git

GitHub

By Chidanand

anandr72@gmail.com

Agenda

- Introduction to Version Control & Git
- Client server v/s Distributed Version Control
- Git Concepts, Hash Values (SHA-1), Data Model
- Basic Git Operations
- GitHub set-up, registration, SSH Keys, Repo Set up and 101 operations, Limitations of personal account
- Advanced Git Operations – Branching Model, remote tracking branches, merge conflicts, fetch, pull, rebase, stash, reset, cherry-pick, revert
- Workflow for sample Java Repository
- GitHub Organizations, Access Permissions, Clone v/s Collaborative approach
- Typical Branching Models
- Jenkins, CI/CD & DevOps (Time Permitting)

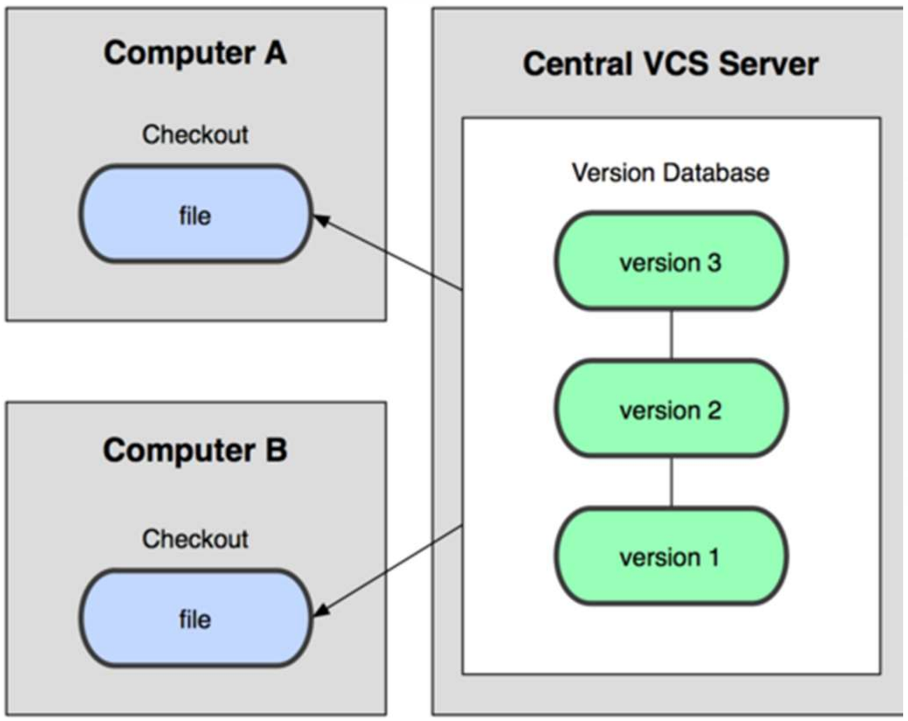
Why Version Control

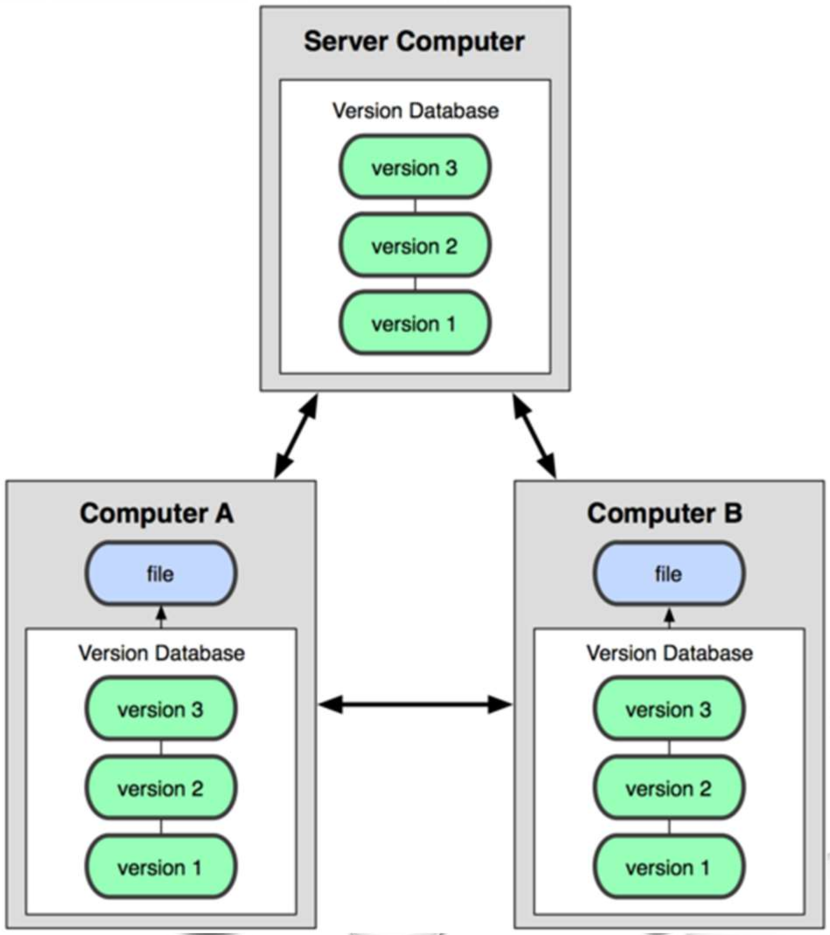
- To manage multiple versions of source code
- Collaboration, simplifies team efforts & concurrent work
- Allows you to go back & forth between versions, diff changes between two source snapshots & across branches
- Popular ones – Mercurial (hg), bazaar, subversion (SVN), Concurrent Version System (CVS), perforce, Clearcase

Typical Version Control Tasks

?

Centralized Vs Distributed Version Control





Centralized Version Control

- Based on the idea that there is a single "central" copy of your project somewhere (probably on a server), and everyone will "commit" their changes to this central copy
- Typical workflow -> Checkout a file -> Edit -> Check it back in
- Where it wins - Programmers no longer have to keep many copies of files on their hard drives manually, because the tool can talk to the central copy and retrieve any version they need on the fly
- Connection is needed for most functions
- CVS, SVN, Perforce, Clearcase etc

Distributed Version Control

- Almost everything is a local copy
- Typical workflow -> clone, commit, pull, push
- Except for pull, push, all other activities can be performed without connection – commit, merge, branching, diffs
- Every clone is an exact replica of the central repo
- Git, Mercurial (hg), Bazaar

Git History

Literal meaning?

- Started with a need for managing Linux kernel development, back in 2005

"I'm an egotistical bastard, and I name all my projects after myself. First 'Linux', now 'git' - Linus Torvalds

Definition

Git is an open source, distributed version control system designed for speed and efficiency

Git variants

- GitHub (https://github.com/) – Cloud hosting & GitHub Enterprise Server (https://enterprise.github.com/home)
- Atlassian Bit bucket (https://bitbucket.org/)
- Gitlab (https://gitlab.com/)
- Unfuddle (https://unfuddle.com/)
- Assembla (https://www.assembla.com/git/)

And many more……………

Git Client Set-up

- Download & Install clients (Windows, Mac, Linux)

 -> https://git-scm.com/downloads

- Git Bash is the bare client that we will use for all our lab exercises
- Create a Directory – "Git_Training" on your computer and please use this directory for all Lab exercises

Git config – Username & EMail

- First time configuration for setting up Username & Email ID
- On your Git bash
  - $git config --global user.name "John Doe"
  - $ git config --global user.email johndoe@example.com

  Check if the configurations were saved

  - $git config --list

Git Config

- Git config – tool that lets you customize your git environment
- 3 levels of configuration
  - System level – All users of this system and all subsequent git repos will inherit this, invoked with –-system (stored in C:\ProgramData\Git\config)
  - Global – specific to this user only (and applies to all his repos), invoked using –-global (stored in C:\Users\<username>\.gitconfig)
  - Local or Repo level (default) –local (stored in Repos .git/config)

Git config

- $git config –-list (lists all your global settings)
- $git config user.name (check for one particular setting)
- $git config –-global --edit (edit/modify settings)

Git config “core.autocrlf” for Cross Platform

- Formatting & white space issues are common when developers use Windows & Linux/Unix/Mac desktops
- Window editors use both Line Feed (LF,\n) and Carriage Return (CR, \r) for identifying new line while Unix/Mac systems use only Line Feed
- Some windows editors silently replace existing LF line endings with CRLF or insert both line-ending characters when the user hits the ENTER key

Git config “core.autocrlf” for Cross Platform

- On Windows machine, set core.autocrlf to true so that Git converts LF endings into CRLF when you check out code
- $git config --global core.autocrlf true
- On Linux/Mac, to ensure that Git converts CRLF to LF on commit
- $git config –-global core.autocrlf input
- If everyone is on Windows, turn off this feature
- $git config --global core.autocrlf false

Git First Repo Demo

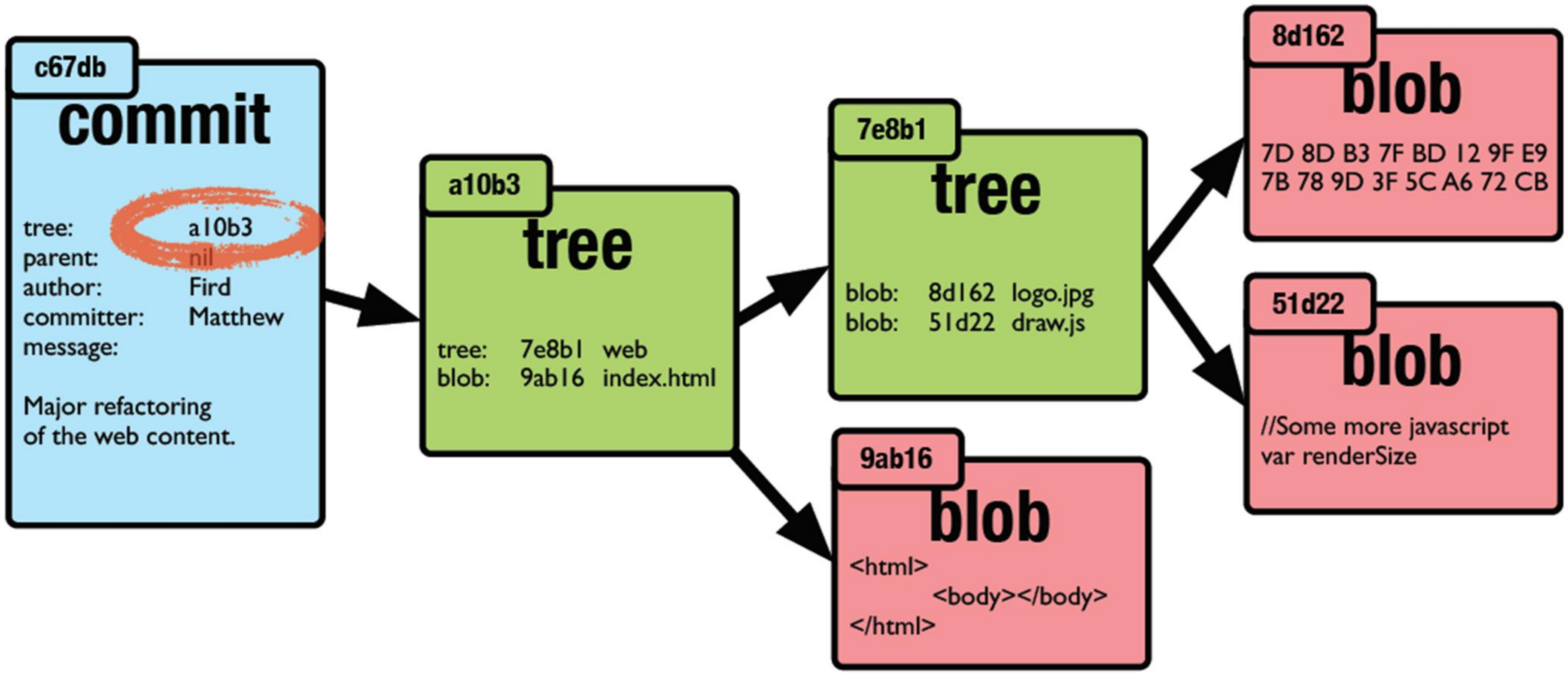
Create your First “Local” Repo

- Create(mkdir) a new directory called “HelloGit” and chdir to it
- $git init
  - Creates the famous ‘.git’ directory (NEVER TOUCH IT)
  - Also creates the default branch called ‘master’
- Add few files & save them
- $git add .
- $git status
- $git commit –am “My First commit”

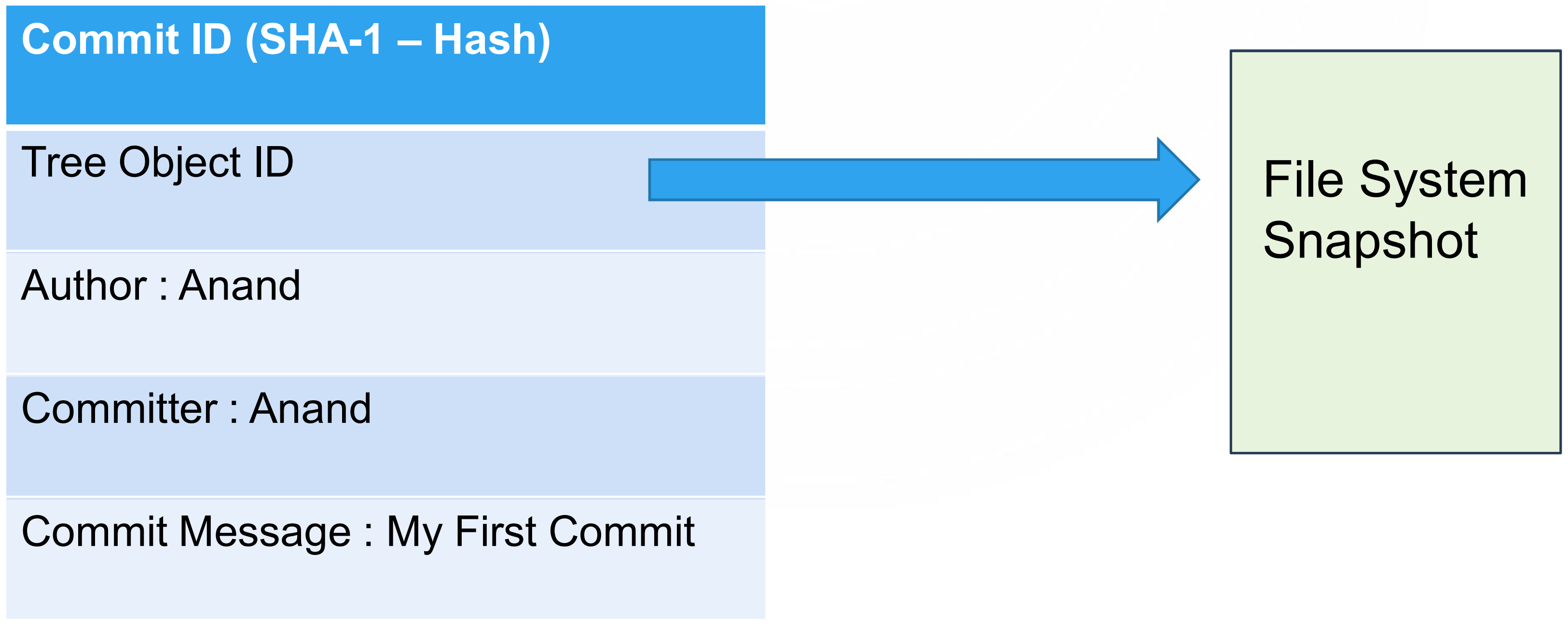
Git Internals

- For each repo, git stores all information about it in a special directory '.git' (usually called as Object Store)
- Git knows about 4 object types
  - Blob – each file that you add to the repo is a blob
  - Tree – each directory structure is turned into a tree
  - Commit – snapshot of your working tree
  - Tag – Unique identifier for a commit or snapshot
- SHA-1 (checksum) – Unique 160 bit hex code identifier for a particular commit

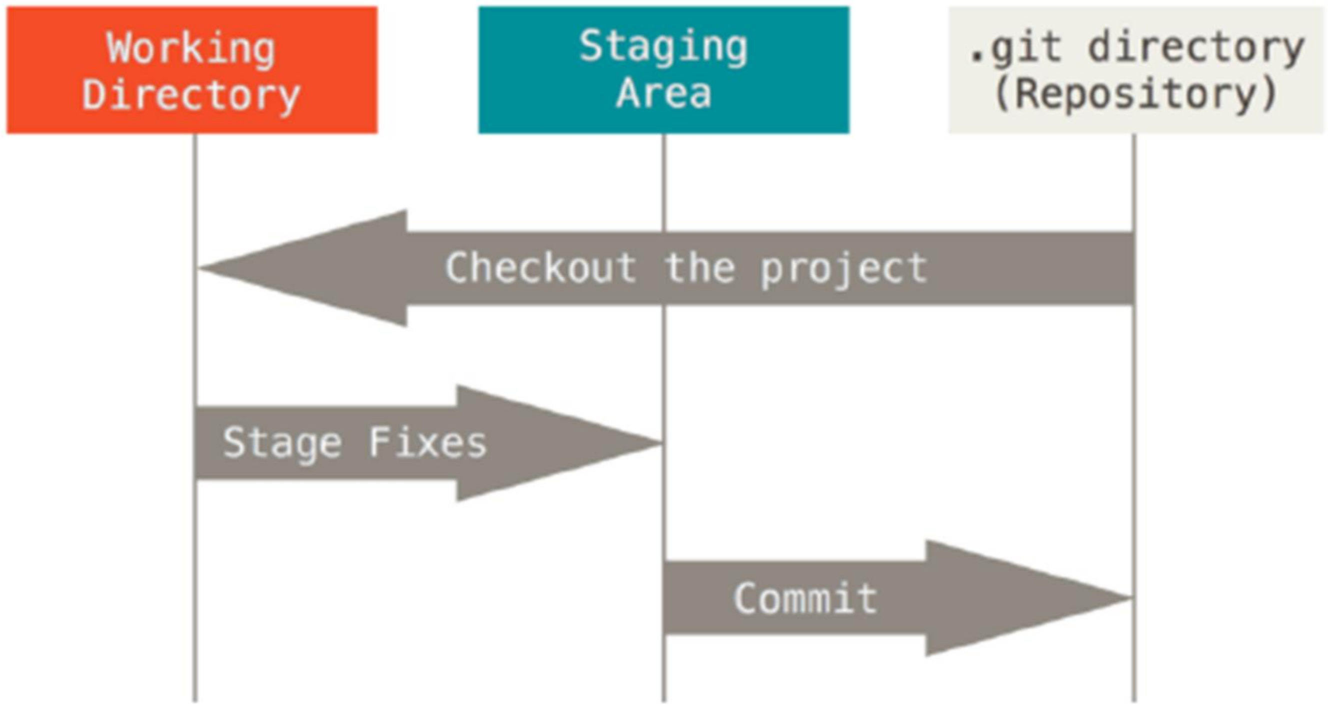
Git Internals – Object Model



Git - Commit

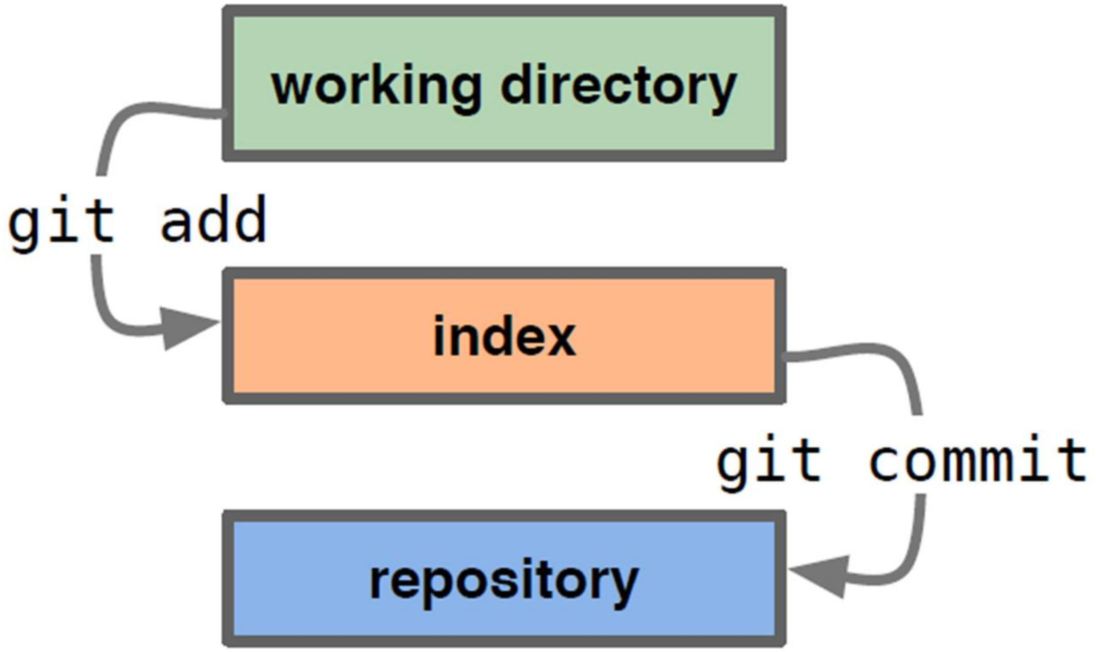

Commit ID (SHA-1 – Hash)
Tree Object ID
Author : Anand
Committer : Anand
Commit Message : My First Commit
File System Snapshot

Git – 3 Status of Repository

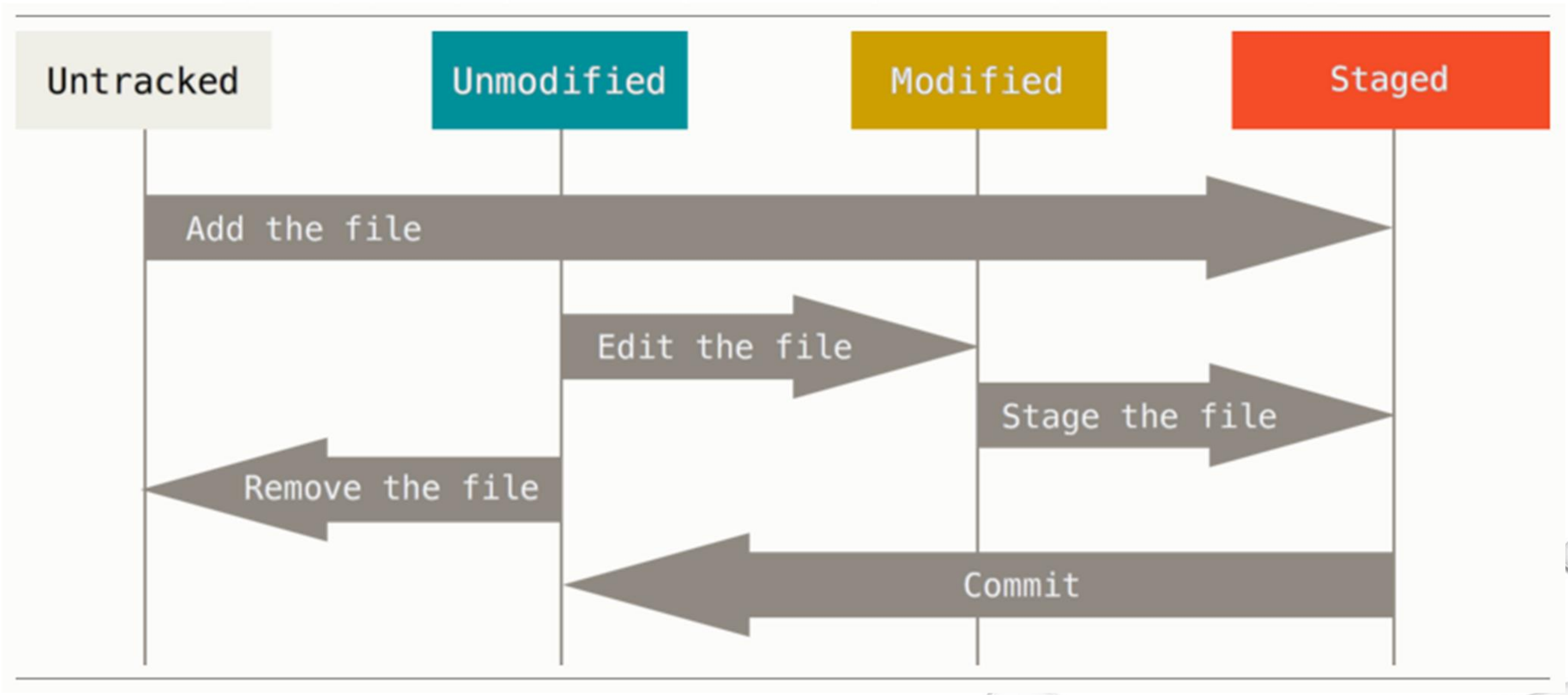


Modified, Staged, Committed

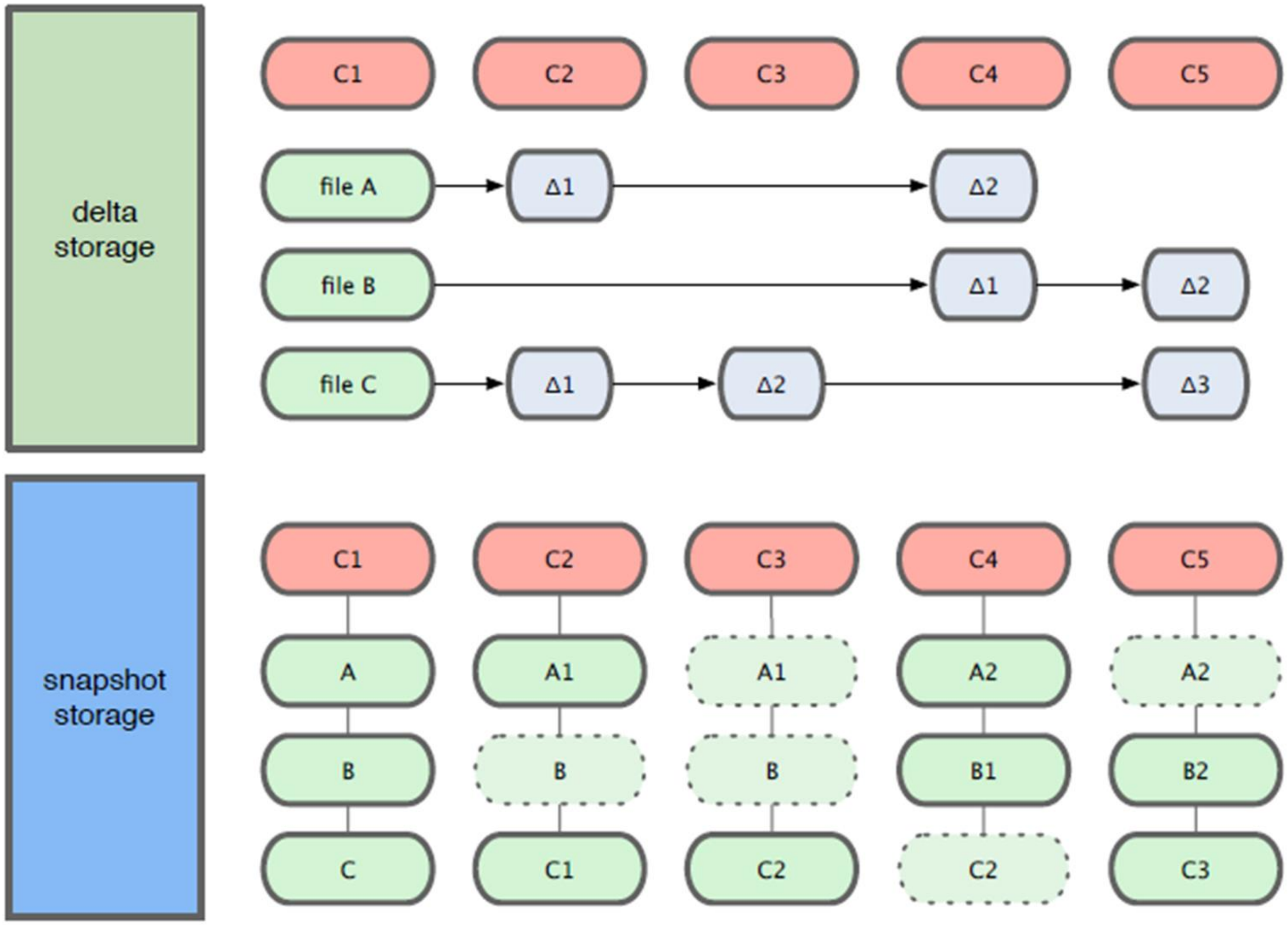
Git Working Model - 3 stages



Git - Files Lifecycle



Git Internals – Snapshot storage



Git Server Details

- Snapshot of Git Client, Server & Workflow!!

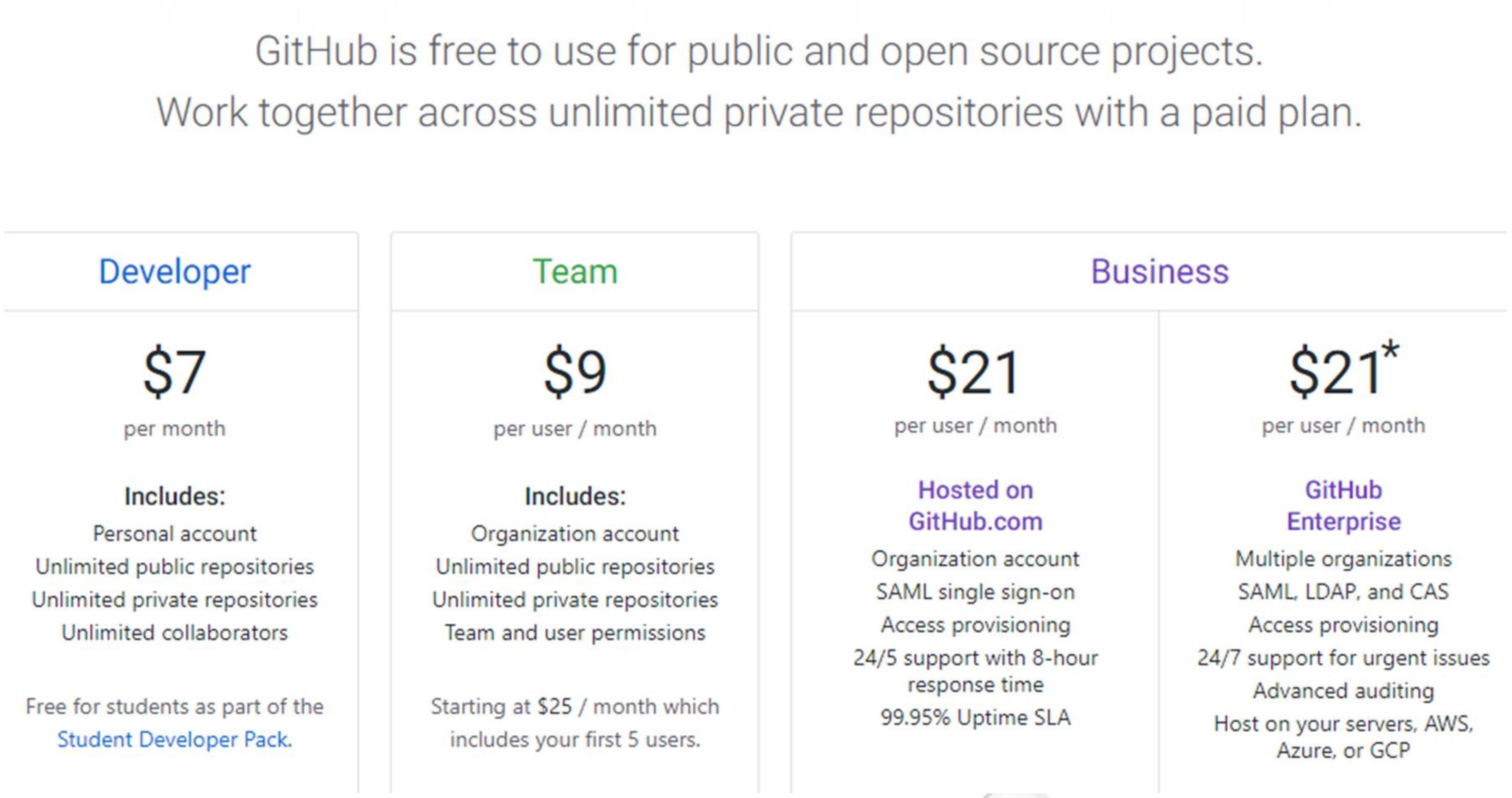
GitHub Server & Client Set-Up

- Register on https://github.com/ (personal email ID)

Free Subscription for individual Developers

Remember : UserName (Email ID) & Password

GitHub Account Subscription Details



GitHub is free to use for public and open source projects.
Work together across unlimited private repositories with a paid plan.

Developer	Team	Business	
$7 per month	$9 per user / month	$21 per user / month	$21* per user / month
Includes:	**Includes:**	**Hosted on GitHub.com**	**GitHub Enterprise**
Personal account	Organization account	Organization account	Multiple organizations
Unlimited public repositories	Unlimited public repositories	SAML single sign-on	SAML, LDAP, and CAS
Unlimited private repositories	Unlimited private repositories	Access provisioning	Access provisioning
Unlimited collaborators	Team and user permissions	24/5 support with 8-hour response time	24/7 support for urgent issues
		99.95% Uptime SLA	Advanced auditing
			Host on your servers, AWS, Azure, or GCP
Free for students as part of the Student Developer Pack.	Starting at $25 / month which includes your first 5 users.		

Create First Repo on GitHub Server

- Demonstration & Exercise to Create “Hello_Server” Repo on GitHub.com
- Add few files to it & commit few changes

Clone Repo using HTTPS

- Demo & exercise

Find HTTPS URL of the Repo and use it to clone the Repo

In Git bash, cd to ""Git_Training" directory

```
$git clone "HTTPS URL OF YOUR REPO"
```

Add 2 new files and edit few more

```
$git add .
$git commit –am "Comments"
$git push –u origin master
```

GitHub SSH Set-Up

- REMEMBER : AT ANY TIME, ONLY ONE SSH KEYS CAN BE ACTIVE ON A DEVELOPER MACHINE/LAPTOP
- BACK-UP Existing Keys (found in USER-HOME\.ssh directory)

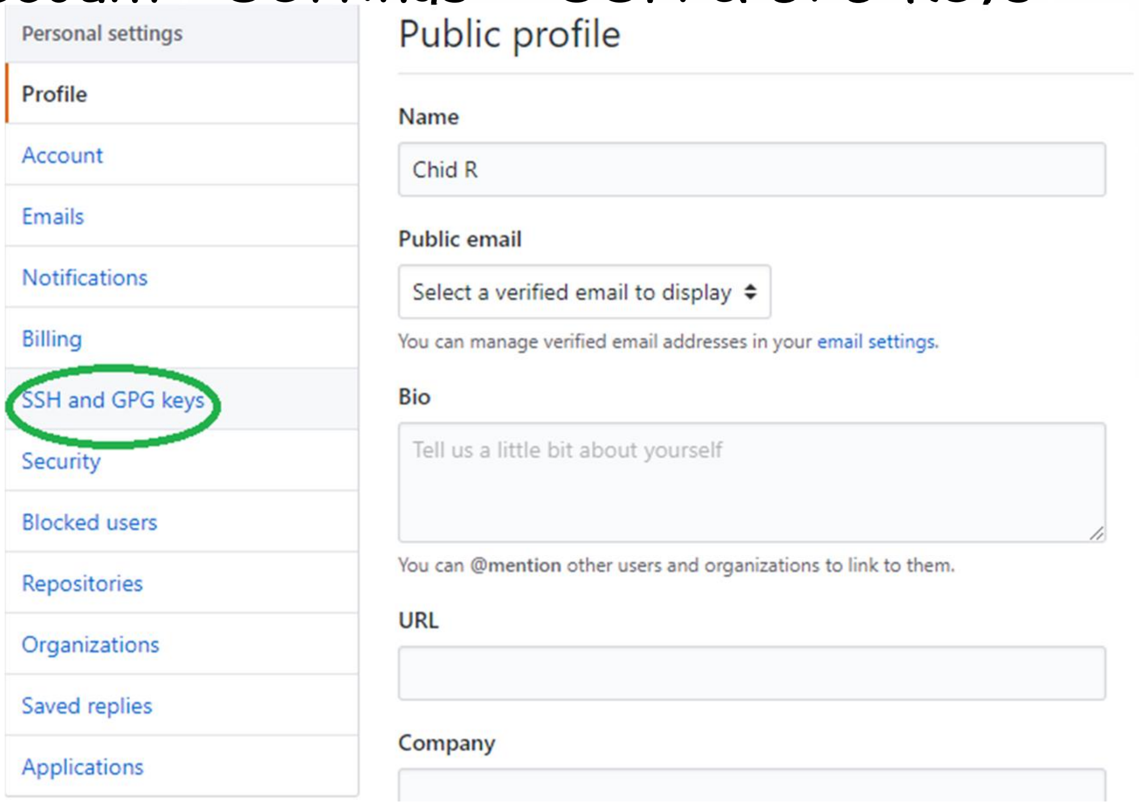
- Create a pair of SSH keys using the following command

```
$ssh-keygen -t rsa -C YOUR_EMAIL_ID
```

Please Note : Enter a passphrase & **REMEMBER IT**

GitHub SSH Set-Up

- Open public key in notepad, copy it over to your GitHub Account : Settings -> SSH & GPG Keys



Public profile

Name

Chid R

Public email

Select a verified email to display

You can manage verified email addresses in your email settings.

Bio

Tell us a little bit about yourself

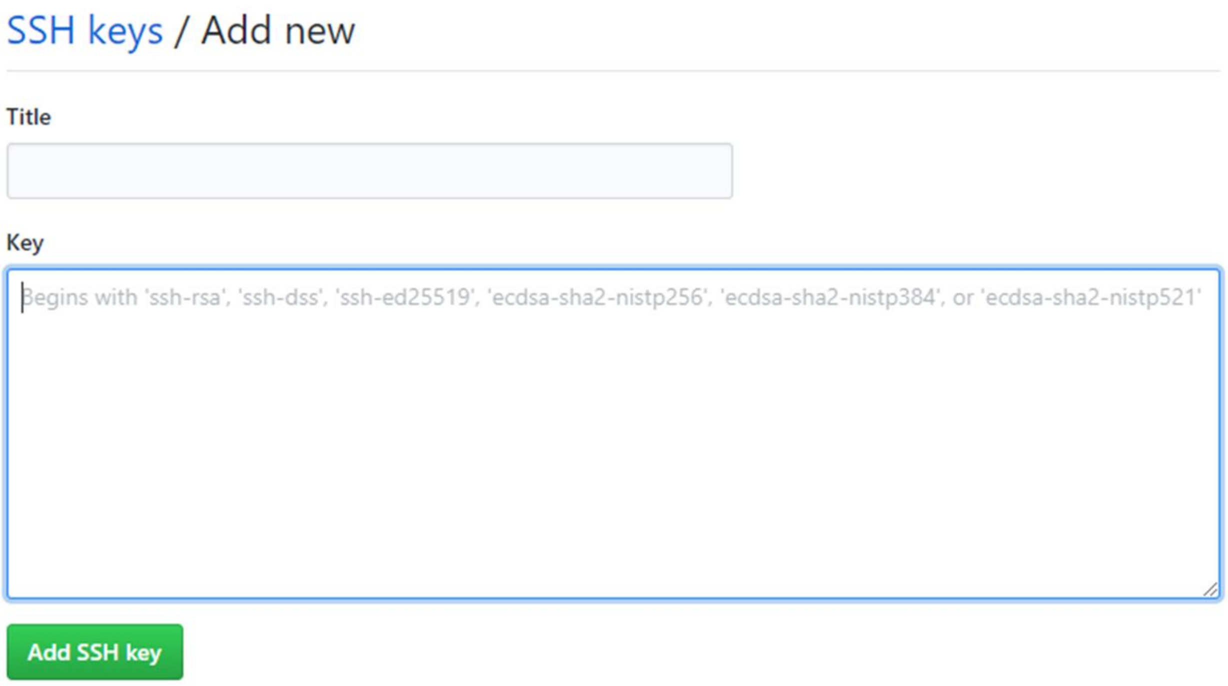
You can @mention other users and organizations to link to them.

URL

Company

GitHub SSH Set-Up

- Paste your Public Key & Save it


SSH keys / Add new

Title

Key

Begins with 'ssh-rsa', 'ssh-dss', 'ssh-ed25519', 'ecdsa-sha2-nistp256', 'ecdsa-sha2-nistp384', or 'ecdsa-sha2-nistp521'

GitHub Set-Up

- Check your SSH connection using the following command

```
$ssh -T git@github.com
```

Well Done!! Your SSH setup is complete!!

Git SSH clone

- Locate the SSH URL of your GitHub Repository
- In Git Bash, create a new Directory (or remove the previous HTTPS Repository since the Repo will get overwritten)
- $git clone "SSH URL OF YOUR REPO"
- After the repo is cloned, add 2 new files to it and make few changes to existing files
- $git add .
- $git commit –am "Comments"
- $git push –u origin master

Git basic workflow

- Init or clone $git init or $git clone
- Edit/change file(s)
- Stage the changes $git add (file)
- Review changes $git status
- Commit changes $git commit –m "Comments"
- Push to repo $git push –u origin master

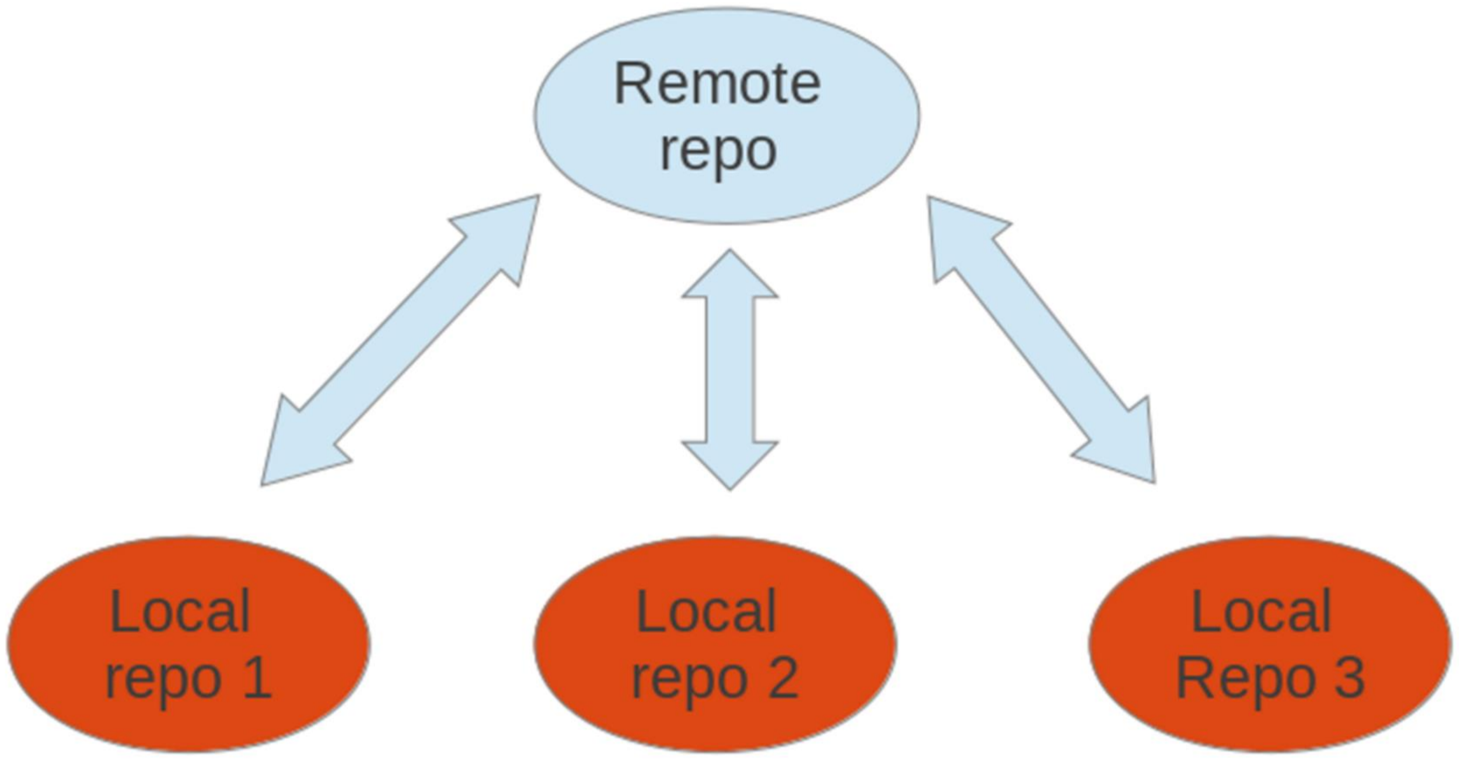
Few Git terminologies

- master – Repositories main branch (most of the times)
- Clone – Copy/replica of a repository
- Commit – check-in in changes
- HEAD – is the reference to the current commit and most of the time it points to latest commit in your branch
- Origin – The default name given to main remote repo
- Pull/fetch – Get latest from remote repo
- Push – Submit or check-in latest changes

Git Inspection commands

- $git status –> Shows status of working directory and staging area
- $git log -> Shows details of committed snapshots
- $git log –-oneline
- $git log –-stat
- $git log <since>..<until>   -> Diff between 2 commits
- $git log --graph -–decorate -–oneline
- $git log <filename>

Git remote



Git remote

- References to versions of repositories that are hosted elsewhere
- By default, there is one remote called ‘origin’ which is the main repository reference
- If teams have need to work/collaborate with various repositories, you can have multiple remotes for a repo

Git remote

- $git remote –v
- $git remote add name URL

Eg : $ git remote add Andy git@bitbucket.org:AnandR72/new_repo.git

Can push/pull to this repo by specifying remote name

Eg : $git push Andy master

Modify URL using:

- $git remote set-url Andy git@bitbucket.org:AnandR72/Repo_new.git
- $git remote remove Andy

.gitignore

- Add .gitignore file when you want to git to exclude them from tracking changes to it
- Can be added in any directory and path can be relative

Regex Syntax:

```
log/*        All files under log directory will be ignored
.classpath
.project
*.DLL
*.dll
*~           All files ending with ~ will be ignored
```

.gitignore contd

- Git never ignores files that are already tracked (???)
- .gitignore affects only new files and if you need to ignore already tracked files, you need to explicitly remove them
- $git rm –-cached filename
- Problem of .classpath & .project in a Java Repo
- What extensions need to be ignored for Python, SCALA and/or PHP projects

.gitignore contd

- Global .gitignore can also be set up that applies to all Git Repos (across repositories)
- Create a .gitignore file in your home directory
- $cd ~/
- $vi .gitignore -> put in all appropriate regex and save it
- $git config --global core.excludesfile ~/.gitignore

Git Exercise

- Create a new Repo on BitBucket called – IgnoreTest
- Create folders 'bin', 'logs', 'images' and 'lib' with some sample contents in it
- Add a .gitignore file to exclude the following:
    - Directories 'bin', 'logs', 'images' & 'lib'
- Commit changes to check if the folder contents are ignored

❖ Problem of .classpath & .project in a Java Repo

❖ $git rm --cached .classpath

Git Exercise

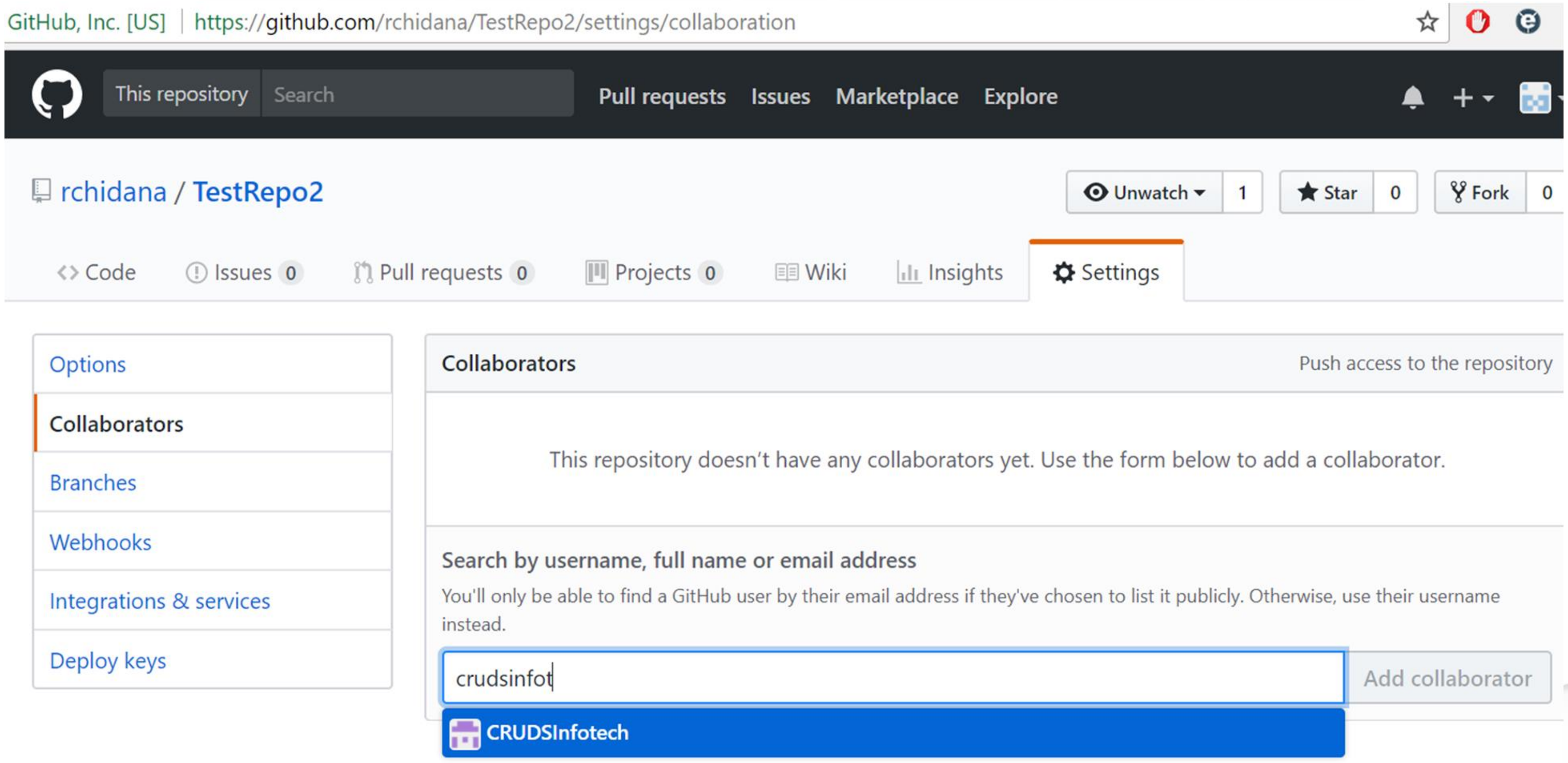
No fun working all alone!!

Collaborative exercises : Group 2 members together and have them set up one repository which will be collaboratively used by each other for the reminder of the lab session

- User1 will create a Repo called – MyDetails
- Add Two Files to it – Name.txt & Address.txt
- Push these changes to GitHub Server
- User1 will share this Repo with User2 by making him a Collaborator

GitHub Repo Sharing

Add Collaborators to your Repo : Settings -> Collaborators


GitHub, Inc. [US] | https://github.com/rchidana/TestRepo2/settings/collaboration

Collaborators — Push access to the repository

This repository doesn't have any collaborators yet. Use the form below to add a collaborator.

Search by username, full name or email address

You'll only be able to find a GitHub user by their email address if they've chosen to list it publicly. Otherwise, use their username instead.

crudsinfot

GitHub Repo Permissions

- READ -> Can only View, Clone or Fork Repo
- WRITE -> Can Contribute to the repo by pushing changes directly from their local repo
- OWNER -> READ, Write, Clone, Delete, Add Collaborators
- ADMIN -> Similar rights as Repo Owner
  - Can change Repo Settings
  - Add, Change or Remove User permissions
  - Give other users administrative access
  - Delete the Repo
  - Transfer Ownership of Repos, Archive Repo

Git Exercise

- User2: Clone "MyDetails"
- User1 : Add "phone.txt" & commit changes
- User2 : Add "empid.txt" & commit changes
- User1 & User2 : Push your changes simultaneously!!

```
! [REJECTED]        MASTER -> MASTER (FETCH FIRST)
```

Git Exercise

- Always remember to pull changes from the main repo before you push any changes to it!!
- $git pull origin master

ALWAYS REMEMBER TO PULL CHANGES FROM THE SERVER BEFORE PUSH!!

Git Exercise

- Amicably between User1 & User 2 – add “hobbies.txt”, “education.txt” and also make updates in other files
- Provide proper commit comments & check-in files

Git Exercise

What are code conflicts? examples?

- User1 : Add new file “name.txt” with your name
- User2 : Add new file “name.txt” with your name
- Commit changes & push it to repo

Git Exercise

- Create more conflicts in 2 or more files & resolve them successfully

Git reset – Undoing things

- $git reset –-soft, --hard, --mixed (default)

Try the following on a local Repo

- Create a local repo called "Reset_Test"
- Add a file "blog.txt" and perform 5 commits to it
- $git reset –-soft HEAD~2 (roll back by 2 commits)
- Make additional changes & check it in again
- $git reset by SHA code
- Difference between soft, hard & mixed!!

Git Exercise

Quiz!!

- Create a Repo on remote server called “Rollback_Test”
- Perform 5 commits to it
- Reset/re-wind head by 1 commit and push it to server!!

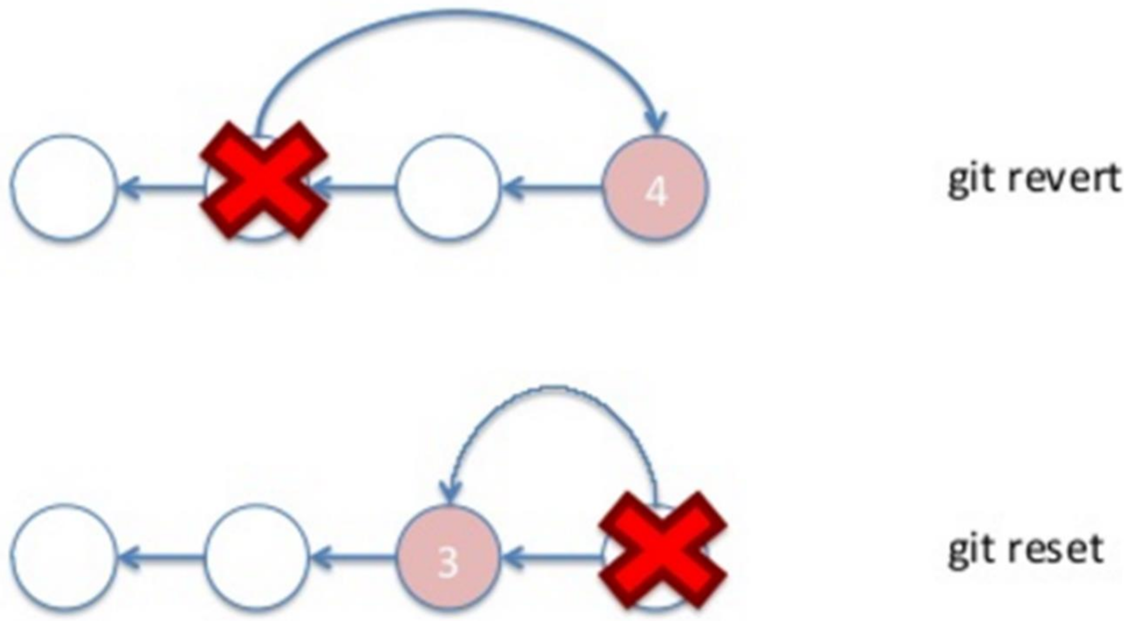
Git revert – Undoing things

- Reset loses contents while revert is a safe way of un-doing changes
- Instead of removing a commit, revert undoes the commit and appends a new commit with resulting content
- Prevents losing history and works better for integrity of revision history & reliable collaboration
- $git revert <commit>

Git revert Exercise

- Create a new Repo “Revert_Repo”
- Add a.txt (first commit), b.txt (second commit), c.txt (third commit)
- Add e.txt and also make changes to a.txt & b.txt
- Commit these changes as fourth commit
- $git revert <third commit>

Git revert v/s reset



Git amend

- Convenient way to modify the last commit message
- $git commit -–amend –m “New Commit Message”
- Exercise
  - Create a Repo with 1 file with multiple functions in it
  - Modify Add function
  - Put in an incorrect commit message – modified Multiply
  - >$git commit –-amend –m “Modified Add Function”

Git amend – Missed out files

- Assume that you just made a commit but missed out few files in it and want to include it in the previous commit
- Add those files to git index after which :
- $git commit –-amend
- $git log –-stat
- Exercise:

Git Reverse workflow

- Add 1 new file & edit another file and git add changes for staging
- $git add .
- $git diff –-staged (shows changes that are being staged)
- $git reset HEAD (unstage changes)
- $git status
- $git checkout modified_file (undo edits & get back to previous snapshot)
- $git clean –n (checks & prompts) while –f (removes new file) and –d (removes directories)

Git rm & move

- To completely remove a file from git index and from the file system
- $git rm File_Name
- To remove file from git index but not from file system
- $git rm –cached File_Name
- To Rename files or folders
- $git mv old_folder new_folder
- Git Copies old_folder contents to new_folder, removes old_folder and adds new_folder in Git index

Git Stash Exercise

- Save Work in Progress tasks
- $git stash
- $git stash save "Work in Progress – fix for defect #2"
- $git stash list
- $git stash pop #applies first stash that was saved
- $git stash apply stash@{0} #stashed changes remains
- $git stash drop stash@{0}

Git Tag

- Git has the option to tag a commit in the repository history so that you find it easier at a later point in time. Most commonly, this is used to tag a certain version which has been released
- 2 types of tags – Light Weight & Annotated
- Lightweight tag is just a pointer to a commit without any additional information about the tag
- $git tag v-1.0.4 ->creates a tag for the current commit
- $git show v-1.0.4 -> Shows details of this tag
- $git tag -> Will show all available tags

Git Tag

- Annotated tags can store a tagging message
- $git tag –a v1.0 –m “First Base Line Version”
- $git show v1.0
- You can tag any commit at any point in time
- $git tag V0.5 SHA-ID-OF-COMMIT

Git Sharing Tag

- By default git push does not transfer tags to the server, you will have to explicitly push tags
- $git push origin v1.0   { tag name}
- To push all tags in one shot
- $git push origin --tags

Git Delete Tag

- To delete a tag locally
- $git tag –-delete V0.1   {tag-name}
- To delete this tag from the server
- $git push –-delete origin V0.1   {tag-name}

Importance of Tag

- ??
- GitHub Release Feature allows you to draft a new release based on tags and also include any other binaries if needed and publish it to users
- Exercise & Demo to release from a given Tag

Branches – Killer feature of Git

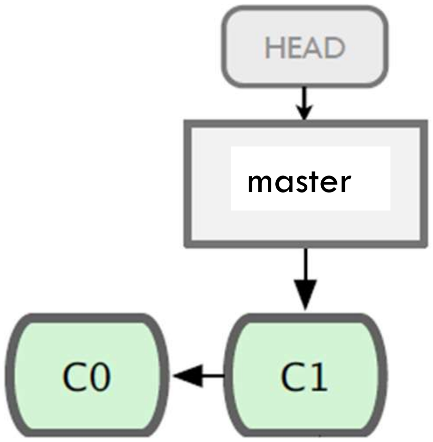
- What is a Git branch & what is the need for creating branches?
- $git branch new_branch (creating new branch)
- $git checkout new_branch (switching to new branch)

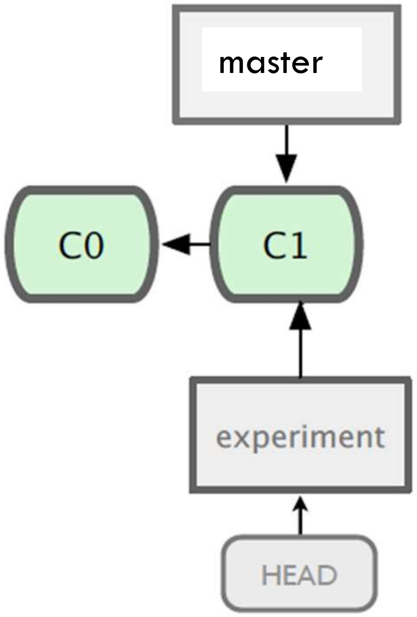
Or

- $git checkout –b new_branch (create & switch to new branch)
- Demo of Branch & file System snapshot

Git Branches

While on ‘master’ branch : $git checkout –b experiment





Git merge Exercise

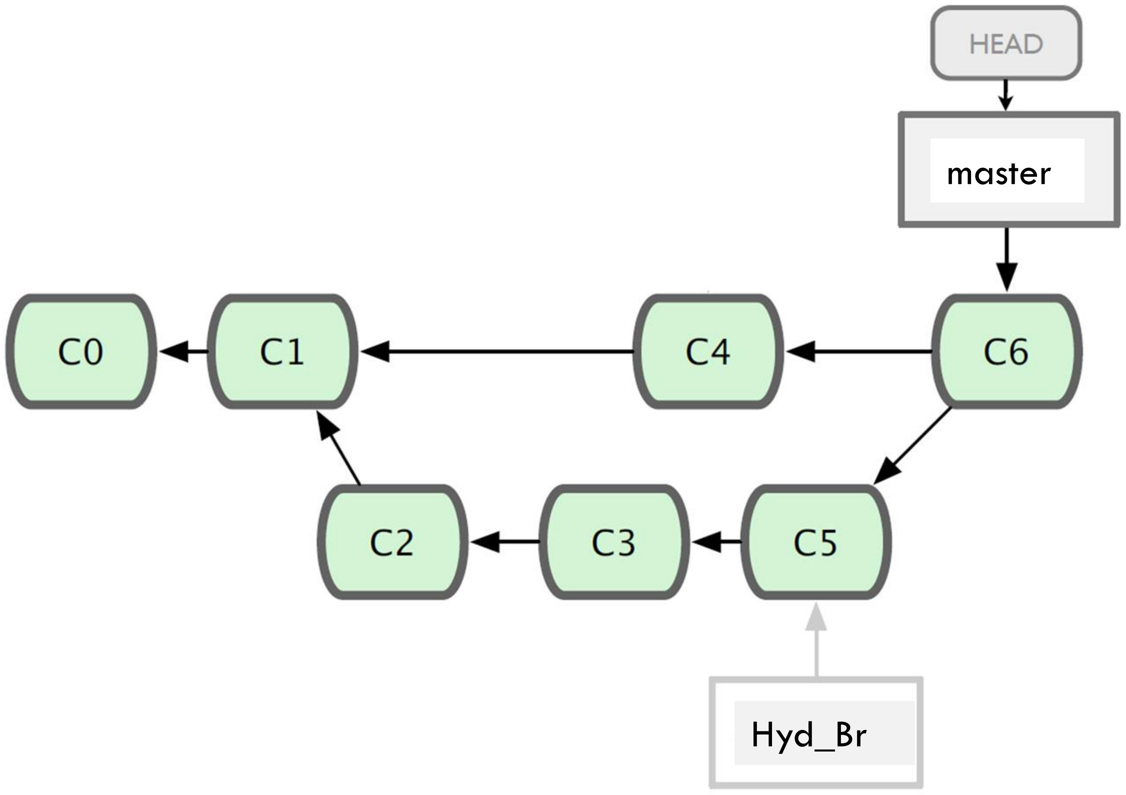
- Create a Repo called “Blogs”
- Add a file “personal.txt” to it and commit it
- Create a new branch from “master” called “travel_blogs”
- Switch to “travel_blogs”, add a new file called “travel.txt” and commit 2 changes to it
- Switch back to “master” and merge “travel_blogs”

“FAST-FORWARD MERGE”

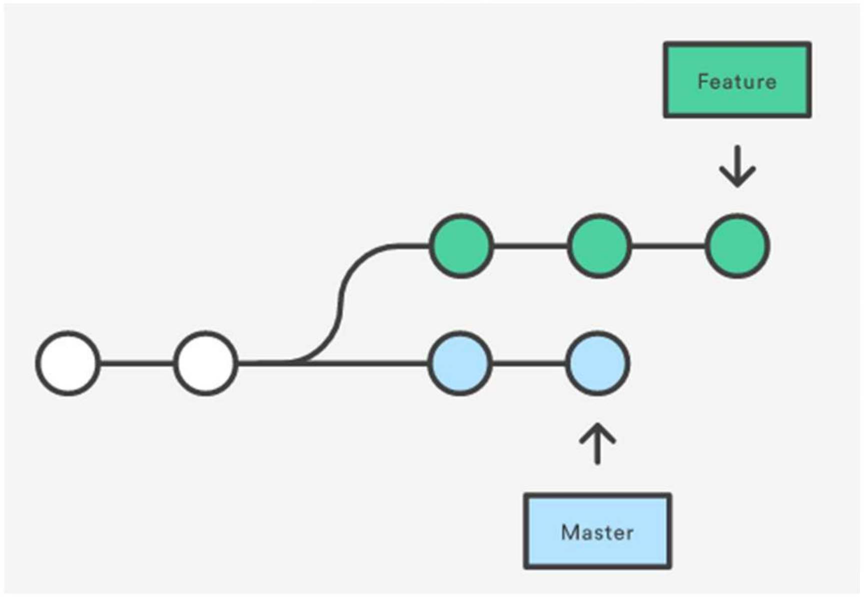
FF merge desired or not??

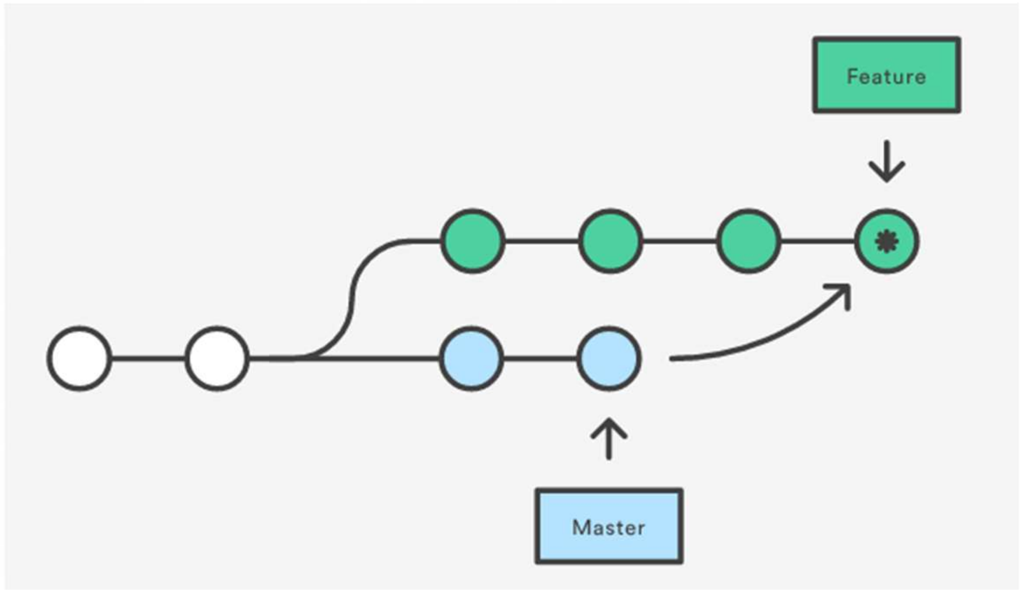
- It is always better to have a commit which explicity denotes branch merges
- Setting to turn off Fast forward merge
- $git config --global --add merge.ff false
- Create one more branch hobbies_blog, check-in hobbies.txt
- Come back to master branch & merge hobbies_blog

Git Exercise
3 way recursive merge Strategy



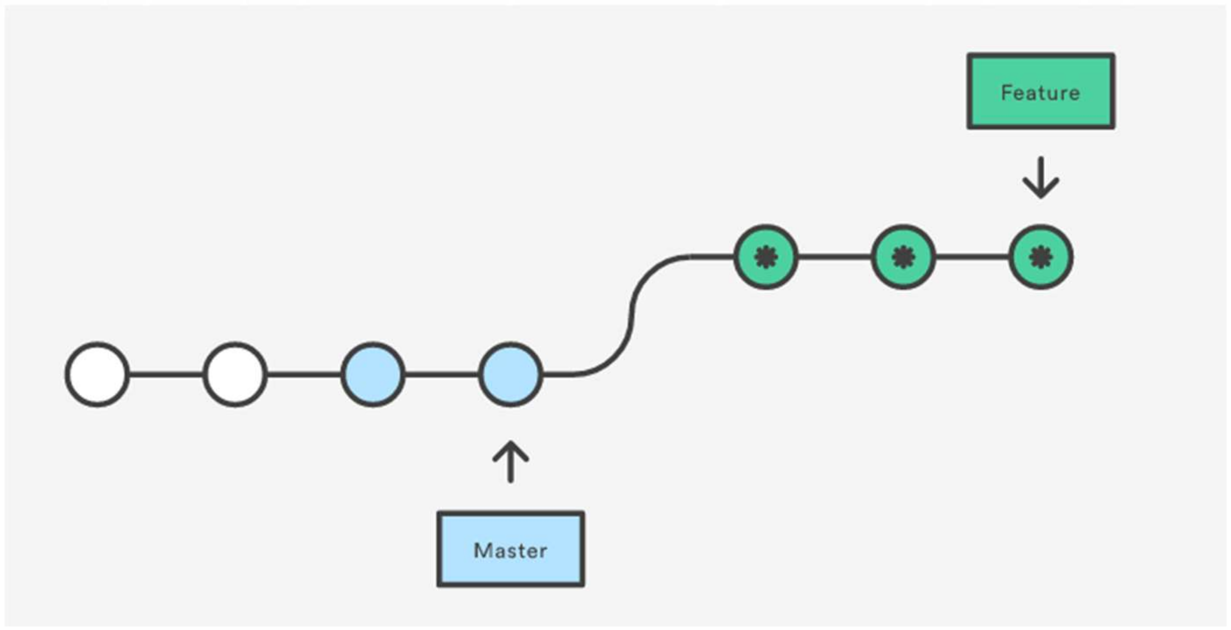
Git Rebase??



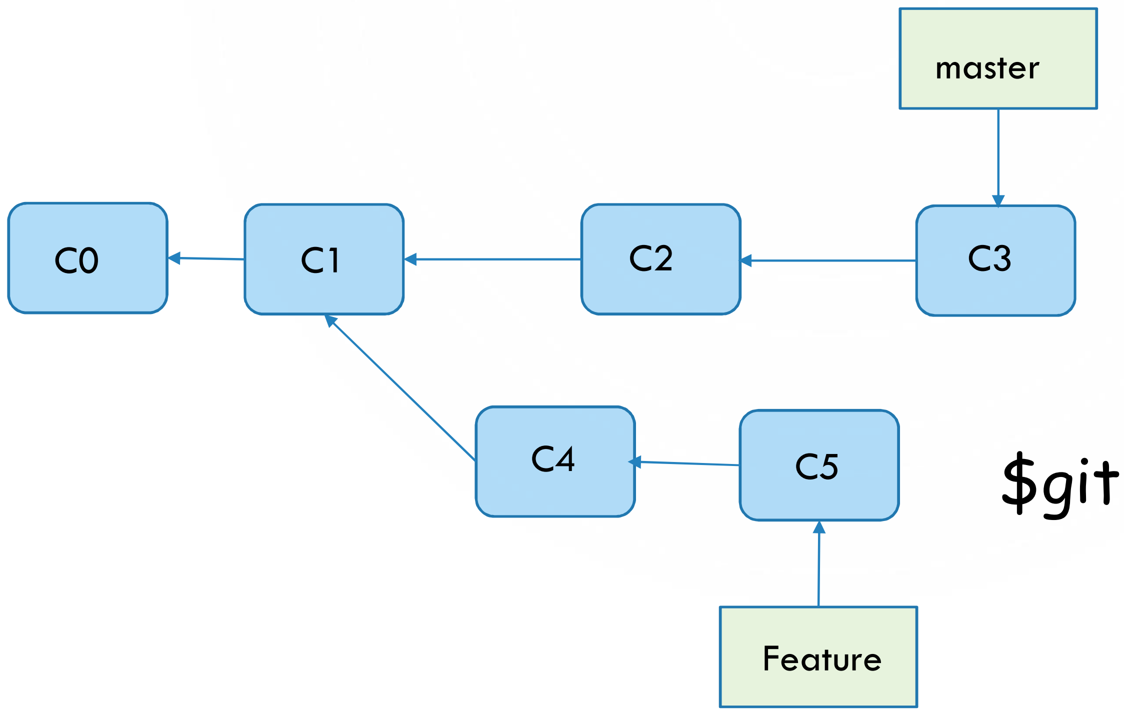


Merge creates extra commit

Where to Use Rebase??



Git Exercise – 12 Rebase



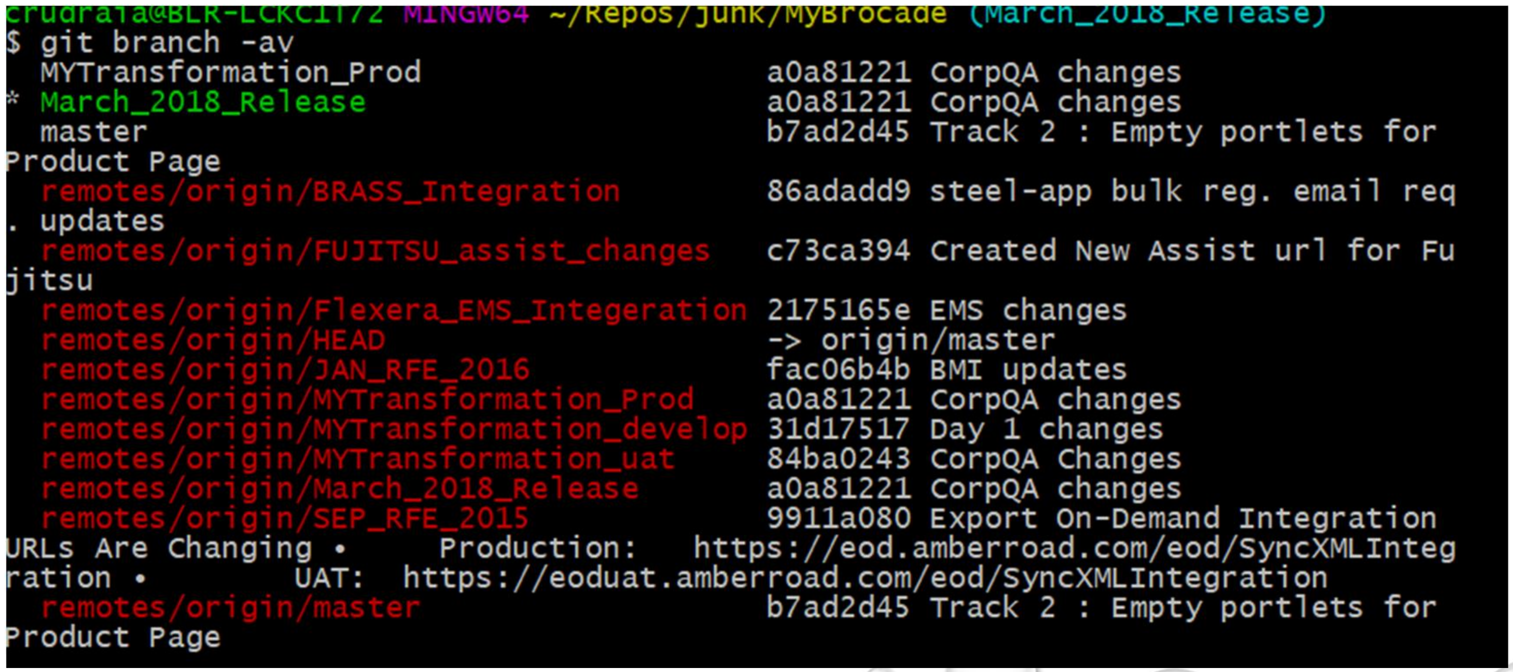
```
$git rebase master
```

Interactive rebase

- Wonderful command that allows you to pick & choose commits, amend commit message and squash commits
- Master -> Commit: M1, M2
- Checkout Feature branch
- Feature -> Commit : F1, F2, F3, F4
- Master -> Commit: M3, M4
- On Feature branch ->git rebase –i master

Git Remote Tracking Branches

- 2 Types of Branches – Local & Remote
- $git branch –av



```
crudraia@BLR-LCKC1172 MINGW64 ~/Repos/junk/MyBrocade (March_2018_Release)
$ git branch -av
  MYTransformation_Prod                          a0a81221 CorpQA changes
* March_2018_Release                             a0a81221 CorpQA changes
  master                                         b7ad2d45 Track 2 : Empty portlets for
Product Page
  remotes/origin/BRASS_Integration               86adadd9 steel-app bulk reg. email req
. updates
  remotes/origin/FUJITSU_assist_changes          c73ca394 Created New Assist url for Fu
jitsu
  remotes/origin/Flexera_EMS_Integeration 2175165e EMS changes
  remotes/origin/HEAD                            -> origin/master
  remotes/origin/JAN_RFE_2016                    fac06b4b BMI updates
  remotes/origin/MYTransformation_Prod           a0a81221 CorpQA changes
  remotes/origin/MYTransformation_develop 31d17517 Day 1 changes
  remotes/origin/MYTransformation_uat            84ba0243 CorpQA Changes
  remotes/origin/March_2018_Release              a0a81221 CorpQA changes
  remotes/origin/SEP_RFE_2015                    9911a080 Export On-Demand Integration
URLs Are Changing •     Production:    https://eod.amberroad.com/eod/SyncXMLInteg
ration •        UAT:  https://eoduat.amberroad.com/eod/SyncXMLIntegration
  remotes/origin/master                          b7ad2d45 Track 2 : Empty portlets for
Product Page
```

Git Remote Tracking Branches

- These are local references to remote branches, any change in remote (server) branches will automatically reflect in local branches when user does a pull
- User 1 -> Create Branch_Demo repo with 3 branches in it -> master(default), develop, rfe and commit some changes in each of these branches
- User 2 to clone this repo

Git Remote Tracking Branches

- User2 to make changes in Develop & RFE branch
- User1 to make changes in master branch
- Both users can fetch changes from other branches and do a diff to find out incoming changes
- $git diff origin/rfe {will show differences between current branch & remote rfe branch}
- $git merge origin/rfe {merge these changes, similar to git pull origin rfe}

Git Remote Tracking Branches Prune

- User1 adds a new Branch “rfe” and pushes to origin
- User2 $ git remote update --prune
- User1 $git branch –d <branchname>   Remove local branch
- User1 $git push origin --delete <branchname> Remove remote branch

Git Checkout & detached HEAD

- Checkout is a powerful command and can be used to switch to branches or to a snapshot (SHA-ID)
- Create a Repo called Checkout_Test
- Commit 5 changes to it – C1,C2,C3,C4 & C5
- $git checkout C3

Git Submodules

- Repositories inside Repository
- Submodules are Git Repositories nested inside a parent repository at a specified path
- Submodules can be located at any place within the parent repository and can be configured using a .gitmodules file located at the root of parent repo
- This file will contain URL for pulling submodules
- Supports adding, updating, synchronizing & cloning

Git Submodules Exercise

- Create a Repo called “Proj_with_Submodule” and push it to the server
- Create a submodule linking one other existing Repo (choose one which has multiple branches)
- $ git submodule add git@github.com:rchidana/Driver_Module.git SubModule
- $git status
- $cat .gitmodules

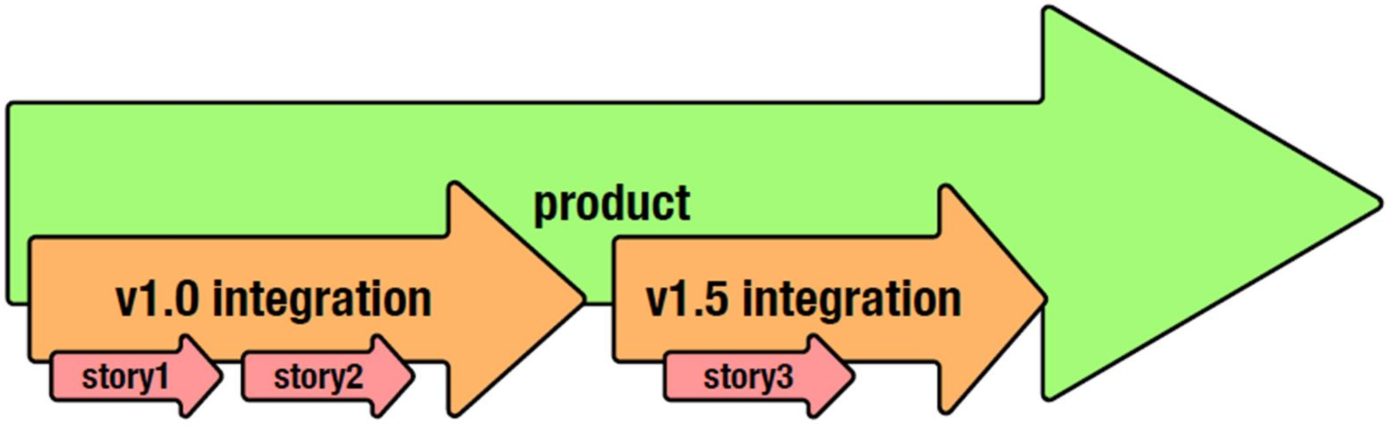
Git Submodules Exercise

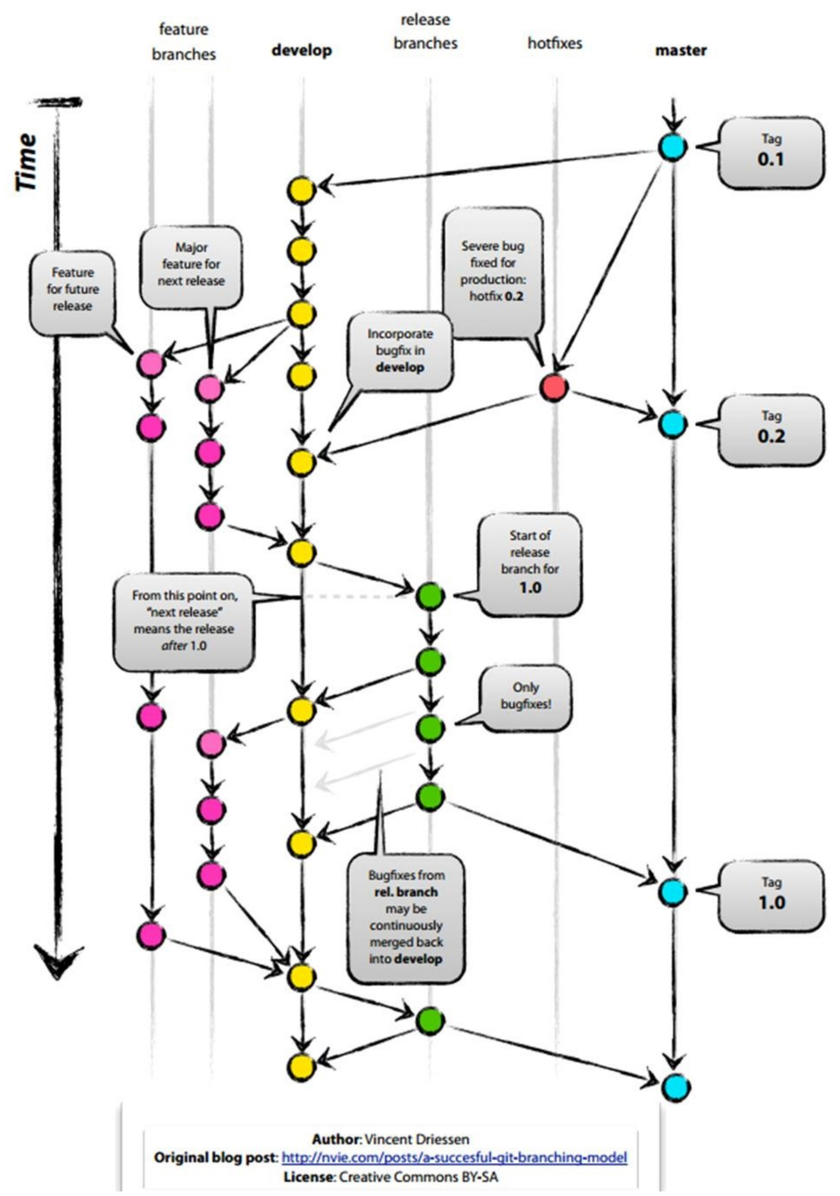
- $git submodule init (older versions)
- $git submodule update { pull in latest code}

- Switch to different branches present in the submodule!!

Git Branch Model

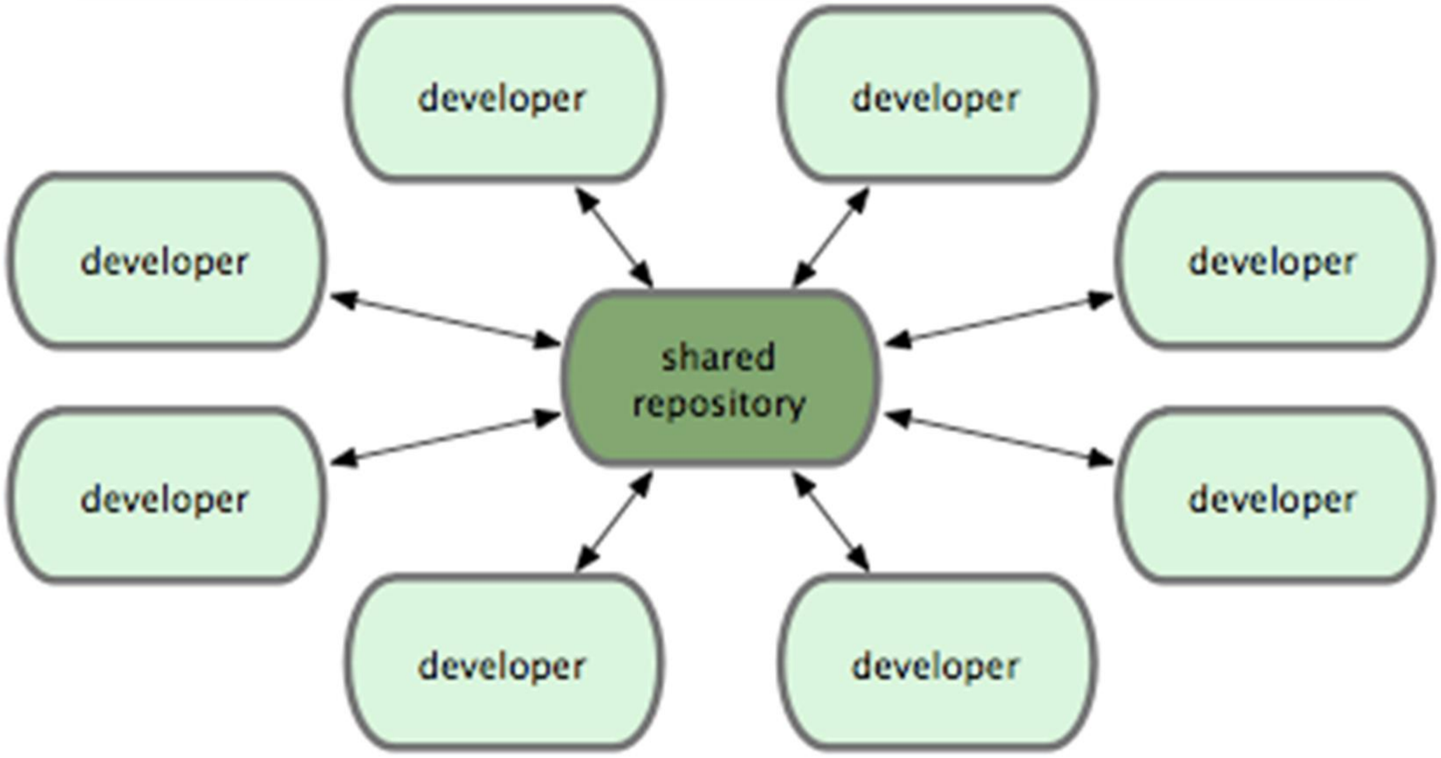
- git branching is very cheap (resource wise)
- branch based on your need



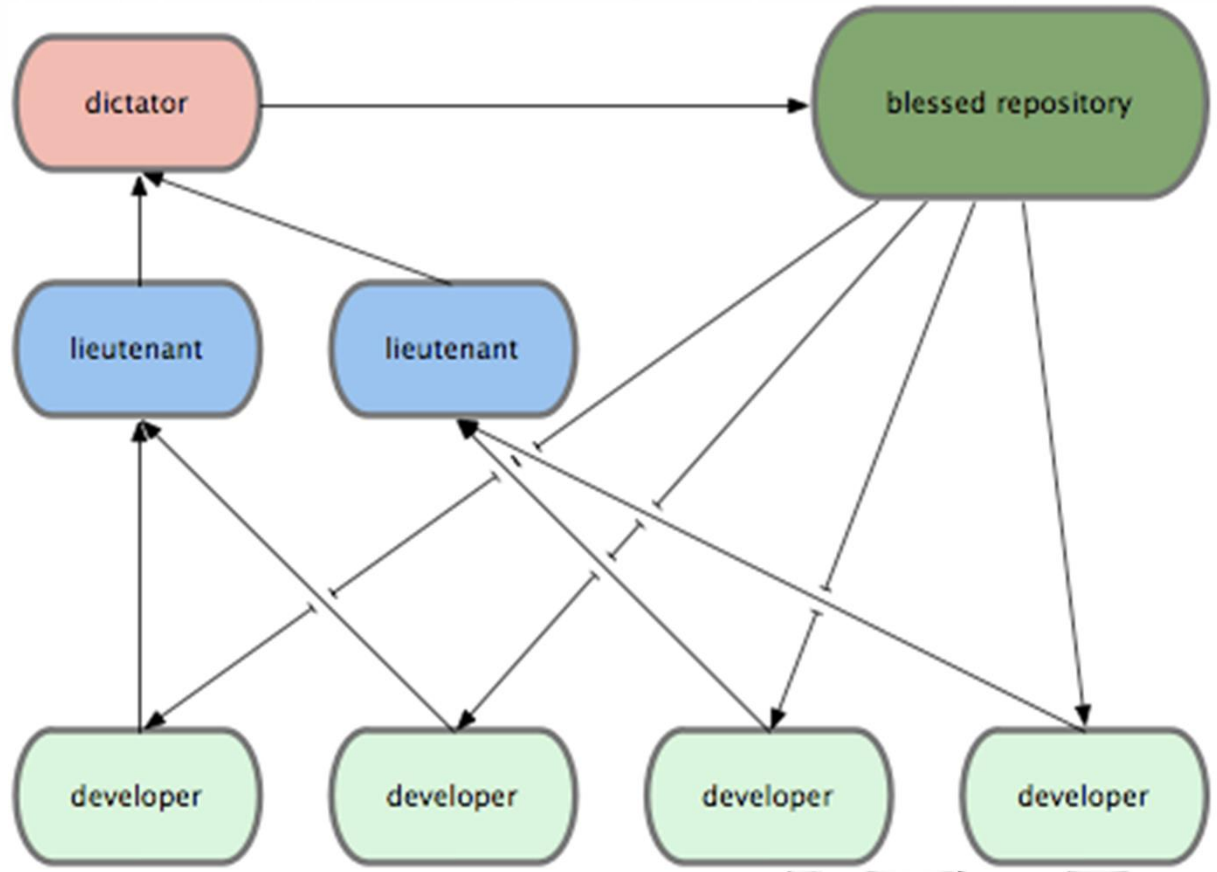

feature branches
develop
release branches
hotfixes
master
Time
Tag 0.1
Feature for future release
Major feature for next release
Severe bug fixed for production: hotfix 0.2
Incorporate bugfix in develop
Tag 0.2
Start of release branch for 1.0
From this point on, "next release" means the release after 1.0
Only bugfixes!
Bugfixes from rel. branch may be continuously merged back into develop
Tag 1.0
Author: Vincent Driessen
Original blog post: http://nvie.com/posts/a-succesful-git-branching-model
License: Creative Commons BY-SA

Git Branch Model

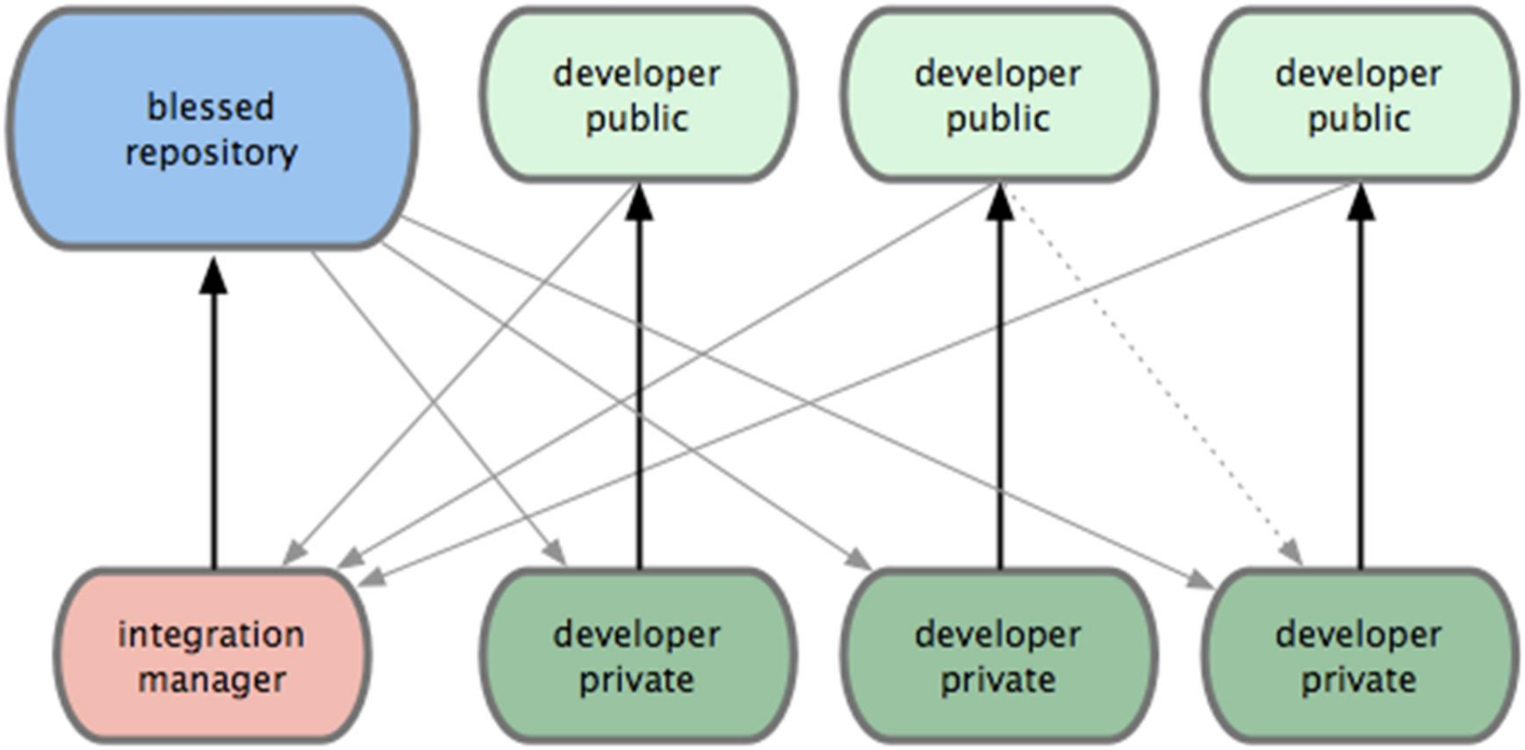
Git Repo Usage – Centralized workflow



Git Repo Usage – Blessed workflow



Git Repo Usage – CI Managed



Git Organizations

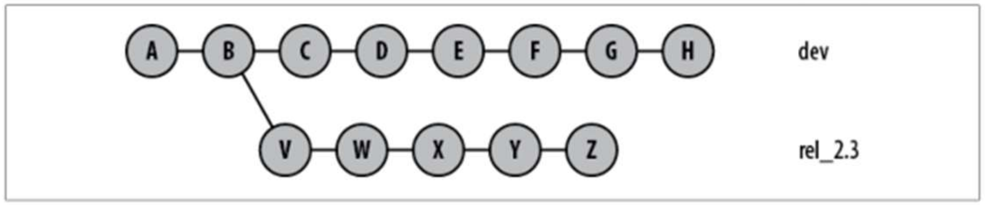
- Granular access to GitHub Repos
- Teams & Organizations can be set up with various permission levels
- Demo

Git PULL Requests & Code Reviews

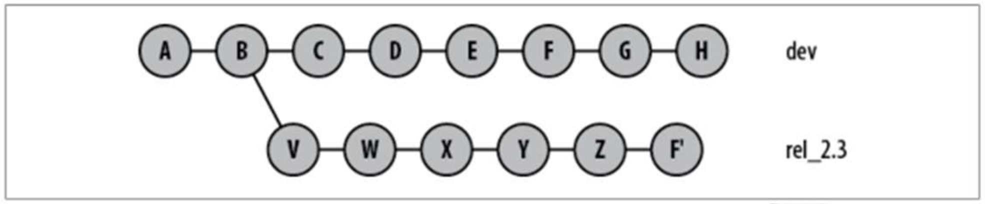
- Mainly applicable for forks
- Demo & Exercise on how to raise Git PULL Request

Git Cherry Pick

- Pick & Choose commits that you want to apply



```
$git checkout rel_2.3
$git cherry-pick D,F,H
```



Git diff

- Lists out differences between various commits
- $git diff HEAD..HEAD~1
- $ git diff 147ce61..03c9a6d (compares 2 commits)
- $git diff master..develop   (compares branches)

References

- Git SCM Book : https://git-scm.com/book/en/v2

BACK UP SLIDES

Git Hooks

- Git has a way to fire off custom scripts when certain important actions occur
- Client & Server Side Hooks
- Client-side hooks are triggered by operations such as committing and merging, while server-side hooks run on network operations such as receiving pushed commits
- Client scripts are present in .git/hooks directory

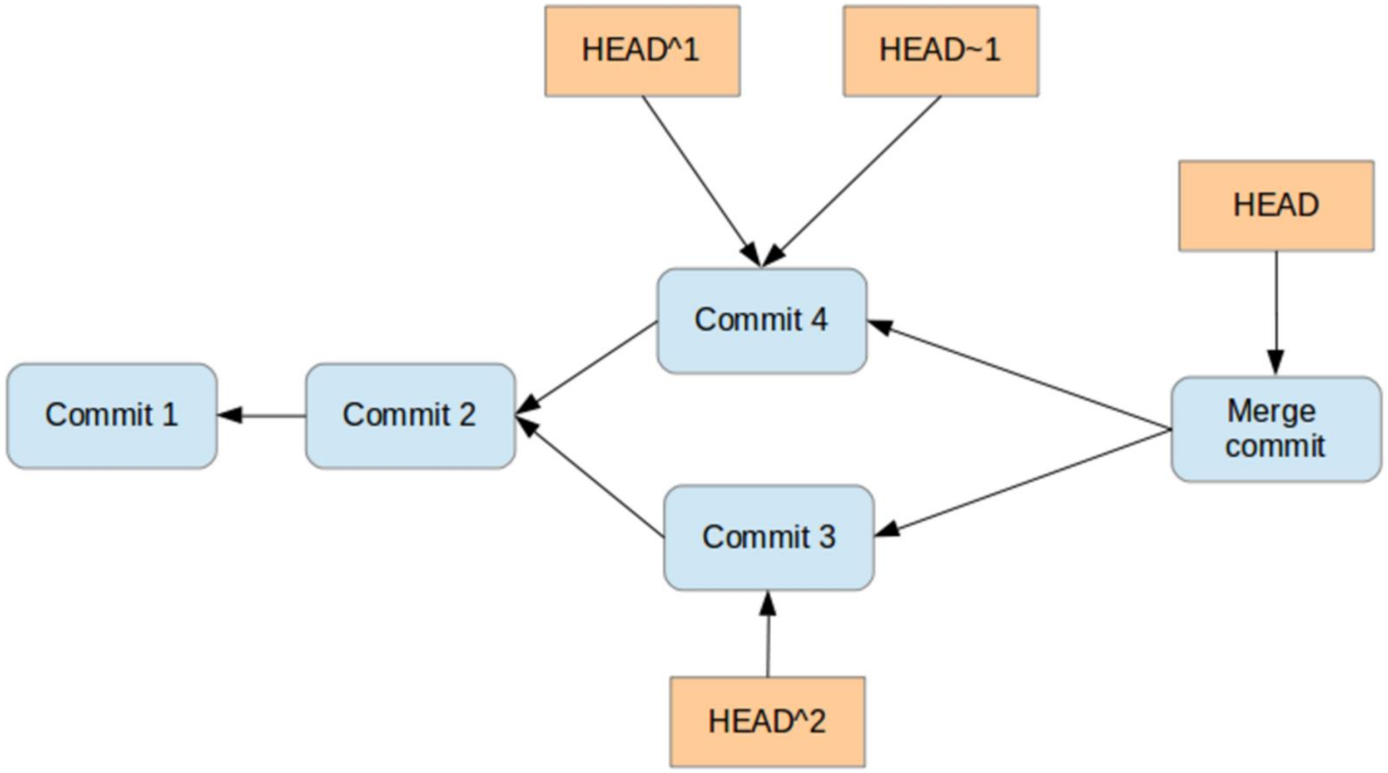
Git reverse workflow – unstage changes

- Add 1 new file & edit another file and git add changes for staging
- $git diff –-staged (shows changes that are being staged)
- $git reset HEAD (unstage changes)
- $git status
- $git checkout –- file (undo edits & get back to previous snapshot)
- $git clean –n (checks & prompts) while –f (removes new file)

Git HEAD~ and HEAD^

- Using Tilde & carrot symbols for commit references
- ~ represents commits older than some reference
- ^ represents parent of a commit and if commit is not a merge, it is illegal
- $git log HEAD~2..HEAD -> Will spit out difference between present HEAD & two commits behind it

Git HEAD~ and HEAD^



Finding all Ignored files

- git status –-ignored

Caching Git Credentials

- Git provided credential helpers so as to avoid repetitive typing in of username & password for Git operations
- $ git config credential.helper 'cache --timeout=300'

- If you want to clear the cache

- $git credential-cache exit

Storing Git Credentials

- Git provides an option to store your passwords unencrypted on disk, protected only by filesystem permissions
- $git config credential.helper store
- User needs to enter credentials once after which there is a .git-credentials file that is created in user home directory which stores credentials in this format :

https://user:pass@example.com