Shabanas-MacBook-Pro:Shopify shabanarumane$ git config --global user.name

Shabana Rumane

Shabanas-MacBook-Pro:Shopify shabanarumane$ git config --global user.email

shabana.rumane@shopify.com

1. *Setup:*
   1. clone an existing git repository into a new directory

Shabanas-MacBook-Pro:LearningGit shabanarumane$ git clone shopify

* 1. view remote repositories
  2. add, remove, and rename a remote repository

1. *Staging:*
   1. describe what the different file statuses (unmodified, new, modified, deleted) mean - git status

New/Untracked file: File is new and has not been part of the working directory

Modified : File has been part of the working directory and modifications have been done. Use git add <filename> #### to add to STAGE i.e. ready to be moved to production or to be COMMITed

Deleted: If the file which has been existing in the working directory and which has previously been “marked”, if that is removed, then when checking the status, it would highlight the file categorized as “Deleted”

Unmodified: All the files that are part of the working directory and no modification/deletion of the file is made, its unmodified. When we do git status, these ones are not explicitly highlighted. It is implied that all the other files are unmodified (except the one in gitignore)

* 1. explain what it means for a file to be ignored

When you want to stage/commit or move files to production, there can be few files viz log files that you do not want to move although changes have been done. If you put them in .gitignore, while checking the status of the working folder, this will not be highlighted even if there are changes done.

* 1. check the staging status of one or more files

git status

* 1. add one or more files to the staging area

When the files are modified (they appear under “Changes not staged for commit”) or new files are added (they appear under “untracked files”), perform git add <filename> for each of the file to move them to Staging area (it will now appear under “Changes to be commited”)

* 1. ignore one or more files or directories

1. *Commits:*
   1. create a new commit for staged files, with a message

There are two ways to commit files into stage:

1. Put comment inline with git commit -m <commit comments>
2. git commit -v This will open an editor showing exactly the changes that have been done. You have to enter the commit comments in here
3. git commit This will open an editor to enable you to enter the commit comments

Shabanas-MacBook-Pro:shopify shabanarumane$ git commit

[master fb1972c7f126] Created a new home card for learning purpose

3 files changed, 106 insertions(+)

create mode 100644 components/home/app/models/home/feed/cards/shab\_29oct\_card.rb

create mode 100644 components/home/test/unit/home/feed/cards/shab\_29oct\_card\_test.rb

To add files to commit without moving to staging area: git commit -a -m <commit comments>

* 1. view the commit history

Use git log

* 1. amend a previous commit

To change a previous commit, make changes to the file you want to change, save, stage and commit with following command git commit --amend

If there was no changes made to the file but the changes need to be done in the commit comment, then run git commit --amend immediately after the last commit and you can change the previous comments

* 1. revert a commit

Use command git reset HEAD <filename> This would move the file from the commit stage (under “Changes to be committed”) to “Changes not staged for commit”

To revert back the changes of the file to the very originial i.e. when it was cloned, give command git checkout --<filename> This will remove all the changes you made to the file and it will be in a state when you first cloned.

* 1. identify the most recently committed version of a file
  2. stash and unstash files
  3. squash multiple commits into a single commit

1. *Diffs:*
   1. show the differences between the working directory and staging

To check the difference between the staging and the changes you have done but not staged - git diff. The result shows by line what changes have been done.

* 1. show the differences between the staging area and the latest commit

To check what changes that you have put in Stage that will go into the commit - git diff --staged. This result will show you what is in staging area and the last commit

* 1. show the differences between the working directory and the most recent commit
  2. show the differences between two commits

1. *Branches:*
   1. identify the current branch

git status This will give you the branch that you are currently on

* 1. create a new branch

git checkout - b <branchname> - This will create a new branch and put you on (or point the HEAD) at the checkout point i.e. the new branch. All the changes done from then on will be tied to the new branch name. To switch to the other or master branch (master is the base), give command git checkout master.

* 1. delete a branch

The process is:

* create a branch - git checkout <branchname>,
* make changes to your repository change/add files,
* commit the changes, git add <filename> - to add to stage first and then git commit -m <commit message> (to add the files to the commit stage)
* Switch back to the base branch - master. git checkout master
* Merge the branch with the master git merge <branchname> to merge the new branch to master
* Delete the branch after it has been used git branch -d <branchname>
  1. checkout an existing branch from a remote repository

git fetch — to get all the updates from all branches

git checkout <branchname> — the existing branch that you are interested in

1. *Fetching, pulling, and conflicts*:
   1. explain the difference between fetching and pulling

git fetch origin gets the new data from the working directory. But it does not integrate the new data on your working files.

git pull origin master not only gets the new data from the working directory but it also merges it with your working files. This can create a merge conflict

* 1. fetch or pull changes from a remote branch

Fetch can be done whenever you want, as it does not impact the files you are working on. Pull should be done only when there is nothing to commit.

* 1. list which files have conflicts when trying to fetch, pull, or merge
  2. resolve conflicts manually and with a merge tool

When you give a command

git checkout master —-this takes you to the main branch

git merge fix2 —— this fix2 is a branch

If the same file has had some difference between the master branch and fix2 branch then the merge will fail and it will state which file has an issue and will ask you to resolve the conflict. If you give git status it will tell you that there was a merge conflict and it has to be resolved. To resolve go to the file, open it. In the file there will be an indicator (<<<<<<<, =======, and >>>>>>>) which will indicate the row where the conflict occurred. After you remove the conflict and remove all unwanted lines, perform git add <filename>. This would stage the file and indicate the resolution of the merge conflict. Once done you can commit the file. git commit. Git would still understand that this commit is part of the merge executed earlier. It would open up the commit message which would contain the description that the commit was because of a merge conflict on the file mentioned. Alternatively more details can be added here

1. *Merging, pushing, and rebasing:*
   1. merge a local branch into another local branch

Use the same method as merging with the master. If branch2 is generated from branch1 and branch2 needs to be merged into branch1 then

git checkout branch1

git merge branch2

* 1. abort a merge

When you have merged a branch and want to undo it. Just give command git merge -- abort. This will revert back to the state before merge. Incase there was a conflict which you resolved and merged and you want to revert back the conflict, just roll back to the commit before the merge happened with "git reset --hard " and start over again

* 1. explain what happens in a fast-forward merge

Say the last commit on the master was C1 and at that point a branch B1 was created and say a few commits were performed in the branch. Now when B1 is merged to the master, the status on the master has not changed since the branch B1 was created. Branch B1 is more like a progress of the master. Hence when the merge happens, the system considers and indicates it to be a fast-forward merge. It is like fast forwarding to the next state without any hurdle in between

* 1. push a local branch to origin
  2. push a local branch to a remote branch
  3. push a local branch to a remote branch that has conflicts with the local branch
  4. explain the problems that can result from rebasing a shared branch
  5. rebase a branch onto master
  6. rebase a branch and push to a remote branch

1. *Pull requests:*
   1. explain what a pull request (PR) is and how it is used

git pull -u origin <branchname/master(if want to create PR for master>

* 1. open, request a review for, merge, and close a pull request
  2. perform a review on somebody else's pull request
  3. revert a merged pull request