Assignment

## Branching Workflows

**Long-Running Branches**

Git uses a simple three-way merge, merging from one branch into another multiple times over a long period is generally easy to do. This means you can have several branches that are always open and that you use for different stages of your development cycle; you can merge regularly from some of them into others.

* master branch — possibly only code that has been or will be released.
* another parallel branch named develop or next that they work from or use to test stability  — it isn’t necessarily always stable, but whenever it gets to a stable state, it can be merged into master.
* topic branches (short-lived branches, like your earlier iss53 branch) when they’re ready, to make sure they pass all the tests and don’t introduce bugs.

### Topic Branches

Topic branches, however, are useful in projects of any size. A topic branch is a short-lived branch that you create and use for a single particular feature or related work. it’s common to create, work on, merge, and delete branches several times a day.

## Remote Branches

Remote references are references (pointers) in your remote repositories, including branches, tags, and so on.

Remote-tracking branch names take the form <remote>/<branch>. For instance, if you wanted to see what the master branch on your origin remote looked like as of the last time you communicated with it, you would check the origin/master branch.

origin” is not special

Just like the branch name “master” does not have any special meaning in Git, neither does “origin”. While “master” is the default name for a starting branch when you run git init which is the only reason it’s widely used, “origin” is the default name for a remote when you run git clone. If you run git clone -o booyah instead, then you will have booyah/master as your default remote branch.

### Pushing

When you want to share a branch with the world, you need to push it up to a remote to which you have write access. Your local branches aren’t automatically synchronized to the remotes you write to — you have to explicitly push the branches you want to share. That way, you can use private branches for work you don’t want to share, and push up only the topic branches you want to collaborate on.

### Tracking Branches

Checking out a local branch from a remote-tracking branch automatically creates what is called a “tracking branch” (and the branch it tracks is called an “upstream branch”). Tracking branches are local branches that have a direct relationship to a remote branch. If you’re on a tracking branch and type git pull, Git automatically knows which server to fetch from and which branch to merge in.

When you clone a repository, it generally automatically creates a master branch that tracks origin/master. However, you can set up other tracking branches if you wish — ones that track branches on other remotes, or don’t track the master branch. The simple case is the example you just saw, running git checkout -b <branch> <remote>/<branch>.

**Upstream shorthand**

When you have a tracking branch set up, you can reference its upstream branch with the @{upstream} or @{u} shorthand. So if you’re on the master branch and it’s tracking origin/master, you can say something like git merge @{u} instead of git merge origin/master if you wish.

### Pulling

While the git fetch command will fetch all the changes on the server that you don’t have yet, it will not modify your working directory at all. It will simply get the data for you and let you merge it yourself. However, there is a command called git pull which is essentially a git fetch immediately followed by a git merge in most cases. If you have a tracking branch set up as demonstrated in the last section, either by explicitly setting it or by having it created for you by the clone or checkout commands, git pull will look up what server and branch your current branch is tracking, fetch from that server and then try to merge in that remote branch.

### Deleting Remote Branches

Suppose you’re done with a remote branch — say you and your collaborators are finished with a feature and have merged it into your remote’s master branch (or whatever branch your stable codeline is in). You can delete a remote branch using the --delete option to git push. If you want to delete your serverfix branch from the server, you run the following:

$ git push origin --delete serverfix

## Rebasing

In Git, there are two main ways to integrate changes from one branch into another: the merge and the rebase.

### The Basic Rebase

The easiest way to integrate the branches, as we’ve already covered, is the merge command. It performs a three-way merge between the two latest branch snapshots and the most recent common ancestor of the two,creating a new snapshot (and commit).

there is another way: you can take the patch of the change that was introduced in C4 and reapply it on top of C3. In Git, this is called rebasing. With the rebase command, you can take all the changes that were committed on one branch and replay them on a different branch.