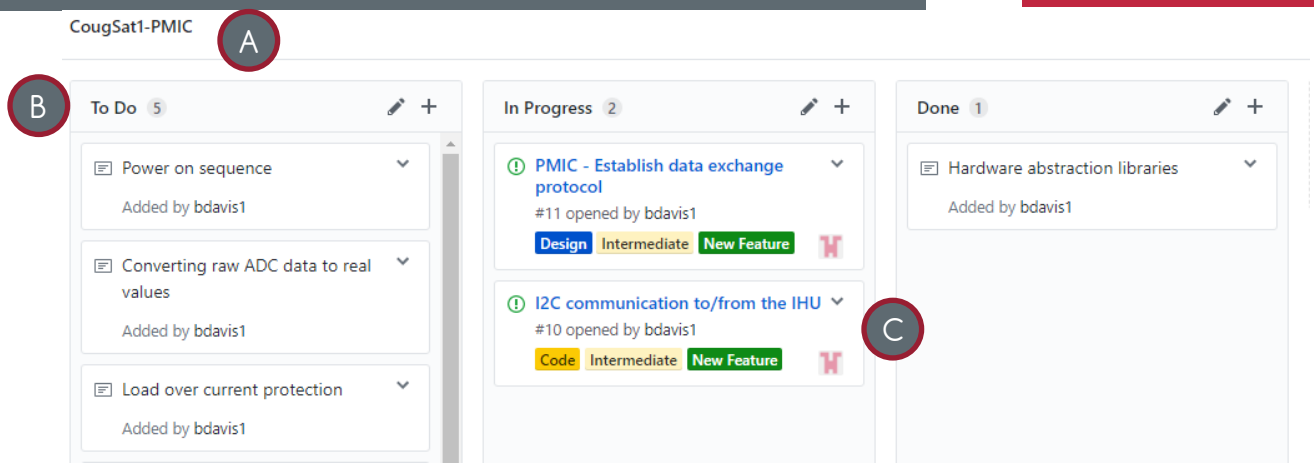



Figure 6

Figure 6 shows a GitHub project.

(A) is the name of the project.

(B) are the columns of the project. They are To Do, In Progress, and Done.

(C) is a card that is also an issue. Cards can be moved between columns by dragging them. A card can be converted into an issue using the dropdown arrow on the card.

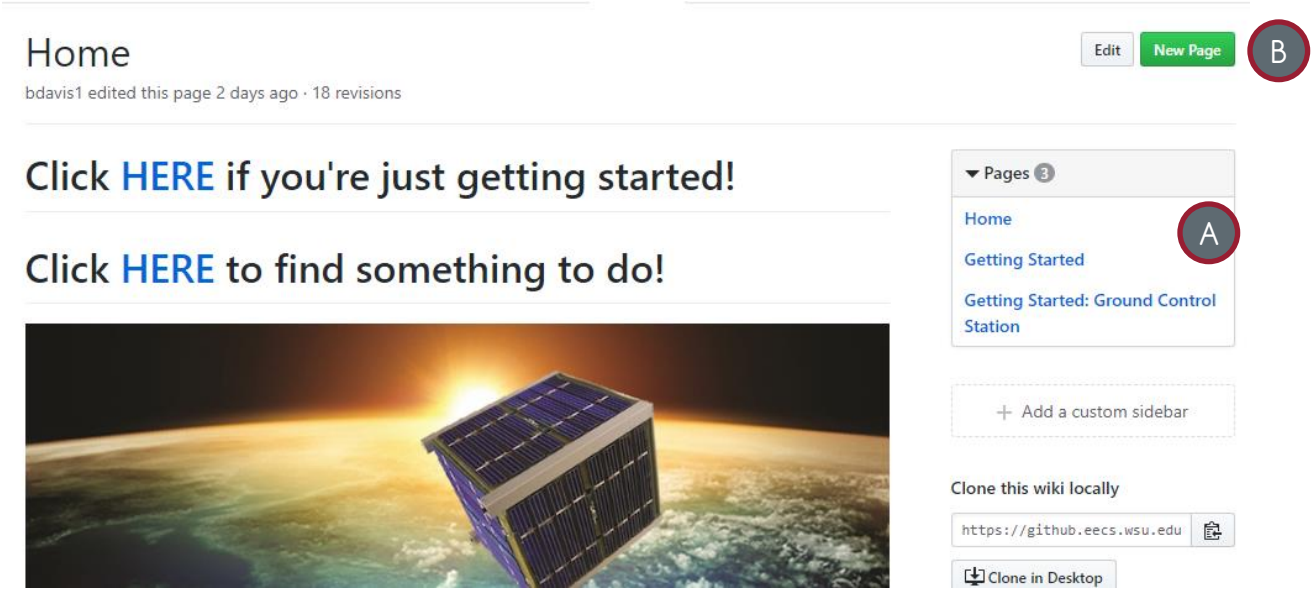



Figure 7

Figure 7 shows the wiki for the repository.

(A) is the list of pages in the wiki.

(B) opens the editor to edit the current page or create a new page.

## 4.2 GitHub Desktop

GitHub Desktop is easiest way to interact with repositories. It only works with the Git system of GitHub, project management features need to be accessed through GitHub Online, see 4.1. It is downloaded here [https://desktop.github.com/](https://desktop.github.com/)

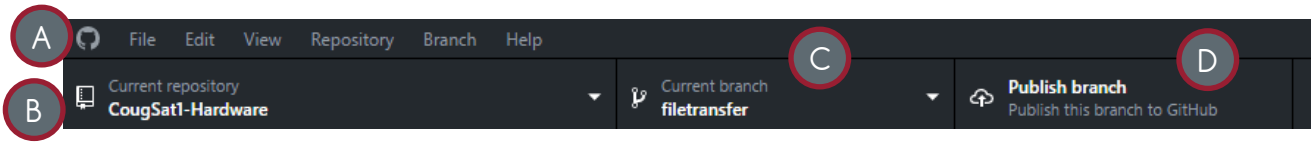



Figure 8

Figure 8 shows the top interface of GitHub Desktop.

(A) is the menu bar. To add a repository, go to File/Clone Repository/Enterprise.

(B) is the current repository.

(C) is the current branch. In the dropdown, you can select a branch or create a new branch based off the current branch.

(D) is the next available command. They are Publish Branch, Push, Pull, Fetch origin. Publish branch tells the server of your new branch, once a pull request is approved, you must republish the branch if you want to continue working. If you don't have to create a new branch after an approved pull request, the branch needs to be deleted first. Fetch origin checks to see if there are changes in the server repository.

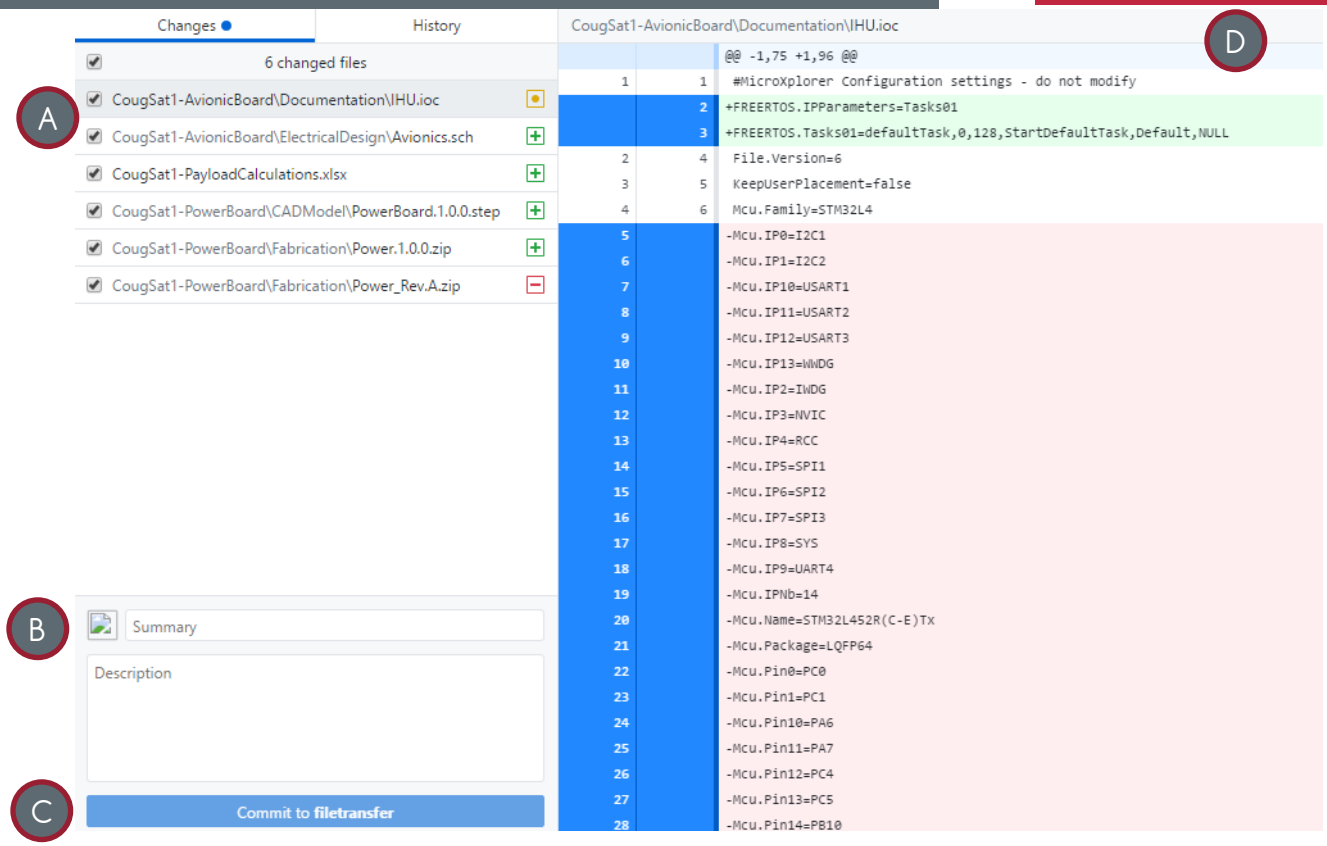



Figure 9

Figure 9 shows the commit interface.

(A) is a list of changed files in the repository. It shows the file and the type of change: Add, Change, Remove.

(B) is the commit message.

(C) commits the changes to the local repository.

(D) shows the changes for the current selected file.

## 4.3 Eclipse Embedded

Git built into Eclipse is accessed by opening the Git perspective through Window/Perspective/Open Perspective/Other-> Git.

Here is a tutorial on how to use Eclipse Git:
https://eclipsesource.com/blogs/tutorials/egit-tutorial/

Pull request need to be made through GitHub Online, see 4.1

## 4.4 Command Line

Git Bash can be downloaded here: https://git-scm.com/downloads

Here is a tutorial on how to use Git Bash:
https://www.youtube.com/watch?v=SWYqp7iY_Tc

Pull request need to be made through GitHub Online, see 4.1

# 5 Glossary

**Branch** – A copy of another branch with changes applied. Also, to branch is to make a branch
**Card** – Tasks listed on a project page, tasks can also be issues
**Commit** – Create a snapshot of changes
**Commit Object** – Contains a set of files, and reference to parent commit objects
**Head** – A reference to a commit object, the most recent version of each branch
**Issues** – A task that can be worked on, can be ideas, enhancements
**Merge** – Turn two branches into one by copying changes
**Milestone** – A collection of associate issues used to track progress on a project
**Pull** – Get changes from server repository or branch and merge them into the local repository or branch
**Pull Request / Merge Request** – Proposed changes to a repository to be approved or rejected by a repository's collaborators
**Push** – Update server repository or branch with local repository commits
**Repository** – A folder containing a set of commit objects and their history

## 6 References

http://who-t.blogspot.com/2009/12/on-commit-messages.html
https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow
http://semver.org/
https://help.github.com/articles/github-glossary/
http://radek.io/2015/08/24/github-issues/