Simple Sequence of GIT Commands

I read the documentation and googled a good bit, but there is no real simple steps to be able to commit your local changes to github. I compiled the following steps and I just want to make sure am doing the right thing. If I changed a file foo.java locally:

1. git status -s //will show me that foo.java has changed
2. git add foo.java //will add it to my local repo
3. git commit -m "my changes" //commit to the local repo
4. git tag "v1.1" //create a tag
5. git push --tags //finally, move the local commit to the remote repo with the new tag. this will prompt for your password. if no tag is set as in step 4, then just

> git push

is enough. right?

I am just trying to make sure that these basic steps for the most use cases is what is required to use github. I am brand new to github, and these steps are working for me, but want to make sure i am not making any fundamental mistake. Please comment if there are any missing steps. Again, am concerned about the most generic day-to-day use (like, am not really concerned about branches, etc. which I will learn on a need basis). Thank you in advance.

git github command push commit

edited May 18 '12 at 18:58

Tomasz Nurkiewicz
191k 24 421 484

asked May 18 '12 at 18:50

RGi
38 1 6

3 Answers

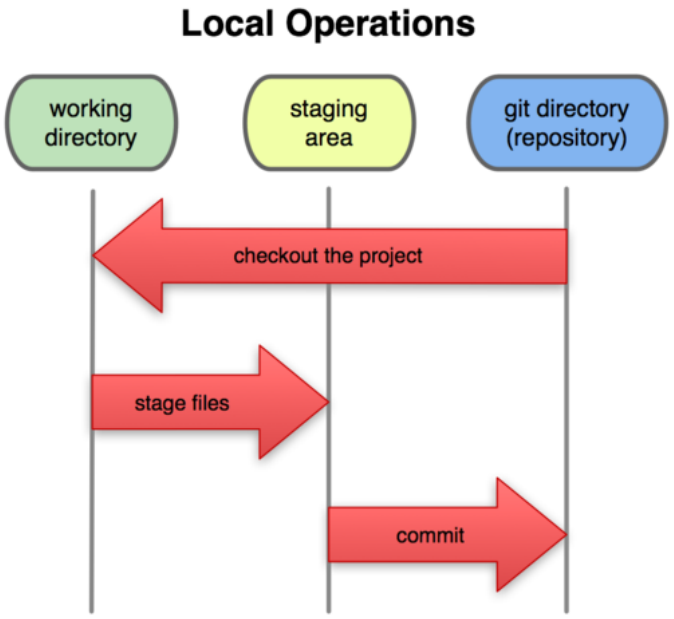
Your steps are fine. To nit-pick slightly, though, about the comments:

The comments about step (2) and (3) are not the best way to think about what's happening, I don't believe.

```
2.git add foo.java     //will add it to my local repo
3.git commit -m "my changes"      //commit to the local repo
```

The step which "adds" your file *to the local repository* is `git-commit`. That's why it's called `commit`; you *commit* changes to the repository. `git-add foo` adds `foo` to the *staging area*, **not** to the repo itself.

Your `git` repository has three "areas", `working`, `staging` and `repository`, depicted here (image taken from the Pro Git book):



You make changes and work in the creatively named "working directory".

When you've made some changes, you want to prepare to make a commit. This is where the "staging area" comes into play. You "stage" the changes that you want to commit, and when you're happy with what the commit will look like, you commit the "staging area" to the "repository". [Note: in the `man` pages, this `staging area` is mostly referred to the `index`].

This allows you a lot of flexibility. You can stage all the changes since your last commit, or you can stage files individually, or you can stage parts of files. You can add and delete files from the staging area without losing changes or messing up the repositories history. That's what the `git add` and `git rm` commands do; they add from the `working directory` to the `staging area` , but they **don't** add directly into the `repository` . (Hopefully the image helps make the distinctions clear).

Your steps are fine. If you want to understand more about branching, commiting, manipulating commits and branches and whatnot, I'd recommend reading the Pro Git book - it's got a whole bunch of pretty pictures and language simple enough that I can understand it ;)

answered May 18 '12 at 20:29

simont
20.9k 7 59 96

After (3), you should be able to call `git push origin master` which will push your current `master` branch to github

answered May 18 '12 at 18:53

ChrisB
1,942 8 30

right, with no tags, if I was using just "git push", it was giving me a warning. So, the right way to do push is then: "git push origin master". Thank you. – RGi May 18 '12 at 18:57

In order to be able to just do a `git push` , you probably need to do a `git push -u origin master` once. That will make your local master branch "track" the remote master branch. After that, you can just do `git push` or `git pull` and it will know what to do. (FYI, this tracking gets setup for you automatically if you get the code via a `git clone` .) – dontangg May 18 '12 at 19:07

I think that that's enough for very basic usage. I'd just like to add two comments:

- It's always a good thing to check what you're adding to the staging area (which is what you're doing with `git add`): either use `git diff` , or do a `git add --patch` , which will start an interactive procedure to let you decide whether to accept or reject each hunk of code you modified. If you messed anything up during this phase, you can always `git reset HEAD` to get the changes back to the working copy (i.e., you would simply undo the add)
- You might want to do steps 2 and 3 together by issuing a `git commit -a -m 'your message'` .

answered May 18 '12 at 20:27

sturmer
194 3 11