GitHub - Manual & Cheat Sheet

Team Avionics and Payload

Why GitHub?

We at Avionics and Payload write code to make the flight computer and other circuitry work. However, this was to be written and contributed by more than one person. However, when code was developed in-house, it was shared amongst contributors via means such as WhatsApp and Gmail.

Such development would often mean messy organization of code. It can result in the wrong code version, untracked changes and confusion amongst contributors. It can also lead to the deletion of important files and code which you felt at one point was not used, but suddenly feel a need to recover them.

GitHub serves as a place where developers can commit their code into a repository, which is shared amongst fellow developers. Here, you can edit changes and push it to a common web repository which can be viewed and tracked by everyone.

This can help make code management better and avoid unprofessional means of sharing code such as WhatsApp and mails. It can also make everyone contribute to the overall development of code and have reviews regarding it.

Proposed Repo Structure at TA

The proposed repo structure at TA can be the following -

1. master
    a. dev

The code in the master is not touched. It is not to be disturbed.
The code in dev is the area where we all can sit and develop the code. This branch contains all code relevant to the below folders-

1. FC code
2. GUI
3. Payload code
4. Other important folders that can emerge overtime (GPS, independent sensor development)
5. README.md

Project Workflow

Below is the project workflow each and everyone must follow while editing code and pushing into GitHub -

1. Setup a local directory and configure git within it (done only once)
2. Pull the code from the repository onto the local. Pull only that branch relevant to you
3. Make changes to the code
4. Stage the commits and commit them to the repository
5. Push code into the dev-branch only

Finally, before integration, we will perform a pull request from dev to master.

Making a new Repository (For Admin of Repo)

1. To create a new repo on GitHub, navigate to your GitHub account on the web and create a new repository
2. Enter the name of the repository. Enter a valid name and separate the words with hyphens (-).
   Example : sample-repo, my-repo.
   This practice is preferred and you will not run into unnecessary command line issues later.
3. Initialise the GitHub repository on the web with a .gitignore file (good practise) and a README.md (markdown file). The README.md file contains all project level overview and description
4. After setting up the repo on the web, we must then create a SSH (Secure Shell) layer for our GitHub repository.

On your local machine, create a new directory and give it the remote repo name. Open the git-bash terminal and set up your SSH key for your repository.

https://docs.github.com/en/authentication/connecting-to-github-with-ssh/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent
https://docs.github.com/en/authentication/connecting-to-github-with-ssh/adding-a-new-ssh-key-to-your-github-account
https://docs.github.com/en/authentication/connecting-to-github-with-ssh/testing-your-ssh-connection

5. After setting up the SSH key, we can then authenticate into GitHub from the local system and clone the code from remote repository

Setting up GitHub Local Repository

1. On the folder created in the local machine which you want to be your local repository, type the following command to initialise the local repository

   git init

2. After initializing the local directory as a local repository, clone the existing repository into the local folder using the following command

   git clone <repository URL>

3. Now, the local directory is the local copy of the remote repository. You can edit and make changes to your code from the local using a proper IDE.

Staging Changes

1. After you have edited your code in the local repository, you have to stage your code before pushing it to GitHub remote
2. To check the status of the repository (in terms of **modified** and **untracked files**), type the following command -

   git status .

3. Now, given that you have modified existing code or added new code, make sure that you add the new files into the staging area before pushing it. To add the modified or new files, type the following command.

git add <filename>

You can then check the status of the local repository again by typing (**git status .**) . Do not add any cached files which may reflect in untracked or modified sections during staging.

4. After adding all the files, you have to commit the changes. Type the following command -

   git commit -m "<your commit message>"

   Make sure that the commit message is clear and crisp. Do not type long commit messages to changes or unrelated info. Keep it crisp.
5. After committing the code, you would have to push it to the remote repository. However, it is a bad practice to push code to the master branch. Instead, you would have to checkout to a branch dedicated to you for development of your code.
   During final code integration, all the branches will be reviewed and then finally merged into the main branch.
   To checkout to a new branch, type the following command-

   git checkout -b <new-branch-name>

6. After checking out into the new branch, type the following command to push your final edited code into the remote repository.

   git push origin <new-branch-name>

7. To finally verify the changes, go to the website and refresh the repository page. You can view your commits and code in the remote repository.
8. Remember that step 5 is to be executed the first time you create a new branch. This should not be done every time. This is because you create your branch only once and work on your branch only. Creating unnecessary branches only increases the size of the repository tree, and makes it hard to integrate and manage code later.
   In order to checkout to another existing branch, type the following command -

   git checkout <your-branch-name>

Cloning Repository to Local

1. To clone the repository to the local machine for editing changes, open the folder where you have initialised as a local repository and type the following commands in sequence -

   git checkout <your-branch-name>

git pull

2. You would then observe that the local repository has all the files cloned from the remote repository. You can then make the existing changes to the code when required.
3. Once changes have been made, stage the changes, commit the code and then push it to the remote repository.