**Merging after master has been updated**

Jeffrey and I pulled the latest version of master. I make a change, push it to my branch, check for any updates to master with the handy Discord™ app, open a pull request, which gets approved and merged with master. Jeffrey makes a change, pushes it to his branch, checks for any updates to master with the handy Discord™ app, and sees an update to master he does not have. His branch, he runs:

1. git fetch origin
   1. This will fetch updates from master, useful in case he did not check Discord.
2. git pull origin master or git merge master
   1. This will pull/merge (depending on your command) master into his branch.

He may or may not run into merge conflicts here. See “Resolving conflicts during merge” if there are conflicts. Assuming there are no conflicts:

1. git add .
   1. This will add all modified files to the staging area (see figure)
2. git commit -m “<reasonable commit message>”
   1. Pls send commit messages we can decipher.
3. git push -u origin <his branch name>

He can then open a pull request, which hopefully gets approved before the next pull request.

**Files Unreal changed, but you specifically did not change**

You changed file1. You use git log to check and see what other things have been modified. For some reason, file2, 3, and 4 have inexplicably been modified as well. You are absolutely sure nothing has changed.

1. git commit <your files>
   1. This will specifically commit files
2. git checkout -- .
   1. The . is, for git’s purpose, the same as \* (don’t actually use \* unless you know what you’re doing). This will checkout the original version of all the uncommitted but modified files.
3. Run git status again to see magic (you will only see your modified files in green, everything else should be gone).
   1. This is actually optional, but you should always check this to make sure things are k.
4. git commit -m “<reasonable commit message>”
   1. You still need to supply a reasonable commit message.
5. Push, open pull request, approve as normal.

**Resolving conflicts during merge**

You’ve merged master into your local working directory, and there are some merge conflicts! If they are files you haven’t touched, those files should be overwritten, but Unreal likes to touch a lot of things you directly haven’t, so conflicts naturally arise that way. Thankfully, Git tells you what files have merge conflicts. Run:

1. git checkout master <file>
   1. This will tell git to choose master’s file and overwrite yours. Do this for every file with a conflict you haven’t touched.
2. git commit -m “Resolved merge conflicts”
3. Push, open pull request, approve as normal.

**Reverting a file to a previous commit**

Some updates were made to master, and file was sent back a couple of commits. You’ll have to first find the hash of the last commit that had a working version of the file that messed up. You can find this by going to the project repo, clicking commits, finding that working commit, and copying that 7-character hash value. You don’t even have to open UE4 for this, but it’s helpful to make sure it actually worked.

1. git checkout <hash> <file>
2. git commit -m “<reasonable commit message>”
3. Push, open pull request, approve as normal.