# **GIT – Version Control System**

## Registering on GITHUB site

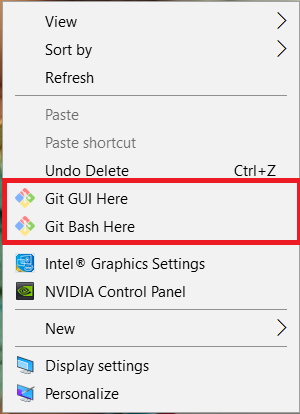
* Open git hub link: <https://github.com>
* Signup by entering your details
* Once Signup is done then you will get a mail from GitHub. Verify your email by clicking on “verify your mail id” link
* Now you can see the  button, click on **New repository** and create your Repository.

## Installation of Git

* Download the Git client from <https://git-scm.com/> and click on downloaded file.
* Click next on all the screens. no need to enter any details.

Verification:

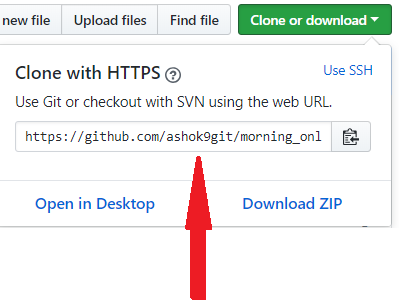
* Right on Desktop, Git Bash and Git GUI should be visible like below.



* Once you are done with Git installation then you can clone your code from github.
* Checkout command:

git clone <repositoryURL>

Note: get the repository URL from GITHUB by copying .. see below pic



* Now you can commit your changes to GitHub repository.

What is GIT?

Git is a **source code management system** and it is **Distributed/Decentralized version control system**.

Developed by **Linus Torvalds** for Linux kernel development.

We have two types of version control systems:

1) Centralized version control system. (CVCS)

2) Distributed/Decentralized version control system. (DVCS)

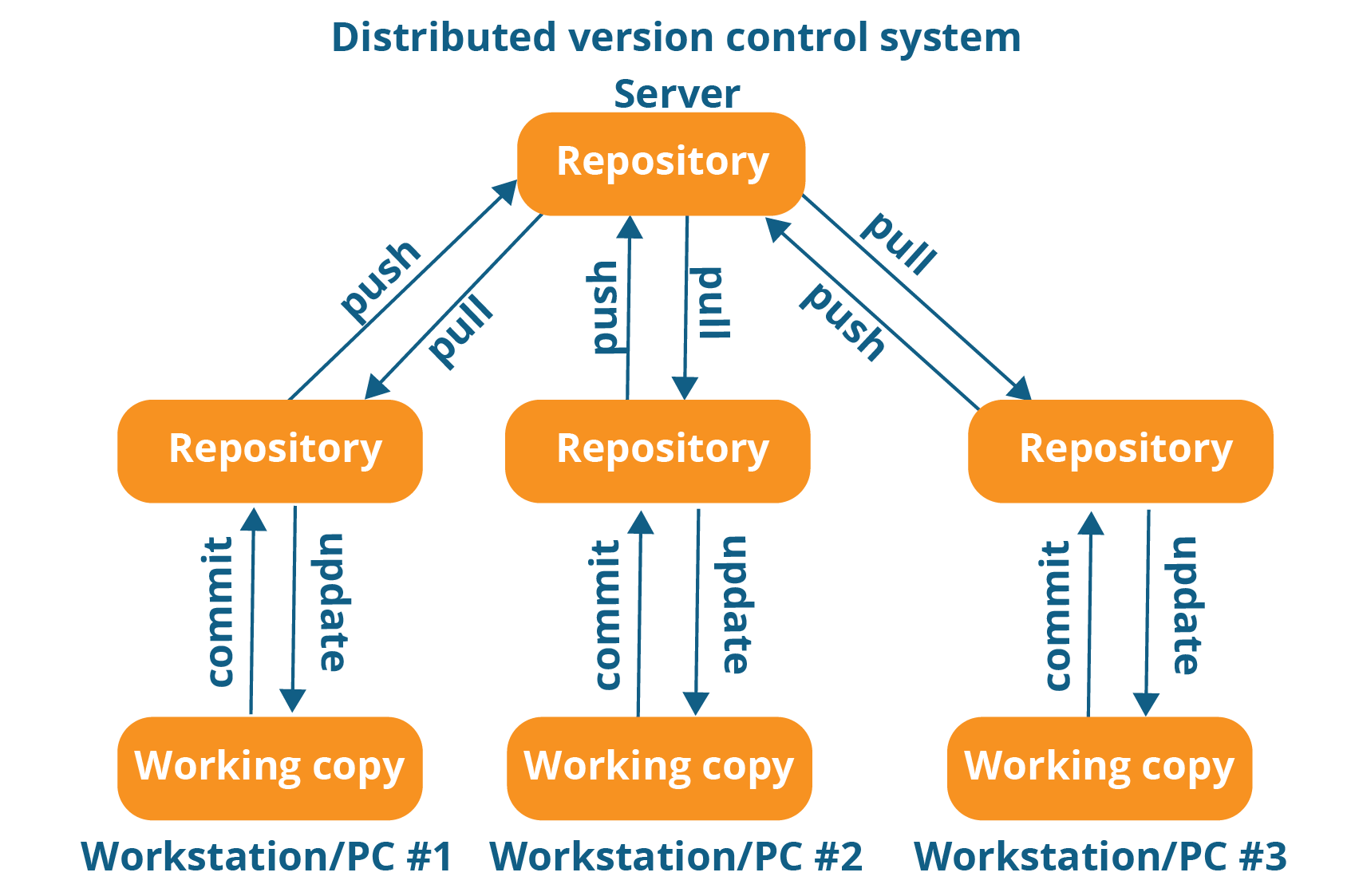
**Centralized version controlling system** holds all the information in the centralized server, so if you server is destroyed/crashed your data related to the repository will be lost. Which states that developers can’t collaborate between them for the code development.

Ex: SVN



**Distributed/ Decentralized version controlling system** as the capability to store the repository in the central server and provides the clients to make a master copy of the repository, so which helps to restore the repository in case of centralized server failure.

Ex: git



**Advantages of using git.**

1) Free and open source

2) Fast and small

3) Implicit backup

4) Security and No need of powerful hardware.

**Commits**

Commit holds the current state of the repository A commit is also named by SHA1 hash. You can consider a commit object as a node of the linked list. Every commit object has a pointer to the parent commit object. From a given commit, you can traverse back by looking at the parent pointer to view the history of the commit. If a commit has multiple parent commits, then that particular commit has been created by merging two branches.

+++

git commit -m "name this commit appropriately to identify the commit with a comment and that you need to show in the repository"

+++

**Branches**

Branches are used to create another line of development. By default, Git has a master branch, which is same as trunk in Subversion. Usually, a branch is created to work on a new feature. Once the feature is completed, it is merged back with the master branch and we delete the branch. Every branch is referenced by HEAD, which points to the latest commit in the branch. Whenever you make a commit, HEAD is updated with the latest commit.

**Tags**  
Tag assigns a meaningful name with a specific version in the repository. Tags are very similar to branches, but the difference is that tags are immutable. It means, tag is a branch, which nobody intends to modify. Once a tag is created for a particular commit, even if you create a new commit, it will not be updated. Usually, developers create tags for product releases.

**HEAD**

HEAD is a pointer, which always points to the latest commit in the branch. Whenever you make a commit, HEAD is updated with the latest commit. The heads of the branches are stored in .git/refs/heads/ directory.

**Revision**

Revision represents the version of the source code. Revisions in Git are represented by commits. These commits are identified by SHA1 secure hashes.

**URL**

URL represents the location of the Git repository. Git URL is stored in config file.

**Basic workflow of git to upload the code to the remote repository:**

----------------------------------------------------------------------------------------------

|  |
| --- |
| **Working Directory** |
| **Staging area/Index** |
| **Local git repository** |
| **Remote repository** |

1) You will modify/edit the files that are created/cloned in the directory.

--> git status (to check the modified files)

2) You add these files to the staging area.

--> git add . or git add \* or git add <filename/directory>

3) You perform the commit to move the files from staging area to the local git repository.

--> git commit -m "comment that you want to add"

4) Atlast you will permanently store the changes to the remote repository.

--> git push -u origin <branch\_name> (ex: git push -u origin master)

→ git push origin <branch\_name>

**Example scenarios:**

1) How to create a new repository on the command line

DevOps@DevOps-PC MINGW64 /e/DevOps\_class/practice\_git

$ mkdir objects

DevOps@DevOps-PC MINGW64 /e/DevOps\_class/practice\_git

$ cd objects

DevOps@DevOps-PC MINGW64 /e/DevOps\_class/practice\_git/objects

$ git init

Initialized empty Git repository in E:/DevOps\_class/practice\_git/objects/.git/

DevOps@DevOps-PC MINGW64 /e/DevOps\_class/practice\_git/objects (master)

2) How to add files/code from working directory to the staging area.

DevOps@DevOps-PC MINGW64 /e/DevOps\_class/practice\_git/objects (master)

$ touch file1 file2

DevOps@DevOps-PC MINGW64 /e/DevOps\_class/practice\_git/objects (master)

$ git status

On branch master

No commits yet

Untracked files:

(use "git add <file>..." to include in what will be committed)

file1

file2

nothing added to commit but untracked files present (use "git add" to track)

DevOps@DevOps-PC MINGW64 /e/DevOps\_class/practice\_git/objects (master)

$ git add .

DevOps@DevOps-PC MINGW64 /e/DevOps\_class/practice\_git/objects (master)

$ git status

On branch master

No commits yet

Changes to be committed:

(use "git rm --cached <file>..." to unstage)

new file: file1

new file: file2

3) How to move the files/code from staging area to the local .git repository.

DevOps@DevOps-PC MINGW64 /e/DevOps\_class/practice\_git/objects (master)

$ git commit -m "new files"

[master (root-commit) 854e549] new files

2 files changed, 0 insertions(+), 0 deletions(-)

create mode 100644 file1

create mode 100644 file2

**git commit -a** --> automatically notice any modified (but not new) files, add them to the index, and commit, all in one step

4) How to push the updated to remote repository:

**git push -u origin master**

5) How to push an existing code into git repository.

**git remote add origin https://github.com/ashok9git/objects\_repo.git**

**git push -u origin master**

6) How to update the global values like email, name to the local git.

**git config --global user.email "you@example.com"**

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

7) How to setup the origin to you local repository and update the code to remote repo:

# **https://github.com/ashok9git/objects\_repo.git** --> its the URL in which you want to push the code modified.

DevOps@DevOps-PC MINGW64 /e/DevOps\_class/practice\_git/objects\_repo (master)

$ git push origin master

Username for 'https://github.com': ashok9git

Counting objects: 12, done.

Delta compression using up to 4 threads.

Compressing objects: 100% (10/10), done.

Writing objects: 100% (12/12), 1.28 KiB | 656.00 KiB/s, done.

Total 12 (delta 5), reused 0 (delta 0)

remote: Resolving deltas: 100% (5/5), completed with 2 local objects.

To https://github.com/ashok9git/objects\_repo.git

f4f193e..563b930 master -> master

8) How to make a copy of the remote repository to the local workstation

#**git clone https://github.com/ashok9git/objects\_repo.git**

9) How clone a particular branch from remote repository

#**git clone --branch <branch\_name> https://github.com/ashok9git/objects\_repo.git**

10) To view the current origin to which it is pointing to

#**git remote -v** --> List all currently configured remote repositories

11) How to view the total number of commits in the repository

#**git log**

12) How to view the metadata information of the commit.

#**git show <commit\_id>**

**BRANCH Scenarios:**

-----------------------------

how to create a branch

# **git branch <name>**

List branches of the remote repository

#**git branch -r**

how to check the total branches created

# **git branch**

how to move to the branch from master to the newly created branch

# **git checkout <branch\_name>**

We can create a branch and switch to that branch in single command.

# **git checkout -b <branch\_name>**

how to delete a branch in the local git repository.

# **git branch -D <branch\_name>**

Make sure that you are not in that particular branch, if yes, move to master branch and delete it.

Note: this will not delete the branch in the remote repository

how to delete a branch in remote repository (git server)

# **git push origin --delete <branch\_name>**

how to rename the branch

# **git branch -m <old\_name> <new\_name>**

how to push all created branches to the remote repository

#**git push --all origin**

how to merge a branch to the master

# **git merge <branch\_name>** --> before doing this you need to be in the master branch

**Tags:**

-------

Tag assigns a meaningful name with a specific version in the repository. Tags are very similar to branches, but the difference is that tags are immutable. It means, tag is a branch, which nobody intends to modify. Once a tag is created for a particular commit, even if you create a new commit, it will not be updated. Usually, developers create tags for product releases.

how to create a tag

DevOps@DevOps-PC MINGW64 /e/DevOps\_class/practice\_git/evening\_batch/online\_evening (master)

$ git tag <tagname> <commitid>

e.g:

DevOps@DevOps-PC MINGW64 /e/DevOps\_class/practice\_git/evening\_batch/online\_evening (master)

$ git tag app\_idea1\_release\_1.9 ccbf3bc1bac8bd709cafded0c112f87847ef7641

how to push the tags to the remote repository

#**git push origin tag v2.0**

#**git push --tags origin**

#**git push origin --tags**

how to check the commit history.

DevOps@DevOps-PC MINGW64 /e/DevOps\_class/practice\_git/evening\_batch/online\_evening (master)

$ git log --pretty=oneline

c73db736116c3e7554e58b54a986e5889eed1583 (HEAD -> master, origin/master, origin/HEAD) asdfas

7a4e6a7b3025b0babe69d172d13a009bf16357ff asdfas

ebec4a4faf34c77b42792479b9b92a0e761ac975 Create newfile

90324d82d447e05a7fa96523d784d469203b6bfa Update ashok.ppt

7965220afa073743243036a48c0416f537e057a6 Prathap clone

5eba1a05b01346a14b9005c6492a94858aed4555 asdf

0f94156d2fa9fe057e3354a835e8e7620b91283f (tag: release\_app\_1.7) fetching

fe9a74c59f6bc91c383b33e8f371274836af9a10 from myrepo

9b5873801d089a36169eb0e9f895f857dbdccab5 practice files

e9042a3bd36d58bae9336d0532991621b6cec1f1 Revertng "idea1"

da29052141a9fa0bce725bda382d54c2d94492a0 Revertng "conflict"

95f41c883c4d3e85ee13b3305aad04009775e656 conflict

ccbf3bc1bac8bd709cafded0c112f87847ef7641 (tag: app\_idea1\_release\_1.9) idea1

0d8cf2c911c578655ed451cd8f1670f029de5b1e intial stash

ed11cd6a350ab6ec8e354137dd16bca6ef489874 sample file

7c82c74e20a886e567fb0cb8c9e6985d0a3a18a9 reseting

09dee051666fd34b094dc52f26adefa40cc77b7b commiting file for git reset

a63ef2eb0b87e13e18dc02b5df1b5c3028d2a80b git ignore file

0ee3857fd12101913017b32013cbd6b18a2a674b batch file

DevOps@DevOps-PC MINGW64 /e/DevOps\_class/practice\_git/evening\_batch/online\_evening (master)

$ git log --oneline

c73db73 (HEAD -> master, origin/master, origin/HEAD) asdfas

7a4e6a7 asdfas

ebec4a4 Create newfile

90324d8 Update ashok.ppt

7965220 Prathap clone

5eba1a0 asdf

0f94156 (tag: release\_app\_1.7) fetching

fe9a74c from myrepo

9b58738 practice files

e9042a3 Revertng "idea1"

da29052 Revertng "conflict"

95f41c8 conflict

ccbf3bc (tag: app\_idea1\_release\_1.9) idea1

0d8cf2c intial stash

ed11cd6 sample file

7c82c74 reseting

09dee05 commiting file for git reset

a63ef2e git ignore file

0ee3857 batch file

how to view the tags which have been created

# **git tag -list or git tag –l or git tag**

how to delete a tag in local git repository

# **git tag -d <tag\_name>**

how to delete the tags in the remote repository

# **git push origin :<tag\_name>**

how to view the data along with the commit details

#**git show v1.4**

**Create the following .gitignore file in the root of your Git directory to ignore the specified directory and file.**

**#echo "\*.jar" >> .gitignore**

**How to update the repository by pointing the HEAD to different commit and update to the remote repository:**

# git reset --hard <commit\_ID>

#git push -u -f origin master

Clone

Clone operation creates the instance of the repository. Clone operation not only checks out the working copy, but it also mirrors the complete repository. Users can perform many operations with this local repository. The only time networking gets involved is when the repository instances are being synchronized.

+++

# git clone <URL of the repository that you want to download>

+++

Pull

Pull operation copies the changes from a remote repository instance to a local one. The pull operation is used for synchronization between two repository instances. This is same as the update operation in Subversion.

+++

# git pull --> without any further options

+++

Push

Push operation copies changes from a local repository instance to a remote one. This is used to store the changes permanently into the Git repository. This is same as the commit operation in Subversion.

+++

# git push -u origin <branch\_name>

# git push origin master

+++

**Stash**

Your code is in progress and suddenly a customer escalation comes. Because of this, you have to keep aside your new feature work for a few hours

stashing will do 2 things

1. whenever you modifing the changes it will take backup.

2. and it revert back with orginal position of the file where you started.

git stash --> create stash, remove changes from working directory(when files create and edit)

git stash list--> list all stash available for the repository

git stash apply stash@{0} -->my 1st idea is good (it only apply it not remove)

git stash pop --> it will apply and remove the last stash array

git stash clear --> remove all backup entrys

Undo/Reverting Changes after adding to staging area :-

=====================================================

i added a file into staging area, now i want to modify it before going to commit it?

Syntax: git reset HEAD <file>

sol is : 1. git reset (mixed is the default option)

2. git reset --mixed --> revert back / undo the changes from staging area

3. git reset --hard --> to remove changes from all the 3 areas

@ working directory

@ staging Area

@ Repository(once file came to staging area, it creats temperary commit id in repo)

4. git reset --soft --> remove changes only in repo temp id (HEAD)

HEAD--> once we add file into staging area it create one temp id in the repo as a reference, this is the latest commit id which we working.

Undo/Reverting Changes after commitid:-

=======================================

1. git revert 256ed01(1st 7char) --> it revert only the content not the complete entry(commit id) and create a new commit id and refering to previous.