**What is Jenkins?**

Jenkins is a self-contained, open source automation server which can be used to automate all sorts of tasks related to building, testing, and delivering or deploying software.

**Continuous Delivery** – involves a manual trigger to production.

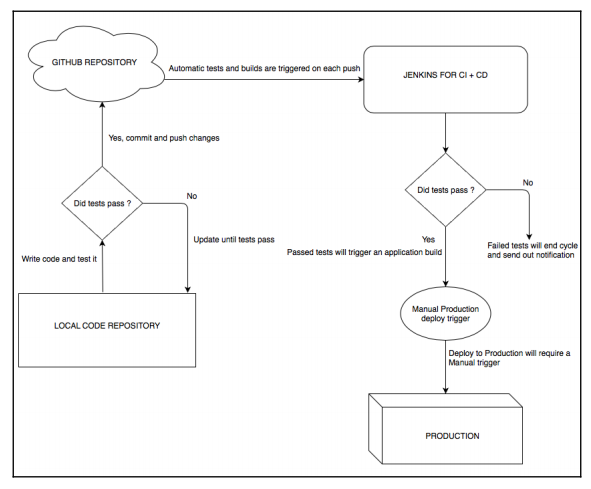
**Continuous Deployment** – involves automatic releases to production.

**Continuous Integration**– is usually the initial part of both Continuous Delivery and Deployment, involving the testing and building of any new or updated source code.

Jenkins has the following advantages over software development:

* It is open source and completely free.
* Issues with tests and builds are detected easily and reported almost immediately.
* Jenkins is platform independent, available on Windows, macOS, and Linux.
* It is easily configurable and customizable for any project.

Below is simple diagram describing the steps for Continuous Delivery



**Installing Jenkins**

Refer below link.

<https://www.blazemeter.com/blog/how-to-install-jenkins-on-windows>

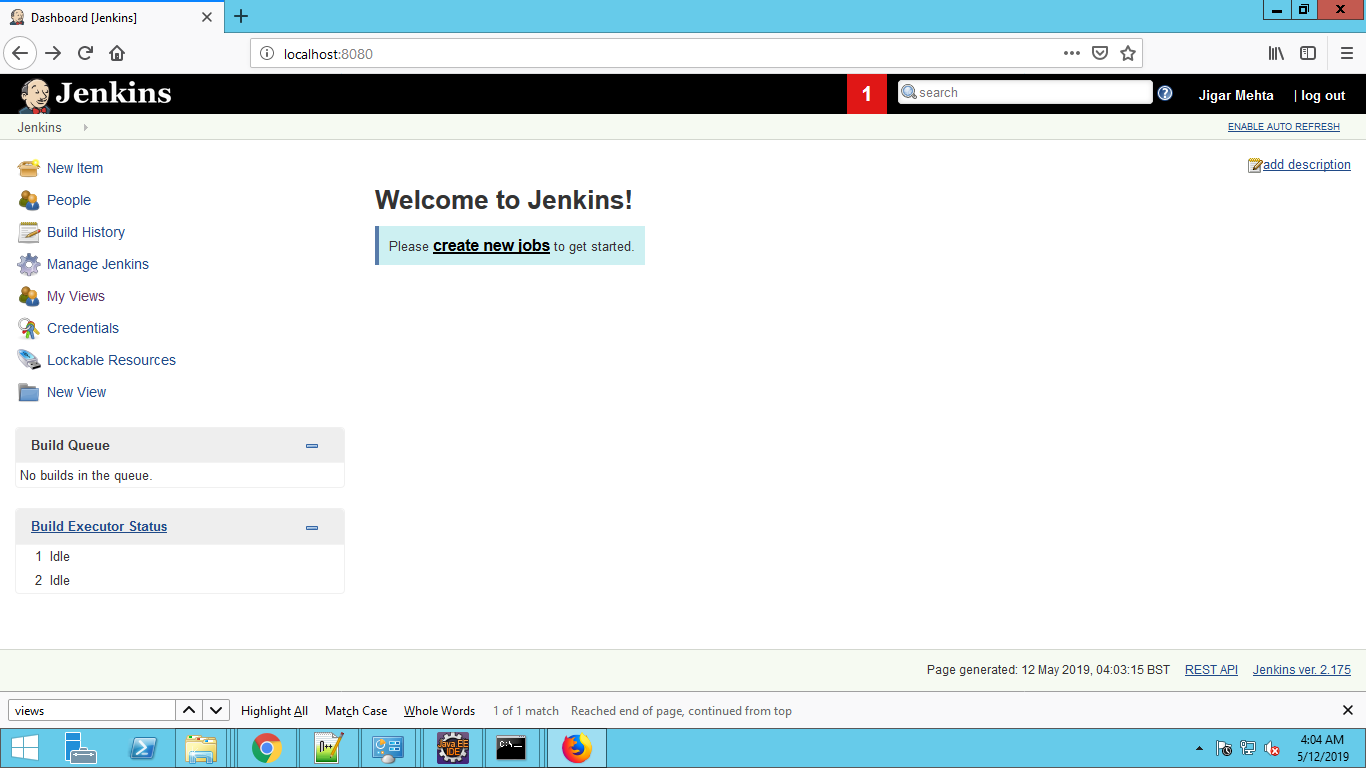
1. Run below command in CMD prompt where Jenkins.war file is located to start Jenkins if not installed as a Windows service

java -jar jenkins.war

1. Navigate to below link in browser.

<http://localhost:8080/>

Below screen will appear



In the top-right corner, there's a drop-down linked to your username. They are

**Builds**: Option to view all pipeline builds

**Configure**: Add a new project or Job

**User Views**: Display custom user views.

**Credentials**: Display credentials, if authorized

|  |  |
| --- | --- |
| **Selection** | **Explanation** |
| New Item | Allows the user to create a new item, which could be a project, pipeline, and so on. |
| People | Lists all the users available. |
| Build History | Shows all builds. |
| Manage Jenkins | Lists all configurations related to the Jenkins server. |
| My Views | Lists all custom user views. |
| Credentials | Lists all of the user and server credentials available. |
| New View | Allows you to create a new view. |

Run below command in Windows batch command in Jenkins Project:

echo BRANCH\_NAME = %BRANCH\_NAME%

echo BUILD\_DISPLAY\_NAME = %BUILD\_DISPLAY\_NAME%

echo BUILD\_ID = %BUILD\_ID%

echo BUILD\_NUMBER = %BUILD\_NUMBER%

echo BUILD\_TAG = %BUILD\_TAG%

echo BUILD\_URL = %BUILD\_URL%

echo CHANGE\_AUTHOR = %CHANGE\_AUTHOR%

echo CHANGE\_AUTHOR\_DISPLAY\_NAME = %CHANGE\_AUTHOR\_DISPLAY\_NAME%

echo CHANGE\_AUTHOR\_EMAIL = %CHANGE\_AUTHOR\_EMAIL%

echo CHANGE\_ID = %CHANGE\_ID%

echo CHANGE\_TARGET = %CHANGE\_TARGET%

echo CHANGE\_TITLE = %CHANGE\_TITLE%

echo CHANGE\_URL = %CHANGE\_URL%

echo EXECUTOR\_NUMBER = %EXECUTOR\_NUMBER%

echo GIT\_AUTHOR\_EMAIL = %GIT\_AUTHOR\_EMAIL%

echo GIT\_AUTHOR\_NAME = %GIT\_AUTHOR\_NAME%

echo GIT\_BRANCH = %GIT\_BRANCH%

echo GIT\_COMMIT = %GIT\_COMMIT%

echo GIT\_COMMITTER\_EMAIL = %GIT\_COMMITTER\_EMAIL%

echo GIT\_COMMITTER\_NAME = %GIT\_COMMITTER\_NAME%

echo GIT\_LOCAL\_BRANCH = %GIT\_LOCAL\_BRANCH%

echo GIT\_PREVIOUS\_COMMIT = %GIT\_PREVIOUS\_COMMIT%

echo GIT\_PREVIOUS\_SUCCESSFUL\_COMMIT = %GIT\_PREVIOUS\_SUCCESSFUL\_COMMIT%

echo GIT\_URL = %GIT\_URL%

echo JENKINS\_HOME = %JENKINS\_HOME%

echo JENKINS\_URL = %JENKINS\_URL%

echo JOB\_BASE\_NAME = %JOB\_BASE\_NAME%

echo JOB\_NAME = %JOB\_NAME%

echo JOB\_URL = %JOB\_URL%

echo NODE\_LABELS = %NODE\_LABELS%

echo NODE\_NAME = %NODE\_NAME%

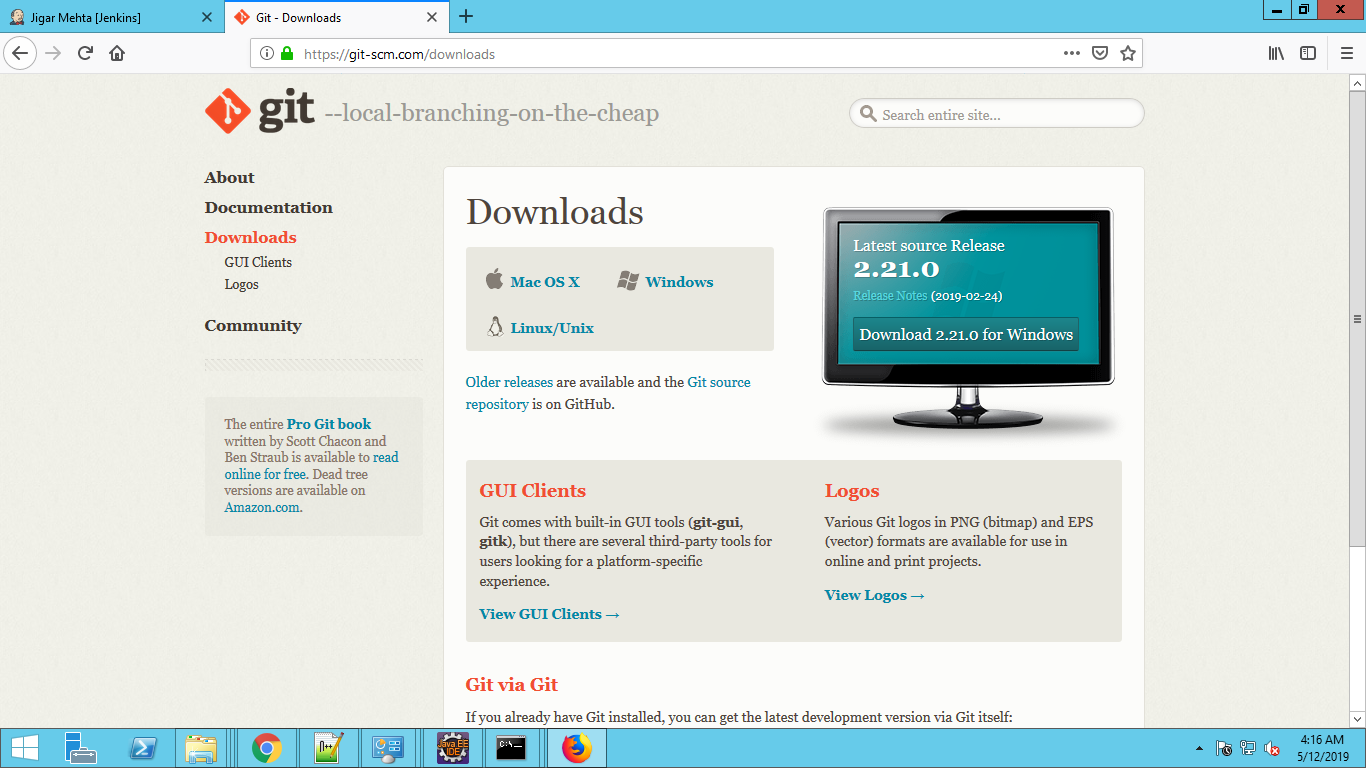
echo SVN\_REVISION = %SVN\_REVISION%

echo SVN\_URL = %SVN\_URL%

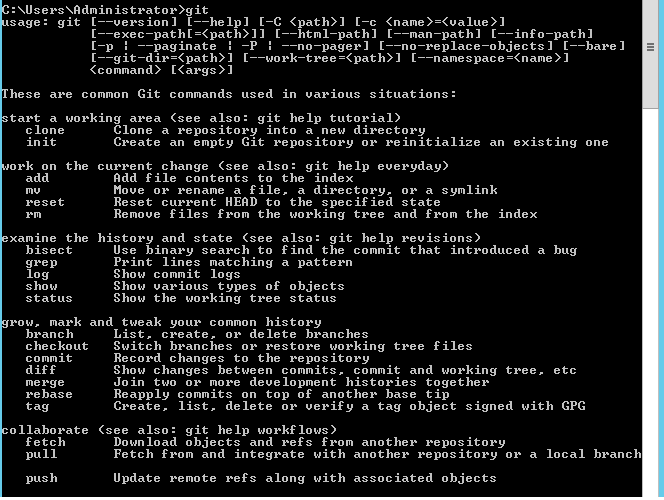
echo WORKSPACE = %WORKSPACE%

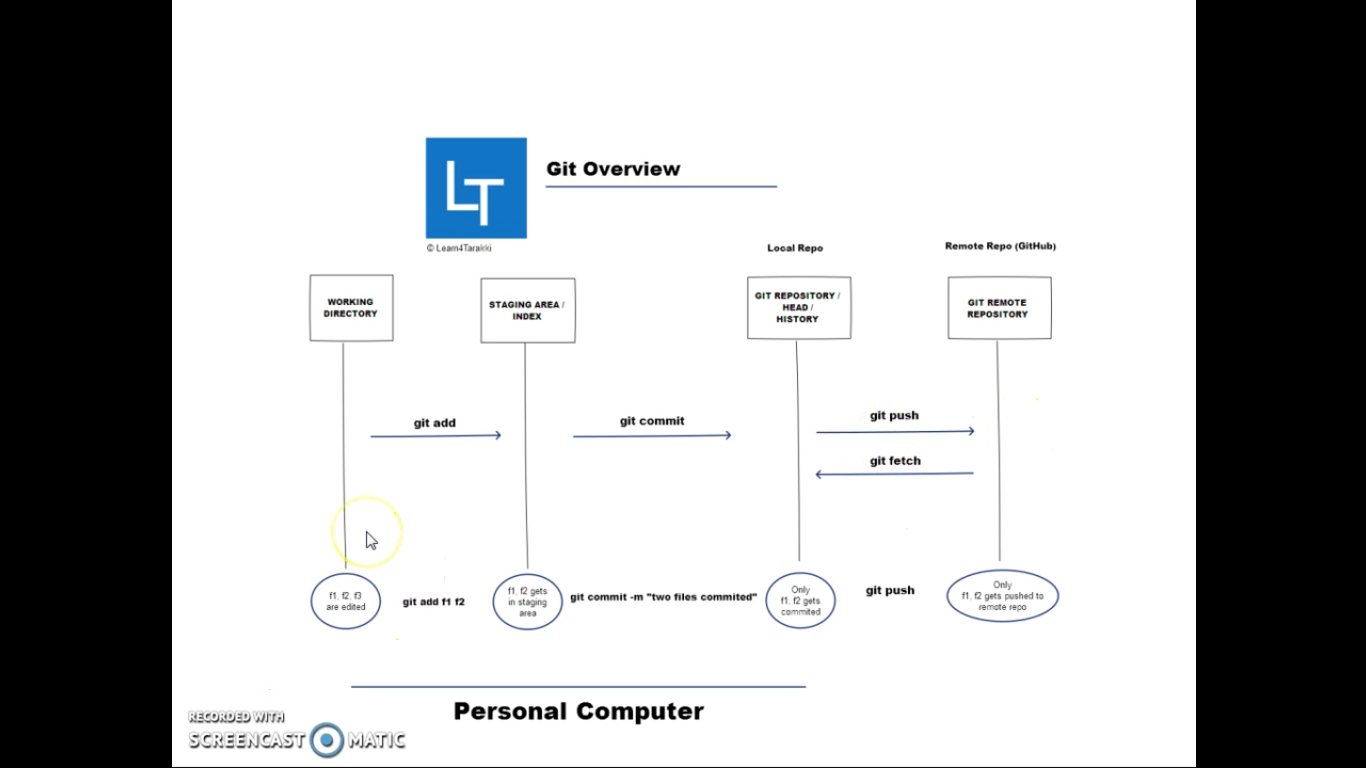
**Install Git:**

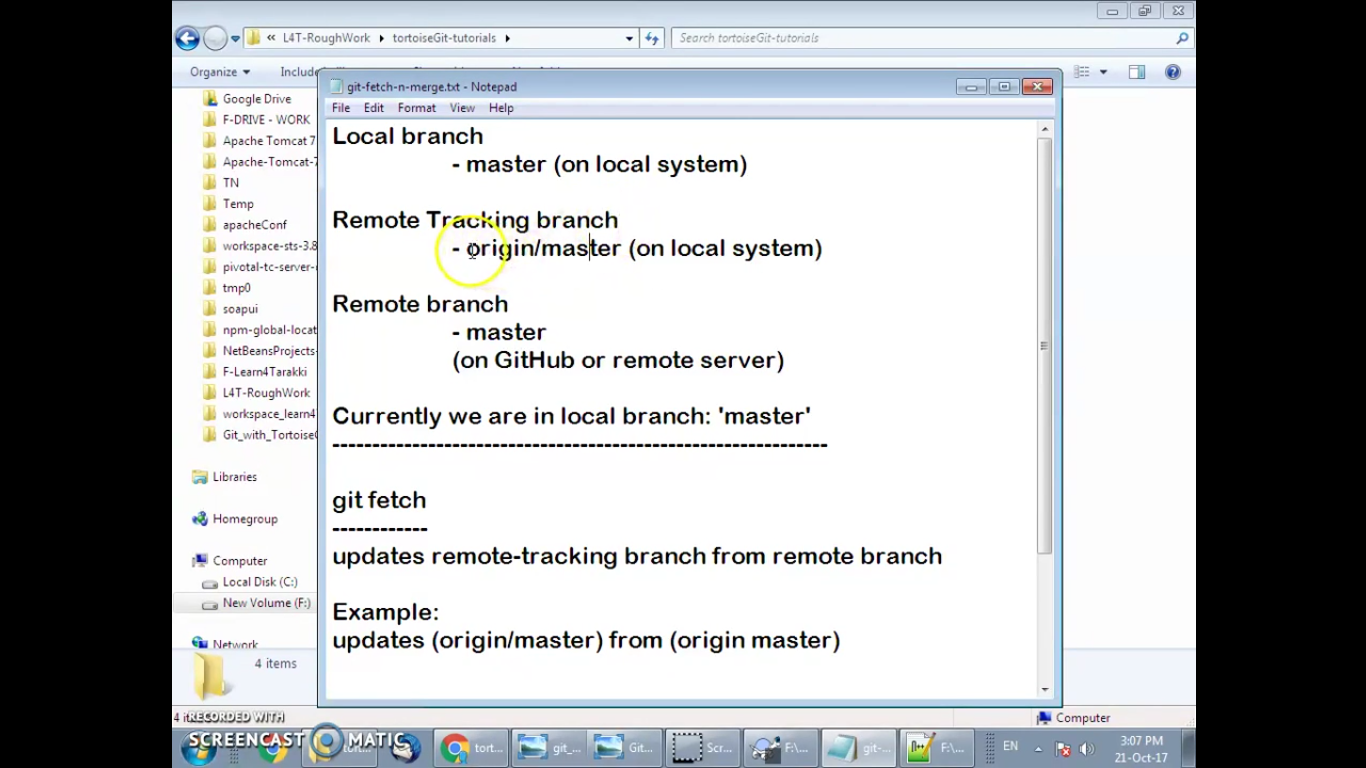
Navigate to <https://git-scm.com/downloads> and Git for Windows

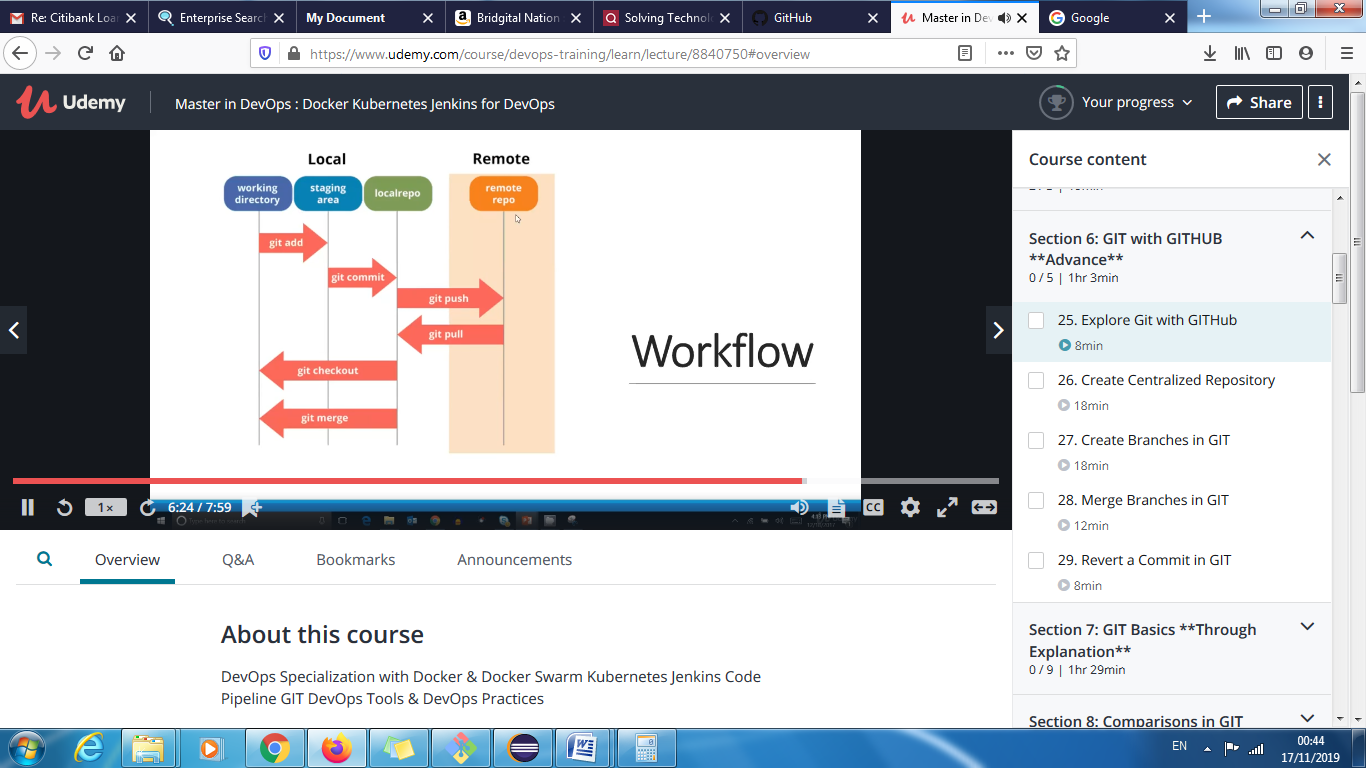


Run git in CMD and it should show below screen that means Git is installed properly







****

**git help -a**

Main Porcelain Commands

add Add file contents to the index

am Apply a series of patches from a mailbox

archive Create an archive of files from a named tree

bisect Use binary search to find the commit that introduced a bug

branch List, create, or delete branches

bundle Move objects and refs by archive

checkout Switch branches or restore working tree files

cherry-pick Apply the changes introduced by some existing commits

citool Graphical alternative to git-commit

clean Remove untracked files from the working tree

clone Clone a repository into a new directory

commit Record changes to the repository

describe Give an object a human readable name based on an available ref

diff Show changes between commits, commit and working tree, etc

fetch Download objects and refs from another repository

format-patch Prepare patches for e-mail submission

gc Cleanup unnecessary files and optimize the local repository

gitk The Git repository browser

grep Print lines matching a pattern

gui A portable graphical interface to Git

init Create an empty Git repository or reinitialize an existing one

log Show commit logs

merge Join two or more development histories together

mv Move or rename a file, a directory, or a symlink

notes Add or inspect object notes

pull Fetch from and integrate with another repository or a local branch

push Update remote refs along with associated objects

range-diff Compare two commit ranges (e.g. two versions of a branch)

rebase Reapply commits on top of another base tip

reset Reset current HEAD to the specified state

revert Revert some existing commits

rm Remove files from the working tree and from the index

shortlog Summarize 'git log' output

show Show various types of objects

stash Stash the changes in a dirty working directory away

status Show the working tree status

submodule Initialize, update or inspect submodules

tag Create, list, delete or verify a tag object signed with GPG

worktree Manage multiple working trees

Ancillary Commands / Manipulators

config Get and set repository or global options

fast-export Git data exporter

fast-import Backend for fast Git data importers

filter-branch Rewrite branches

mergetool Run merge conflict resolution tools to resolve merge conflicts

pack-refs Pack heads and tags for efficient repository access

prune Prune all unreachable objects from the object database

reflog Manage reflog information

remote Manage set of tracked repositories

repack Pack unpacked objects in a repository

replace Create, list, delete refs to replace objects

Ancillary Commands / Interrogators

annotate Annotate file lines with commit information

blame Show what revision and author last modified each line of a file

count-objects Count unpacked number of objects and their disk consumption

difftool Show changes using common diff tools

fsck Verifies the connectivity and validity of the objects in the database

gitweb Git web interface (web frontend to Git repositories)

help Display help information about Git

instaweb Instantly browse your working repository in gitweb

merge-tree Show three-way merge without touching index

rerere Reuse recorded resolution of conflicted merges

show-branch Show branches and their commits

verify-commit Check the GPG signature of commits

verify-tag Check the GPG signature of tags

whatchanged Show logs with difference each commit introduces

Interacting with Others

archimport Import a GNU Arch repository into Git

cvsexportcommit Export a single commit to a CVS checkout

cvsimport Salvage your data out of another SCM people love to hate

cvsserver A CVS server emulator for Git

imap-send Send a collection of patches from stdin to an IMAP folder

p4 Import from and submit to Perforce repositories

quiltimport Applies a quilt patchset onto the current branch

request-pull Generates a summary of pending changes

send-email Send a collection of patches as emails

svn Bidirectional operation between a Subversion repository and Git

Low-level Commands / Manipulators

apply Apply a patch to files and/or to the index

checkout-index Copy files from the index to the working tree

commit-graph Write and verify Git commit-graph files

commit-tree Create a new commit object

hash-object Compute object ID and optionally creates a blob from a file

index-pack Build pack index file for an existing packed archive

merge-file Run a three-way file merge

merge-index Run a merge for files needing merging

mktag Creates a tag object

mktree Build a tree-object from ls-tree formatted text

multi-pack-index Write and verify multi-pack-indexes

pack-objects Create a packed archive of objects

prune-packed Remove extra objects that are already in pack files

read-tree Reads tree information into the index

symbolic-ref Read, modify and delete symbolic refs

unpack-objects Unpack objects from a packed archive

update-index Register file contents in the working tree to the index

update-ref Update the object name stored in a ref safely

write-tree Create a tree object from the current index

Low-level Commands / Interrogators

cat-file Provide content or type and size information for repository objects

cherry Find commits yet to be applied to upstream

diff-files Compares files in the working tree and the index

diff-index Compare a tree to the working tree or index

diff-tree Compares the content and mode of blobs found via two tree objects

for-each-ref Output information on each ref

get-tar-commit-id Extract commit ID from an archive created using git-archive

ls-files Show information about files in the index and the working tree

ls-remote List references in a remote repository

ls-tree List the contents of a tree object

merge-base Find as good common ancestors as possible for a merge

name-rev Find symbolic names for given revs

pack-redundant Find redundant pack files

rev-list Lists commit objects in reverse chronological order

rev-parse Pick out and massage parameters

show-index Show packed archive index

show-ref List references in a local repository

unpack-file Creates a temporary file with a blob's contents

var Show a Git logical variable

verify-pack Validate packed Git archive files

Low-level Commands / Synching Repositories

daemon A really simple server for Git repositories

fetch-pack Receive missing objects from another repository

http-backend Server side implementation of Git over HTTP

send-pack Push objects over Git protocol to another repository

update-server-info Update auxiliary info file to help dumb servers

Low-level Commands / Internal Helpers

check-attr Display gitattributes information

check-ignore Debug gitignore / exclude files

check-mailmap Show canonical names and email addresses of contacts

check-ref-format Ensures that a reference name is well formed

column Display data in columns

credential Retrieve and store user credentials

credential-cache Helper to temporarily store passwords in memory

credential-store Helper to store credentials on disk

fmt-merge-msg Produce a merge commit message

interpret-trailers add or parse structured information in commit messages

mailinfo Extracts patch and authorship from a single e-mail message

mailsplit Simple UNIX mbox splitter program

merge-one-file The standard helper program to use with git-merge-index

patch-id Compute unique ID for a patch

sh-i18n Git's i18n setup code for shell scripts

sh-setup Common Git shell script setup code

stripspace Remove unnecessary whitespace

External commands

flow

lfs

cherry Find commits yet to be applied to upstream

diff-files Compares files in the working tree and the index

diff-index Compare a tree to the working tree or index

diff-tree Compares the content and mode of blobs found via two tree objects

for-each-ref Output information on each ref

get-tar-commit-id Extract commit ID from an archive created using git-archive

ls-files Show information about files in the index and the working tree

ls-remote List references in a remote repository

ls-tree List the contents of a tree object

merge-base Find as good common ancestors as possible for a merge

name-rev Find symbolic names for given revs

pack-redundant Find redundant pack files

rev-list Lists commit objects in reverse chronological order

rev-parse Pick out and massage parameters

show-index Show packed archive index

show-ref List references in a local repository

unpack-file Creates a temporary file with a blob's contents

var Show a Git logical variable

verify-pack Validate packed Git archive files

Low-level Commands / Synching Repositories

daemon A really simple server for Git repositories

fetch-pack Receive missing objects from another repository

http-backend Server side implementation of Git over HTTP

send-pack Push objects over Git protocol to another repository

update-server-info Update auxiliary info file to help dumb servers

Low-level Commands / Internal Helpers

check-attr Display gitattributes information

check-ignore Debug gitignore / exclude files

check-mailmap Show canonical names and email addresses of contacts

check-ref-format Ensures that a reference name is well formed

column Display data in columns

credential Retrieve and store user credentials

credential-cache Helper to temporarily store passwords in memory

credential-store Helper to store credentials on disk

fmt-merge-msg Produce a merge commit message

interpret-trailers add or parse structured information in commit messages

mailinfo Extracts patch and authorship from a single e-mail message

mailsplit Simple UNIX mbox splitter program

merge-one-file The standard helper program to use with git-merge-index

patch-id Compute unique ID for a patch

sh-i18n Git's i18n setup code for shell scripts

sh-setup Common Git shell script setup code

stripspace Remove unnecessary whitespace

External commands

flow

lfs

(END) 1 [sig] bash 10344! sigpacket::process: Suppressing signal 18 to win32 process (pid 9796)

3704118 [sig] bash 10344! sigpacket::process: Suppressing signal 18 to win32 process (pid 9796)

cherry Find commits yet to be applied to upstream

diff-files Compares files in the working tree and the index

diff-index Compare a tree to the working tree or index

diff-tree Compares the content and mode of blobs found via two tree objects

for-each-ref Output information on each ref

get-tar-commit-id Extract commit ID from an archive created using git-archive

ls-files Show information about files in the index and the working tree

ls-remote List references in a remote repository

ls-tree List the contents of a tree object

merge-base Find as good common ancestors as possible for a merge

name-rev Find symbolic names for given revs

pack-redundant Find redundant pack files

rev-list Lists commit objects in reverse chronological order

rev-parse Pick out and massage parameters

show-index Show packed archive index

show-ref List references in a local repository

unpack-file Creates a temporary file with a blob's contents

var Show a Git logical variable

verify-pack Validate packed Git archive files

Low-level Commands / Synching Repositories

daemon A really simple server for Git repositories

fetch-pack Receive missing objects from another repository

http-backend Server side implementation of Git over HTTP

send-pack Push objects over Git protocol to another repository

update-server-info Update auxiliary info file to help dumb servers

Low-level Commands / Internal Helpers

check-attr Display gitattributes information

check-ignore Debug gitignore / exclude files

check-mailmap Show canonical names and email addresses of contacts

check-ref-format Ensures that a reference name is well formed

column Display data in columns

credential Retrieve and store user credentials

credential-cache Helper to temporarily store passwords in memory

credential-store Helper to store credentials on disk

fmt-merge-msg Produce a merge commit message

interpret-trailers add or parse structured information in commit messages

mailinfo Extracts patch and authorship from a single e-mail message

mailsplit Simple UNIX mbox splitter program

merge-one-file The standard helper program to use with git-merge-index

patch-id Compute unique ID for a patch

sh-i18n Git's i18n setup code for shell scripts

sh-setup Common Git shell script setup code

stripspace Remove unnecessary whitespace

External commands

flow

lfs

(END)4587302 [sig] bash 10344! sigpacket::process: Suppressing signal 18 to win32 process (pid 9796)

cherry Find commits yet to be applied to upstream

diff-files Compares files in the working tree and the index

diff-index Compare a tree to the working tree or index

diff-tree Compares the content and mode of blobs found via two tree objects

for-each-ref Output information on each ref

get-tar-commit-id Extract commit ID from an archive created using git-archive

ls-files Show information about files in the index and the working tree

ls-remote List references in a remote repository

ls-tree List the contents of a tree object

merge-base Find as good common ancestors as possible for a merge

name-rev Find symbolic names for given revs

pack-redundant Find redundant pack files

rev-list Lists commit objects in reverse chronological order

rev-parse Pick out and massage parameters

show-index Show packed archive index

show-ref List references in a local repository

unpack-file Creates a temporary file with a blob's contents

var Show a Git logical variable

verify-pack Validate packed Git archive files

Low-level Commands / Synching Repositories

daemon A really simple server for Git repositories

fetch-pack Receive missing objects from another repository

http-backend Server side implementation of Git over HTTP

send-pack Push objects over Git protocol to another repository

update-server-info Update auxiliary info file to help dumb servers

Low-level Commands / Internal Helpers

check-attr Display gitattributes information

check-ignore Debug gitignore / exclude files

check-mailmap Show canonical names and email addresses of contacts

check-ref-format Ensures that a reference name is well formed

column Display data in columns

credential Retrieve and store user credentials

credential-cache Helper to temporarily store passwords in memory

credential-store Helper to store credentials on disk

fmt-merge-msg Produce a merge commit message

interpret-trailers add or parse structured information in commit messages

mailinfo Extracts patch and authorship from a single e-mail message

mailsplit Simple UNIX mbox splitter program

merge-one-file The standard helper program to use with git-merge-index

patch-id Compute unique ID for a patch

sh-i18n Git's i18n setup code for shell scripts

sh-setup Common Git shell script setup code

stripspace Remove unnecessary whitespace

External commands

flow

lfs

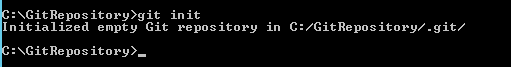
(END)

**Upload project on Github:**

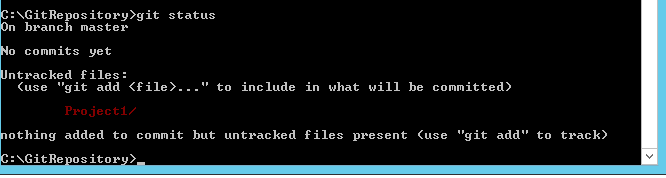
Create a local folder ‘GitRepository’ in C drive. This will be used as Git Repository

In the CMD prompt go to c:\ GitRepository and run ‘git init’. See below screenshot with result.

.git folder (hidden) is created in this folder.



Run ‘git status’. See below result



Git commands:

<https://dzone.com/articles/top-20-git-commands-with-examples>

<https://github.com/joshnh/Git-Commands>

**git config**

It is a good idea to introduce you to Git with your name and public email address before doing any operation. The easiest way to do so is:

**Local config can be found at below location**

D:\Eclipse SimRel WorkSpace\Sharepoint\_Automation\_2019\.git\config

**Global config can be found at below location**

C:\Users\jigar.mehta1\.gitconfig

**System Config can be found at below location**

C:\Program Files\Git\mingw64\etc\gitconfig

Sample config file

[core]

bare = false

repositoryformatversion = 0

filemode = false

symlinks = false

ignorecase = true

logallrefupdates = true

age=13

name = jigarlocal

name = mehtalocal

[remote "Sharepoint\_Automation\_2019"]

url = https://github.com/jigarmehta1999/Sharepoint\_Automation\_2019.git

fetch = +refs/heads/\*:refs/remotes/Sharepoint\_Automation\_2019/\*

Usage: git config --global user.name “[name]”

Usage: git config --global user.email “[email address]”

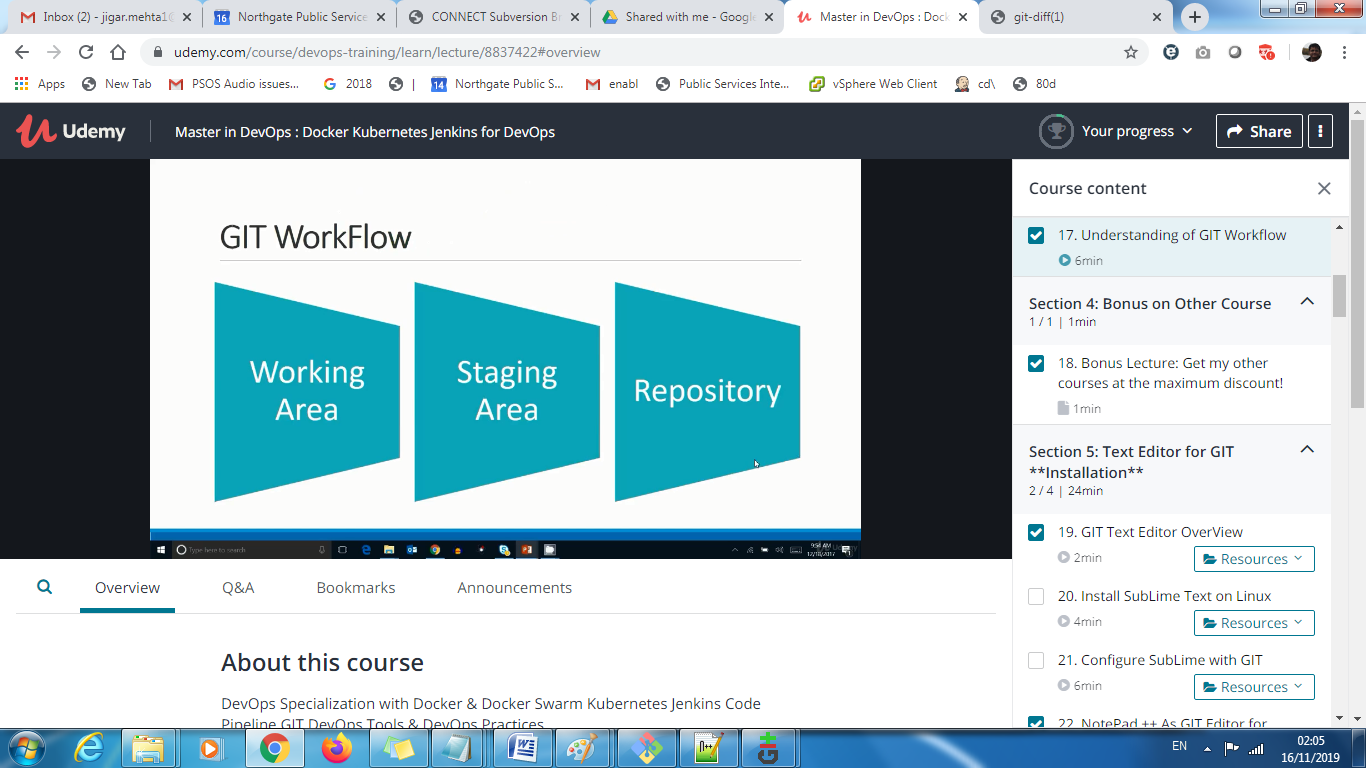
e.g.

git config --global user.name “jigarmehta1999”

git config --global user.email “[jigar.digital@gmail.com]”

git status

git log



This command sets the author name and email address respectively to be used with your commits.

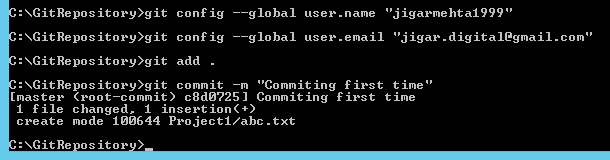
### Basic Snapshotting

| **Command** | **Description** |
| --- | --- |
| git status | Check status |
| git add [file-name.txt] | Add a file to the staging area |
| git add –A | Add all new and changed files to the staging area |
| git commit -m "[commit message]" | Commit changes |
| git rm -r [file-name.txt] | Remove a file (or folder) |

Run below commands

git add . (Adds files to staging area)

git commit -m "hello1"



Run below commands:

Step1: Connect to Github with username and repository name

**Syntax:** git remote add origin https://github.com/[username]/[repository-gitname].git

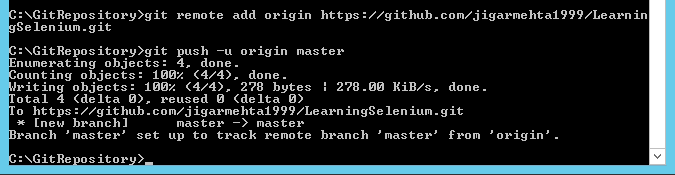
E.g.

git remote add origin <https://github.com/jigarmehta1999/LearningSelenium.git>

Push changes to remote repository (and remember the branch)

Syntax: git push -u origin [branch name]

git push -u origin master

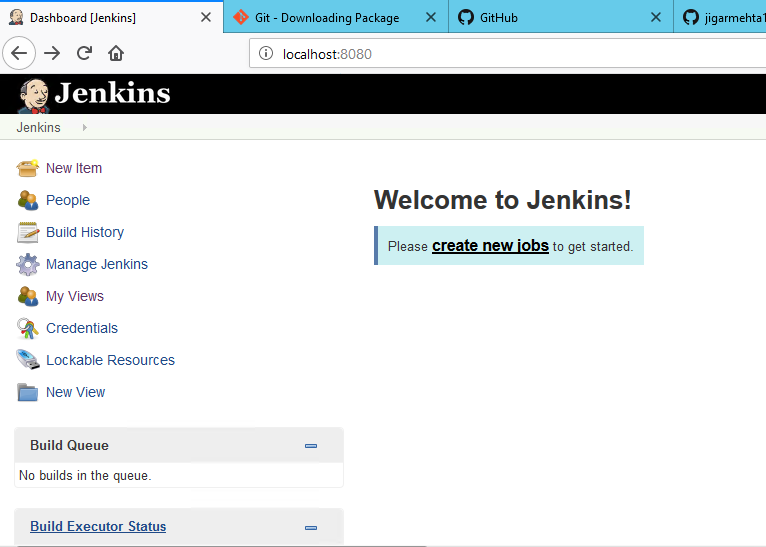


**List of all Git Commands**

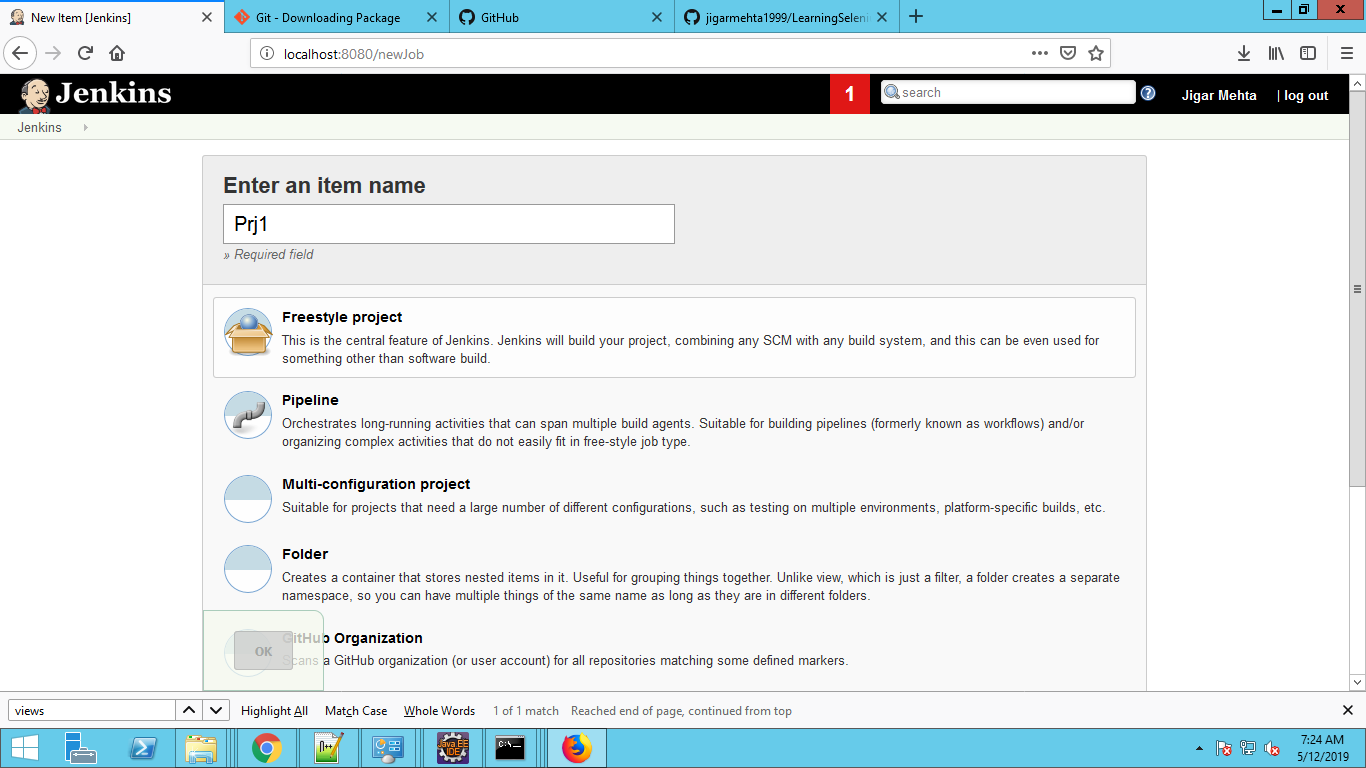
|  |
| --- |
| D:\GitRepository>git help -a |
| See 'git help <command>' to read about a specific subcommand |
|  |
| Main Porcelain Commands |
| add Add file contents to the index |
| am Apply a series of patches from a mailbox |
| archive Create an archive of files from a named tree |
| bisect Use binary search to find the commit that introduced a bug |
| branch List, create, or delete branches |
| bundle Move objects and refs by archive |
| checkout Switch branches or restore working tree files |
| cherry-pick Apply the changes introduced by some existing commits |
| citool Graphical alternative to git-commit |
| clean Remove untracked files from the working tree |
| clone Clone a repository into a new directory |
| commit Record changes to the repository |
| describe Give an object a human readable name based on an available ref |
| diff Show changes between commits, commit and working tree, etc |
| fetch Download objects and refs from another repository |
| format-patch Prepare patches for e-mail submission |
| gc Cleanup unnecessary files and optimize the local repository |
| gitk The Git repository browser |
| grep Print lines matching a pattern |
| gui A portable graphical interface to Git |
| init Create an empty Git repository or reinitialize an existing one |
| log Show commit logs |
| merge Join two or more development histories together |
| mv Move or rename a file, a directory, or a symlink |
| notes Add or inspect object notes |
| pull Fetch from and integrate with another repository or a local branch |
| push Update remote refs along with associated objects |
| range-diff Compare two commit ranges (e.g. two versions of a branch) |
| rebase Reapply commits on top of another base tip |
| reset Reset current HEAD to the specified state |
| revert Revert some existing commits |
| rm Remove files from the working tree and from the index |
| shortlog Summarize 'git log' output |
| show Show various types of objects |
| stash Stash the changes in a dirty working directory away |
| status Show the working tree status |
| submodule Initialize, update or inspect submodules |
| tag Create, list, delete or verify a tag object signed with GPG |
| worktree Manage multiple working trees |
|  |
| Ancillary Commands / Manipulators |
| config Get and set repository or global options |
| fast-export Git data exporter |
| fast-import Backend for fast Git data importers |
| filter-branch Rewrite branches |
| mergetool Run merge conflict resolution tools to resolve merge conflicts |
| pack-refs Pack heads and tags for efficient repository access |
| prune Prune all unreachable objects from the object database |
| reflog Manage reflog information |
| remote Manage set of tracked repositories |
| repack Pack unpacked objects in a repository |
| replace Create, list, delete refs to replace objects |
| Ancillary Commands / Interrogators |
| annotate Annotate file lines with commit information |
| blame Show what revision and author last modified each line of a file |
| count-objects Count unpacked number of objects and their disk consumption |
| difftool Show changes using common diff tools |
| fsck Verifies the connectivity and validity of the objects in the database |
| gitweb Git web interface (web frontend to Git repositories) |
| help Display help information about Git |
| instaweb Instantly browse your working repository in gitweb |
| merge-tree Show three-way merge without touching index |
| rerere Reuse recorded resolution of conflicted merges |
| show-branch Show branches and their commits |
| verify-commit Check the GPG signature of commits |
| verify-tag Check the GPG signature of tags |
| whatchanged Show logs with difference each commit introduces |
|  |
| Interacting with Others |
| archimport Import a GNU Arch repository into Git |
| cvsexportcommit Export a single commit to a CVS checkout |
| cvsimport Salvage your data out of another SCM people love to hate |
| cvsserver A CVS server emulator for Git |
| imap-send Send a collection of patches from stdin to an IMAP folder |
| p4 Import from and submit to Perforce repositories |
| quiltimport Applies a quilt patchset onto the current branch |
| request-pull Generates a summary of pending changes |
| send-email Send a collection of patches as emails |
| svn Bidirectional operation between a Subversion repository and Git |
|  |
| Low-level Commands / Manipulators |
| apply Apply a patch to files and/or to the index |
| checkout-index Copy files from the index to the working tree |
| commit-graph Write and verify Git commit-graph files |
| commit-tree Create a new commit object |
| hash-object Compute object ID and optionally creates a blob from a file |
| index-pack Build pack index file for an existing packed archive |
| merge-file Run a three-way file merge |
| merge-index Run a merge for files needing merging |
| mktag Creates a tag object |
| mktree Build a tree-object from ls-tree formatted text |
| multi-pack-index Write and verify multi-pack-indexes |
| pack-objects Create a packed archive of objects |
| prune-packed Remove extra objects that are already in pack files |
| read-tree Reads tree information into the index |
| symbolic-ref Read, modify and delete symbolic refs |
| unpack-objects Unpack objects from a packed archive |
| update-index Register file contents in the working tree to the index |
| update-ref Update the object name stored in a ref safely |
| write-tree Create a tree object from the current index |
|  |
| Low-level Commands / Interrogators |
| cat-file Provide content or type and size information for repository objects |
| cherry Find commits yet to be applied to upstream |
| diff-files Compares files in the working tree and the index |
| diff-index Compare a tree to the working tree or index |
| diff-tree Compares the content and mode of blobs found via two tree objects |
| for-each-ref Output information on each ref |
| get-tar-commit-id Extract commit ID from an archive created using git-archive |
| ls-files Show information about files in the index and the working tree |
| ls-remote List references in a remote repository |
| ls-tree List the contents of a tree object |
| merge-base Find as good common ancestors as possible for a merge |
| name-rev Find symbolic names for given revs |
| pack-redundant Find redundant pack files |
| rev-list Lists commit objects in reverse chronological order |
| rev-parse Pick out and massage parameters |
| show-index Show packed archive index |
| show-ref List references in a local repository |
| unpack-file Creates a temporary file with a blob's contents |
| var Show a Git logical variable |
| verify-pack Validate packed Git archive files |
|  |
| Low-level Commands / Synching Repositories |
| daemon A really simple server for Git repositories |
| fetch-pack Receive missing objects from another repository |
| http-backend Server side implementation of Git over HTTP |
| send-pack Push objects over Git protocol to another repository |
| update-server-info Update auxiliary info file to help dumb servers |
|  |
| Low-level Commands / Internal Helpers |
| check-attr Display gitattributes information |
| check-ignore Debug gitignore / exclude files |
| check-mailmap Show canonical names and email addresses of contacts |
| check-ref-format Ensures that a reference name is well formed |
| column Display data in columns |
| credential Retrieve and store user credentials |
| credential-cache Helper to temporarily store passwords in memory |
| credential-store Helper to store credentials on disk |
| fmt-merge-msg Produce a merge commit message |
| interpret-trailers add or parse structured information in commit messages |
| mailinfo Extracts patch and authorship from a single e-mail message |
| mailsplit Simple UNIX mbox splitter program |
| merge-one-file The standard helper program to use with git-merge-index |
| patch-id Compute unique ID for a patch |
| sh-i18n Git's i18n setup code for shell scripts |
| sh-setup Common Git shell script setup code |
| stripspace Remove unnecessary whitespace |
|  |
| External commands |
| flow |
| lfs |

**Connect Jenkins to Github**

Create a project ‘Prj1’ in Jenkins by clicking ‘New item’



Select Freestyle project



Project -> Staging -> Local Git Repo -> Github

**Git branch**

Git stores a branch as a reference to a commit. In this sense, a branch represents the tip of a series of commits. A branch represents an independent line of development. Branches serve as an abstraction for the edit/stage/commit process.

The git branch command lets you create, list, rename, and delete branches.

git branch

List all of the branches in your repository. This is synonymous with git branch --list.

git branch <branch>

Create a new branch called <branch>. This does not check out the new branch.

git branch -d <branch>

Delete the specified branch. This is a “safe” operation in that Git prevents you from deleting the branch if it has unmerged changes.

git branch -D <branch>

Force delete the specified branch, even if it has unmerged changes. This is the command to use if you want to permanently throw away all of the commits associated with a particular line of development.

git branch -m <branch>

Rename the current branch to <branch>.

git branch -a

List all remote branches.

git branch branch\_name

Create a new branch i.e. branch\_name

$ git remote add new-remote-repo https://bitbucket.com/user/repo.git

# Add remote repo to local repo config

$ git push <new-remote-repo> crazy-experiment

# pushes the crazy-experiment branch to new-remote-repo

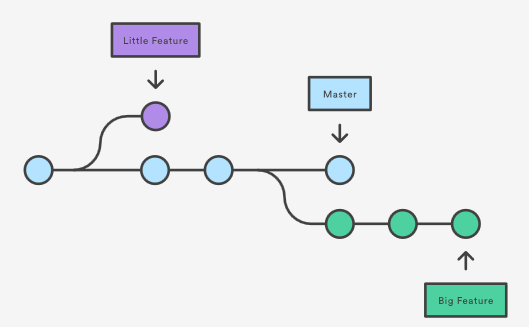
**Deleting branch on local**

git branch -d crazy-experiment

git branch -D crazy-experiment

**Deleting branch on remote github**

git push –d origin remote\_github\_branch\_name



git init

git remote add origin <https://github.com/jigarmehta1999/sample.git>

git pull origin master

git add .

git commit –m “type commit message”

git push origin master

Note: Branch in a GIT is pointer to Commit

git branch branch\_name (create new local branch reference in refs/heads/)

git checkout branch\_name (to move to working on a branch branch\_name)

git branch (List all available branches in the local .git/refs/heads

To rename an existing branch

$git branch –m old\_branch\_name new\_branch\_name

To delete an existing branch.

$git branch –d branch\_name

jigar.mehta1@MUM-LAP-409 MINGW64 ~/sample1 (master)

$ git branch -help

usage: git branch [<options>] [-r | -a] [--merged | --no-merged]

or: git branch [<options>] [-l] [-f] <branch-name> [<start-point>]

or: git branch [<options>] [-r] (-d | -D) <branch-name>...

or: git branch [<options>] (-m | -M) [<old-branch>] <new-branch>

or: git branch [<options>] (-c | -C) [<old-branch>] <new-branch>

or: git branch [<options>] [-r | -a] [--points-at]

or: git branch [<options>] [-r | -a] [--format]

Generic options

-v, --verbose show hash and subject, give twice for upstream branch

-q, --quiet suppress informational messages

-t, --track set up tracking mode (see git-pull(1))

-u, --set-upstream-to <upstream>

change the upstream info

--unset-upstream Unset the upstream info

--color[=<when>] use colored output

-r, --remotes act on remote-tracking branches

--contains <commit> print only branches that contain the commit

--no-contains <commit>

print only branches that don't contain the commit

--abbrev[=<n>] use <n> digits to display SHA-1s

Specific git-branch actions:

-a, --all list both remote-tracking and local branches

-d, --delete delete fully merged branch

-D delete branch (even if not merged)

-m, --move move/rename a branch and its reflog

-M move/rename a branch, even if target exists

-c, --copy copy a branch and its reflog

-C copy a branch, even if target exists

-l, --list list branch names

--create-reflog create the branch's reflog

--edit-description edit the description for the branch

-f, --force force creation, move/rename, deletion

--merged <commit> print only branches that are merged

--no-merged <commit> print only branches that are not merged

--column[=<style>] list branches in columns

--sort <key> field name to sort on

--points-at <object> print only branches of the object

-i, --ignore-case sorting and filtering are case insensitive

--format <format> format to use for the output

git diff

git checkout . or git checkout filename.ext (discard changes in working directory)

git reset HEAD . or git reset HEAD filename.ext (to unstage from staging area)

git rest HEAD~ (to undo changes after committing the code)

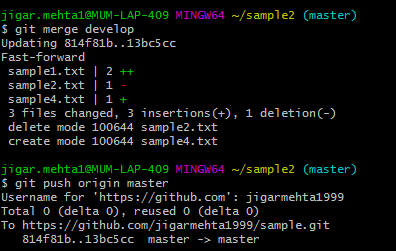
git status -> will shows the above two commands

git log -> will history if all changes

git rm (to delete files from local git)

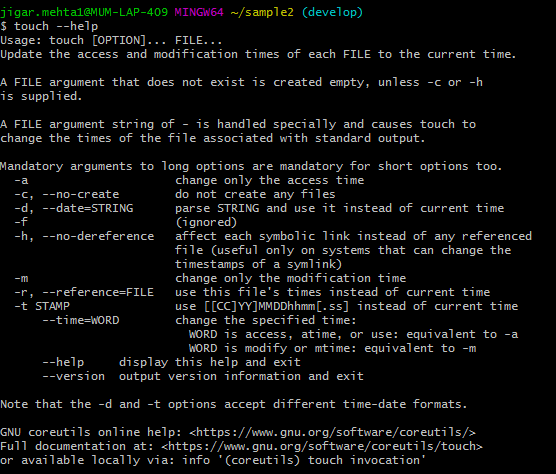
git clone <Url of .git >

In order to merge a branch use commands in below screenshot



**Git checkout**

In Git terms, a "checkout" is the act of switching between different versions of a target entity.

touch <filename>

echo “Hello World” > testfile.txt

Above command will remove existing text from testfile.txt and add “Hello World”

echo “Hello World” >> testfile.txt

Above command will append text “Hello World” to testfile.txt

$ git mv -help

usage: git mv [<options>] <source>... <destination>

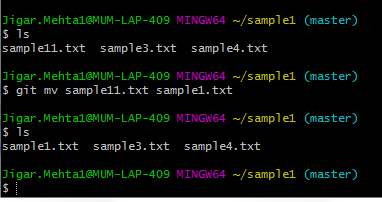
git mv will change the name of file in working area as well as staging area

-v, --verbose be verbose

-n, --dry-run dry run

-f, --force force move/rename even if target exists

-k skip move/rename errors



git log

git log –help

usage: git log [<options>] [<revision-range>] [[--] <path>...]

or: git show [<options>] <object>...

-q, --quiet suppress diff output

--source show source

--use-mailmap Use mail map file

--decorate-refs <pattern>

only decorate refs that match <pattern>

--decorate-refs-exclude <pattern>

do not decorate refs that match <pattern>

--decorate[=...] decorate options

-L <n,m:file> Process line range n,m in file, counting from 1

git log –oneline –graph –decorate

git log --since="2 days ago"

--follow

Continue listing the history of a file beyond renames (works only for a single file).

E.g.

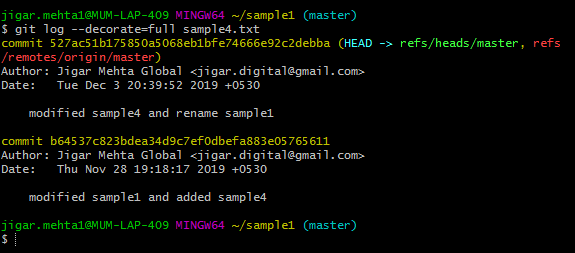
git log --follow <filename.ext>

Shows all changes/commit made for filename.ext

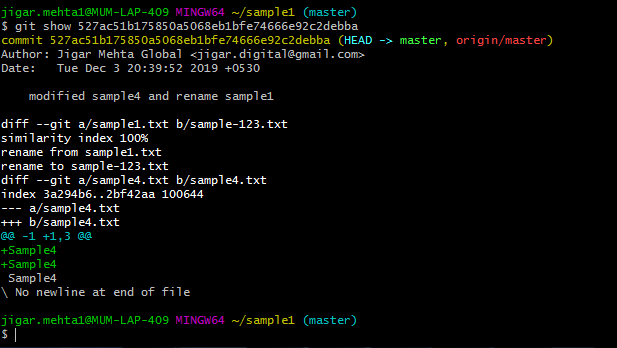
--decorate[=short|full|auto|no]

Print out the ref names of any commits that are shown.

E.g.



git show <commitid>



alias

jigar.mehta1@MUM-LAP-409 MINGW64 ~/sample1 (master)

$ git config --local alias.decoratefull "log --decorate=full sample4.txt"

The above command will create below entry in locan config file.

[alias]

decoratefull = log --decorate=full sample4.txt

Command aliases for the git command wrapper - e.g. after defining "alias. decoratefull = log --decorate=full sample4.txt ", the invocation "git decoratefull" is equivalent to ‘git log --decorate=full sample4.txt’.

jigar.mehta1@MUM-LAP-409 MINGW64 ~/sample1 (master)

$ git decoratefull

commit 527ac51b175850a5068eb1bfe74666e92c2debba (HEAD -> refs/heads/master, refs/remotes/origin/master)

Author: Jigar Mehta Global <jigar.digital@gmail.com>

Date: Tue Dec 3 20:39:52 2019 +0530

modified sample4 and rename sample1

commit b64537c823bdea34d9c7ef0dbefa883e05765611

Author: Jigar Mehta Global <jigar.digital@gmail.com>

Date: Thu Nov 28 19:18:17 2019 +0530

modified sample1 and added sample4

jigar.mehta1@MUM-LAP-409 MINGW64 ~/sample1 (master)

$

.gitignore

In order to stop git to move file from working directory to staging area add that file or folders in .gitignore file

Specific File: TestFile.txt

File pattern: \*.log

Folder: TestResources/

Git can only ignore files that are untracked - files that haven't been committed to the repository, yet. That's why, when you create a new repository, you should also create a .gitignore file with all the file patterns you want to ignore.

However, of course, not everything goes perfect... and files slip through that you later would like to see ignored.

**Preparing the Cleanup**

Before cleaning up your repository, you should conduct two important preparations:

1. Make sure your .gitignore file is up-to-date and contains all the correct patterns you want to ignore.
2. Commit or stash any outstanding local changes you might have. Your working copy should be clean before you continue.

In three steps, you can clean up your repository and make sure your ignored items are indeed ignored:

$ git rm -r --cached .

$ git add .

$ git commit -m "Clean up ignored files"

Working Area -> Staging Area -> Git Repo

Below command shows changes in Working Area and Staging area

$ git diff

$ git diff filename.ext

Below command shows changes in Working Area and Git Repo (i.e. last commit)

$ git diff HEAD

$ git diff HEAD filename.ext

Below command shows changes in Staging Area and Git Repo (i.e. last commit)

$ git diff --staged HEAD

$ git diff --staged HEAD filename.ext

Below command shows changes between two commits in Git Repo

$ git diff previous\_commitid new\_commitid

The above can also be written as below. It will show differences in all the modified files in Git Repo

$ git diff HEAD~ HEAD

Below command will show difference in Git Repo for filename.ext only

$ git diff HEAD~ HEAD filename.ext

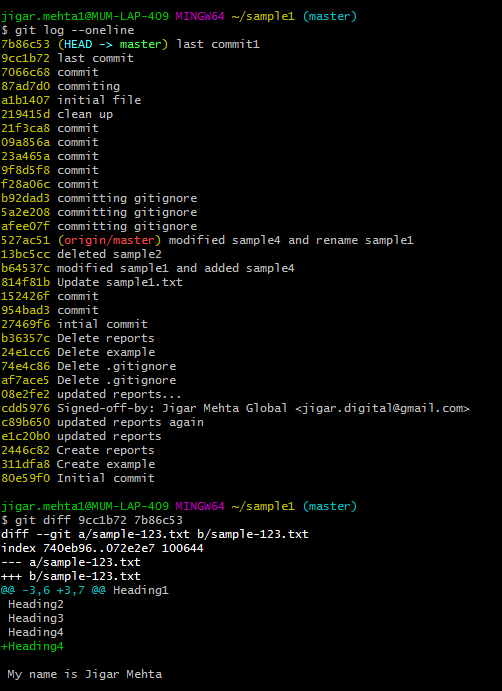
Below command will show difference 2 branches

$ git diff branchname1 branchname2

Branching and Merging

Below will list all local and remote branch names

$git branch –a



**When to use Git rebase?**

**Below process is also called Centralized Workflow**

**User1 and User2 both take a branch from Master at same commit.**

**Changes made by User1 and push it to Remote server on master branch**

git status # View the state of the repo

git add <some-file> # Stage a file

git commit # Commit a file</some-file>

git push origin master

**Changes made by User2 and try to push it to Remote server on master branch**

git status # View the state of the repo

git add <some-file> # Stage a file

git commit # Commit a file</some-file>

git push origin master

But, since her local history has diverged from the central repository, Git will refuse the request with a rather verbose error message:

error: failed to push some refs to '/path/to/repo.git'

hint: Updates were rejected because the tip of your current branch is behind

hint: its remote counterpart. Merge the remote changes (e.g. 'git pull')

hint: before pushing again.

hint: See the 'Note about fast-forwards' in 'git push --help' for details.

git pull --rebase origin master

If while rebasing there are conflicts then Git will pause the rebase at the current commit and output the following message, along with some relevant instructions:

CONFLICT (content): Merge conflict in <some-file>

Run git status to see where the problem is. Conflicted files will appear in Unmerged paths

# Unmerged paths:

# (use "git reset HEAD <some-file>..." to unstage)

# (use "git add/rm <some-file>..." as appropriate to mark resolution)

#

# both modified: <some-file>

Edit the files in listed in unmerged paths and stage the file in usual fashion i.e.

git add <some-file>

git rebase –continue

Git will move on to the next commit and repeat the process for any other commits that generate conflicts.

If these go wrong or you have no idea what’s going on then execute the following command and you’ll be right back to where you started:

git rebase –abort

After done synchronizing with the central repository publish changes to remote server:

git push origin master

Stashing: