Version Control System: Git



(Audience Handbook)

Index

* [VCS](#_6g5hjhnorwtr)
* [Why Git](#_dus7yx3d9z7w)
  + [Distributed Vs Centralized VCS](#_e8gh8eg7z3po)
* [Typical Workflow](#_bk0p78dyl7uf)
* [Installation](#_4a2e2pmm50gt)
* [Repository Creation](#_z61em029b9dp)
* [Clone Existing Repos](#_f2fwed26fccw)
* [Add/Remove Changes](#_nte15na9z0a8)
* [Get Status](#_ppwijckrg2ub)
* [Commit Changes](#_4p7u58c5dvq6)
* [Branching](#_3nrl8iuz5qv3)
* [Merging](#_9dpuqgswrq7p)
* [Git Remote](#_qdz5yraeci1h)
* [Push](#_eid6xtskmjv5)
* [Fetch](#_20tsjgkb57yt)
* [Pull](#_69q68ptms2il)
* [Git Advance](#_aligm03wejbg)
  + [Git Diff](#_jdqvtbsm42z3)
  + [Git Log](#_y132mae0r7as)
  + [Git Stash](#_lu4uf97jqwak)
  + [Git Reset](#_n1hupj2uevbk)
  + [Git Rebase](#_hnenqlwj662u)
* [Git Fork](#_hs7suvf1tu1a)
* [Git Pull Request](#_28t39jl9asia)

## 

## 

## 

## 

## VCS

Version Control System are the tools that help a software team manage changes to source code over time. Version control software keeps track of every modification to the code in a special kind of database. Developing software without using version control is risky, like not having backups. Version control can also enable developers to move faster and it allows software teams to preserve efficiency and agility as the team scales to include more developers.

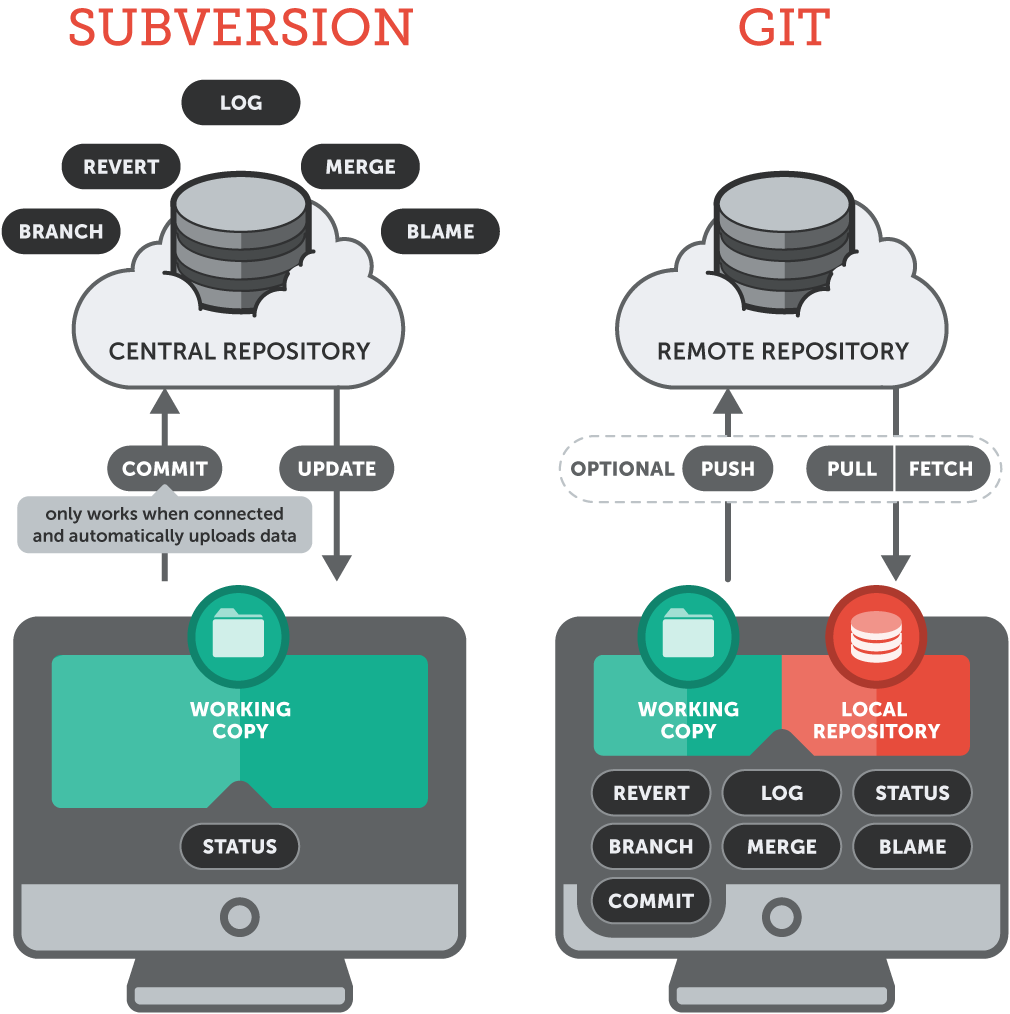
Good version control software supports a developer's preferred workflow without imposing one particular way of working. Ideally it also works on any platform, rather than dictate what operating system or tool chain developers must use.

## Why Git

Git is a version control system that offers the familiar capabilities of CVS or Subversion. Git offer almost all of its features for use offline and without a central server.

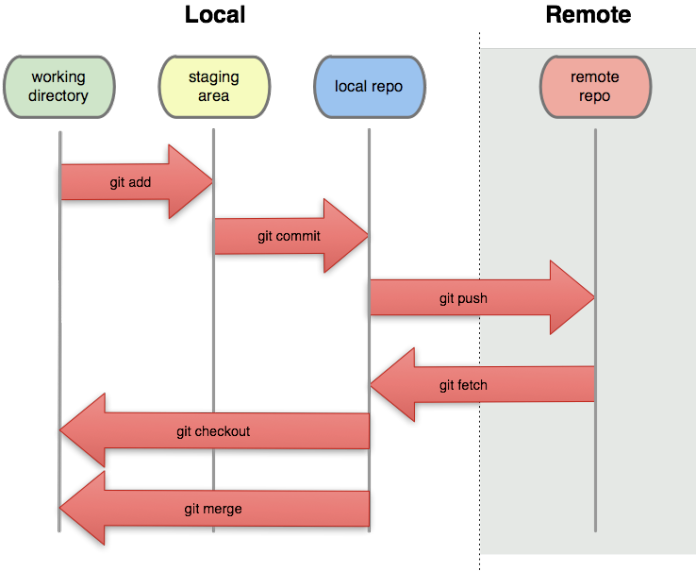
### Distributed Vs Centralized VCS

If you are familiar with centralized version control systems like Subversion, there will be some adjustments to make in your migration to Git. The first is that there is no central server. The full history of the repository lives on every user’s machine that has cloned (checked out) a copy of the repository. This is the essence of a Distributed Version Control System (DVCS).



## Typical Workflow

Below is the typical workflow for git. Here you can figure out that git provides most of its functionality on the local machine of user. Only the some specific task of data transfer and sharing are done with remote system.



## Installation

For most platforms, you can simple copy the binaries to a folder that is on the executable search $PATH. Git is primarily written in C, which means there is a unique distribution for each supported platform.

For Linux(Debian based) you can install it via “apt” package manager.

$ apt-get install git

## Repository Creation

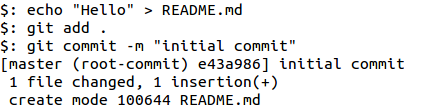
Now git has been installed so you can try play with this. First try to create a repository. Follow below command to create a directory and then initialize this as git repo.

$ git init



$ git add .

$ git commit –m ’initial commit‘



## Clone Existing Repos

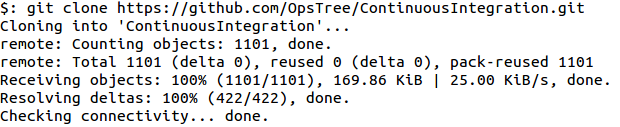
An equally common use case for Git is starting from someone else’s repository history. This is similar to the checkout concept in Subversion or other centralized version control systems.

The syntax to pull down a local copy of an existing repo is:

$ git clone https://github.com/OpsTree/ContinuousIntegration.git

or

$ git clone [git@github.com](mailto:git@github.com):OpsTree/ContinuousIntegration.git



The protocol difference often signifies whether you have read-only or writeable access to the origin repository. The final syntax, which accesses an SSH exposed repository, is the most common write-enabled protocol.

## Add/Remove Changes

When you are ready to put files into the next commit, they must be first staged with the add command.

$ git add <file name, folder name, or wildcard>

$ git add .

$ git add \*.java

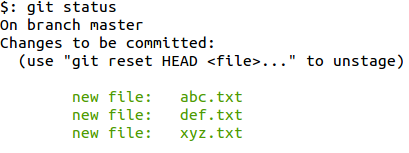


Specifying a folder name as the target of a git add recursively stages files in any subdirectories.

## Get Status

To check the current status of a project’s local directories and files (modified, new, deleted, or untracked) invoke the status command:

$ git status

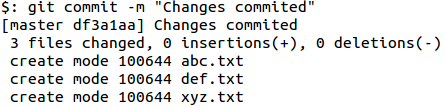


## Commit Changes

Once all desired blobs are staged, a commit command transactionally saves the pending additions to the local repository.

$ git commit

$ git commit –m”<your commit message>”



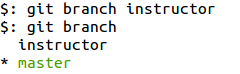
## Branching

Branching superficially appears much the same as it does in other version control systems, but the difference lies in the fact that Git branches can be targeted to exist only locally, or be shared with (pushed to) the rest of the team. The concept of inexpensive local branches increases the frequency in which developers use branching, opening it up to use for quick private experiments that may be discarded if unsuccessful, or merged onto a well-known branch if successful.



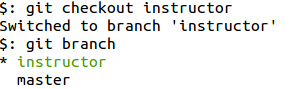
$ git branch <new branch name> <from branch>

$ git branch <new branch name>



Checking out (switching to) a branch is as simple as providing its name:

$ git checkout <branch name>



Can create a new branch from a branch and switch into this using a single command.

$ git checkout -b <new branch name> <from branch>

To list the complete set of current local and remote branches known to Git:

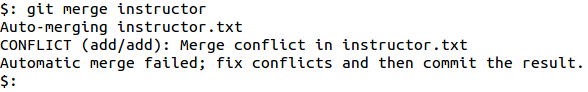
$ git branch -a

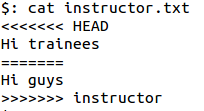
## Merging

Like other popular VCSs, Git allows you to merge one or more branches into the current branch.

$ git merge <branch one>

$ git merge <branch one> <branch two>





If any conflicts are encountered, which is rare with Git, a notification message is displayed and the files are internally marked with >>>>>>>>> and <<<<<<<< around the conflicting portion of the file contents. Once manually resolved, git-add the resolved file, then commit in the usual manner.

## Git Remote

While full paths to other repositories can be specified as a source or destination with the majority of Git commands, this quickly becomes unwieldy and a shorthand solution is called for. In Git-speak, these bookmarks of other repository locations are called **remotes**.

A remote called **origin** is automatically created if you cloned a remote repository. The full address of that remote can be viewed with:

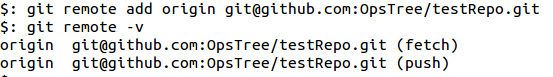
$ git remote -v

## 

To add a new remote.

$ git remote add <remote name> <remote address>

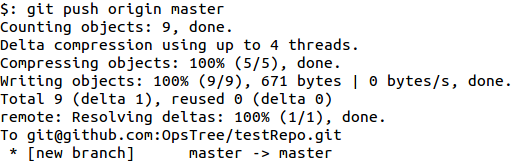
$ git remote add <remote name> [git@github.com](mailto:git@github.com):matthewmccullough/ts.git



## Push

Pushing with Git is the sending local changes to a colleague or remote repository with sufficiently open permissions as to allow you to write to it.

$ git push <remote name> <branch name>

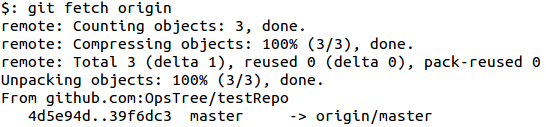


## Fetch

To retrieve remote changes without merging them into your local branches, simply fetch the blobs. This invisibly stores all retrieved objects locally in your .git directory at the top of your project structure, but waits for further explicit instructions for a source and destination of the merge.

$ git fetch <remote name>

$ git merge <remote name/remote branch>



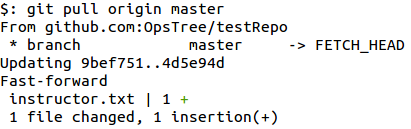
## Pull

Pulling is the combination of a fetch and a merge as per the previous section all in one seamless action.

$ git pull

$ git pull <remote name>

$ git pull <remote name> <branch name>



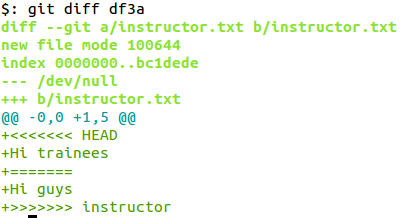
## Advanced git operations

### Git Diff

A patch-style view of the difference between the currently edited and committed files, or any two points in the past can easily be summoned.’

$ git diff

$ git diff 32d4..

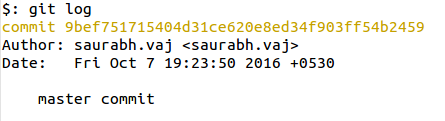


git diff: everything unstaged diffed to the last commit

### Git Log

The full list of changes since the beginning of time, or optionally, since a certain date is right at your fingertips, even when disconnected from all networks:

$ git log



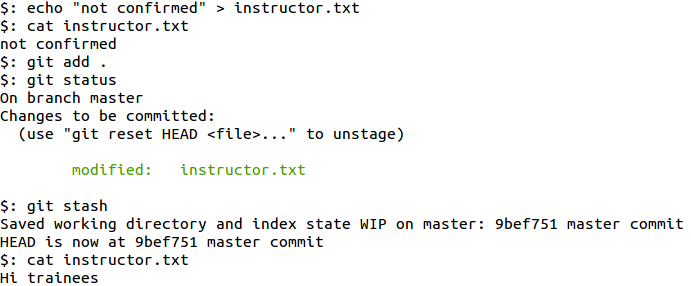
$ git log --since=yesterday

$ git log --since=2weeks

### Git Stash

Git offers a useful feature for those times when your changes are in an incomplete state, you aren’t ready to commit them, and you need to temporarily return to the last committed.This feature is named “stash” and pushes all your uncommitted changes onto a stack.

$ git stash



### Git Reset

If you want to abort your current uncommitted changes and restore the working copy to the last committed state.

$ git reset --hard

Resetting with the hard option recursively discards all of your currently uncommitted (unstaged or staged) changes.

### Git Rebase

Rebasing is the rewinding of existing commits on a branch with the intent of moving the “branch start point” forward, then replaying the rewound commits. This allows developers to test their branch changes safely in isolation on their private branch just as if they were made on top of the mainline code, including any recent mainline bug fixes.

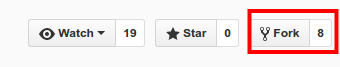
$ git rebase <source branch name>

$ git rebase <source branch name> <destination branch name>

## Git Fork

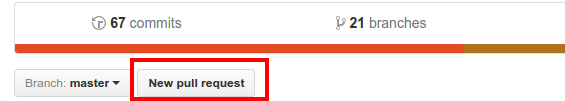
A fork is a copy of a repository. Forking a repository allows you to freely experiment with changes without affecting the original project.

Most commonly, forks are used to either propose changes to someone else's project or to use someone else's project as a starting point for your own idea.



## Git Pull Request

Pull requests let you tell others about changes you've pushed to a repository on GitHub. Once a pull request is opened, you can discuss and review the potential changes with collaborators and add follow-up commits before the changes are merged into the repository.



## Assessment

1. Clone a remote repo and perform branching and merging.
2. Create a bare git repo and implement distributed version control system.