Why Command Line / Terminal?

Git was originally designed as a command line tool and graphical clients came later. As a result, new features make it onto the command line well before they are integrated into a graphical client.
online help; nearly all online assistance websites, blogs, or other tutorials, use the command line as the standard way of communicating how Git does something.
Only the command line provides all the raw **power** of Git.Much like my online help argument, going command line allows me one set of commands in Git, that will work on Windows, the Mac OS, and even Linux.

Why Source Control

Backup: Source control is a type of backup.
Versioning/ History : It's not just a normal backup that's difficult to access or restore;no, instead, it's an ongoing backup of every version of your source you have made; creating a history, or a trail, of changes. This allows you to **undo changes** that you've made or restore from a previous known state. In doing so, you could **compare** various revisions of your code; for example, what has changed in the last two weeks.
Collaboration / Teamwork.
Branching : Source control supports the concept of branching off to make changes in isolation. This encourages experimentation, but in a safe way, without immediately impacting ongoing new development.
Code Review : some source control systems provide excellent tools for reviewing code for collaborative learning and improving the quality of the project.

IMP : Always commit after small work or changes.

Source control Options
Two major types: **centralized** and **decentralized / distributed**
Centralized :
free and open source options : Subversion and CVS,
Commercial : ClearCase, Perforce, and TFS, as well as many others.
The main concept with centralized systems, is that there is a centralized server that is the ultimate source of a collection of versioned files. This normally means that a network connection is required to the central server for most operations, including just editing a file.
decentralized / distributed:
Doesn't mandate a central source of truth, and allows for most operations to be local.
Source control systems that fit this model include **Mercurial and Git**

What is GIT?

Git is a decentralized and distributed version control system. While Git is decentralized, most people still choose to use it with a central server to serve as the main repository(remote) for a project or team.
Another key benefit of being distributed is that most operations in Git are local; there are only a few commands that require a network connection. Otherwise, you can work completely disconnected;

Install Git in mac : sudo xcode-select --install //Install xcode command line developer tool.
Also you can use package manager like (Homebrew, MacPorts) to install git. But it might lead to and issue. Fo use command line developer tool to install git.

Key Concept

Repository : Collections of version **controlled files** are kept together in a repository. The repository also contains the **history** of changes and any special **configuration**.

Three local states related to files being managed by Git: 1) working directory, 2) staging area, and 3) Git repository or the commit history.
The **working directory** is the directory or folder on your computer that holds all the project or application files. Files within the working directory, may or may not be managed by Git.
Git Repository : Normally, within the working directory is a hidden folder called the ".git" folder that contains the actual git repository. The Git repository manages the Git commit history, that is all the changes that are finalized and permanently part of the Git repository.
Staging area : In-between(working directory and Git repository) is the Git staging area, often referred to as the Git index, that is a holding area for queueing up changes for the next commit. Since files in the staging area haven't been committed yet, you can move the files in and out of the staging area without impacting the Git repository, and its history of changes.

Remote Repository(ex. Github)

Master Branch : branches work like they do in other source control systems; they are timelines that contain your changes. In Git, branches contain commits; when we start off, Git provides us with a default branch named master

Git installation

For windows : http://git-for-windows.github.io
For mac : in terminal use “git version”

For windows : Use git bash for git
For Mac : Use terminal for git

Terminal Command

To see more about command https://ss64.com/osx/
pwd : print working directory
mkdir : to create directories or folders
cd : changing directory
ls : lists directory contents of files and directories.
ls -l : lists directory contents of files and directories. With more info
ls -a : lists directory contents of files and directories, These hidden files all have dots (.) in front of their names
rm : Delete files and folders
unzip zip-file-path : Unzip given file to directory selected on terminal
mv: Move or rename files or directories

Basics

Set global git username : git config --global user.name “bhavesh”
Set global git email : git config --global user.email “bhavesh@gmail.com”
View config setting (username, email etc) : git config --global --list
Another option to see/edit/delete stored config setting, You have to look into git config file, use “mate ~/.gitconfig” or “git config --global -e” to see git config if TextMate is set as default git editor. (described below)
Clone remote repository : git clone “http-clone-url”
Status of repository : git status : this command tells which branch you are on, and uncommitted changes of repository and tell to pull/push from remote if you are behind/ahead . If not, it tells branch is up to date. (We use the "git status" command to see if there are any changes between the working directory, the staging area, our local repository, and our remote repository.)
Uncommitted changes/after changes => those is in working repository
Add changes into staging area / pre-commit : git add “file path”
Add changes into local Git repository : git commit -m “commit-msg”
Modify last commit msg : git commit --amend
Push changes of local Git repository to remote repository :
git push remote-name branch-name
Ex : git push origin master

How can I set up an editor to work with Git on Windows?

See video 16 & 17
https://stackoverflow.com/questions/10564/how-can-i-set-up-an-editor-to-work-with-git-on-windows

TextMate as editor for git

Go to TextMate >> Preferences >> Terminal >> Install - shell support
Now type “mate” in terminal TextMate will be opened.
Configure TextMate with Git
git config --global core.editor "mate -w"
To test/see config setting open in TextMate
After set above use
git config --global -e

Git basics

1) **Start with a fresh project (git init)**

git init new-project-folder-name : Initialized empty Git repository in “new-project-folder-name” folder
After init .git folder is created. Change directory of terminal to .git folder and command “ls”
You will see, following folders:

HEAD	config	hooks	objects
branches	description	info	refs

Use “git status” to see status of working repository’s current branch with staging area, git repository(local) and remote repository.

Use “git add file-path-wish-to-add to add new/existed-changed file in git index or staging area. (precommit)

Use "git rm --cached <file>..." to unstage

On git commit (without extra parameter) git will invoke default editor(Text mate) for commit message. After save and close editor with commit message (cmd+s and cmd+w). Then commit will be happened with msg entered in editor.
After every commit, you can see **sha1 unique key** for each commit. Eg [master **7f5c162**]

2) Adding Git to an Existing Project (git init)

Use git init command in the existing project
Add all files and folder of working directory in git staging area use “git add .” (dot)
Commit msg incline use “git commit -m ‘commit-msg’”

3) Starting on GitHub by Joining an Existing Project (git clone)
To clone use “git clone “hhtp-clone-url””
git push remote-repository-name branch-name : command is used to push local git directory changes to remote remote git repository

Before make any push, make sure local git is up to date with remote repository (Best practise, always follow), To that pull remote code using **git pull remote-repository-name(remote-reference) branch-name** command.

git commit -am “commit-msg” (git add and git commit -m in one line -a for add and -m commit msg)

git ls-files : Gives list of files that git is tracking.

git add “file path” it adds changes(not entire file) to git staging. (So one file’s changes can be in working directory or staging area)

Add all files recursively by using **git add . (Dot)** add all file recursively from this folder onward.

Backing out commited changes / from staging area to working directory : git reset HEAD “file-path”

Discard changes from working directory : git checkout -- “file-path”

git mv old-file-name/path new-file-name/path : Rename/move git file name/path. If we don't use “git mv” to rename/move file but we rename/move directly using finder or “mv” command. Git shows two changes one the file is deleted or one new file is added, that time use “git add -A” command (comment adds file deleted, renamed or moved in staging area.) or “git add -u” after git add file-name/path. https://git-scm.com/docs/git-add

Delete file tracked by git : git rm file/name-path : git delete file and stage deleting changes. You can unstage by “git reset HEAD file-name/path” and “git checkout -- file-path” and you can get your file by before commiting.
If you deleted file using “rm” or from file finder so use “git add -A” or git add -u” after git add file-name/path to tell git about deleted file.

Git manual : use “git help command-key“ ie git help add

“git log” gives commit history in reverse order

git log --abbrev-commit : shorten the commit hash

git log --oneline --graph --decorate :

git log f848f76...8832388 : specify commit range
git log --since=”3 days ago” : brings 3 days ago commits

git log -- file-name : commits on specific file

git show commit-hash-key : display commit details and changes for that commit.

Git alias
git config --global alias.your-alias-command "git-command-that-want-to-be-alias"

Ex : git config --global alias.**hist "log --all --graph --decorate --oneline"**
Now use “git hist” gives response “git log --all --graph --decorate --oneline”

You can modify alias after create, To see stored alias You have to look into git config file, use “mate ~/.gitconfig” or “git config --global -e” to see git config if TextMate is set as default git editor.

Git Ignore(unwanted file)
To create gitignore file use “mate .gitignore”, git ignores the files that is mentioned in .gitignore.
Git ignore pattern expression example:
Specific file : Myfile.ext
File pattern : *.ext
Folder : my-folder/

.gitignore file is text file like any other file, and it needs to added in git.
git add .gitignore and git commit -m “added git ignore”

Visual Diff / Merge Tool

P4Merge for window/Macox (But we use SourceTree app)

P4Merge for windows/mac : www.perforce.com or
https://www.perforce.com/downloads/visual-merge-tool

Setup P4Merge in only windows by : Control panel >> Advanced system settings >> Environment variable >> system variable >> path (ADD P4Merge application path)

P4Merge for window/mac git configuration : git config --global merge.tool p4merge
git config --global mergetool.p4merge.path “p4merge-application-path”
git config --global mergetool.prompt false
git config --global diff.tool p4merge
git config --global difftool.p4merge.path “p4merge-application-path”
git config --global difftool.prompt false

Git Comparison

git diff : Shows changes between working directory and staging area (What currently is staged)
git difftool : Shows changes between working directory and staging area in difftool if it is set.

git diff HEAD : Shows changes between working directory and local git repository (Last local commit)
git difftool HEAD : same as git diff HEAD but open in difftool

git diff --staged HEAD : Shows changes between Staging area and local git repository (Last local commit)
git difftool --staged HEAD : same as git diff --staged HEAD but open in difftool

Note : HEAD : local git last commit u can say

git diff -- <file-name> : git diff shows difference between multiple files if multiple file has been changed but u can specify file name to see changes in one file.
And same trick git difftool -- <file-name>

git diff 8832388 HEAD 148c66d : Difference b/w git repo commit(8832388) to git repo at HEAD in git local repository

git diff HEAD HEAD^ : Difference b/w git repo and git repo at 1 commit before HEAD in git local repository

git diff 8832388 148c66d : Difference b/w two commits in git local repository

git diff master origin/master : Comparing local master branch vs remote master branch

git diff <branch-name> <branch-name> : shows difference between two branches

Branching

Beast practice is we should create feature/Topic branches and merge back them in master branches.
git branch : gives list of all local branches

git branch -a : gives list of all branches and -a give both local and remote branches. In output * means current active branch.

git branch <new-branch-name> : create local branch of name mentioned

git checkout <branch-name> : Switch to given branch.

git branch -m <existing-branch-name> <new-branch-name> : Change branch name.

git branch -d <branch-name> : delete the branch (can not delete current checkouted branch)

git checkout -b <new-branch-name> : Create branch before checking out into the branch

Merging

git merge <branch-name> : branch-name is name of source branch that need to merge into current local branch (Fast forword merge) (Doesn't preserve as separate branch)

git merge <branch-name> --no-ff : Merge but disable fast forword capability. It show merge commit in log.

git merge <branch-name> -m "Commit-msg" : (preserve as separate branch in log)

Above three merge work fines if we don't have conflicts in branches.

But what if you have conflicts:
git merge <branch-name> gives error similar like below
"CONFLICT (content): Merge conflict in simple.html
Automatic merge failed; fix conflicts and then commit the result."
Action to take:

1) See differences and identify conflicts : git diff <branch-name> <branch-name> // or difftool
2) Merge with conflicts : git merge <branch-name>

<<<<<<< HEAD : changes from local last changes (Target)
>>>>>>> Branch-name : changes from local source branch changes (source)

3) git mergetool : and do manually merge using merge tool like p4merge
4) git commit -m “commit-msg”

Note : conflict-file.**orig** file is created to save your file before merge. You should add it in gitignore : *.orig

Rebase

In Git, there are two main ways to integrate changes from one branch into another: the merge and the rebase.

Take the patch of the commit change of current branch and apply after destination branch’s commit changes.

Use when current branches changes you want to ahead of destination branch.

git rebase <branch-name>

When you gets conflicts while rebasing
Abort rebase **:** git rebase --abort : abort/cancel rebase

You have to manually resolve conflicts
git rebase <branch-name>
git mergetool
Solve conflicts manually then
git rebase --continue

Rebase local branch with remote branch
git fetch origin master : Download objects and refs from another repository
git pull --rebase origin master

Stashing

Use git stash when you want to record the current state of the working directory and the index, but want to go back to a clean working directory. The command saves your local modifications away and reverts the working directory to match the HEAD commit.

1) git stash : In Work In Progress (WIP. Uncommitted changes) you don't want to commit but you want to go back to your local git HEAD commit. You can use git stash. After

apply this your uncommitted(WIP) changes will be saved (in shash list). But stash only tracked file (Which added in git not new created file)
2) git stash list : Shows all WIP (Uncommited work but saved in stash) on the branch
3) git stash apply : If you want your WIP changes you have to run this command using that your working directory gets all uncommitted and stashed changes back in uncommitted state
4) git stash drop : after you done with stash you can remove stash.

Stash untracked files, You can do one thing out of two.

1) If you sure you going to add/track the untracked file later, You can add it to git staging area (make it as tracked file) by using git add.
2) If you are not sure you want to add this file in git but want to stash it , Then use “git stash -u”.

git stash pop : Combine => git stash apply + git stash drop

Multiple stash:
git stash save "save-msg" : Stash with stash message , Last stash is index zero (First in - last out)

git stash list : shows all saved stashes.

git stash apply stash@{index} : where index is reference/number of the stash you want to apply. Use reference if you multiple stash.

git stash drop stash@{index} : remove particular stash, given in stash index

git stash clear : Empty all stash list.

Stash into branch
If you want all non working directory changes in new branches :
git stash branch <branch-name> : First Switch to new branch, second stash all changes, Third dropped stash

Tagging

git tag <tag-name> : (Lightweight tag) create tag : It's simply a marker on particular commit.

git tag --list : To see list of tags

git show <tag-name> : shows commit that add that tag

git tag --delete <tag-name> : Delete tag

Annotated tag: is similar to Lightweight tag expect it has little extra informations.
git tag -a <tag-name> : Create tag and opens to editor (mate) to write tag info.
git show <tag-name> for annotated tag you can see more information than lightweight tag : Tag info, Author name of tag, Date etc.
Or can use **git tag <tag-name> -m "Tag-Info"**

To compare two tags : git diff <tag1-name> <tag2-name> OR git difftool <tag1-name> <tag2-name>

Tagging specific (previous) commit : use commit sha key at last like this
git tag -a <tag-name> <Commit-SHA>

Updating an Existing Tag Location : If you want to edit tag location from one location to another,
There is couple of approaches :

1) Delete tag and create new tag on corresponding commit location
2) Force update : git tag -a <existing-tag-name> -f <new-commit-SHA>
   Note: if you don't provide <new-commit-SHA> tag update to last/ recent commit location.

Push Tag in Github : git push origin <local-tag-list>

Push all tags on github/server : git push origin <branch-name> --tags
i.e git push origin master --tags

Remove/Delete tag from remote : git push origin :<tag-name>
i.e. git push origin :v-0.9-beta : the tag will delete from remote repository but still have in local repository.

Reflog

Reflogs tells what we done in last 60 days. Using git reflog you can see all git actions, Example to understand : Suppose you reset commit. That commit is disappeared from git log but you can back-up, To backup you will need reset commited SHA which you will get from "git reflog"

Cherry Pick

Pick/Apply any particular commit of another branch to selected branch without merging that two branches. (Note : another branch can be same as selected branch)

git cherry-pick <commit-SHA> : Given one or more existing commits, apply the change each one introduces, recording a new commit for each.

Git flow