# Git Hub

## Getting Started

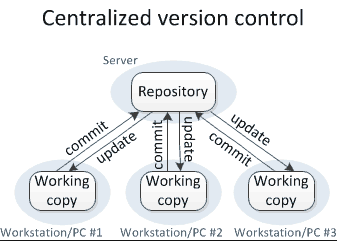
**Version Control:** It is a system that records changes to a file or set of files over time so that we can recall specific versions later.

**Local Version Control Systems:** It is a simple database which keeps all the changes of the files in the disk in a special format.

(RCS ∷ Revision Control System)

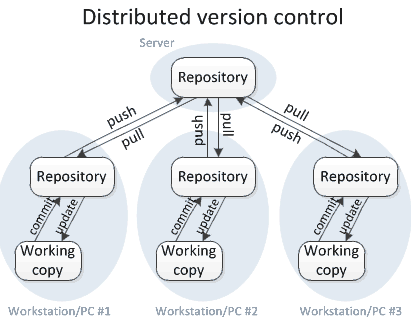
**Centralized Version Control Systems:** It’s downside is when the server goes down, no one can collaborate and it becomes devastating when the hard disk corrupts.

(SVN ∷ Apache Subversion)



**Distributed Version Control Systems:** It saves the snapshot of the file as well as the mirror repository (Versions Database)

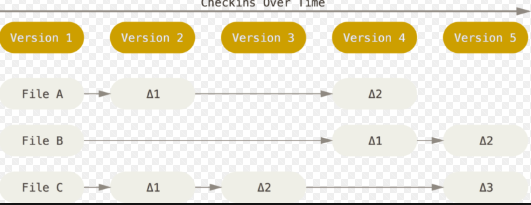
(Github)



## Git Basics

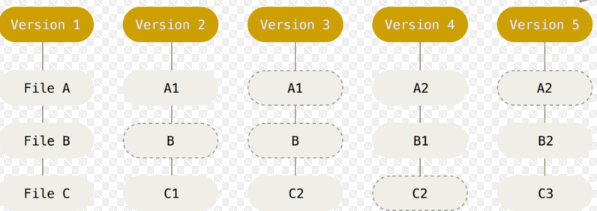
### How Subversion works

It stores changes



### How GitHub works

It stores data as snapshots of the project over time. It stores links to previous files



### Git Has Integrity

It uses hash to address the database. It uses SHA-1 which is 40 characters long

### The Three States

**Committed** ∷ It safely stored the data in local database

**Modified** ∷ The file has been changed but not saved in local database

**Staging** :: It is marked as modified file in its current version which will go to next snapshot

### Understanding Directories

**Git Directory ::** It stores the metadata and object based database for our project

**Working Directory** ∷ It uses a single check out from Git directory to our disk

**Staging Area ::** It is a simple file, generally contained in our Git directory, that stores information about what will go into our next commit.

## Setting Up GitHub

**Global Username**

git config –global user.name “Photon Khan”

**Global Email**

git config –global user.email [khan.photon@gmail.com](mailto:khan.photon@gmail.com)

**Global Editor**

git config –global core.editor code –wait

**Global Configuration List**

git config –list

**GitHub Help**

git help, git help [config], git [config] help

**Exit Github**

Press q

## Getting a Git Repository

### Initializing a Repository in an Existing Directory (Not in GitHub yet)

**Initializing Git Directory**

git init

**Create a Commands Text File**

dir > commands.txt

**Create a Readme File**

dir > README

**Creates a Folder**

mkdir > hello

### Cloning an Existing Repository (Exists in GitHub)

**Cloning a github file**

git clone git∷//github.com/mp5maker/notes

### Recording changes to the repository

**Checking the status of our files**

**Untracked File:** It means the that Git sees a file we didn’t have in the previous snapshot.

**Tracking New File**

git add README

git add commands.txt

Now they are staging modified files. After that if we modify the files it won’t be in the staging section. Then we need to again mention git add

### Ignoring Files

**Create a file named .gitignore**

> dir .gitignore

**Regex**

**\*.[oa] 🡪** Ignore any files ending in .o or .a

**\*~ 🡪** Ignore all the files that ends with ~ ‘tilde’

**!lib.a 🡪** But do not ignore lib.a file

**/TODO 🡪** Ignore the root todo files

**build/** 🡪 Ignore all the files in the build directory

### Viewing our staged and unstaged changes

**Differences between the staged and the non-staged file**

git diff

**Differences between the staged files versus the previous commit**

git diff –cached

**Git Commit**

git commit -m “First Commit”

**Git Commit without staging**

git commit -a -m ‘Added new benchmark”

### Removing from the commit

**Git Remove which has already been staged**

git rm --cached extra.txt

### Moving Files (Renaming the file or location)

**Moving Files**

git add extra.txt

git mv extra.txt rmextra.txt

### Logging the commits

**Git Log**

git log

**Git log differences(-p) in each commit by limit by 2(-2)**

git log -p -2

**Abbreviated stats of each commit**

git log --stat

**Git log using pretty**

git log –pretty=oneline

**Git log using pretty with formatting**

git log –pretty=format: “%H %an %ar %s”

%H 🡪 Commit Hash

%ar 🡪 Author Relative Data

%ad 🡪 Author Date

%an 🡪 Author Name

%s 🡪 Subject

**Git log Graph**

git log –pretty=format:“%h %s” –graph

**Limit Log Output**

git log –since=2.weeks

**Graphical User Interface**

gitk

### Changing the Commit

**Changing the last commit**

git commit --amend

**Unmodifying a modified file**

git checkout –filename ∷ Discard changes in the working directory

git reset HEAD filename ∷ Discard to unstage

### Working with remotes

**Showing our remotes**

git remote -v ∷ If we have cloned a repository, then it shows the origin

**Removing the remotes**

git remote rm origin

**Adding remotes**

git remote add [shortname] [url]

git remote add gt http∷github.com/mp5maker/github

**Fetching the remote**

git fetch [remote-name]

### Git Branches

**Check all the branches**

git branch

**Remove the branches**

git branch -d [branch-name]

### Pushing to the remote

#### Set up the git SSH keys with github

Profile Picture 🡪 Settings 🡪 SSH & GPG Keys 🡪 Follow the instructions to generate SSH

**Pushing to Remote**

git push [remote-name] [branch-name]

git push origin master

This only works if we cloned from a server to which we have write access and if nobody has pushed in the meantime. If I and someone else clone at the same time and they push upstream and then we push upstream, my push will rightly be rejected. We need to pull their work first then we can push

### Inspecting a remote

**Show Remote Name**

git remote show [remote-name]

**Renaming the remote**

git remote rename test photon

### Tagging

**List our tags**

git tag

This is used to mark release points (v1.0, v1.1)

**Creating tags**

git tag -a v1.0 -m ‘my version 1.0’

**Signed tags**

git tag -s v1.5 -m ‘my signed version 1.5’

We can also sign our tags with GPG, we need to have a private key for that.

**Lightweight tags**

git tag v1.4-lw

**Verifying tags**

git tag -v v1.0

**Pushing the tag**

git push origin v1.0

**Git Aliases**

git config –global alias.co checkout

git config –global alias.br branch

git config --global alias.ci commit

git config --global alias.st status

git config --global.alias.visual “!gitk”

### Basic Branching and Merging

#### **Story**

* Work on a website
* Create a branch for a new story we are working on
* Do some work in that branch

New Issue Arrives

* Revert back to our production branch
* Create a branch to add the hotfix
* After it’s tested, merge the hotfix branch, and push to production
* Switch back to our original story, and continue working

### Workflow

First Name it issue 53

* git checkout -b iss53
* git add .
* git commit -m “Added Footer”

New Issue Arrives

* git checkout master
* git checkout -b “hotfix”
* git add .
* git commit -m “Broken Email Address”
* git checkout master
* git merge hotfix
* git branch -d hotfix
* git branch master
* git merge iss53

Conflicted Files (Git didn’t resolved the problem, it just paused)

* git status
* git mergetool

### Show the Branch Issues

**See the last commit of each branch**

git branch -v

**To see the branches that are already committed into the branch we are on**

git branch --merged 🡪 Opposite 🡪 git branch --no-merged

**Tracking branches**

git checkout –track origin/master

**Give me the list of all the remote branches**

git branch -r

**Creates a new branch with the master**

git branch [remote-name] [branch-name]

### Tracking Branches

It means the local branches that have direct relationship with remote branches

Generally, the master branch is the tracking branch.

**Delete Remote Branches**

git push [remote-name] :[branch-name]

**Rebase/Shifting branches**

git rebase master

**Topic branch off another topic branch**

git rebase –onto master server client

### Git Server

There are lots of protocol: Local Protocol, SSH Protocol, Git Protocol, HTTPS Protocol

**Local Protocol**

git clone /opt/git /project.git

git clone <file:///opt/git/project.git>

git remote add local\_proj /opt/git/project.git

**SSH Protocol**

git clone ssh://user@server.project.git

git clone [user@server.project.git](mailto:user@server.project.git)

Encrypted Protocol

### Track Branches Visual

**Install GitWebUI**

Linux

wget -O - https://raw.githubusercontent.com/alberthier/git-webui/master/install/installer.sh | bash

Windows/Mac

curl https://raw.githubusercontent.com/alberthier/git-webui/master/install/installer.sh | bash

**Run**

git webui --port[port-number]

### Forking Projects

If we want to contribute to an existing project to which we don not have push access. Github encourages forking the project. This way we get a writable copy of any repository by clicking the “fork” button.

### Upstream

We must configure a remote that points to the upstream repository in Git to sync changes we make in a fork with the original repository. This allows us to sync changes made in the original repository with the fork.

**Check my remotes**

git remote -v

origin https://github.com/YOUR\_USERNAME/YOUR\_FORK.git (fetch)

origin https://github.com/YOUR\_USERNAME/YOUR\_FORK.git (push)

**Example**

origin git@github.com:mp5maker/github.git (fetch)

origin git@github.com:mp5maker/github.git (push)

**Adding Upstream (After the fork)**

git remote add upstream <https://github.com/ORIGINAL_OWNER/ORIGINAL_REPOSITORY.git>

**Check my remotes again**

git remote -v

origin https://github.com/YOUR\_USERNAME/YOUR\_FORK.git (fetch)

origin https://github.com/YOUR\_USERNAME/YOUR\_FORK.git (push)

upstream https://github.com/ORIGINAL\_OWNER/ORIGINAL\_REPOSITORY.git (fetch)

upstream https://github.com/ORIGINAL\_OWNER/ORIGINAL\_REPOSITORY.git (push)

### Commit Guidelines

git diff –check

git describe master