# ARC FTC Programming Guide

Jackson Isenberg

## 1 Introduction

Members are expected to follow these practices explicitly and carefully. The purpose of this is to ensure proper management and flow of programs and also solid organization. It also increases member participation and improves upon the code review process.

## 2 Organization

The organization for our club is located here.

### 2.1 Repositories

In addition to extraneous projects, all teams will store their seasonal code in repositories here. The programming leads of each team will create a public repository that will be the central "hub" for their team's code. After the end of the season, the repository will be archived. This repository can only be pushed to by members of the organization (the programming leads). The programming leads generally do not write a lot of the code and instead mostly do code review of pull requests. Exceptions can be made for when software teams are relatively small.

#### 2.1.1 Branches

The org repository will have two branches: a stable `main` branch and an unstable `dev` branch. `dev` is short for development. This is where code is pushed and tested. All pull requests from forks should be squash-and-merged into this branch. We will talk more about how pull requests should be handled in a later section.

### 2.2 Forks

Programming team members are expected to fork from the main team repository in the organization. Forks work well for when you want to make a pull request, which is what members should do. Members will be tasked with writing specific portions of code, collaborating, and will push these changes to their individual forks. When a task is finished, the member will make a pull request from their fork to the `dev` branch of the org repository.

If the lead programmer is also working on code with the team members, they too should make a fork.

#### 2.2.1 Branches

The `main` branch of forked repositories should be used to pull from upstream (the org repository) `dev` branch. Forks are used for testing and production of code. Whenever a new feature is introduced, a branch should be made in the fork. The branch should follow a name scheme of the day's date it was produced. For example, a feature introduced on September 17th, 2021 would be on a branch named `20210917`. Do **NOT** delete these branches. They should be kept as a way of keeping track of which features were produced when.

## 3 Git

Git is a powerful tool that we will use to do all of our version control. You should already have Android Studio installed, which is where you will edit your code. Git is how you will produce proper version control. You can download Git with this link.

## 3.1 Local Development

### 3.1.1 Clone

Local development starts at the clone phase. The first step is to clone your fork of the org repository onto your computer. You can do this by navigating to the desired location for your code folder, right clicking and opening bash, and using the `git clone <repository-link>` command, but replace `<repository-link>` with the actual link to the your fork (found by clicking the green drop-down on GitHub and copying the https link). To paste you will need to press the "insert" option on your keyboard.

### 3.1.2 Set Upstream

Navigate into the project root directory and open a bash session. You will only need to do this once after doing your clone. Use the command `git remote add upstream <repository-link>` but replace `<repository-link>` with a link to the main repository in the org.

### 3.1.3 Checkout

Every time before you start working, navigate into your root directory in file explorer and open a bash session. Do `git checkout main`. First, pull changes from upstream with `git pull --set-upstream upstream dev`. You will then create a new branch with the day's date following the previously stated conventions. You can do this with the command `git checkout -b <branch-name>` except replace `<branch-name>` with the formatted date.

### 3.1.4 Commits

When you finish working in Android Studio, build your code. You can do this by clicking on the hammer in the top right. If everything builds successfully, you can make a commit. Navigate into the root folder of your project in file explorer. Open git bash and use the `git add -A` command. This adds all of your changes, additions, and deletions to your current stash (commit queue). Then use the `git commit -m "message"` command, except replace `"message"` with an explanatory commit message, such as "added servo gate subsystem". This clears the stash and adds everything to a current commit. You should commit often. This is our way of saving progress. When needed, at least once or week, and/or the current task is complete, push your code. This is done very simply with the push command `git push --set-upstream origin <branch-name>`.

## 3.2 Project Development

Programming leads will clone the FTC SDK for that season, create a new repository in the org, set that org repository as a remote called "repo" with the command `git remote add repo <repository-link>`. Then, checkout a new branch called `dev` via `git checkout -b dev` and then modify the README to have the team number at the top. Add these changes through `git add -A` and then make a commit with the message "initial commit". Then do `git push --set-upstream repo dev`. This is all that needs to be done.

# 4 Pull Requests

## 4.1 Fork to Dev

First, merge your feature branch into your main branch. Do this by opening a pull request in the fork of your current feature branch to the main branch. There should not be any merge conflicts. If there are, solve them. If not, rebase and merge into main. Then, make a PR from main into the org repository's dev branch. If there are merge conflicts, fix them. If not, request a review from the programming lead and/or one mentor. Once it passes the review, it will be squashed and merged into `dev`.

## 4.2 Dev to Master

The programming lead will pull changes from `dev` into their local computer and test the code on a robot/mechanism/prototype. The lead programmer will then fix any issues if they arise (or this task can be delegated to the team). If it passes and has the desired behavior, then the programming lead will make a PR from `dev` into `main` and add a mentor for review. Once it passed review, it will be merged normally.