Git

* Keep track of changes
* Especially text changes
* Version 1, version 2, version 3
* Version control system
* Source code management (SCM)
* Distributed version control
* Open source and free software
* Compatible with Unix-like systems (Linux, Mac OS X, Solaris) and Windows
* Faster than other SCMs (100x in some cases)
* Better safeguards against data corruption

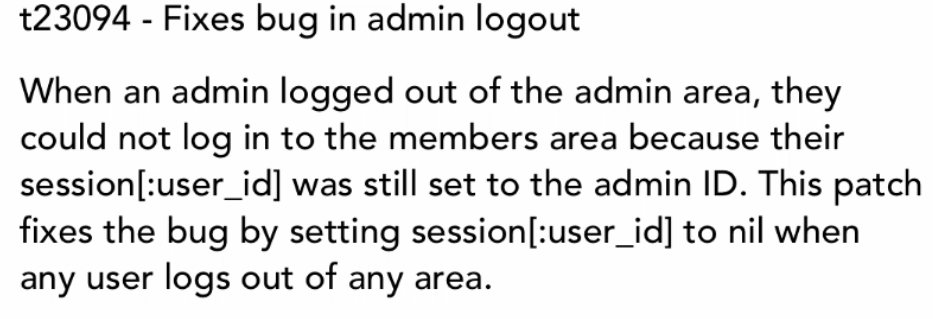
Distributed Version Control

* Different users (or teams of users) maintain their own repositories, instead of working from a central repository
* Changes are stored as “change sets” or “patches”
* Track changes, not versions
* Different from CVS and SVN, which track versions
* Change sets can be exchanged between repositories
* “merge in change sets” or “apply patches”
* No single master repository; just many working copies
* Each their own combination of change sets
* Imagine changes to a document as sets A,B,C,D,E,F
* Repo 1: A,B,C,D,E,F
* Repo 2: A,B,C,D
* Repo 3: A,B,C,E
* Repo 4: A,B,E,F
* No need to communicate with a central server
* Faster
* No network access required
* No single failure point
* Encourages participation and “forking” of projects
* Developers can work independently
* Submit change sets for inclusion or rejection
* 3 types of Configuration in Git
* System
* In this configuration changes are applicable for all users
* In Linux it will in /etc/gitconfig
* In windows it will in Program Files\Git\etc\gitconfig
* User
* User level configuration
* In Linux it will available in ~/.gitconfig
* In windows $Home\.gitconfig
* Project
* Configuration based on project to project
* my\_project/.git/config
* Auto Completion : i.e. means if we something write like help then we write just h and hit tab it will now complete that word for us
* Auto completion is already installed in windows in linux and mac it should installed.
* Git initialize means get everything ready to start tracking.
* Git commit means tell git to track the change.
* Tell git to add all changes that have been made to this entire project, everything that’s been

git add .

. is short for this directory

* Git commit means tell git to put it in permanent memory, to put in the repository.
* When work with git repository follow the following steps
* make changes
* add the changes
* commit changes to the repository with a message
* Writing commit message
* Commit message that describes the changes that we’re making in that commit set, so added file to project, that would be more descriptive saying that we added this first file.
* Commit message best practices
* Short single-line summary (less than 50 characters)
* Optionally followed by a blank line and a more complete description
* Keep each line to less than 72 characters
* Write commit messages in present tense, not past tense
* “fix bug” or “fixes bug”, not “fixed bug”
* Bullet points are usually asterisks or hypens
* Can add “ticket tracking numbers” from bugs or support requests
* Can develop shorthand for your organization
* “[css,js]”
* “bugfix:”
* “#38405 - ”
* The same set of conventions for their commit messages.
* Be clear and descriptive
* Bad: “Fix typo”
* Good: “Add missing > in project section of HTML”
* Bad: “Update login code”
* Good: “Change user authentication to use Blowfish”
* Bad: “Updates member report, we should discuss if this is right next week”
* Example of git commit



* Viewing the commit Log
* Return commit id, author, date & commit message
* Syntax: git log
* No. of git commit: git log –n 5
* Which will return first 5 git commits
* If we want to see commits of a specific date then type
* git log --since=2012-06-15
* Until a specific date

git log --until=2015-11-15

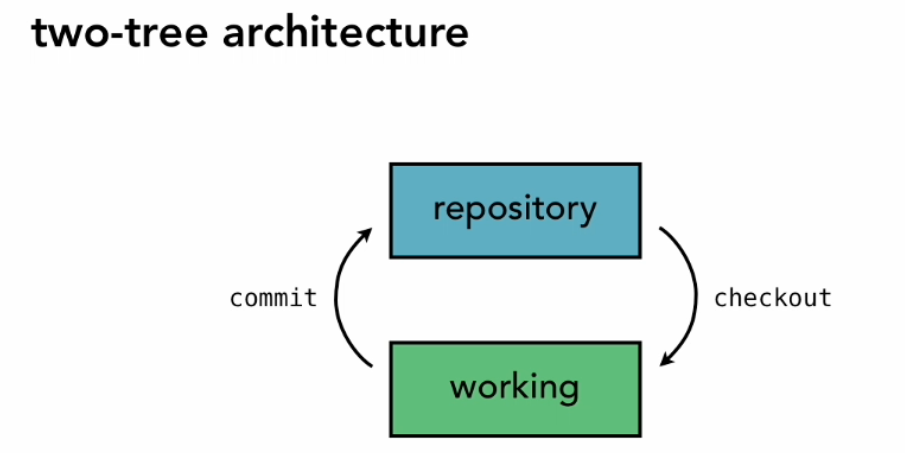
* To see log file using author

git log --author=”hasanmbstu13”

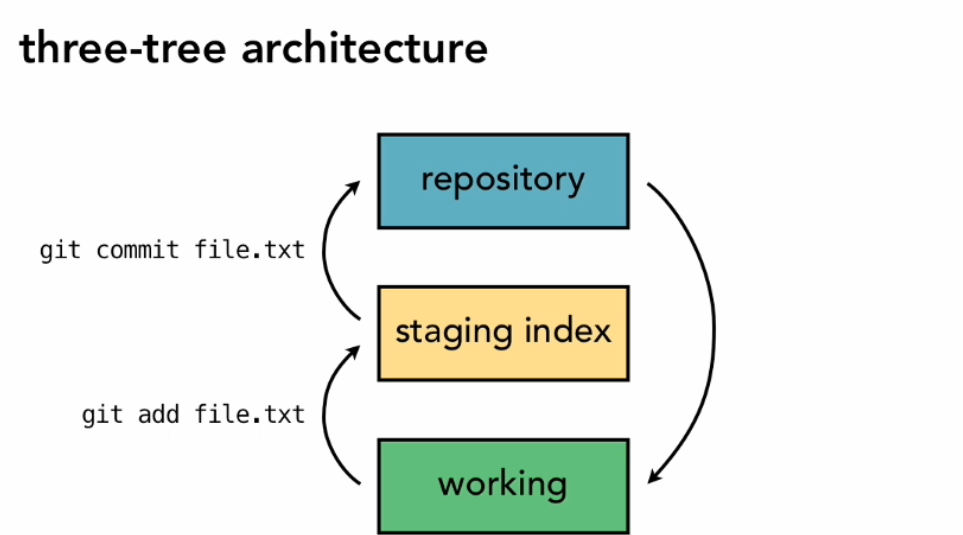
* The grep command allows you to search one file or multiple files for lines that contain a pattern.

git log --grep=”Init”

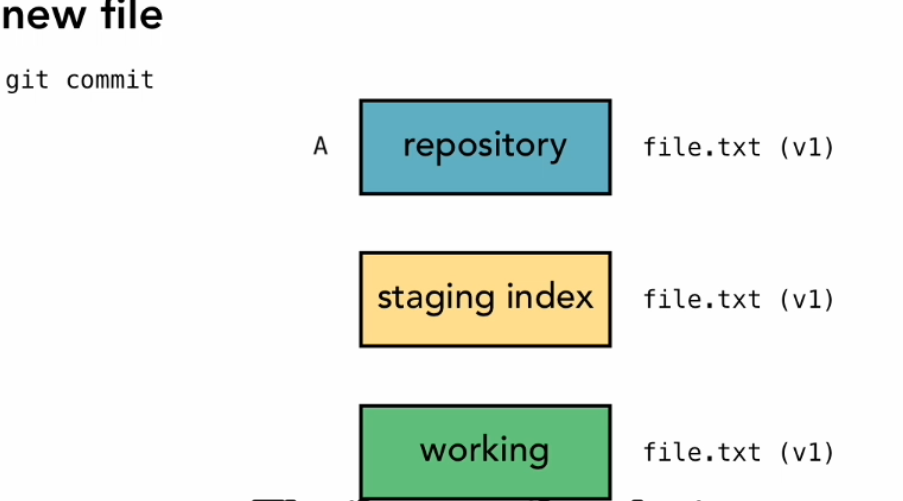
* Two-tree architecture



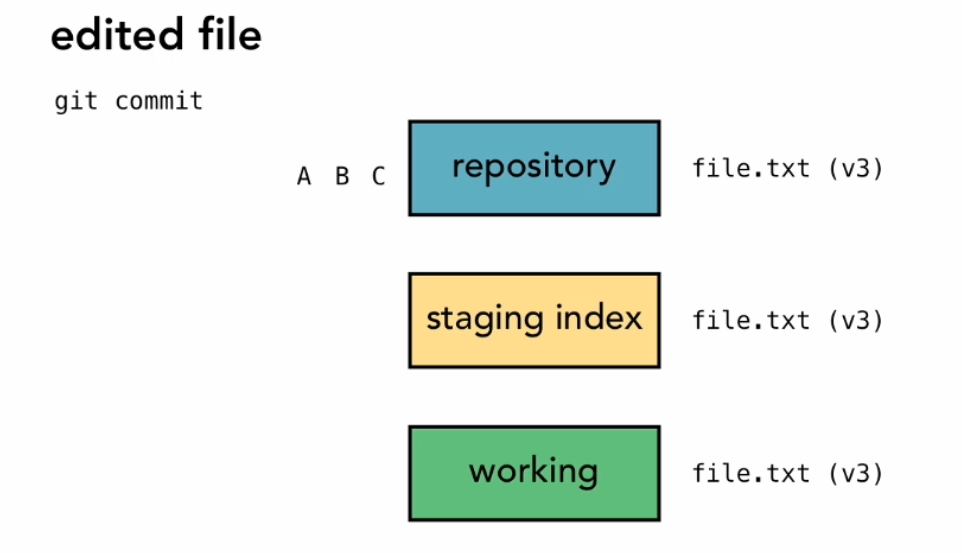
* If repository is shared repository and many people working from it, they may commit their changes to the repository. And if I haven’t checked out a copy recently to get those changes, then my working copy doesn’t have their changes. So once again the repository and the working tress will not have the same information in them.
* Three-tree architecture



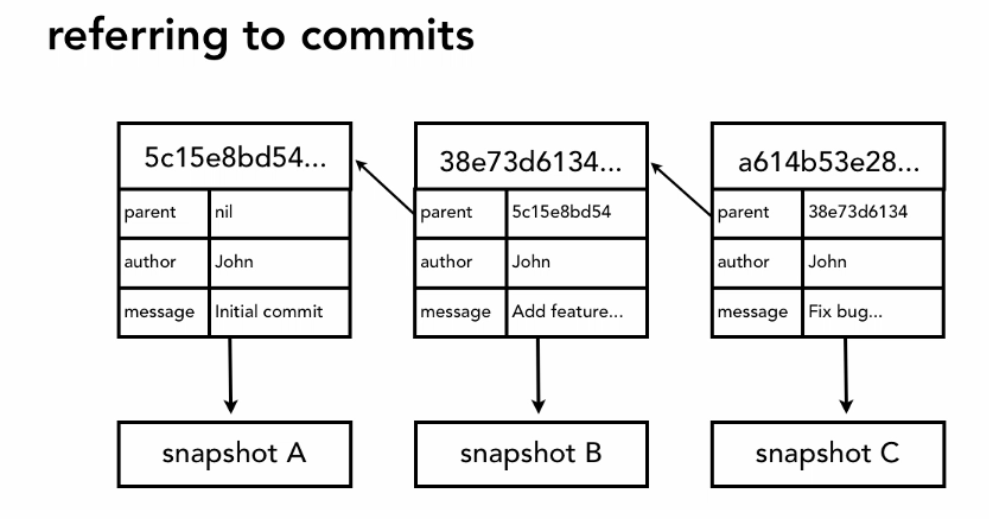
* Add file in git



* First work in the working repository or working directory
* Add file in the staging index
* After that commit the file push in the repository
* Edited File



* Referring to commits
* When we submit these changes to the repository at that point Git generates a checksum for each change set
* A checksum is a number that’s generated by taking data and feeding it into an algorithm
* Checksum algorithms convert data into a simple number
* Same data always equals same checksum
* Data integrity is fundamental
* Changing data would change checksum
* Git uses SHA-1 hash algorithm to create checksum
* 40-character hexadecimal string (0-9,a-f)
* Example: 0c280fd6b4dbcb57c8a42da86130953e5f532ce1
* Referring to commits Snapshots



* HEAD
* Is a reference variable
* We call this variable a pointer, because its purpose is to reference or point to, a specific commit in the repository as we make new commits the pointer is going to change or move to point to a new commit.
* Head always pointer to “tip” of current branch in repository
* Last state of repository, what was last checked out
* Points to parent of next commit
* Where writing commits takes place
* Head pointer points the place where we left off in our repository for the things that we’ve committed
* By default the branch that we’re working on master branch that’s our main branch
* Head always points to the tip of the currently checked out branch form the repository.
* Head is the tip of the currently checked out branch.
* To add a file in git repository

1. git add second\_file.txt

* To stage a file commit the file
* To reported git conditions using

1. git status

* To add multiple files at once

git add first\_file.txt second\_file.txt

* diff is used to compare two files.
* To get details about changes that were in our working directory.
* diff as the term that it uses show us a diff between the old version and the new version.

1. git diff

* To see comparison between single files before and after modified

1. git diff first\_file.txt

* git diff only return the working directory
* If we want to see the comparison of version in staging directory

1. git diff –staged

* Git shortcut

1. F – forward; b – backward; q – quit

Delete file in Git

* Two way is available to delete a file

1. The first is to move the file out of the working directory, in this case into the trash, and then tell git about it.
2. The second is to tell Git to do the delete for us, and add it to the staging directory all at the same time.
3. To remove a file in git

git rm delete-first-file.txt

1. After that add a commit

Move and Rename files in Git

* Two ways are available to move or rename a file

1. The first, is that we can just go into the regular operating system and do all of our moving and renaming and then come back and tell fit about those changes, and tell it to stage those changes.
2. The second, is that we can do it from Git, and let Git handle working in the operating system for us, just like it did with delete.