# Git allows groups of people to work on the same documents (often code) at the same time, and without stepping on each other's toes. It's a distributed version control system.

Our terminal prompt below is currently in a directory we decided to name "gitbox". To initialize a Git repository here, type the following command:

**git init**

# Good job! As Git just told us, our "gitbox" directory now has an empty repository in /.git/. The repository is a hidden directory where Git operates, now check status of repository

**git status**

# Create a file called gitcat.txt in the gitbox repository for you and check status of repo

Option - create file in folder itself or use bash command - touch

**git status**

# Adding changes

Good, it looks like our Git repository is working properly. Notice how Git says gitcat.txt is "untracked"? That means Git sees that gitcat.txt is a new file.To tell Git to start tracking changes made to gitcat.txt, we first need to add it to the staging area by using git add.

**git add gitcat.txt**

# Git is now tracking our gitcat.txt file. Let's run git status again to see where we stand:

**git status**

#committing

Notice how Git says changes to be committed? The files listed here are in the Staging Area, and they are not in our repository yet. We could add or remove files from the stage before we store them in the repository.

To store our staged changes we run the commit command with a message describing what we've changed. Let's do that now by typing:

**git commit -m "Add cute gitcat story"**

# Create few file and directory in your local git directory as following

**Create directory - gitfamily**

Create following files in gitfamily directory or folder

gitfamily/ bule\_gitcat.txt

gitfamily/black\_gitcat.txt

Now create few file outside

yellow\_gitcat.txt

organe\_gitcat.txt

# check status of repo -

**git status**

Now you notice that we've added a bunch of .txt files into your directory below.

Luckily, we can add all the new files using a wildcard with git add. Don't forget the quotes!

**git add '\*.txt**'

# Now commit all changes

**git commit -m "Add all the gitcat txt files"**

#History

Check for history of your repository

**git log**

# Remote repositories <Create github user account > github.com / create repository

Create empty github repository

Now add your local repository details to github repository

**git remote add origin “git hub repo url”**

Example - [https://github.com/devopsleap/gitworkshop](https://github.com/devopsleap/gotworkshop)

# Pushing remotely

The push command tells Git where to put our commits when we're ready, and now we're ready. So let's push our local changes to our origin repo (on GitHub).

The name of our remote is origin and the default local branch name is master. The -u tells Git to remember the parameters, so that next time we can simply run git push and Git will know what to do. Go ahead and push it!

**Git pull origin master --allow-unrelated-histories ["Esc" + ":" + "WQ" After ]**

**git push -u origin master**

# Pulling Remotely

Now invite your friend to github repositories and ask them to add some commits

Now you need to pull latest changes

# Differences

**git diff HEAD**

# Now add some files to stage under gitfamily folder

**git add gitdog.txt (create file in local git folder)**

**git status**

# Resitting the stage

So now that gitdog is part of the family, gitcat is all depressed. Since we love gitcat more than gitdog, we'll turn his frown around by removing gitdog.txt.

You can unstage files by using the git reset command. Go ahead and remove

Add file to gitfamily/gitdog.txt.

**git status**

**git reset gitfamily/gitdog.txt**

# Undo everything to gitcat.txt stage

Files can be changed back to how they were at the last commit by using the command: git checkout -- <target>. Go ahead and get rid of all the changes since the last commit for gitcat.txt

**git checkout -- gitcat.txt**

# Branching Out

When developers are working on a feature or bug they'll often create a copy (aka. branch) of their code they can make separate commits to. Then when they're done they can merge this branch back into their main master branch.

We want to remove all these pesky gitcats, so let's create a branch called clean\_up, where we'll do all the work:

**git branch clean\_up**

# Ok, so you're in the clean\_up branch. You can finally remove all those pesky gitcats by using the git rm command which will not only remove the actual files from disk, but will also stage the removal of the files for us.

You're going to want to use a wildcard again to get all the gitcats in one sweep, go ahead and run:

**git rm '\*.txt'**

# Committing Branch Changes

Now that you've removed all the cats you'll need to commit your changes.

Feel free to run git status to check the changes you're about to commit.

**git commit -m "Remove all the cats"**

# Switching back to master

**git checkout master**

# Preparing to Merge

Alrighty, the moment has come when you have to merge your changes from the clean\_up branch into the master branch. Take a deep breath, it's not that scary.

We're already on the master branch, so we just need to tell Git to merge the clean\_up branch into it:

**git merge clean\_up**

# Removing unwanted branch

**git branch -d clean\_up**

# Final Push

Here we are, at the last step. I'm proud that you've made it this far, and it's been great learning Git with you. All that's left for you to do now is to push everything you've been working on to your remote repository, and you're done!

**git push**