GitFlow

This document describes the code workflow methology used. Some extra background reading can be found [here](https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow), but is not required. This document will start with a overview and then provide some examples that are in the context of this project. The diagram below will be explained later.

git-start(8)(4).png

# Overview

We have 2 main branches (stable and development) and then many other branches known as feature branches. The stable branch contains tested code that we know works and any bugs in it should be minor. The development branch will contain code that compiles and may of been tested land side but needs to be tested on the boat in the water. Ideally when the boat is taken out to be sailed it will run whatever is in the development branch.

A feature branch is branch where development on a specific feature(A feature can be any significant code fix or addition, a example may be a new waypoint manager) happens, it is completely isolated from features everyone else is working on. Each feature should have its own branch and once the code is ready to be tested on the boat it is then merged into the development branch for testing on the boat. After it has been tested on the boat, any fixes that need to be made for it should occur in the feature branch which should then be once again merged into the development branch.

Once a feature is considered complete and tested it should be merged into the stable branch, this can either be done using a release branch or by just merging the development branch. A release branch should be used when a new feature needs to be tested while another set of features that are in the development branch are ready to be merged into master.

In a ideal world the stable branch should contain last sprints codebase and develop should contain the current sprints code.

# Examples

In this example we will look at two features(WaypointManager and SystemLogger) being developed by two separate people(Mary and Ben).

Mary is working on a new waypoint manager and ben is working on a new system logger. Both of them may touch similar code areas to ensure their features work correctly. If they were working on the same code branch this could lead to a lot of code merge conflicts when they commit code. So to solve this problem they use Gitflow.

They start with a stable branch called master and development branch called develop. At the start the development branch is identical to the stable branch.

git-start.png

Mary begins work on her assigned feature and creates a new branch based on the branch called master. She uses the following command:

git checkout master # this commands switches her codebase(The working branch) to the master branch

git checkout -b waypoint\_manager # this command creates a new branch based on her current working branch which is the branch develop

git-start(1).png

Mary makes two code commits on her branch using the commands git add and git commit.

git-start(2).png

Ben creates a new branch for his feature using the same commands Mary did.

git-start(3).png

Ben then makes a single code commit.

git-start(4).png

Mary decides she is happy with her code and wants to merge it into the develop branch for testing on the boat. She uses the following commands to do so.

git push # This makes a copy of her branch on github so that others can look at it and check it out.

git checkout develop # switches to the develop branch

git pull origin develop # this will pull the newest version of the develop branch onto her machine

git checkout waypoint\_manager # Switches back to her branch she has been working on (red branch)

git merge develop # this command merges develop into her branch, this is good practice. Any merge conflicts will come up here and should be fixed before merging the code back into develop.

git-start(6).png

At this point Mary’s branch (The red one) has her code in it and any other code that was put in the branch develop after she created her branch. As nothing else has been put in her branch, when she merges develop into her branch git will say something along the lines of nothing to do here. So as she has no merge conflicts she will now merge her code into the develop branch, essentially updating it. This is done using the following commands:

git checkout develop # Switches her working branch to the develop branch

git merge waypoint\_manager # This merges her branch into the develop branch.

As Mary merged develop into her branch first there should be no code conflicts as she should have all the changes made in develop.

git-start(7).png

Ben makes one more commit and then decides his code is ready for the boat. So he adds his commit, pushes his code to the cloud. he then checks he has the latest version of develop by checking out the branch develop and doing a git pull. The develop branch now has all the code Mary has written. Ben then checkouts his branch again and merges develop into his branch.

git-start(8)(1).png

Mary and Ben edited similar lines of code and so have when Ben merged the develop branch into his branch he had some merge conflicts. After fixing this he can then merge his branch back into develop.

git-start(8)(2).png

At this point both of the new features are in the develop branch. After some testing on the boat these two new features are seen as working and complete so the develop branch is merged back into the stable branch, master.

git-start(8)(3).png

# Common Git Commands

## Cloning the repository

git clone https://github.com/pophaax/sailingrobot.git

cd sailingrobot

git submodule init # initialises all the submodules (Only need to do this once)

git submodule update #Updates the submodule to the latest version

## Branch Operations

To change to a new branch:

git checkout [name of branch]

To create a new branch

git checkout -b [name of branch]

To merge one branch into another branch you need to checkout the branch you want the code to be merged into then run the following command:

git merge [name of the branch you wish to merge into the working branch]

## Adding code and commiting

Once you have added code it needs to be added to a commit and then commit this should be done often.

Adding code is simple, first its good to see what code has actually changed by using git status. Lines that are green in git status are files that have already been added to the commit.

git status

Adding code to a git commit:

git add -u # This command adds all changes to existing files to the commit

git add [path of file] # This command will add a single file or folder

To commit code:

git commit -m “type a message about what you committed in these quotes”

Only code that has been “staged” (Added) will be commited.

## Updating and uploading code

To update the current working branch:

git pull origin [branch-name]

To upload all the current git commits in your branch to github

git push origin [branch-name]