**Creating a new branch**

**Notes:**

* Please make sure that you specify the "up stream" or "parent" when creating a branch. If you do not it will use the current branch you are on and this will cause problems down the road for you.
* You also should never create a branch with uncommitted changes. Please try not to use git stash to get around this ... it can cause problems.
* You can make the <branch\_name>  the task in order to automatically link the diff with a task.

**Using 'git checkout'**

    > git checkout -b <branch\_name> <up stream "parent">

**Using 'git branch'**

    > git branch <branch\_name> <up stream "parent">

**Example**

    > git checkout -b deallogic origin/master

    > git checkout -b T12345 origin/master

**Switching Branches**

* You should only switch branches if you have no uncommitted changes, that is your 'git status' should be clean\*

    > git checkout <branch\_name>

**Example**

    > git checkout deallogic

**From creating a branch to pushing code (with code review via Github PR)**

**0. (optional) Install**[**hub**](https://github.com/github/hub) , a CLI wrapper for git  with some extra features. Mainly to create PRs from the command line in Step 3.

(First, test if you have hub installed already - hub --version . If you do, you can continue to Step 1.)

Follow the [Hub Installation](https://github.com/github/hub#installation) docs for your platform.

If you are on Ubuntu and the instructions still say "we do not recommend snap":

* Find the [latest binary](https://github.com/github/hub/releases/latest) for your platform (you probably want the "Linux 64-bit" one), copy the link address
* Download it onto your machine - e.g. wget <url>
* Unzip the tarball - e.g. tar -xvf <file.tgz>
* Go into the new folder that was extracted, and run the install file - sudo ./install
* Test that hub is now installed - hub --version

Now that hub is installed, you'll need to authenticate on your first use. To do this:

* Generate a github Personal Access Token to be used with hub ([context](https://github.com/github/hub/issues/2655#issuecomment-735836048) for the curious) - <https://github.com/settings/tokens>
  + Click "Generate New Token"
  + Add whatever "Note" you want to label this as
  + Set "Expiration" to what you prefer
  + Under "Select scopes", tick at least repo and gist
* When running hub pull-request for the first time, you'll be asked for your github username and password. For the password, enter your Personal Access Token string
* Should work now

**1. Starting a new task/project**  
    > git checkout -b <task\_name/project> origin/master

**2. Getting the latest changes**  
    > git fetch origin master  
    > git rebase origin/master  
**3. Submitting code for code review -- create a Pull Request (PR)**  
    > git commit -a -m "This is the title of my commit :)"  
  
    # Two options to create your PR:  
    # (1) push your changes to a remote branch, and git will provide you with a link to create a PR  
    > git push origin <current\_local\_branch\_name>:<new\_remote\_branch\_name>  
  
    # (2) use `[hub](https://github.com/github/hub)` to create a PR. need `-p` arg to push AND request a pull request at once.  
    > hub pull-request -p  
  
    # Fill in the PR template, including Summary, Test Plan, etc.

**4. If a reviewer requests changes...**

    # Make the changes  
    > git commit -am "new commit description"  # add the change as a new commit  
     > git push origin <current\_local\_branch\_name>:<remote\_branch\_name>

Please make changes to your PR as new commits - this preserves the previous PR review history and makes it easier for code reviewers to see what's new (thereby making it easier to review our PR and get your changes merged in).

**4a. If your unit test pipeline is failing...**

If the test failure is related to your code change, follow Step 4 to fix the test and update your PR.

If the test failure is NOT related to your code change, and you want to rebase with the latest master to rerun the pipeline:

    # Make the changes  
    > git rebase origin/master  # rebase your branch on the latest master  
    > git push origin +<your\_remote\_branch\_name>  # force push your new updated branch to the remote

Please do this either **before** your PR has been reviewed by anyone, or **after** it has been approved, to make it as easy as possible for your reviewers to deal with your PR.  
(force-pushing the branch renders the previous PR review history mostly useless making it harder to review what changed)

**5. Once your PR is approved, merge your commit into `master` via the PR webpage**

    **Workflow Example**

    > git checkout -b readme\_test origin/master  
    > echo "THIS IS AN EXAMPLE" > README  
    > git commit -m "Create README file for example"  
    > hub pull-request  # and fill in the template  
  
    PR is created and reviewed → someone points out that there needs to be a "!" at the end of the sentence.  
     
    Open README and change content to  THIS IS AN EXAMPLE!  
    > git commit -a --amend  
    > git push origin +readme\_test  
  
    Alice approves the PR → click "Squash and merge" on the webpage.

     If there is a merge conflict:  
    Fetch/rebase and resolve the conflict on your local branch  
     > git fetch  
      > git rebase origin/master

    Push the rebased branch to remote  
    > git push origin +readme\_test

    Have your PR approved again → merge it in via the webpage

**Handling Merge Conflicts**

**1. Getting latest changes**

    > git fetch  
    > git rebase origin/master

**2. Git told you the automatic merges failed**

    > First, rewinding head to replay your work on top of it...  
    > Applying: TEST: DO NOT PUSH  
    > Using index info to reconstruct a base tree...  
    > Falling back to patching base and 3-way merge...  
    > Auto-merging scripts/models/creation/wikipedia/hadoop\_make\_wiki.py  
    > CONFLICT (content): Merge conflict in scripts/models/creation/wikipedia/hadoop\_make\_wiki.py  
    > Failed to merge in the changes.  
    > Patch failed at 0001 TEST: DO NOT PUSH  
    >  
    > When you have resolved this problem run "git rebase --continue".  
    > If you would prefer to skip this patch, instead run "git rebase --skip".  
    > To restore the original branch and stop rebasing run "git rebase --abort".

**3. Have a look with git status**  
    > git status  
    > # Not currently on any branch.  
    > # Unmerged paths:  
    > # (use "git reset HEAD <file>..." to unstage)  
    > # (use "git add/rm <file>..." as appropriate to mark resolution)  
    > #  
    > # both modified: scripts/models/creation/wikipedia/hadoop\_make\_wiki.py  
    > #  
    > no changes added to commit (use "git add" and/or "git commit -a")  
  
**4. Manually merge the file**  
    > vim scripts/models/creation/wikipedia/hadoop\_make\_wiki.py  
  
...  
<<<<<<< HEAD  
features = self.full\_path('hadoop', self.feature\_count\_folder+'\_merged')  
topics = self.full\_path('hadoop', self.topic\_normalized\_folder + '\_merged')  
model\_stats = self.full\_path('hadoop', os.path.join(self.model\_stats\_folder, 'part-00000'))  
=======  
this is to try and force a merge  
features = '[hdfs:///forcemegre/$/%s](hdfs://wiki.wish.site/forcemegre/$/%25s)' % eature\_count\_folder+'\_merged')  
topics = '[hdfs:/please force](http://hdfs/please%20force) a merge conflict' % self.full\_path('hadoop', sellder+'\_merged')  
please force a merge conflict  
model\_stats = '[hdfs:///user/$USER/%s/part-00000](hdfs://wiki.wish.site/user/$USER/%25s/part-00000)' % self.full\_path('hadoop', self.model\_stats\_folder)  
>>>>>>> TEST: DO NOT PUSH  
... 

**5. 'git add' to tell git it has been merged**       
    > git add  scripts/models/creation/wikipedia/hadoop\_make\_wiki.py  
    > git status

    > # Not currently on any branch.  
    > # Changes to be committed:  
    > #   (use "git reset HEAD <file>..." to unstage)  
    > #  
    > #       modified:   scripts/models/creation/wikipedia/hadoop\_make\_wiki.py  
    > #

**6. 'git rebase --continue' to continue with your rebase**  
    > git rebase --continue  
    > Applying: TEST: DO NOT PUS

Note:

If you encountered this issue after running git add and then git continue,

nothing to commit, working tree clean

madong:clroot madong$ git rebase --continue

Applying: PRODUCT-23353 add strings to i18n

No changes - did you forget to use 'git add'?

run git rebase --skip, and resolve the conflict. Once conflict is resolved, run git rebase --continue.

run git status and see if your branch is diverged after rebase. If it did, run git push -f to force update your branch

**Reverting changes to a File**

**Reverting to last committed state**

    > git checkout -- example\_file.py

**Reverting to an old version of a file**  
    > git checkout <sha1> example\_file.py

**Reverting all your uncommitted changes**

     > git reset --hard

**Finding the last person who modified the code ('blame')**

This will tell you who last modified the code you're looking at, and when.  Looking up the git commit will also provide you the diff responsible.

    > git blame /path/to/file  # can add a `-L <line\_number>` optional argument to blame from that line  
  
    > git show <sha1>  # to get more info on commit once you find it  
  
  
    # The last commit to modify the code may be a formatting-related commit and not the original change  
    # If so, you can do:  
    > git blame <sha1>^ /path/to/file  # blames starting from the last commit before <sha1>

We also recommend download any git blame packages into your code editor to make this process even quicker.

**Going back in time**

> git reset <sha1 of the commit to go back to>

**Unamending a Commit**

**1. Backup the code you need (you can patch this later if things go wrong)**

> git diff HEAD~1 > ~/Desktop/mycommit.diff  
  
 **2. Find the amend in your reflog**  
  
> git reflog  
73cab2f HEAD@{0}: commit (amend): Someone else's commit  
05e8444 HEAD@{1}: checkout: moving from master to mybranch

**3. reset to before the amend**  
> git reset --soft HEAD@{1}

**Adding a commit from master to production**

1. Find the commit hash in master that you want to move to production

2. Delete your current production branch, if you have one

   > git branch -D production

3. Fetch updated refs, including the production ref. This will prevent the need to rebase production later, assuming no one pushes to production before you do.

  > git fetch -a

4.  Check out the production branch  
    
  > git branch -b production --track origin/production

5. Cherry-pick commit from master on your local production branch

  > git cherry-pick hash

6. Push your updated production back to origin on github

  > git push origin production:production

**Working with other remote branches**

Essentially, master and production are just remote branches with a special purpose. As needed, you can always create a new remote branch, push to it and rebase from it.  
  > git push origin <your local branch name>: <your remote branch name>

However, please always remember to delete the remote branch that you created when it is no longer in use.  
  > git push origin : <your remote branch name>

**Merging Multiple Commits in the Same Branch**

Let's say you accidentally did another  git commit -a  in your working branch and now you end up with 2 commits, where the second commit is an incremental change of the first, they should belong together.  
     > git rebase --interactive  
This should prompt you within an editor window, you generally would want to select "pick" for the first commit, and "squash" for everything else. Afterwards they'd become ONE.  
More reading [here](https://www.google.com/url?q=https%3A%2F%2Frobots.thoughtbot.com%2Fgit-interactive-rebase-squash-amend-rewriting-history&sa=D&sntz=1&usg=AFQjCNFaLJoSLrKNcrY7W1_7gfHLaPsuyQ)

**The Ultimate Overwrite: Force Branch Update**

\*note:  **Don't do this unless you know what you're doing.**

Find your commit hash where you want to revert to. Let's say we want to go back to hash  6ef745ab103360b81dc5c39ef6896ea0417295be.

> git log  
  
commit 511b218392bb662c0542eac23cac81fcb02baadd  
  
Blah Blah  
  
commit daf7aa753dde9280ac3e2c63fb23d36a9382790b  
  
Blah Blah II  
commit 6ef745ab103360b81dc5c39ef6896ea0417295be  
  
Blah Blah III

> git checkout  417295be  
  
> git branch new\_upper  
  
> git checkout new\_upper  
  
> git push origin -f new\_upper:your\_target