Getting git submodule to track a branch

A submodule in a git repository is like a sub-directory which is really a separate git repository in its own right.

There is a special **git submodule** command included with git, and this command takes various arguments in order to add/update/delete your submodules.

You can set the submodule to track a particular branch (requires git 1.8.2+), which is what we are doing with Komodo, or you can reference a particular repository commit (the later requires updating the main repository whenever you want to pull in new changes from the module - i.e. updating the commit hash reference).

One special thing to note about submodules, is that by default they are first initialized into a detached head state (like an anonymous git branch), so if you want to work on this submodule code later, you'll first need to update the information (checkout) to be able to push your changes correctly.

Creating

To create a submodule, you create a new repository to host your module code and then link that into the main repository project. For example, we've created a trackchanges repository (a new feature coming in Komodo 9) that we're linking into the main Komodo Edit repository as a submodule:

Using the **-b** argument means we want to follow the master branch of the trackchanges repository, and after running this command we'll have an empty src/modules/trackchanges directory.

You'll need to run a special submodule initialize command (i.e. after cloning the main repository) to fetch the code for the first time:

```
git submodule update --init
```

Updating

Now, git uses the *update* command to also fetch and apply updates, but this time the arguments are slightly different:

```
git submodule update --remote
```

This will update the trackchanges submodule we added earlier, to the latest version.

Editing

Now that we have a our submodule playing nicely, it would be useful if we could go and edit this submodule code in place, making commits and pushes, which is exactly what we'll do next. It requires a few additional git commands to change from a detached head into a proper remote branch.

```
cd src/modules/trackchanges
git checkout master
```

This gets you into a state where you can edit, commit and push your changes back to the trackchanges repository.

Note that for GitHub you may wish to change from using *https://* into a*ssh://* remote url (to be able to authenticate using ssh) - to do that you'll want to edit the submodule config file, which you can get from the ".git" file in your submodule:

```
cat .git
gitdir: ../../../.git/modules/src/modules/trackchanges
```

This *gitdir* directory holds the git config file for your remote, so update that accordingly.

One thing to note about submodule editing, if you run the *git submodule update –remote* command, your module will go back into being in a detached head state, so you'll need to remember to re-checkout the next time you want to edit it again.

Status

One important thing to note, is that after you make new commits in the submodule, or have pulled in new commits from the submodule, is that**git status** will show the submodule as modified, like this:

```
#   modified:   src/modules/trackchanges (new commits)
```

which is annoying - as we thought we had told git to track the branch and the submodule is now updated to the latest branch commit… what's going on?

This happens to be a limitation of submodule branch tracking - *git submodule add -b* simply adds information about a branch in the .gitmodule file and allows you the option to manually update the submodule object to the latest commit of that specified branch. Your main repository still thinks your submodule is at the original commit (i.e. when the initial *git submodule add -b* was run) - and when you run **git submodule update** (without the –remote option) your submodule will be reset back to that original commit.

You have to go and update that submodule commit reference to the latest code in the remote branch to avoid this:

```
git add src/modules/trackchanges
git commit -m "Update submodule tracking to the latest commit"
```

This additional commit step is somewhat of a pain - and I'm hopeful that future versions of git come up with a more stream-lined method to keep track of the submodule remote branch.

Summary

In summary, git submodules are a flexible way to de-couple your code base, but it does require some additional learning of git commands in order to manage and update these separate code bases. All in all I think this a worthy trade-off and much better than trying to utilize third-party alternatives (like *Git Subtree* or *Google's Repo*).

There are a lot of git resources out there - this Git tutorial is a good thorough reference, otherwise you'll likely find what you need on Stack Overflow