Git:
For version control system.

Git was created in 2005. Distributed version control., open source and free software. Compatible with Unix-like systems and also windows.

Distributed version control:
Maintain own repositories instead of working from a central repository. Track changes, not versions(different from CVS).

Imagine changes to document as sets A,B,C,D,E,F.
Repo: A,B,C,D,E,F
Repo: A,B,C,D
Repo: A,B,C,E
Not right or wrong.

No central server:
Faster;
No network access;
No single failure point;
All repos are considered equally.

Who use Git?
Track edits;
Share changes;
Not afraid of command-line tools;

Not useful for tracking non-text files.

Download Address:
http://git-scm.com/downloads

Command:
Which git: Find address
Git version: git –version

Configuration:
System Level: /etc/gitconfig
User Level: ~/.gitconfig (~:Home directory)
Project Level: my_project/.git/config
Command:
Git config --system
Git config –global

E.G.

```
git config --global user.name "alexliu0809@hotmail.com"
git config --global user.name "AlexLiu"
git config --global core.editor “notpad.exe” //set text
editor
git config --global color.ui true

Git auto-completion(not for windows):
Download from GitHub:
Cd ~
Curl –OL
https://github.com/git/git/raw/master/contrib/completion/
git-completion.bash
Rename file:
Mv ~/git-completion.bash ~/.git-completion.bash
Edit
~/.bash_profile or ~/.bashrc
```

Git initializing a repository.
Cd ~
Cd ./MyProject/
Git init
Ls –la //check if repository created successfully.

First Commit:
//make changes
//add
//commit

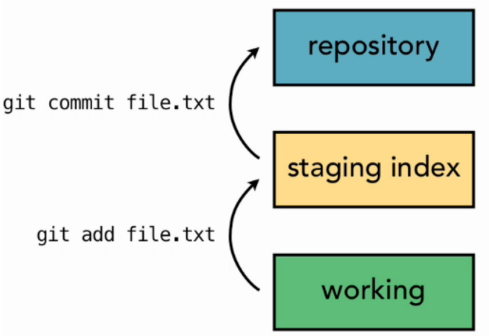
//make changes
Git add . //add everything
Git commit –m “initial commit”

Writing commit messages.
Provide enough information for updates.
“Add:xxx”
“Change:xxx”

See commit Messages:
Git log
Git log –grep=”Init” //message have “init”

Git concepts:
Three-tree architecture.

Repository
Staging Index
Working Space.



Make changes of 10 files. You could add 5 of them to stage, then commit. Five of them on repository and working space. Another 5 changes stay on the working space. Not committed.

Git Workflow
New file:
Add the file to working space.
Use add command, the file exist in staging index now.
Commit, the file is pushed to repository.

Edit:
Change in working space. (v2)
Add push v2 to staging index.
Commit v2, to repository.
The repository will save all the history versions, rather than the newest one.

Referring to Commits.
V1,V2,V3(could represent many files, not only single file).
Generates a checksum for each change set. (SHA-1 Hash Algorithm)
So check checksum(Commit ID in git log) if the data has changed or not.

Git uses HEAD to reference the commit. HEAD points to current branch in repository (points to parent of next commit). It's related to branches. HEAD always points to the newest commit or the check-out branch.

Making Changes to Files.

Use “git status” to check status. Untracked files(cannot tell changes).

ADD: //Add to staging index.
“Git add .” “Git add second_file.txt”

Commit Changes: //after change something
Use add to add to staging index.
Commit –m “Message”

See changes of Files.
“git diff” //compare the changes to every changed file in working spac with comparison to the newest ones in repository and staging index.
“git diff --staged” //stage means cached, compare staging and repo

Delete files. //first tracked in our repo
//commit first and delete, git status to track

1.Drag file to trash, and use “git rm first_file” + “git commit”
//delete first file. And this will be cached. Then commit
Notice: Cannot delete on the disk and then commit.

2.easier. “git rm file_to_delete” + “Commit –m “abc””
//not in the trash, vanished
//the op is staged. Then commit

Renaming. //deleted+new untracked
1. change name + “git add file_name” + “git rm file_name” +”commit”
//after these, git can find that it’s renamed.

2.”git mv file_name new_file_name” + “commit”
 //easier. Move=Rename

Using git with a real project
//Make a real project
//cd the folder
“git init”
“git log” //error
“git status”//All untracked
“git add .”//add everything
“git commit –m “Real project””

“git commit –am “Message”” //add and commit, only for edit, not for deleted files.

Undoing Changes.
Undoing working space changes.
“git status”
“git diff”
//“git checkout” //not recommended, used in branch
“git checkout --index.html” //don't change branch, get the file

Unstaging files.
“git reset HEAD text.html”

Undoing commits we’ve made.
The chain on the line of Git will have to change. But we could change the last commit. //add something to the previous commit
“git commit --amend --m “Message”” //made some changes, commit again, use amend the last commit

Retrieving files from old versions. (Reversion)
“git checkout SHAID --filename.txt” //get file into workspace and your staging index.
“get status” //put the file into staged and your work space.

Reverting A commit
“git revert SHAID”//get the old version, and commit it. Not for complicated situations.

Resets…To be seen

Remove untracked files:
“git clean” //-n,-f
“git clean -n”//tell the files that will be removed.
“git clean -f”//remove files not added into staging index.

Branches.
Try your new ideas in new branches.
Git will switch between branches and we don't have to worry about losing files.

Merge: Commit branch to Master in the last place.
//Create new branch, Head don't move. New commit to branch. Head moves to new commit.

See branch:
“git branch” //see all the branch

Create branch
“git branch my_branch” //create new branch, HEAD don't move. The new branch have all the old versions.

Switch to new branch
“git checkout branchname” //switch to new branch
“git commit –am “add to new branch”” //commit to new branch

Switch back
“git checkout master”//switch back to master.

Create and switch to a new branch
“git checkout –b branch2” //create branch off the current branch. You’re in master, branch from master, you’re from branch1, branch from branch1.

Switch between branches with unsaved files.
1. scrap the unsaved changes by using checkout file.
2. Commit so that it’s permanently saved
3. Stash the changes.
//if there’s no conflict, you can switch between branches. But it’ll have untracked files. Make it clean. Clean switch

Compare Branches:
“git diff branch1..branch2”
//compare brach1 and branch2

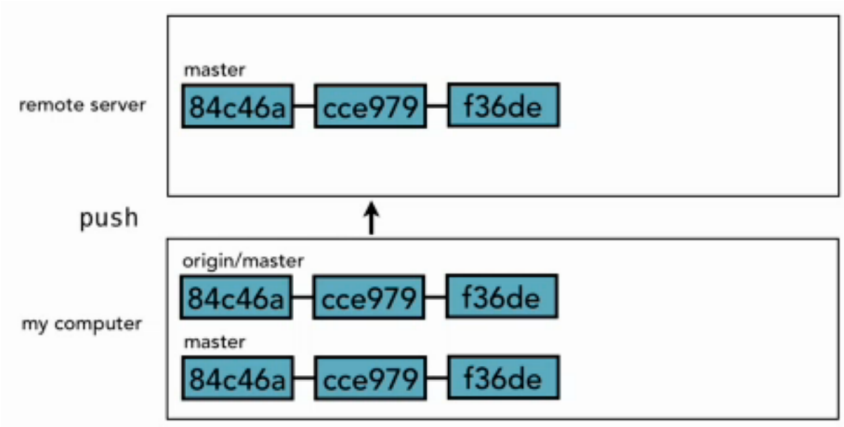
Rename branches:
“git branch –-move branch1 branch1newname” //rename

Delete branch:
“git branch branch_to_delete”

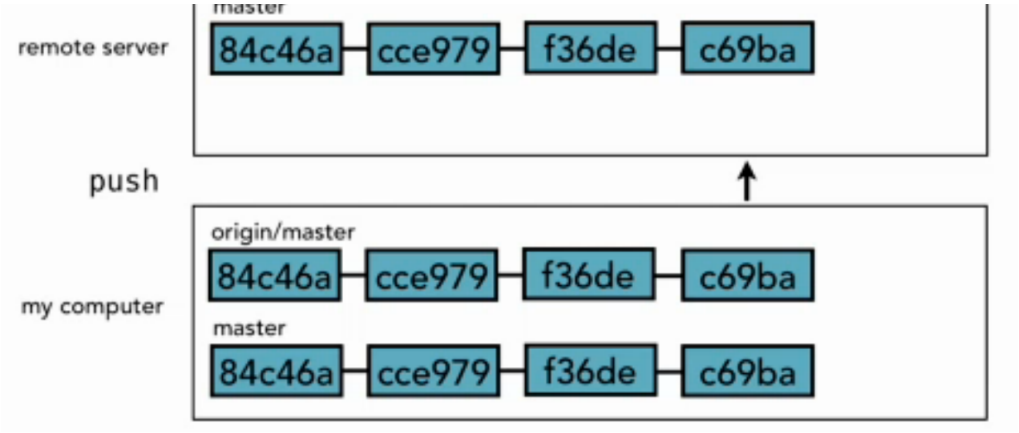
“git branch -d branch_to_delete”
//delete that branch. //Careful.
//Cannot delete the branch you’re on.
//Generally not able to delete the commits that are not merged.
//Or use -D to delete unmerged branch.

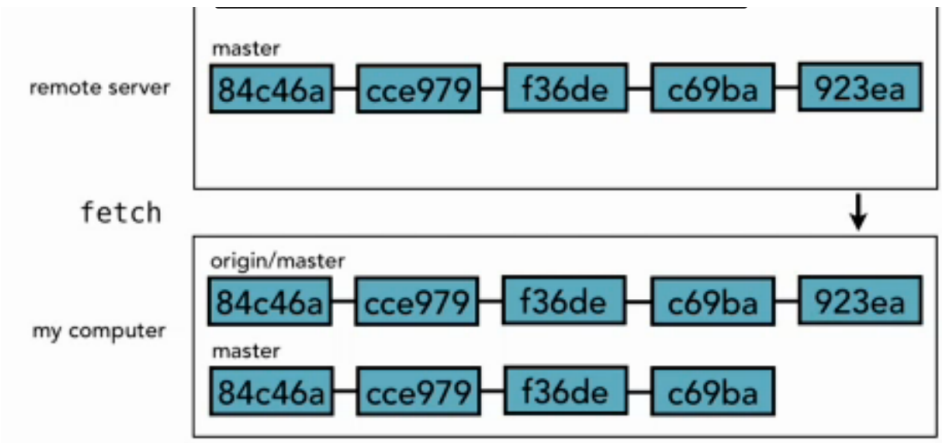
Merge branch:
“git checkout master” //to receive the changes.
“git merge branch_to_merge” //merge the branch to master.
//the changes are brought to receiving branch.
//The two branches are the same now (with not new commit)

GitHub Examples.
Big picture:

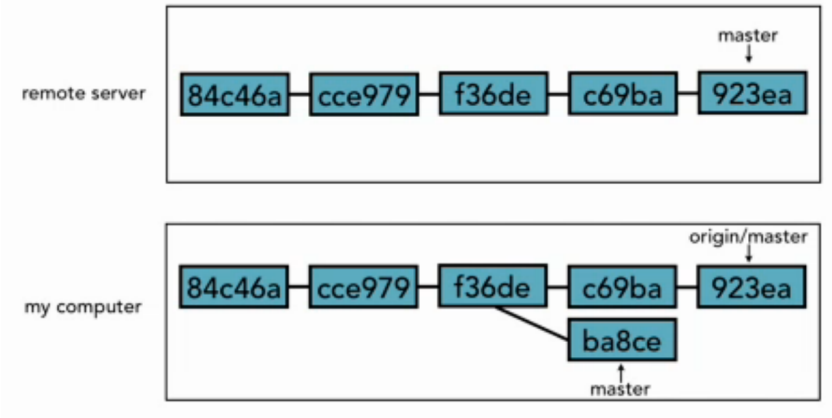


Origin is always synchronized with remote server.





//after fetch, in the origin, but not in our master. After you merge.
//Don't have two copies. Use Pointers.



//After fetch, I didn't push. need to be merged. After that we push, our changes will be pushed.

Generally, working with remote procedure: Commit locally, fetch, merge, push.

GitHub Account:
"git remote add <alias> <url>" //Connection with remote.
//origin(name for github remote) xxx.git. Origin synchronized with remote
"get remote -v" //url for fetching and pushing.
//"git remote rm origin" //remove origin remote

//Set up is done.
//Push
"git push -u origin master" //push master to github.
//password and username
//push our code online.
"git branch -r" //remote branch
"git branch -a" //both branches.

//Clone to local.
"git clone http://xxx.name.git newName" //create a name folder in the current folder and clone all the stuff from remote.
//still two repo, not one
//brought down master by default.

We push:
"git push origin master" //push master branch
Other people need to fetch: //don't update automatically.
"git fetch origin"

//Fetch from the remote.
Merge yours with origin:
“git checkout master”
“git merge branch_to_merge” //origin/master
//Push Again.
“git push origin master.”

Every time we try to upload our code, follow these steps:
1.Commit your source code to local depository:
“git add .”
//add all
“git commit –m “Message””
//commit
2.Fetch from remote repository:
“git fetch origin”
//fetch data to origin
3.Merge your master with origin/master
“git checkout master”
//switch to local master branch
“git merge origin/master”
//synchronize master with remote repository.
4.Upload your code
“git push origin master”
//push your code.

//Git pull= git fetch + git merge. //Not for freshers.
“git pull”
//Always fetch first.
//Fetch before you push.
//Fetch Often.