iSENSE Mobile Applications:

Guide On How to Use Git and Github, Test Applications, and Publish to the Play Store

Written by Mike Stowell

Revisions by Jeremy Poulin

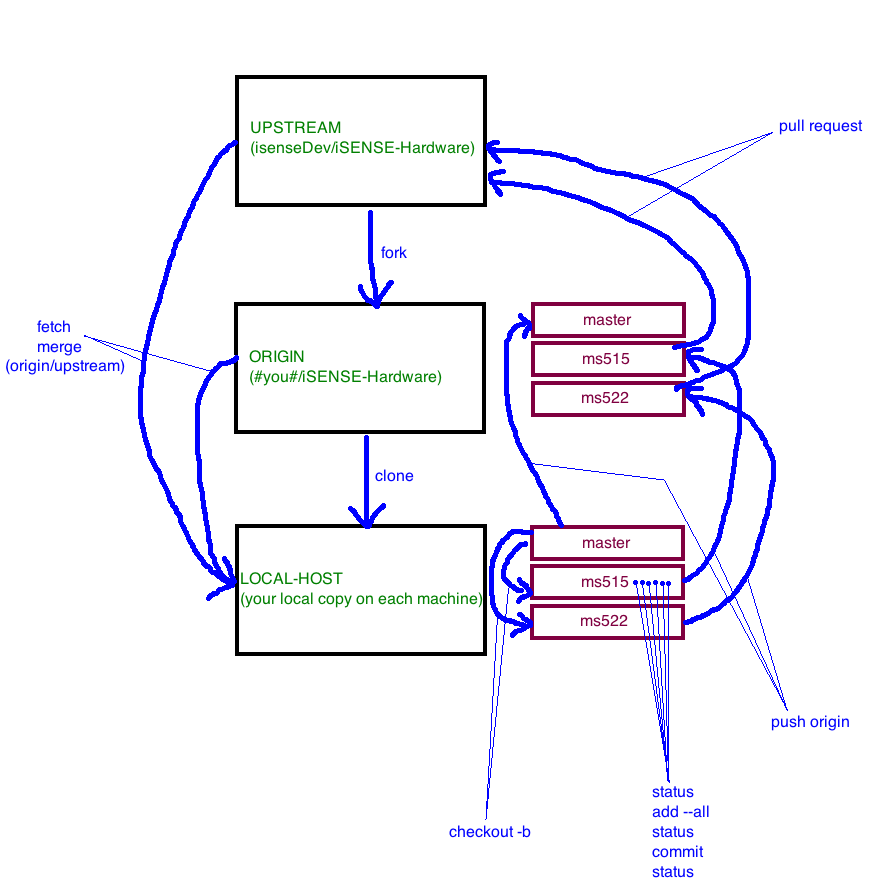
Revision 1

**Getting Started With Git**

If you’re beginning to use Git on a new machine, go to help.github.com to learn how to set up Git for your particular OS. If you’re using Windows, good luck. (If you’re reading this document, you probably already have an account on github.com. If not, sign up for one.)

Please note: make sure when installing git on your machine, download the command line tools and not the Github desktop client, as this client does not allow you to follow our particular workflow model easily and precisely.

**Workflow Model**



**Beginning Development**

So you’ve followed the tutorial on Github and you’ve already learned about forking your own copy of isenseDev/iSENSE-Hardware and cloning YOUR copy (not isenseDev’s) of iSENSE-Hardware to your machine. If you type

$ git remote –v

You should see origin (which should contain in the link #you#/iSENSE-Hardware). If you don’t see an “upstream” remote, we need to add one. Do this with:

$ git remote add upstream \

https://github.com/isenseDev/iSENSE-Hardware.git

Now you have a pointer to your repository (origin) and the top-level repository (upstream).

You’ll be assigned issues on the Github issue tracker page (https://github.com/

isenseDev/iSENSE-Hardware/issues?sort=created&state=open).

To begin working on a particular issue #, you want to create a new “branch” on your local-host. First, make sure your local master branch is up to date by running

$ git fetch upstream or $ git pull upstream master

$ git merge upstream/master

while on your master branch (indicated by typing “git branch” and seeing an “\*” next to the word “master”. If you’re not on your master branch, type “git checkout master”). Now you can create a new branch. Name this branch as your first initial, last initial, and issue #. For example, if you are John Doe working on issue 515, do this like:

$ git checkout –b jd515

You are now free to begin development on that issue and THAT ISSUE ONLY.

**When you’re done for the day**

Let’s assume you haven’t finished the bug/issue yet, but you’re done working for today. There’s several commands for you to execute. These are:

$ git status

$ git add --all // (that’s two dashes)

$ git status

$ git commit -m “changes\_I\_have\_made”

$ git status

Yes, you really WILL get familiar with “git status”. This command shows you every file you’ve worked on since you began development on this branch. In other words, it informs you what you have modified, added, or deleted. When running the initial “git status”, make sure it only reports files you know you’ve changed. If it looks like something else has changed that you don’t want saved (for example, you looked at the API code and accidently modified it but don’t want to push it), you will not be able to run the convenient “git add --all” command later. We’ll explain this soon.

If all changes seem sane, run “git add --all”. This adds all files you’ve added/modified/deleted to git’s memory so it knows that when you run a commit command, it will track those files. If I told you not to run “git add --all”, you’ll need to run “git add” on every single file you’ve added/modified and “git rm” on every single file you’ve deleted. Yes, it’s a pain. However, if you already added a file you didn’t want tracked, you can remove this by running:

$ git rm --cached file\_you\_want\_untracked

Again, run “git status”. It should now say “changes to be committed.” If everything here is correct, you are ready to move on.

Run “git commit –m “changes\_I\_have\_made””, of course replacing “changes\_I\_have\_made” with a message explaining what the changes you are committing entail. MAKE SURE THIS MESSAGE IS DESCRIPTIVE. This is important so that if something breaks in the future, we know exactly which commit broke it. When things break, we have the ability to revert your code to a previous commit, so I re-iterate, MAKE SURE THIS MESSAGE IS DESCRIPTIVE. As an aside, if you type “git commit” without –m, your default text editor will be displayed for you to enter your commit message into.

Run “git status” one more time. There should be no more changes to be committed.

Now that everything you’ve worked on your local machine is ready to push up to your Github repository, you can execute:

$ git push origin branch\_name

where “branch\_name” is the name of your branch you are currently developing on (don’t remember your branch name? – use “git branch” to figure that out).

Now your changes are on YOUR Github repository and can be accessed by any other machine with a pointer to that repository. So, if you want to work from another machine now, after doing

$ git fetch upstream

$ git merge upstream/master

(which should be done at the start of ANY TIME you begin developing for the day) you can do

$ git fetch origin

$ git checkout –b origin/branch\_name

where “branch\_name” is, again, the branch you are currently developing on. Keep in mind that you could be working on more than one branch at a time, but we’ll get to that later.

**Upon completing your assigned issue**

You are still going to push your changes up to your repository using the steps explained in the section above. However, now you’re ready to test. Yes, TEST YOUR CHANGES. A LOT.

Of course, being a good software developer, you’ve been testing your app every step of the way while working on an issue. Wait, you haven’t? Get in that habit.

Every Friday, people around the lab will be testing Android applications. If you want your application tested, there’s a few steps you need to follow.

First, either generate a signed .apk file (explained later) that’s ready to push to the Play Store or get your unsigned .apk file from the /bin folder of your application. Add this .apk file to the Android/Tests directory, following the steps and reading over the details in “test\_sheet.txt” (found in that directory).

When applications are tested, users will comment in test\_sheet.txt regarding your new application. If anyone reports any bugs, you must fix these using the same development cycle described earlier. Otherwise, your application will be ready to be pull requested – the final step.

**Submitting a Pull Request**

When you believe your application is COMPLETELY STABLE, has NO BUGS/ISSUES with it, and IS IN A STATE THAT CAN BE PUSHED TO THE PLAY STORE AT ANY TIME, you may submit a “pull request”. This is done on Github, not on the command line.

Github changes it’s UI quite often, but as of right now, if you view your iSENSE-Hardware repository, change to your development branch (using the dropdown that probably says “master” right now). From here, click the little green button to the left to submit a pull request.

A pull request will ask you to name it (briefly describe what the pull request is) and provide a description. In the description, you MUST put the issue number you’re working on after a “#”. For example, if you’re working on issue 515, your description may look like:

“Fixed menu setup from not showing up, redesigned setup dialog.

#515”

By doing so, Github will create a link from your pull request to that issue so that on the issue page, it will state which pull requests are related to that issue.

Once you’ve submitted a pull request, WAIT. DO NOT MERGE THEM YOURSELVES, UNLESS GRANTED PERMISSION TO DO SO. A higher-seniority Android developer (most likely Mike) will see that you have submitted a pull request and will first make sure you have no remaining bugs left in the Android/Tests folder. If this is the case, this person will merge your pull request, and your development for that issue is complete.

**After Your Changes Are Pulled In**

You are no longer working on that assigned issue now that you’ve completed it. You may close this issue on Github by going to your issue on the issue page and clicking the “close” button (if you do not have permission to close an issue, ask someone else to close it for you).

Now it’s time to update your master branch. In the terminal, run

$ git checkout master

to check out your master branch, and get your repository’s master branch up to date with

$ git fetch upstream

$ git merge upstream/master

$ git push origin master

Perform those steps on every machine you are developing on. Now your master branch, both locally and on your repository, is up-to-date, containing all changes you and anyone else who has merged a pull request in has made.

Typically at this point, you can remove branches locally and on your repository on Github after you have completed development on them. However, it may be safe to hold on to the last 4 or 5 branches you’ve developed on in case things go wrong or changes seem to “disappear.” Once you are ready to remove a branch, however, you can do this locally and on your repository by executing in the terminal:

$ git branch –D my\_branch

$ git push origin :my\_branch

If you prefer syntactic sugar, git push origin :my\_branch can also be written:

t

$ git push origin --delete my\_branch

**Multiple Branch Development**

Let’s say you have two issues assigned to you, both completely unrelated. You may already be working on issue 555, for example. However, it is requested that you fix issue 558 as soon as possible. How can you fix issue 558 if you don’t think issue 555 will be complete for quite some time? It’s time you learn about multiple branches.

If you’ve been following the workflow model described earlier, you have been constantly pushing your changes up to your repository and ensuring you are fetching/merging all the latest, stable changes from upstream. Good. Before working on your new issue, ensure that you push up all your changes from the other branch to your repository.

Now, you should

$ git checkout master

to get to your master branch. Don’t worry, all of your code you were just working on isn’t lost since you just pushed it to your repository. Now follow the regular steps in ensuring your master branch is up to date. That is, perform:

$ git fetch upstream

$ git merge upstream/master

Now you can use:

$ git checkout –b new\_branch

to begin working on the new issue. Remember, for example, if your name is John Doe and the new issue is #558, name this branch “jd558.”

Work on this issue and perform the entire git workflow model just as you have been before. But remember, this is issue 558. This issue is unrelated to issue 555, so make sure when on your 558 branch that you are ONLY working on issue 558. Should you want to go back to working on issue 555, push all your 558 changes to your repository and switch your branch back to the 555 branch by running:

$ git checkout jd555 // (replace “jd” with your initials or we will laugh)

Now you know how to work on two unrelated bugs at once. However, at some point, you’re going to finish issue 558. Perform the whole push/pull request routine as usual for BRANCH 558 ONLY. Do NOT incorporate issue 555 changes into that pull request (in other words, only perform one pull request at a time… never queue up pull requests).

Once your 558 branch has been merged with the top-level upstream repository via an approved pull request, and once you are back to developing on your 555 branch, you must make sure you merge with upstream before pushing/pull requesting branch 558. In other words, as soon as you see your pull request approved for 558, switch to branch 555 and perform:

$ git fetch upstream

$ git merge upstream/master

to receive the latest changes at upstream (which incorporates your changes from 558 as well). Now your 555 branch includes a completed, already-merged version of your code from 558 as well as what you’re working on currently for issue 555. Now once you are done developing 555 and have tested it, you can submit that branch for a pull request. Once merged into upstream, upstream will now contain both issue fixes.

**Other Notes Regarding Git/Github**

* upstream, the top-level isenseDev/iSENSE-Hardware repository, only has one branch: this is named “master” (hence why you are always fetching/merging the “master” branch from upstream). To re-iterate, always make sure you are only pull requesting when your code is stable. If unstable code gets into upstream/master via a bad pull request, no one can fetch a stable version, nor can we sign that version to the Play Store on a moment’s notice.
* Sometimes when merging branches/merging from upstream, you may get a giant, annoying message that says something like “MERGE CONFLICT”. These are difficult to deal with, and should you see one, come to someone that knows about these to give you a personal tutorial on what to do with these. A merge conflict occurs when your current working code conflicts with code on the repository and git can’t figure out how to merge it itself. Conflicts may be common when you’re working on multiple branches if you aren’t careful with what files you are changing. You can avoid most conflicts by only working on one development branch at a time.
* Some clarification on forking/cloning: forking is only done once when you set up your account on Github.com. You only need to replicate isenseDev/iSENSE-Hardware’s repository once. HOWEVER, cloning the repository via command line is what you will do once for each machine you are working on. Fork gives you a replica repository in the cloud, clone gives you a copy of your repository locally on each machine.
* NEVER pull request your master branch! If you follow the workflow model correctly, you’ll notice that your master branch is no different than upstream’s stable, working master branch. The only reason you have a master branch in the first place that is a copy of upstream’s is so that you can create new branches that start off with the most up-to-date code

**Publishing to the Play Store**

Once your application has been thoroughly tested and pull requested, you can publish the application to the Google Play Store.

First, go through this checklist:

* Is my code on my local copy a direct version of what is in upstream’s repository (i.e. I haven’t made any changes locally yet)?
* Have I ensured that my application is not using dev? (i.e. api.useDev(true) should ABSOLUTELY NOT EVER be ANYWHERE in your application)
* Have I incremented the version number in the manifest?
* Have I updated the version code in the manifest?

If you pass through this checklist, then you’re (probably) ready.

It’s time for you to sign your application as a signed .apk file. If you don’t have the iSENSE Android keystore signing certificate (call it our “keystore” for short and everyone will know what you’re talking about), ask anyone who has it to e-mail it to you, and hold onto that e-mail.

To sign an application in Eclipse, go to File -> Export, and export an Android application. Select your application. Now it’ll ask for the keystore. Select our keystore by browsing your local file system and enter the password (which you should also ask from the person who sent you the keystore file). After that, you will be asked to select an alias. As of right now, all apps are signed under the “car-ramp-physics” alias for convenience. Use it, don’t mind it. The password is the same as the keystore password. The last thing you need to do is save the .apk file somewhere. Save it in our Android/Google\_Play\_APK\_Files folder tracked by git, and MAKE SURE YOU CHANGE THE APK FILE NAME to the name of your application appended with the version number, using all underscores. For example, if you are signing Data Walk version 3.5, the default .apk file name may be

DataWalk.apk

but you should change it to

DataWalk\_v3\_5.apk

Your application is now exported as a SIGNED .apk. The Google Play Store only accepts SIGNED .apk files. (For reference, an UNSIGNED .apk file is generated any time you simply run your application from Eclipse onto an emulator/device in the /bin folder of your project).

To put your new .apk file on the Play Store, you need to sign into g-mail with the iSENSE account (again, ask someone for this). Once signed in, search Google for “google play developer console,” and it’ll probably be the first link (the dev console link seems to change frequently which is why I won’t paste the current, long link here).

If you see a list of our apps, you’re in the right place. Select your application and go to the .apk tab. Select the option to upload a new .apk file, drag & drop your new .apk onto the window, and should you have done everything correctly, your .apk file will upload. Once done, select “publish” in the top right corner and within around 2 to 3 hours, you’re new .apk file will be featured on the Play Store.

However, you’re not done here. You should now visit the store-listing tab for your application. Go check these things:

* Make sure the 512x512 icon file on the Play Store listing is the exact same icon as the one used by your application.
* Update the screenshots of your application if your UI changed, even if it’s a very minor update. Try to get at least 3 screenshots for your app (although Google only requires 2).
* Update the description of your app by changing it to say what your updated for that version number. For example, if I just published Car Ramp Physics v3.1 with linear acceleration updates, my new description may be:

v3.1: Added linear acceleration capability to the application.