Collaborating on Projects with Git

Edit Page | Page History

Jump to Section

- Overview
- Setup Git
- Git Workflow
  - Pulling Updates
  - Resolving Merge Conflicts
  - Adding New Features
  - Creating Pull Requests
  - Reverting Changes
- Google Maps Access Across Computers

Overview

This is a guide focused on collaborating on Android projects together with teammates using Git.

Setup Git

First, you need to properly setup git for your project:

```
git init
```

Next, setup a **.gitignore** file at the **root** with the contents from this file to ignore files that shouldn't be shared between collaborators.

If you have **already committed files** and need to **remove them after adding the ignore**, you can run this command to remove them before committing.

```
git rm -r --cached .
```

You can now add the initial files to git using the SourceTree / Github client or by typing:

```
git add .
git commit -am "Initial commit"
```

Next, make sure you have **setup a repository on github** and then add that repo as the origin:

```
git remote add origin git@github.com:myusername/reponame.git
```

and now go ahead and push the code to Github with:

```
git push origin master
```

You can also use your favorite Git GUI (for example the Github client) to do a lot of this process as well.

Git Workflow

Pulling Updates

The following outlines how to collaborate with others using git. When first starting a session working on a project, we need to pull any updates pushed by other collaborators:

```
git checkout master
git pull origin master
```

Resolving Merge Conflicts

A merge conflict occurs when two branches have changed the same part of the same file, and then those branches are merged together. For example, if you make a change on a particular line in a file, and your colleague working in a repository makes a change on the exact same line, a merge conflict occurs.

When this sort of conflict occurs, Git writes a special block into the file that contains the contents of both versions where the conflict occurred:

```
the number of planets are
<<<<<<< HEAD
nine
=======
eight
>>>>>>> branch-a
```

To complete this type of merge, use your text editor to resolve the conflict, then add the file and commit it to complete the merge. In this case, Git has trouble understanding which change should be used, so it asks you to help out. Refer to this excellent github guide to resolve these edit merge conflicts.

Adding New Features

New features should be added in special feature branches that allow changes to be made in isolation. First, we can create a new branch to work on:

```
git checkout -b my_branch_name
```

Now changes can be locally committed to the branch:

```
git add .
git commit -am "Initial commit"
```

You can check your current branch:

```
git branch
```

and push changes back to github in a branch with:

```
git push -u origin new_branch_name
```

While working on the branch, be sure to rebase with master to pull in mainline changes:

```
git checkout master
git pull origin master
git checkout your_branch
git rebase master
```

If there are conflicts, they have to be resolved in the files and then you need to run:

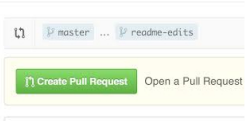
```
git rebase --continue
```

until all conflicts have been resolved.

Creating Pull Requests

When a branch is ready to be merged, a pull request should be opened. Make sure your branch is up to data with master with the rebasing steps above. Then push branch to github with `git push -u origin new_branch_name`.

Go the repository on github and look for:



Press "Create Pull Request" and then if there's an associated issue then in the **description part of the PR** list the issue number you are closing like this: `closes #23` to automatically close the related issue when this is merged.

Tag your team members so they can review your pull request. When you get approval from your team members then merge the pull request back into the mainline.

Reverting Changes

If you have a commit from the past that is working and you made changes locally and want to revert all of your changes since that commit, you can run:

```
git reset --hard
```

If you want to revert your current code to a past commit:

```
git log
...find the commit i.e c1fc1c2d1aa1d37c...
git reset --hard c1fc1c2d1aa1d37c
```

That it! After running that all of your local uncommitted changes will be reverted. For more information, check out this Github guide on how to revert almost anything with Git.

Google Maps Access Across Computers

Often when collaborating on a project with others, you need to have **maps work across multiple computers**. The problem is that the map key fingerprint is different from computer to computer and thus by default maps will only work on the computer that was used to generate the key.

The simplest fix is described in detail within this stackoverflow post but in short you can get the `debug.keystore` from one of the team members, check that into git and then instruct other team members to replace their `debug.keystore` file with the one from repository. See also this link and this guide.

- Adding New Features
- Creating Pull Requests
- Reverting Changes
- Adding New Features
- Creating Pull Requests
- Reverting Changes
- Google Maps Access Across Computers