A version control system is a piece of software that helps the developers on a software team work together and also archives a complete history of their work.

There are three basic goals of a version control system (VCS):

1. We want people to be able to work simultaneously, not serially.
2. When people are working at the same time, we want their changes to not conflict with each other.
3. We want to archive every version of everything that has ever existed — ever.

A filesystem is two-dimensional: Its space is defined by directories and files. In contrast, a repository is threedimensional: It exists in a continuum defined by directories, files, and time. A version control repository contains every version of your source code that has ever existed.

A consequence of this idea is that nothing is ever really destroyed. Every time you make some kind of change to your repository, even if that change is to delete something, the repository gets larger because the history is longer.

Each change adds to the history of the repository. We never subtract anything from that history.

The essential difference between a Centralized Version Control System (CVCS) and a DVCS is the notion of a *repository instance*.

In a CVCS, the repository exists in one place on a central server. Every piece of software that is used to access the repository includes a network client.

Most operations interact with a local repository instance, not a network server. The only time networking code gets involved is when the repository instances are being synchronized. Every developer has his own private repository instance.

In practice, virtually all DVCS teams have a central server. With a CVCS, a central server happens because it is inherent in the centralized model. With a DVCS, a central server happens because of the team’s decision to have one.

**Git Cheat Sheet**

git diff Show unstaged changes between your index and working directory.

**git config --global –edit** Open the global configuration file in a text editor for manual editing.

git diff HEAD Show difference between working directory and last commit.

git diff –cached Show difference between staged changes and last commit

**git log --grep=”<pattern>”** Search for commits with a commit message that matches <pattern>.

|  |  |
| --- | --- |
| Create | |
| $ git init | Fom existing data |
| $ git clone | From existing repository |

|  |  |  |
| --- | --- | --- |
| Show | | |
| $ git status | | Files changed in working directory |
| diff | $ git diff | Changes made to tracked files |
| $ git diff --staged | Shows file differences staging and the last revision |
| $ git diff C1 C2 | What changed between C1 and C2 |
| log | $ git log | History of changes |
| $ git log –p File | History of changes for file with diffs |
| $ git log –S “Foo” | Show commits that make add or remove a certain string |
| $ git log -- grep=’day of week’ | Search commits that contain a log message: |
| $ git blame File | | Who changed what and when in a file |
| $ git show C1 | | A commit identified by ID |
| $ git show C1:File | | A specific file from a specific commit |
| $ git branch | | All local branches |
|  | |  |
|  | |  |

$ git log

Lists version history for the current branch

$ git log --follow [file]

Lists version history for a file, including renames

$ git diff [first-branch]...[second-branch]

Shows content differences between two branches

$ git show [commit]

Outputs metadata and content changes of the specified commit

Revert



$ git reset [file]

Unstages the file but preserve its contents

$ git reset [commit]

Undoes all commits after [commit], preserving changes locally

$ git reset --hard [commit]

Discards all history and changes back to the specified commit

Undo last commit:

git reset --hard HEAD~1

Squash last n commits into one commit:

git rebase -i HEAD~5

ou

git reset --soft HEAD~5  
git add .  
git commit -m "Update"  
git push -f origin master

Revert the previous commit:

git revert HEAD  
git commit

Revert the changes from previous 3 commits without making commit:

git revert --no-commit HEAD~3..

Checkout a tag:

git checkout tagname

git checkout -b newbranchname tagname

Checkout a branch:

git checkout destination\_branch

Use -m if there is merge conflict:

git checkout -m master // from feature branch to master

Checkout a commit:

git checkout commit\_hash

git checkout -b newbranchname HEAD~4

git checkout -b newbranchname commit\_hash

git checkout commit\_hash file

Checkout a file:

git checkout c5f567 -- Relative/Path/To/File

Get their changes during git rebase:

git checkout --ours foo/bar.java  
git add foo/bar.java

Get their changes during git merge:

git pull -X theirs

git checkout --theirs path/to/the/conflicted\_file.php

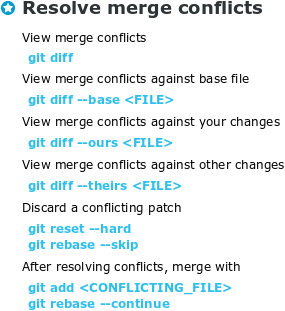
git checkout --theirs .  
git add .

git checkout branchA  
git merge -X theirs branchB

Merge commits from master into feature branch:

git checkout feature1  
git merge --no-ff master

Resolve merge conflicts



$ git rm [file]

Deletes the file from the working directory and stages the deletion

$ git rm --cached [file]

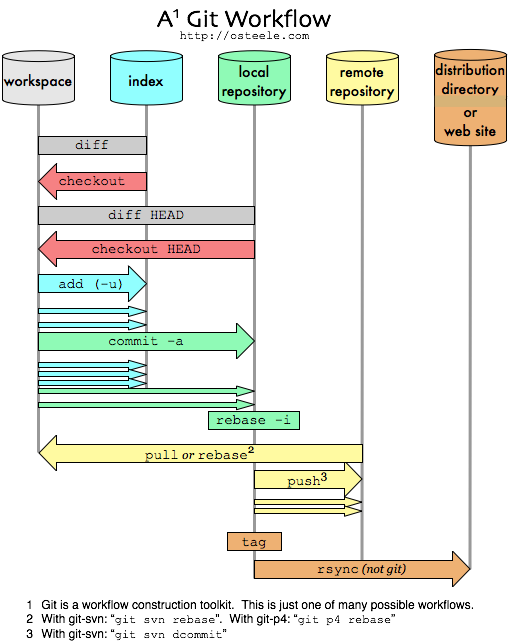
Removes the file from version control but preserves the file locally

$ git mv [file-original] [file-renamed]

Changes the file name and prepares it for commit



“git diff” tells me what I’ve changed since the last checkpoint; “git diff head” shows what’s changed since the last commit. “git checkout .” reverts to the last checkpoint; “git checkout head .” reverts to the last commit. And “git stash” and “git checkout -m -b” operate on the changes since the last commit, which is what I want.



**Setup**

**Show current configuration:**

$ git config --list

**Show repository configuration:**

$ git config --local --list

**Show global configuration:**

$ git config --global --list

**Show system configuration:**

$ git config --system --list

**Set a name that is identifiable for credit when review version history:**

$ git config --global user.name “[firstname lastname]”

**Set an email address that will be associated with each history marker:**

$ git config --global user.email “[valid-email]”

##### Set global editor for commit

$ git config --global core.editor vi

## Configuration Files

##### Repository specific configuration file [–local]:

<repo>/.git/config

##### User-specific configuration file [–global]:

~/.gitconfig

##### System-wide configuration file [–system]:

/etc/gitconfig

## Create

##### Clone an existing repository:

There are two ways:

Via HTTP

$ git clone http://domain.com/user/repo.git

##### Create a new local repository in the current directory:

$ git init

##### Create a new local repository in a specific directory:

$ git init <directory>

## Local Changes

##### Changes in working directory:

$ git status

##### Changes to tracked files:

$ git diff

##### See changes/difference of a specific file:

$ git diff <file>

##### Add all current changes to the next commit:

$ git add .

##### Add some changes in <file> to the next commit:

$ git add -p <file>

##### Commit all local changes in tracked files:

$ git commit -a

##### Commit previously staged changes:

$ git commit

##### Commit with message:

$ git commit -m 'message here'

##### Commit skipping the staging area and adding message:

$ git commit -am 'message here'

##### Change last commit:

Don’t amend published commits!

$ git commit -a --amend

##### Amend with last commit but use the previous commit log message

Don’t amend published commits!

$ git commit --amend --no-edit

##### Move uncommitted changes from current branch to some other branch:

$ git stash

$ git checkout branch2

$ git stash pop

##### Restore stashed changes back to current branch:

$ git stash apply

#### Restore particular stash back to current branch:

* {stash\_number} can be obtained from git stash list

$ git stash apply stash@{stash\_number}

##### Remove the last set of stashed changes:

$ git stash drop

## Search

##### A text search on all files in the directory:

$ git grep "Hello"

##### In any version of a text search:

$ git grep "Hello" v2.5

## Commit History

##### Show all commits, starting with newest (it’ll show the hash, author information, date of commit and title of the commit):

$ git log

##### Show all the commits(it’ll show just the commit hash and the commit message):

$ git log --oneline

##### Show all commits of a specific user:

$ git log --author="username"

##### Show changes over time for a specific file:

$ git log -p <file>

##### Display commits that are present only in remote/branch in right side

$ git log --oneline <origin/master>..<remote/master> --left-right

##### Who changed, what and when in <file>:

$ git blame <file>

##### Show Reference log:

$ git reflog show

##### Delete Reference log:

$ git reflog delete

## Branches & Tags

##### List all local branches:

$ git branch

#### List local/remote branches

$ git branch -a

##### List all remote branches:

$ git branch -r

##### Switch HEAD branch:

$ git checkout <branch>

##### Checkout single file from different branch

$ git checkout <branch> -- <filename>

##### Create and switch new branch:

$ git checkout -b <branch>

##### Create a new branch from an exiting branch and switch to new branch:

$ git checkout -b <new\_branch> <existing\_branch>

#### Checkout and create a new branch from existing commit

$ git checkout <commit-hash> -b <new\_branch\_name>

##### Create a new branch based on your current HEAD:

$ git branch <new-branch>

##### Create a new tracking branch based on a remote branch:

$ git branch --track <new-branch> <remote-branch>

##### Delete a local branch:

$ git branch -d <branch>

##### Rename current branch to new branch name

$ git branch -m <new\_branch\_name>

##### Force delete a local branch:

You will lose unmerged changes!

$ git branch -D <branch>

##### Mark the current commit with a tag:

$ git tag <tag-name>

##### Mark the current commit with a tag that includes a message:

$ git tag -a <tag-name>

## Update & Publish

##### List all current configured remotes:

$ git remote -v

##### Show information about a remote:

$ git remote show <remote>

##### Add new remote repository, named <remote>:

$ git remote add <remote> <url>

##### Download all changes from <remote>, but don’t integrate into HEAD:

$ git fetch <remote>

##### Download changes and directly merge/integrate into HEAD:

$ git remote pull <remote> <url>

##### Get all changes from HEAD to local repository:

$ git pull origin master

##### Get all changes from HEAD to local repository without a merge:

$ git pull --rebase <remote> <branch>

##### Publish local changes on a remote:

$ git push remote <remote> <branch>

##### Delete a branch on the remote:

$ git push <remote> :<branch> (since Git v1.5.0)

OR

$ git push <remote> --delete <branch> (since Git v1.7.0)

##### Publish your tags:

$ git push --tags

## Undo

##### Discard all local changes in your working directory:

$ git reset --hard HEAD

##### Get all the files out of the staging area(i.e. undo the last git add):

$ git reset HEAD

##### Discard local changes in a specific file:

$ git checkout HEAD <file>

##### Revert a commit (by producing a new commit with contrary changes):

$ git revert <commit>

##### Reset your HEAD pointer to a previous commit and discard all changes since then:

$ git reset --hard <commit>

##### Reset your HEAD pointer to a previous commit and preserve uncommitted local changes:

$ git reset --keep <commit>

