# Mobile Git Flow

## Introduction

This document defines the git branching strategy, flow and procedures to be followed by mobile developers whilst developing mobile apps. The target audience for this document is any stakeholder who wishes to discover the Git processes the mobile teams follow and any developer engaging on the mobile project to reference this document as a requisite guideline.

As a developer or stakeholder please reach out at anytime to the Android or iOS lead at anytime for any queries or suggestions to be considered.

## Git Flow

The below diagram depicts and summarises the general git flow for the mobile team in various scenarios. Each subsection below describes each scenario in more detail.

### Feature Development

1. Branch off Dev
2. Name branch according to convention
3. Develop
4. Local Test
5. Update Chang Log
   a. On your branch document each major change under the unreleased section. See change log in the project for more detail.
6. Run auto format on each file
7. Pull Dev branch into local feature branch and resolve any conflicts for merge request.
   a. Dev branch may have merged signed off changes(other feature branches) that need to be included in the feature branch going back into dev.
8. Create PR to Dev and gain approval
9. Distribute build for Manual QA Testing
   a. Pre-emptive deployments to QA for very simple works or items you are confident wont receive change requests whilst in CR
10. Merge to Dev

    a. Only once code review and manual testing has passed

**Creating a Release**

1. Branch off Dev to a release branch
2. Name Release Branch according to convention
3. Update Change Log
4. Update Build version code and version name
5. Run auto format on each file
6. Create a PR to QA
    a. Gain sign off
    b. This will trigger a deploy to QA personnel, amongst other automation
7. Create a PR to Staging
    a. Gain sign off
    b. This will trigger a deploy to internal or external beta testers
8. Create a PR to Master
    a. Only push to Master when 100% confident, this will trigger a publish to the App Stores

**Creating a Hotfix**

1. Branch off the latest release branch to form a new release branch
2. Name Release Branch according to convention
3. Branch your hotfix branch off your new release branch
4. Name Hotfix Branch according to convention
5. Apply Fix
6. Local Test
7. Update Change Log
8. Run auto format on each file
9. Create a PR back into your new release branch
10. Update Build version code and version name
11. Create a PR to QA

    a. Gain sign off
    b. This will trigger a deploy to QA personnel, amongst other automation

12. Create a PR to Staging
    a. Gain sign off
    b. This will trigger a deploy to internal or external beta testers
13. Create a PR to Master
    a. Only push to Master when 100% confident, this will trigger a publish to the App Stores

### Rolling back a Release

1. Branch off the release branch you want to roll back to, to form a new release branch
2. Name Release Branch according to convention
3. Local Test
4. Update Change Log
5. Run auto format on each file
6. Update Build version code and version name
7. Create a PR to QA
    a. Gain sign off
    b. This will trigger a deploy to QA personnel, amongst other automation
8. Create a PR to Staging
    a. Gain sign off
    b. This will trigger a deploy to internal or external beta testers
9. Create a PR to Master
    a. Only push to Master when 100% confident, this will trigger a publish to the App Stores

## Branching

### Naming Conventions

- Feature Branches
    - [Developer Name]\[Ticket Type]\[Ticket ID]_Short_Description
- Release Branches
    - Release\[App Version]_Release
    - Rollback\[App Version]_Rollback

- Hotfix Branches
    - Hotfix\[Ticket ID]_Hotfix
- Pair / Peer Programming
    - Pair\[Ticket ID]_Short_Description

**Branch Management**

- Hand Over
    - When needing to take over a task from another developer, branch off of their existing branch to create your own copy. Name the branch as per convention.
- Pair / Peer Programming
    - When working closely with another developer on a task, create a common branch and name it as per convention.
    - This common branch will serve as a central branch for all developer's works to be merged into.
    - Assign or share the responsibility of keeping this branch up to date with the Dev branch once a day.

**Pull Request**

- Approvals
    - Add each developer of your platform
    - All developers for the respective platform have to approve for now at least
    - Allow 24 hours as a rule of thumb for all developers to have reviewed your PR, after this period start following up with those who you are still awaiting feedback from. Should you still not get feedback raise this in standup as a blocker.
- Description
    - Ticket ID
    - Summary of the work and how it was achieved
- Comments
    - Add high level comments to your PR
    - Add detailed comments to more complicated parts of your PR
    - Referencing in which order to review files can be quite beneficial
    - Comments on a PR is really up to you as a developer, treat your fellow developers as you would like to be treated. If you have to ask should I comment, you probably should and will result in a quicker review turn around

**Commit Messages**

- Click Up integration
    - Start with your ticket ID, then 2 line breaks. This will link your commit to your ticket in click up.
    - Commit messages should have a basic level of what you have changed / added / removed.