

1. `git --version`

2. `git init`

This creates a hidden folder, `.git`, which contains the plumbing needed for Git to work.

3. `git status`

Review the resulting list of files;

4. `git add <file/directory name #1> <file/directory name #2> < ... >`

If all files in the list should be shared with everyone who has access to the repository,

a single command will add everything in your current directory and its subdirectories:

`git add .`

This will "stage" all files to be added to version control, preparing them to be committed in your first commit.

Commit all the files that have been added, along with a commit message:

5. `git commit -m "Initial commit"`

This creates a new commit with the given message. A commit is like a save or snapshot of your entire project.

Adding a remote

To add a new remote, use the `git remote add` command on the terminal, in the directory your repository is stored at.

The `git remote` add command takes two arguments:

1. A remote name, for example, `origin`
2. A remote URL, for example, `https://<your-git-service-address>/user/repo.git`

6. `git remote add origin https://<your-git-service-address>/owner/repository.git`

Clone a repository

`cd <path where you would like the clone to create a directory>`

`git clone https://github.com/username/projectname.git`

Sharing code

`git init --bare /path/to/repo.git`

`git remote add origin ssh://username@server:/path/to/repo.git`

`git push --set-upstream origin master`

Adding `--set-upstream` (or `-u`) created an upstream (tracking) reference which is used by argument-less Git commands, e.g. `git pull`.

Setting your user name and email

`git config --global user.name "Your Name"`

`git config --global user.email mail@example.com`

Remove a global identity

`git config --global --remove-section user.name`

`git config --global --remove-section user.email`

Learning about a command

`git status --help`

`git help status`

`git checkout -h`

Set up SSH for Git

Linux open your Terminal

check to see if you have any existing SSH keys. List the contents of your ~/.ssh directory:

`$ ls -al ~/.ssh`

Lists all the files in your ~/.ssh directory

if you already have a public SSH key. By default the filenames of the public keys are one of the following:

id_dsa.pub
id_ecdsa.pub
id_ed25519.pub
id_rsa.pub

If you see an existing public and private key pair listed that you would like to use on your Bitbucket, GitHub (or similar) account you can **copy the contents of the id_*.pub file.**

create a new public and private key pair with the following command:

`$ ssh-keygen`

Add you SSH key to the ssh-agent. Notice that you'll need te replace id_rsa in the command with the name of your private key file:

```
$ ssh-add ~/.ssh/id_rsa
```

Git Installation

```
$ apt-get install git
```

Git Log

git log

will display all your commits with the author and hash in reverse chronological order – that is, the most recent commits show up first.

Prettier log:

```
git log --decorate --oneline --graph
```

Since it's a pretty big command, you can assign an alias:

```
git config --global alias.lol "log --decorate --oneline --graph"
```

To use the alias version:

history of current branch :

```
git lol
```

combined history of everything in your repo :

```
git lol --all
```

Colorize Logs:

```
git log --graph --pretty=format: ' %C(red)%h%Creset -%C(yellow)%d%Creset %s  
%C(green)(%cr)%C(yellow)<%an>%Creset '
```

The format option allows you to specify your own log output format:

Parameter	Details
%C(color_name)	option colors the output that comes after it
%h or %H	abbreviates commit hash (use %H for complete hash)
%Creset	resets color to default terminal color
%d	ref names
%s	subject [commit message]
%cr	committer date, relative to current date
%an	author name

Online log

```
git log --oneline
```

//will show all of your commits with only the first part of the hash and the commit message.

```
git log -2 --oneline //if you wish to list last 2 commits logs
```

Filter logs:

git log --after '3 days ago'

git log --after 2016-05-01

An alias to **--after** is **--since** .

Flags exist for the converse too: **--before** and **--until** .

You can also filter logs by author . e.g.

git log --author=author

Show the contents of a single commit:

git show 48c83b3

git show 48c83b3690dfc7b0e622fd220f8f37c26a77c934

Working with Remotes

Show information about a Specific Remote:

git remote show origin

Print just the remote's URL:

git remote get-url origin

Set the URL for a Specific Remote:

git remote set-url remote-name url

Get the URL for a Specific Remote

git remote get-url <name>

By default, this will be

git remote get-url origin

Changing a Remote Repository

git remote set-url <remote_name> <remote_repository_url>

Example: git remote set-url heroku https://git.heroku.com/fictional-remote-repository.git

Staging

Staging All Changes to Files:

git add -A

or

git add .

Show Staged Changes:

git diff --cached

Staging A Single File:

git add <filename>

Stage deleted files:

git rm filename

To delete the file from git without removing it from disk, use the --cached flag

git rm --cached filename

Ignoring Files and Folders

in a **.gitignore** file:

Lines starting with `#` are comments.

Ignore files called 'file.ext'

file.ext

Comments can't be on the same line as rules!

The following line ignores files called 'file.ext # not a comment'

file.ext # not a comment

Ignoring files with full path.

This matches files in the root directory and subdirectories too.

i.e. otherfile.ext will be ignored anywhere on the tree.

dir/otherdir/file.ext

otherfile.ext

Ignoring directories

Both the directory itself and its contents will be ignored.

bin/

gen/

Glob pattern can also be used here to ignore paths with certain characters.

For example, the below rule will match both build/ and Build/

[bB]uild/

Without the trailing slash, the rule will match a file and/or

a directory, so the following would ignore both a file named `gen`

and a directory named `gen`, as well as any contents of that directory

bin

gen

Ignoring files by extension

All files with these extensions will be ignored in

this directory and all its sub-directories.

***.apk**

***.class**

It's possible to combine both forms to ignore files with certain
extensions in certain directories. The following rules would be
redundant with generic rules defined above.

java/*.apk
gen/*.class

To ignore files only at the top level directory, but not in its
subdirectories, prefix the rule with a `/`
/*.apk
/*.class

To ignore any directories named DirectoryA
in any depth use ** before DirectoryA
Do not forget the last /,
Otherwise it will ignore all files named DirectoryA, rather than directories
****/DirectoryA/**

This would ignore
DirectoryA/
DirectoryB/DirectoryA/
DirectoryC/DirectoryB/DirectoryA/
It would not ignore a file named DirectoryA, at any level
To ignore any directory named DirectoryB within a
directory named DirectoryA with any number of
directories in between, use ** between the directories
DirectoryA//DirectoryB/**
This would ignore
DirectoryA/DirectoryB/
DirectoryA/DirectoryQ/DirectoryB/
DirectoryA/DirectoryQ/DirectoryW/DirectoryB/

To ignore a set of files, wildcards can be used, as can be seen above.
A sole '*' will ignore everything in your folder, including your .gitignore file.
To exclude specific files when using wildcards, negate them.
So they are excluded from the ignore list:
!.gitignore

Use the backslash as escape character to ignore files with a hash (#)
\##

Exceptions in a .gitignore file:

***.txt**
!important.txt

The above example instructs Git to ignore all files with the .txt extension except for files named important.txt .

Ignore files that have already been committed to a Git repository:

git rm --cached <file>

This will remove the file from the repository and prevent further changes from being tracked by Git. The --cached option will make sure that the file is not physically deleted.

Clear already committed files, but included in .gitignore:

Remove everything from the index (the files will stay in the file system)

\$ git rm -r --cached .

Re-add everything (they'll be added in the current state, changes included)

\$ git add .

Commit, if anything changed. You should see only deletions

\$ git commit -m 'Remove all files that are in the .gitignore'

Update the remote

\$ git push origin master

Git Diff

Show differences in working branch:

git diff

Show changes between two commits:

git diff 1234abc..6789def # old new

Show the changes made in the last 3 commits:

git diff @~3..@ # HEAD -3 HEAD

Note: the two dots (..) is optional, but adds clarity.

Show differences for staged files:

git diff --staged

git diff --cached

git status -v

Show differences for a specific file or directory

git diff myfile.txt

Show differences between current version and last version

git diff HEAD^ HEAD

Undoing

Return to a previous commit:

git checkout 789abcd

//To temporarily jump back to that commit, detach your head

To roll back to a previous commit while keeping the changes:

git reset --soft 789abcd

To roll back the last commit:

git reset --soft HEAD~

To permanently discard any changes made after a specific commit, use:

git reset --hard 789abcd

To permanently discard any changes made after the last commit:

git reset --hard HEAD~

Undo changes to a file or directory in the working copy.

git checkout -- file.txt

Used over all file paths, recursively from the current directory, it will undo all changes in the working copy.

git checkout -- .

To only undo parts of the changes use --patch . You will be asked, for each change, if it should be undone or not.

git checkout --patch -- dir

To undo changes added to the index.

git reset --hard

Without the --hard flag this will do a soft reset.

Merging

git merge incomingBranch

git merge --abort

Committing

git commit -m "Commit message here"

git commit -am "Commit message here"

Note that this will stage all modified files in the same way as git add --all .

If your latest commit is not published yet (not pushed to an upstream repository) then you can amend your commit.

git commit --amend -m "New commit message"

Aliases

```
git config --global alias.ci "commit"  
git ci -m "Commit message..."
```

Configuration

Parameter	Details
--system	Edits the system-wide configuration file, which is used for every user (on Linux, this file is located at \$(prefix)/etc/gitconfig)
--global	Edits the global configuration file, which is used for every repository you work on (on Linux, this file is located at ~/.gitconfig
--local	Edits the repository-specific configuration file, which is located at .git/config in your repository; this is the default setting

Change the core.editor configuration setting.

```
$ git config --global core.editor nano
```

Auto correct typos

```
git config --global help.autocorrect 17
```

To see the current configuration.

```
$ git config --list
```

To edit the config:

```
$ git config <key> <value>
```

```
$ git config core.ignorecase true
```

If you intend the change to be true for all your repositories, use **--global**

```
$ git config --global user.name "Your Name"
```

```
$ git config --global user.email "Your Email"
```

```
$ git config --global core.editor vi
```

Branching

Goal	Command
List local branches	git branch
List local branches verbose	git branch -v
List remote and local branches	git branch -a OR git branch --all
List remote and local branches (verbose)	git branch -av
List remote branches	git branch -r
List remote branches with latest commit	git branch -rv
List merged branches	git branch --merged
List unmerged branches	git branch --no-merged
List branches containing commit	git branch --contains [<commit>]

To create a new branch, while staying on the current branch, use:

git branch <name>

The branch name must not contain spaces and is subject to other specifications listed here. To switch to an existing branch :

git checkout <name>

To create a new branch and switch to it:

git checkout -b <name>

Delete a remote branch

git push origin --delete <branchName>

Delete a branch locally

\$ git branch -d dev

\$ git branch -D dev

Rename the branch you have checked out:

git branch -m new_branch_name

Rename another branch:

git branch -m branch_you_want_to_rename new_branch_name

Pulling

When you are working on a remote repository (say, GitHub) with someone else, you will at some point want to share your changes with them. Once they have pushed their changes to a remote repository, you can retrieve those changes by pulling from this repository.

git pull

You can pull changes from a different remote or branch by specifying their names

git pull origin feature-A

Cloning Repositories

Shallow Clone

Cloning a huge repository (like a project with multiple years of history) might take a long time, or fail because of the amount of data to be transferred. In cases where you don't need to have the full history available, you can do a shallow clone:

git clone [repo_url] --depth 1

to instead get the last 50 commits:

git clone [repo_url] --depth 50

Regular Clone

git clone <url>

git clone <url> [directory]

Clone a specific branch

git clone --branch <branch name> <url> [directory]

Renaming

Rename Folders

```
git mv directoryToFolder/oldName directoryToFolder/newName
```

Rename a local and the remote branch

```
git checkout old_branch
git branch -m new_branch
git push origin :old_branch
git push --set-upstream origin new_branch
```

Renaming a local branch

```
git branch -m old_name new_name
```

Pushing

General syntax

```
git push <remotename> <object>:<remotebranchname>
```

Example

```
git push origin master:wip-yourname
```

Will push your master branch to the wip-yourname branch of origin (most of the time, the repository you cloned from).

Delete remote branch

Deleting the remote branch is the equivalent of pushing an empty object to it.

```
git push <remotename> :<remotebranchname>
```

Example

```
git push origin :wip-yourname
```

Will delete the remote branch wip-yourname

Instead of using the colon, you can also use the --delete flag, which is better readable in some cases.

Example

```
git push origin --delete wip-yourname
```

```
git push --set-upstream origin master
```

Show

For commits:

Shows the commit message and a diff of the changes introduced.

Command	Description
git show	shows the previous commit
git show @~3	shows the 3rd-from-last commit

Git Remote

Display Remote Repositories:

```
$ git remote
```

```
$ git remote -v
```

Change remote url of your Git repository:

```
git remote set-url origin https://localhost/develop/myrepo.git
```

Remove a Remote Repository:

```
git remote rm dev
```

Add a Remote Repository:

```
git remote add <name> <url>
```

Rename a Remote Repository:

```
git remote rename dev dev1
```

Code with Harry Notes:

- **git status**
- **git init**
- **git add --a || git add .** (working dir ==> staging area)
- **git add file1.txt file2.txt**
- **git commit -m "Initial commit"** (staging area ==> repository)
- **git log** (get all commit logs)
- **git clone <url of repo> foldername**
- **rm -rf .git** (remove .git init file)
- **touch .gitignore**
- **view .gitignore** (this file contains list of file or directories to be ignored)(blank folder is by default ignored)
- **git diff** (it will be empty if no difference)
- **git diff --staged** (to compare current staged file with previous commit)
- **git rm file.txt** (remove from staged)
- **git mv file.txt dir/dir2/** (move + staged)
- **git mv file.txt newName.txt** (rename +staged)
- **git rm --cached file.txt** (then add that file in .gitignore) (Untrack a file)
- **git log** (commit hash + Author + Date + Message)
- **git log -p** (log + changes made in each commit) (log +diff)
- **git log -p -3**
- **git log --stat** (log + insert/delete info)
- **git log --pretty=oneline**
- **git log --pretty=short**
- **git log --pretty=full**
- **git log --since=2.days**
- **git log --since=2.months**
- **git log --since=2.years**
- **git log --pretty=format:"%h-- %an"**
- **git commit --amend** (to amend current stage to previous commit)
- **git restore --staged file.txt** (to unstage file.txt)
- **git checkout -- file.txt** (to restore to previous commit) (to unmodify the changes)(undo)
- **git checkout -f** (to loose all new changes)

- **git config --global alias.last 'log -p -1'** (git last ==== git log -p -1)
- **git remote** (it store url)(to check stored url)
- **git remote add origin <url>**
- **git remote -v** (to get stored url and url name)
- **git remote set-url origin <new-url>**
- **git push -u origin master**

Connect PC with github (SSH)

1. **ssh-keygen -t rsa -b 4096 -c "neeraj.kumar@berylsystems.com"** (let passphrase empty)
2. **eval \$(ssh-agent -s)**
3. **ssh-add ~/.ssh/id_rsa**
4. **tail ~/.ssh/id_rsa.pub** (copy this token)

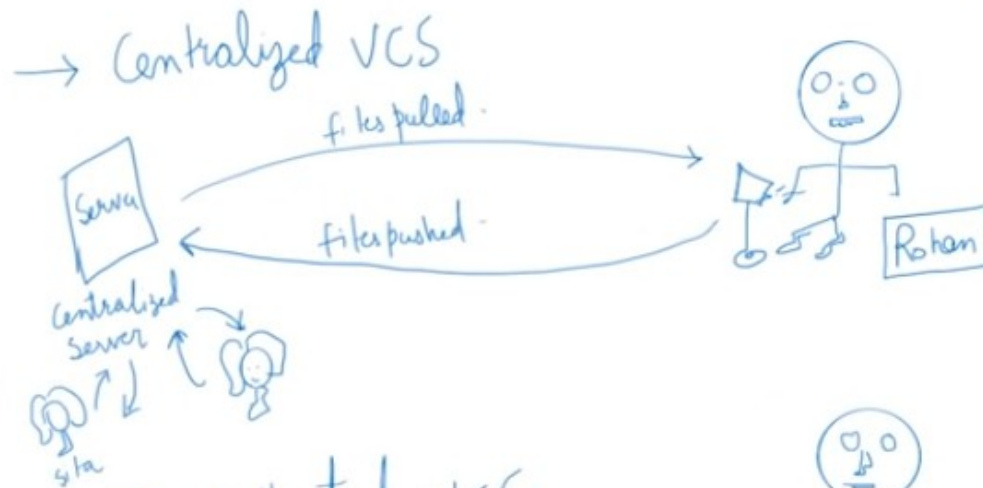
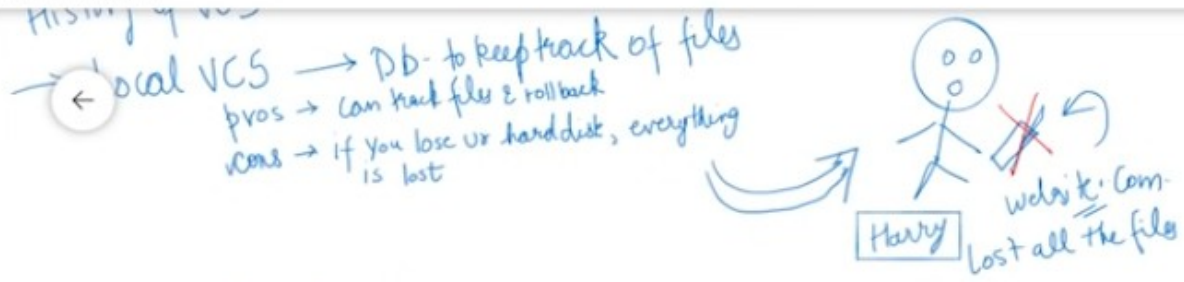
In Github:

5. Main Security ==> SSH and GPG Key ==>
 Title: Neeraj PC BS.HP_Probook
 Key: <paste the token copied>
press Add
6. **git push.....**

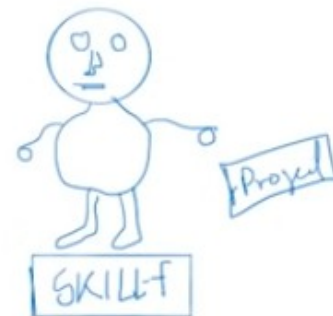
-
- **git checkout -b branch_name** (it will create new branch & switch to it)
 - **git checkout master** (to switch to master branch from current branch)
 - **git branch** (to check all available branches)
 - **git branch -v** (branch name + last commit hash code + message)
 - **git branch --merged** (list of merged branches)
 - **git branch --no-merged** (list of unmerged branches)
 - **git branch -d branch_name** (to delete a merged branch)
 - **git branch -D branch_name** (to delete a unmerged branch)
 - **git merge branch_name** (it will merge the given branch to master)
 - **Conflicts to be handled Manually ==> then git add . And commit it**
 - **git push -u origin branch_name** (to push the branch on github)
 - **git push origin branch_name: newBranch** (it will create a new branch newBranch & start tracking from branch_name ON GITHUB)
 - **git push -d origin branch_name** (to delete a branch on github)
-

PIC Notes

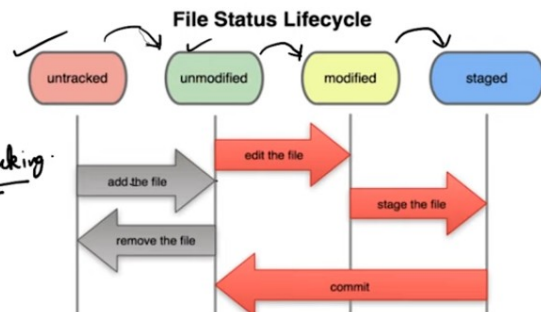
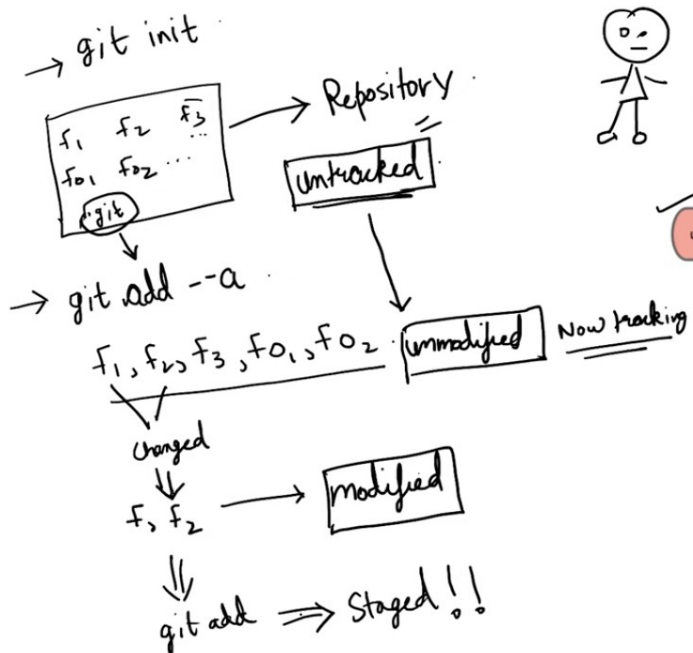
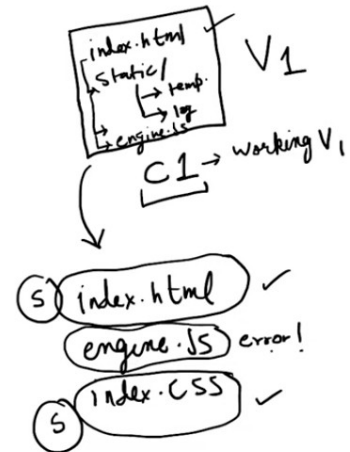
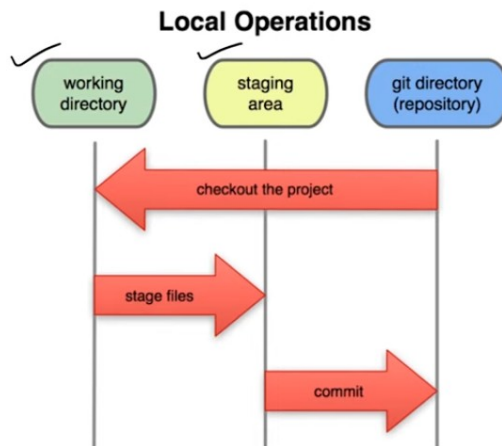
Microsoft Whiteboard



→ Distributed VCS



Git – Three stage architecture



Configuration

System

/etc/gitconfig

Program Files\Git\etc\gitconfig

User

~/.gitconfig

\$HOME\.gitconfig

Project

my_project/.git/config

Configuration

System

git config --system

User

git config --global

Project

git config


```

kevin$ git config --global core.editor "atom --wait"
kevin$ git config --global color.ui true
kevin$ cat .gitconfig
[user]
    name = Kevin Skoglund
    email = someone@nowhere.com
[core]
    editor = atom --wait
[color]
    ui = true
kevin$ _

```

Commit Message Best Practices

A short single-line summary (less than 50 characters)

Optionally followed by a blank line and a more complete description

Keep each line to less than 72 characters

Write commit messages in present tense, not past tense

“Fix for a bug” or “Fixes a bug,” not “fixed a bug”

```

Author: Kevin Skoglund <someone@nowhere.com>
Date: Tue Apr 9 10:51:31 2019 -0400

Initial commit
kevin$ git log --since=2019-01-01
commit 33abc0bee9a90d151e1858a1454a1368933f4c46 (HEAD -> master)
Author: Kevin Skoglund <someone@nowhere.com>
Date: Tue Apr 9 10:51:31 2019 -0400

Initial commit
kevin$ git log --since=2020-01-01
kevin$ git log --until=2020-01-01
commit 33abc0bee9a90d151e1858a1454a1368933f4c46 (HEAD -> master)
Author: Kevin Skoglund <someone@nowhere.com>
Date: Tue Apr 9 10:51:31 2019 -0400

Initial commit
kevin$ git log --author="Kevin"
commit 33abc0bee9a90d151e1858a1454a1368933f4c46 (HEAD -> master)
Author: Kevin Skoglund <someone@nowhere.com>
Date: Tue Apr 9 10:51:31 2019 -0400

Initial commit
kevin$ _

```

```
first_git_project — -bash — 80x24
kevin$ git log --grep="Init"
commit 33abc0bee9a90d151e1858a1454a1368933f4c46 (HEAD -> master)
Author: Kevin Skoglund <someone@nowhere.com>
Date: Tue Apr 9 10:51:31 2019 -0400

    Initial commit
kevin$ git log --grep="Bugfix"
kevin$ git log --grep="Bug_"
```

Revert a commit:

`git revert <Commit SHA Code>`

Retrieve old version

`git checkout <Commit SHA Code> -- file.txt` (it will retrieve the file.txt from old commits)

To delete untracked file in work directory

`git clean [options]`

`-i` == informative

`-n` == it will not delete but it will list files which can be deleted

`-f` == force it will remove that files

List of last committed files and directories

`git ls-tree HEAD`

To add empty directory to github

create a new file with name .gitkeep in the empty directory

`which git` // path of git will be shown ==> /usr/bin/git

NOTE: We cannot push file of size more than 100 mb on github using git
if we require to upload a file of size greater than 100mb we have to install
git lfs install

Remove a Remote:

`git remote remove remote_name`

Rename a Remote

```
git remote rename current_name new_name
```

Disable Auto correct in Config

```
git config --global help.autocorrect 0
```

