Git with GitHub

By

Anand

Agenda

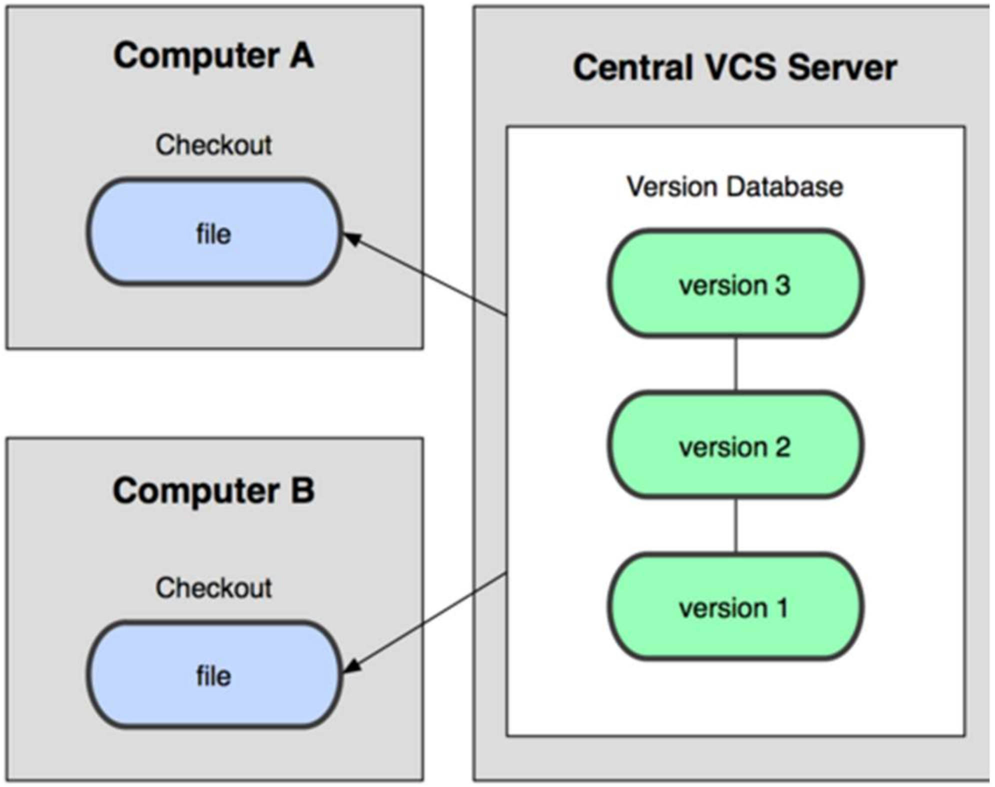
- Introduction to Version Control & Git
- Client server v/s Distributed Version Control
- GitHub set-up, registration, SSH Keys, Repo Set up and 101 ops
- Git Internals, snapshot storage, .git folder contents, git config
- Conflicts – when & why, how to resolve conflicts
- Introduction to Branches, Web-Hook Set up

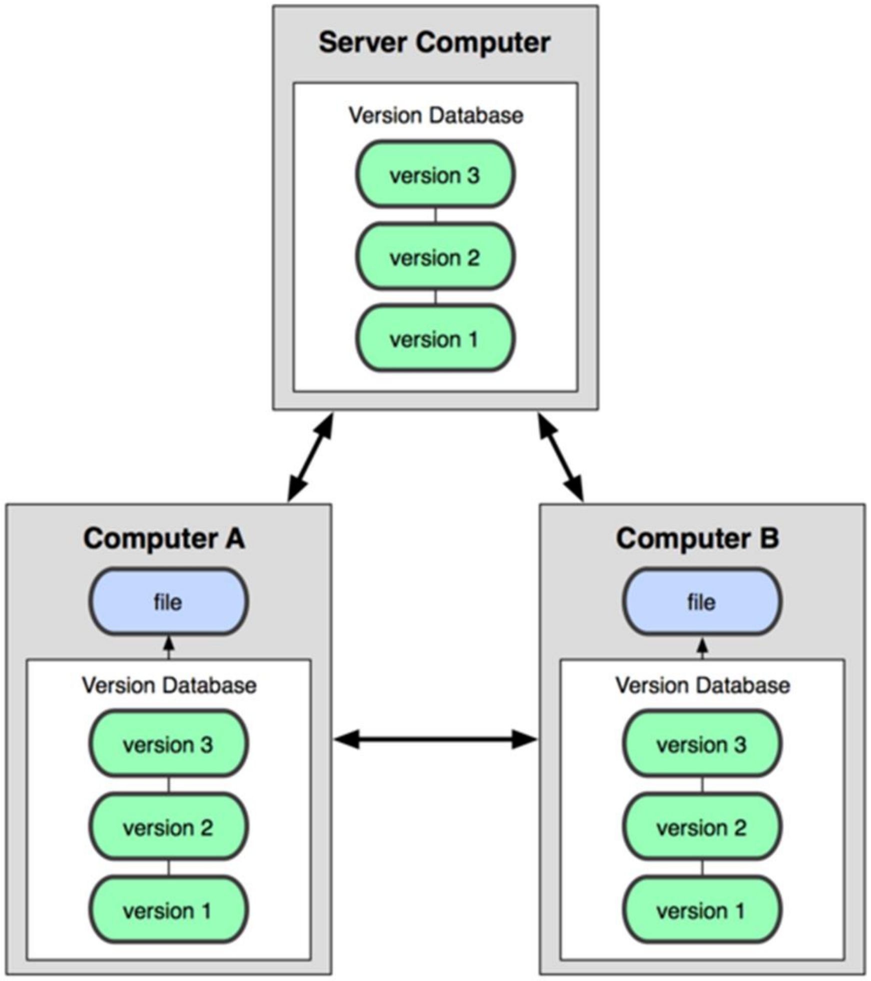
Why Source Version Control?

- 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 v/s Distributed VCS





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/Operations
- 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 b**d, 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 Flavours

- 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/download/win
- Git Bash is the bare client that we will use for most of our lab exercises
- Create a Directory – “Training” on your computer and please use this directory for all Lab exercises

Git – Config username & email

- First time configuration for Git Client

```
>git config --global user.name “John Doe”
>git config --global user.email johndoe@gmail.com
```

- Check if the configurations are set correctly

```
>git config --list
```

Create First Local Repository

- Create a folder – ‘FirstRepo’ under ‘Training’ folder & cd to it

```
>git init
```

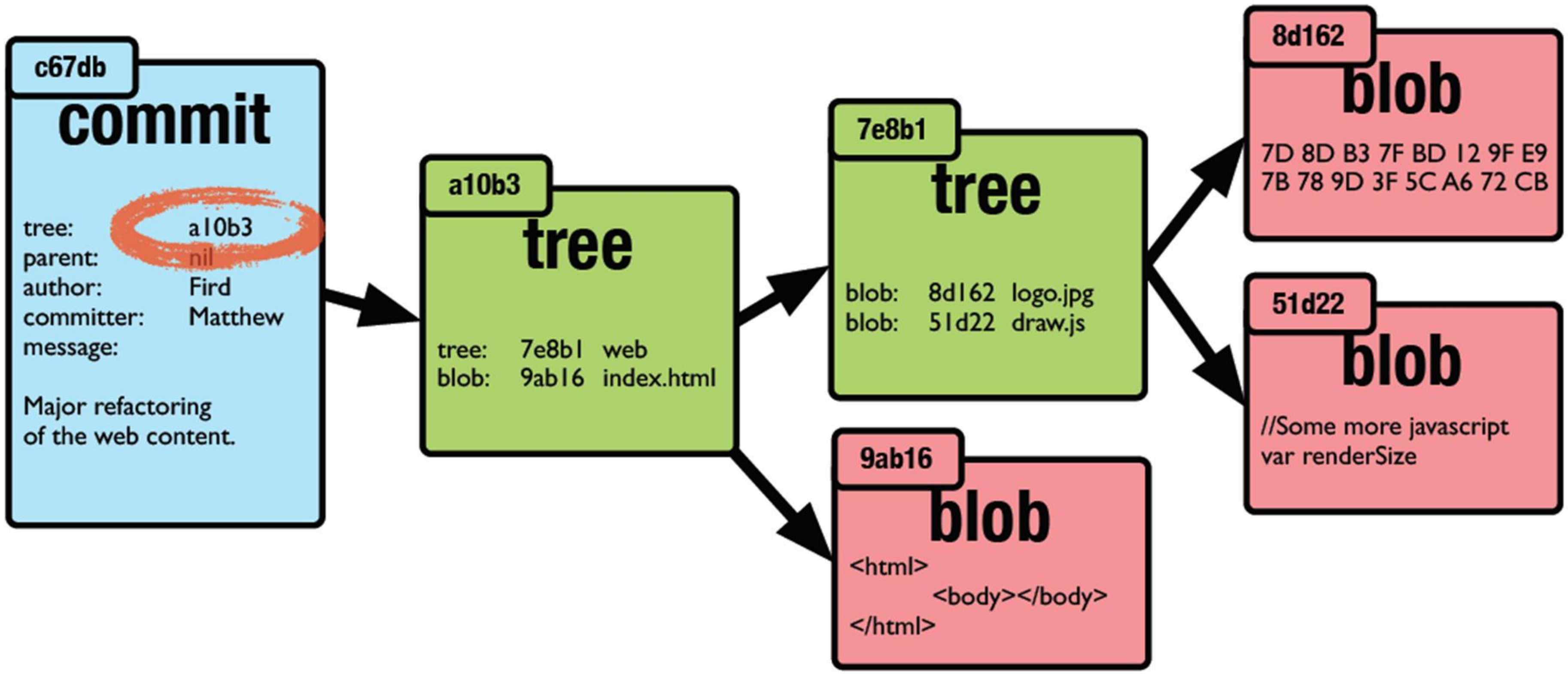
- Creates the famous .git folder, add two files to this Repo

```
>git add .
>git commit –am “First Commit”
>git status
>git log --oneline
```

Git Internals

- 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 Object Model



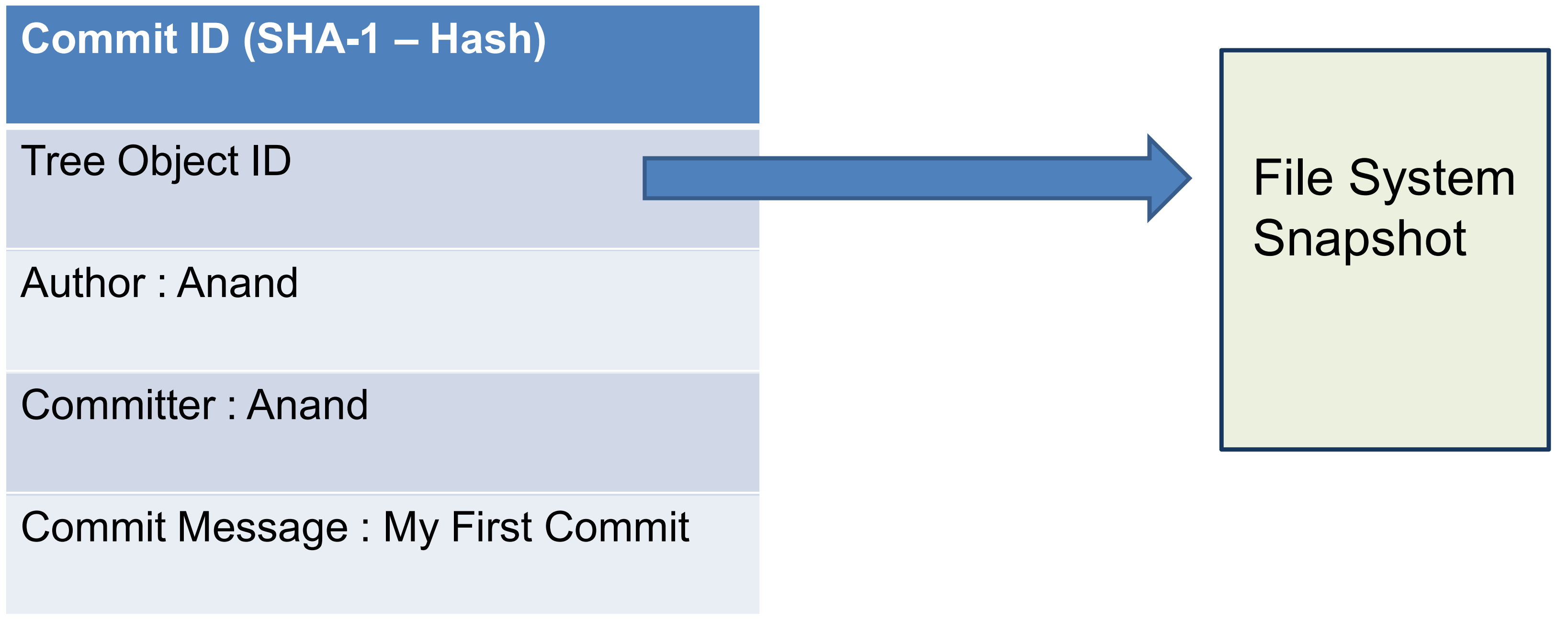
GitHub Registration

- Sign up for GitHub Free personal Account
- Please use your personal email IDs
- Remember your username & password

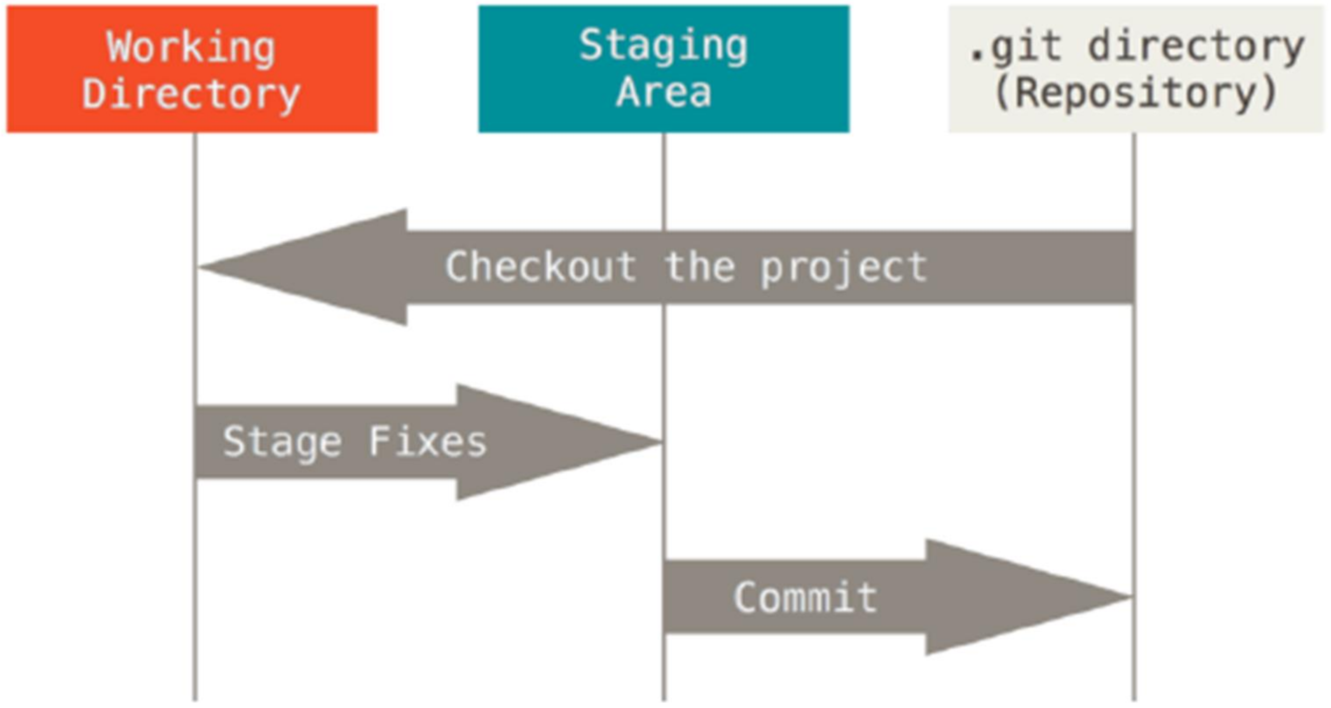
GitHub Create FirstRepo

- Create FirstRepo placeholder on GitHub Server
- Connect the client with GitHub server and push local Repository on to the Server using HTTPS URL

Git Commit

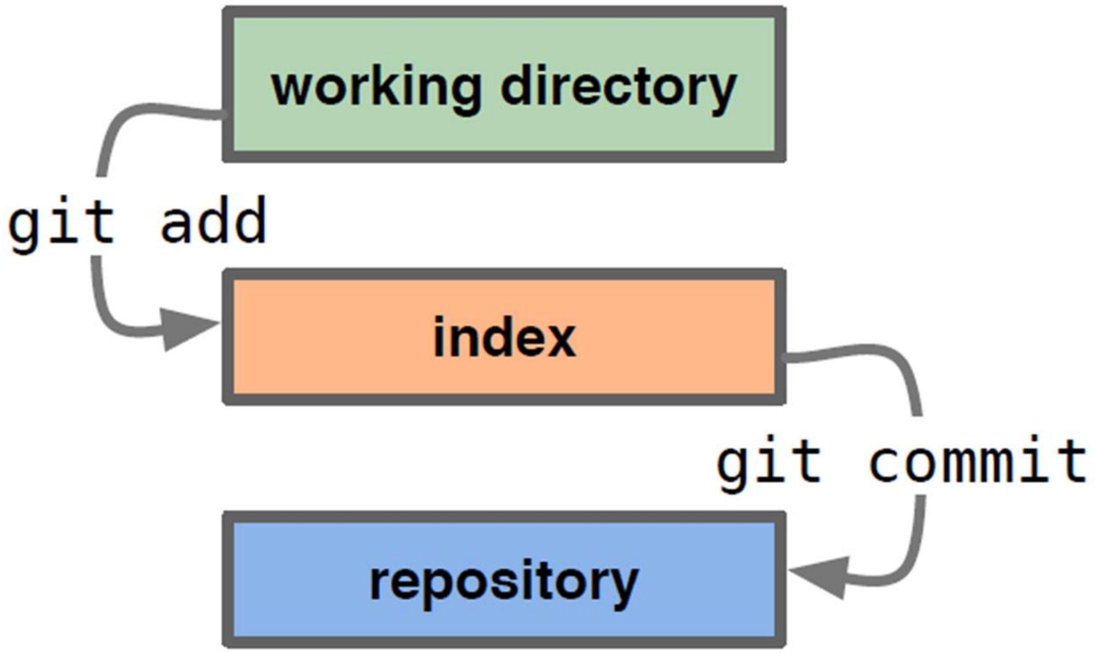

Commit ID (SHA-1 – Hash)
Tree Object ID
Author : Anand
Committer : Anand
Commit Message : My First Commit
File System Snapshot

Git – Repo Status

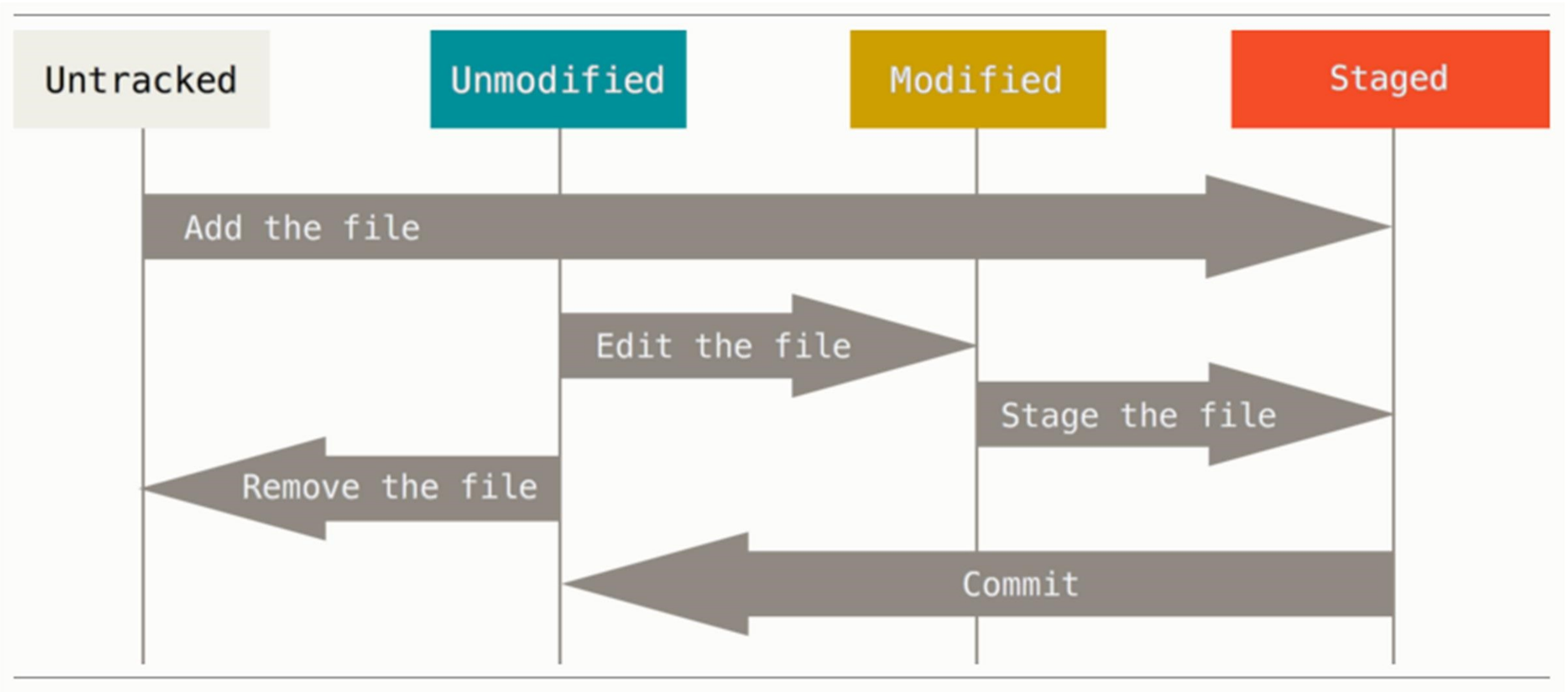


Modified, Staged, Committed

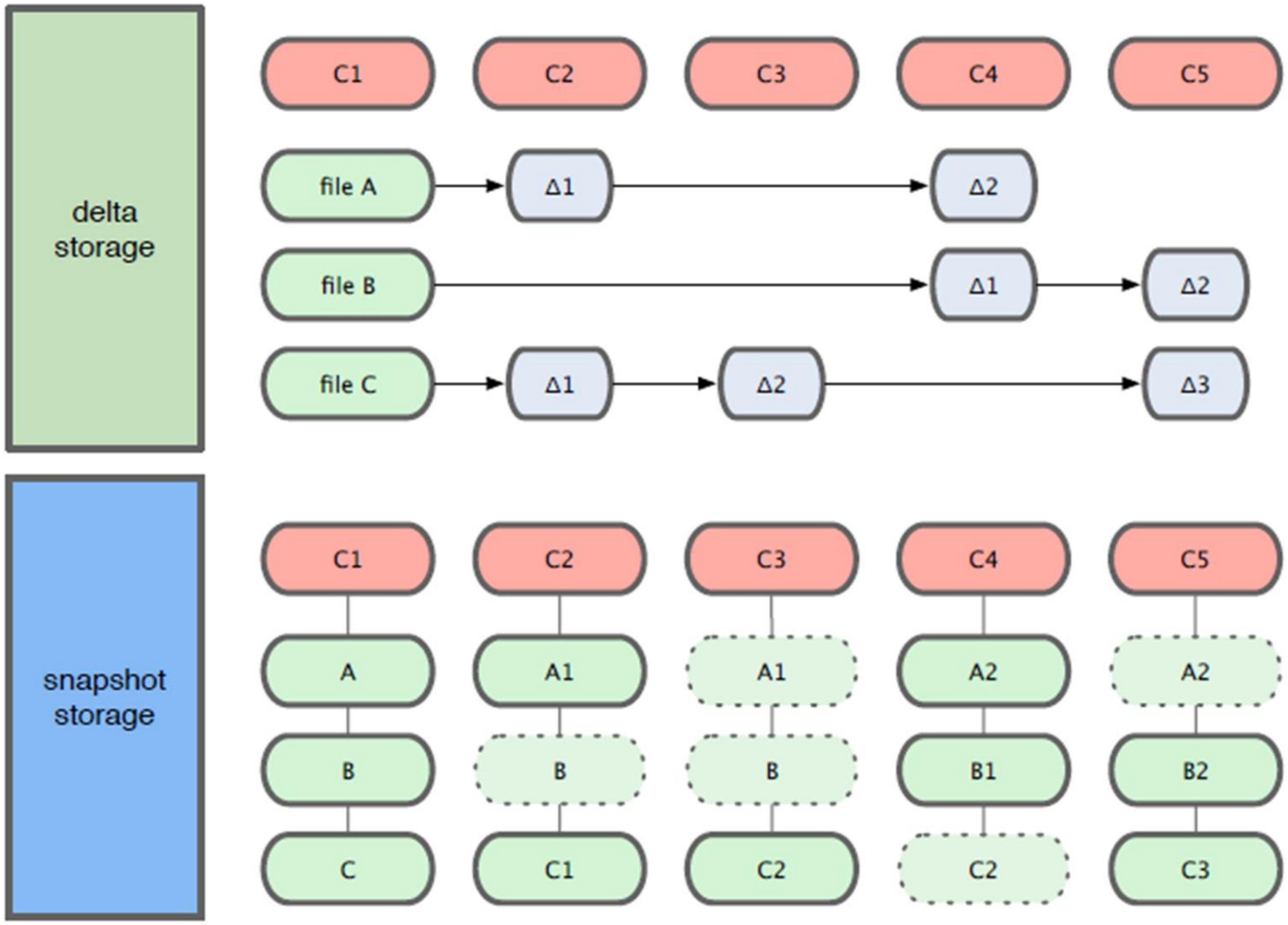
Git Working Model



File Lifecycle



Git – Snapshot Storage



Git Server Details

- Architecture of Git Client, Server & Workflow!!

GitHub SSH Set-Up

- BACK-UP Existing Keys (found in USER-HOME\.ssh directory)

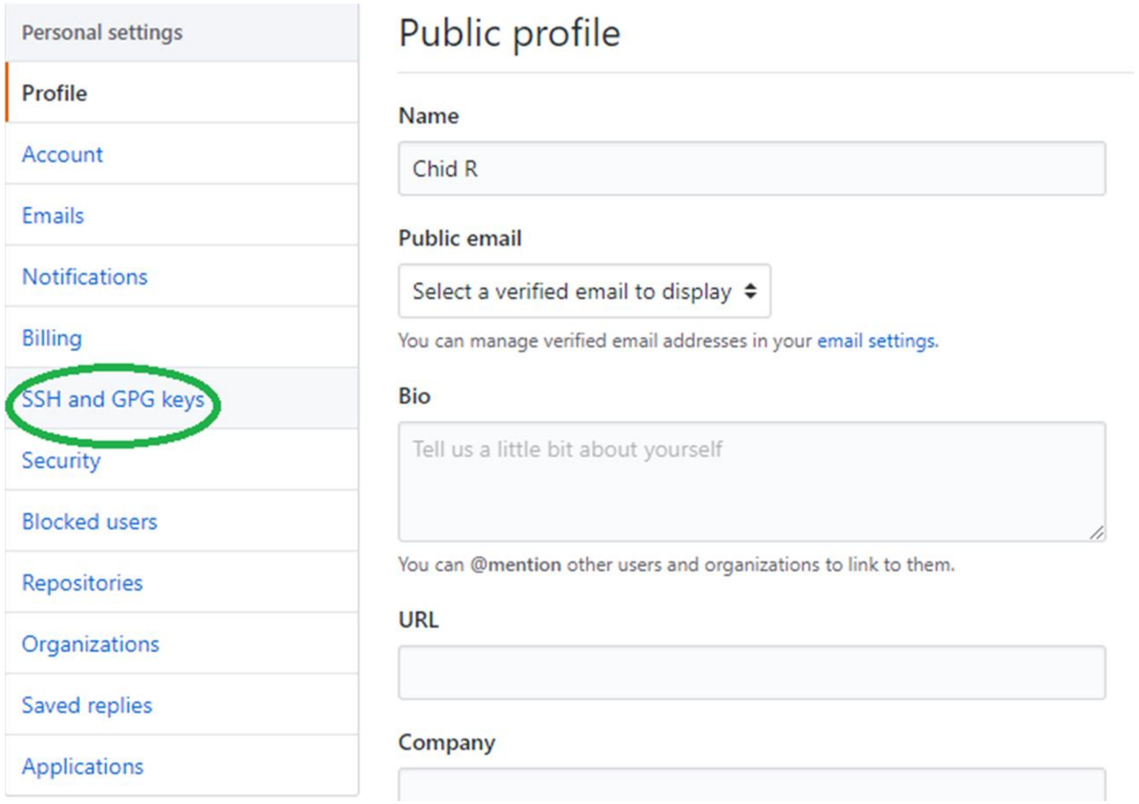
- Create a pair of SSH keys using the following command

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

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

GitHub SSH Set-Up

- Copy SSH Public Key from your desktop and paste it on your GitHub Account -> Setting -> SSH & GPG Keys



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

- Check your SSH connection using the following command

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

Well Done!! Your SSH setup is complete!!

GitHub SSH clone

- Create a new Repo called “SecondRepo” on the Server
- Initialize this with a README
- Clone this Repo onto your Client using its SSH URL
- Add/Update files, commit and then PUSH to server

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

>git log -> Shows details of committed snapshots

>git log –-oneline

>git log –-stat

>git log <since>..<until>   -> Diff between 2 commits

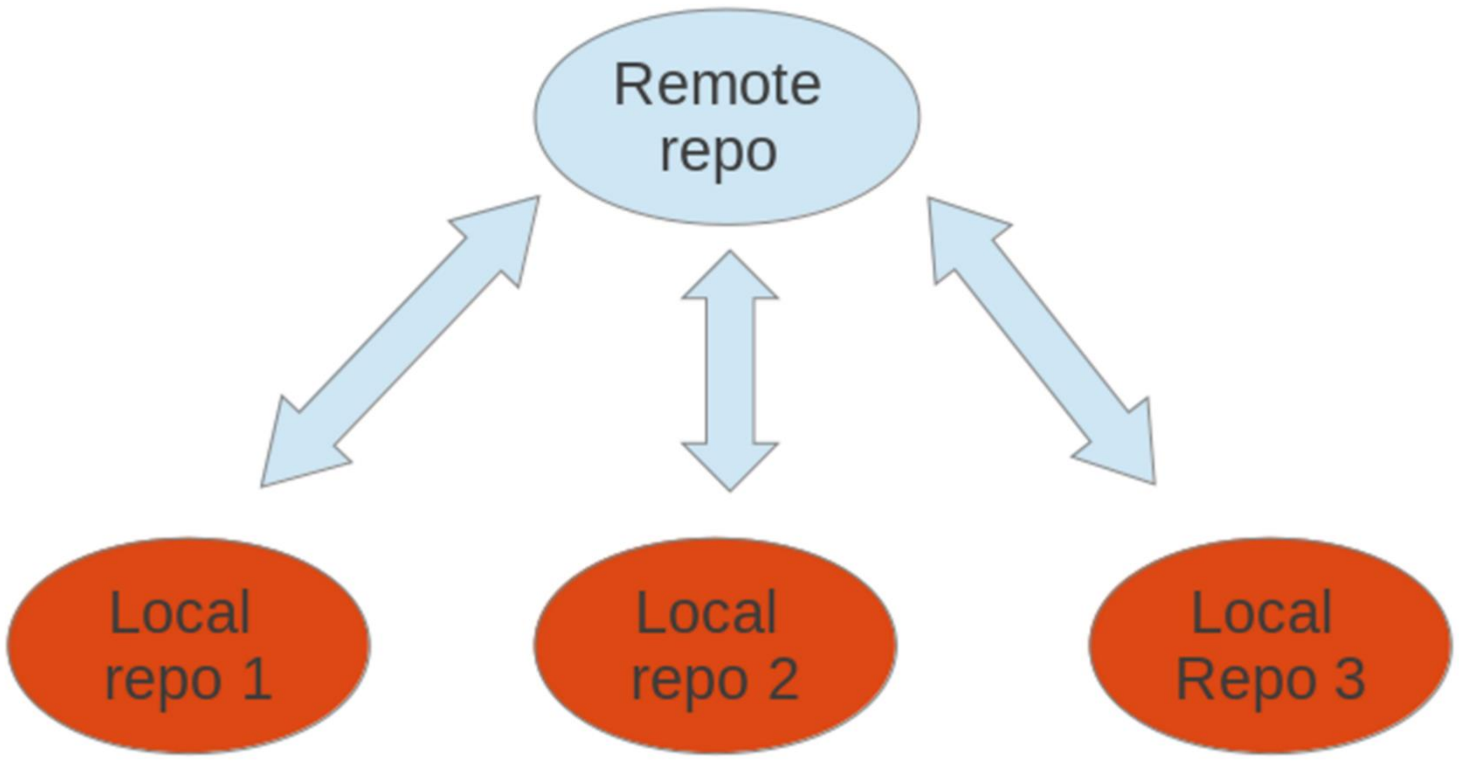
>git log --graph -–decorate -–oneline

>git log <filename>

Git Inspection commands

```
>git log –author="Chid R"
>git log –author="Chid R" –p Hello.py
>git log --oneline master..origin/Bug_Fix  -> displays all
commits that are in Bug_Fix but not in master
>git show SHA-OF-SOME-COMMIT
> git show SHA-OF-SOME-COMMIT --name-only
```

Git Remote



Git Remote - Origin

- Remote is nothing but a reference to Server with which the local Repository is tied up
- Every Local repo SHOULD have at least one remote called “origin” which is the default location to which the local changes will get synced up
- Can Local Repos have more than one Remote??

Git remote

- Adding a new remote

```
>git remote add REMOTE-NAME URL
>git remote –v
>git remote set-url REMOTE-NAME NEW-URL
>git remote remove REMOTE-NAME
```

Git and empty folders

- How does Git treat folders?? With content in it and otherwise??
- Create a new folder in one of the repositories and check if this folder gets tracked??

.gitignore

- Add .gitignore file when you want to git to stop tracking files/changes

Regex Syntax:

log/* *All files under log directory will be ignored*

.classpath

.project

*.DLL

*.dll

*~ All files ending with ~ will be ignored

Git Conflict 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 – Blogs
- Add one file 'Info.txt' with some content in it
- Push these changes to GitHub Server
- User1 will set up WRITE permission for User2

Git Exercise

No fun working all alone!!

- User2 will clone “Blogs” Repository
- User1 & User2 will now add the following files to the Repo
  - ❖ Name.txt and Address.txt
- User1 and User2 will commit changes locally and then push changes to the server at the same time!!
- What happens??

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 Conflicts - Exercise

- What are Code Conflicts??
- How to resolve them??
- Exercise to create conflicts and resolve them amicably

Branches – Killer feature of Git

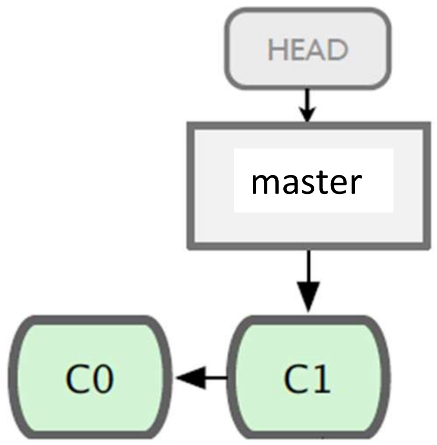
- What is a Git branch & what is the need for creating branches?
- $git branch new_branch (creating new branch)
- $git checkout new_branch (switching to new branch)

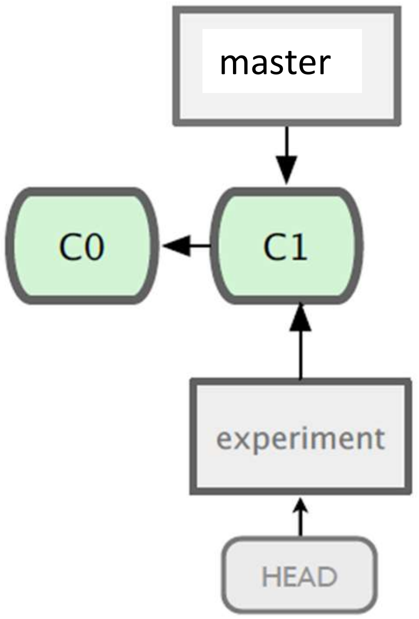
Or

- $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 Branch_Hello Exercise

- Create a Repo called "Branch_Hello"
- Add a file "Hello.bat/Hello.sh" to it and commit it
- Create two new branches from "master" called "Defect" and "RFE"
- Add Branch specific print statements in the bat/sh file
- Commit them to specific branches & push them to server

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

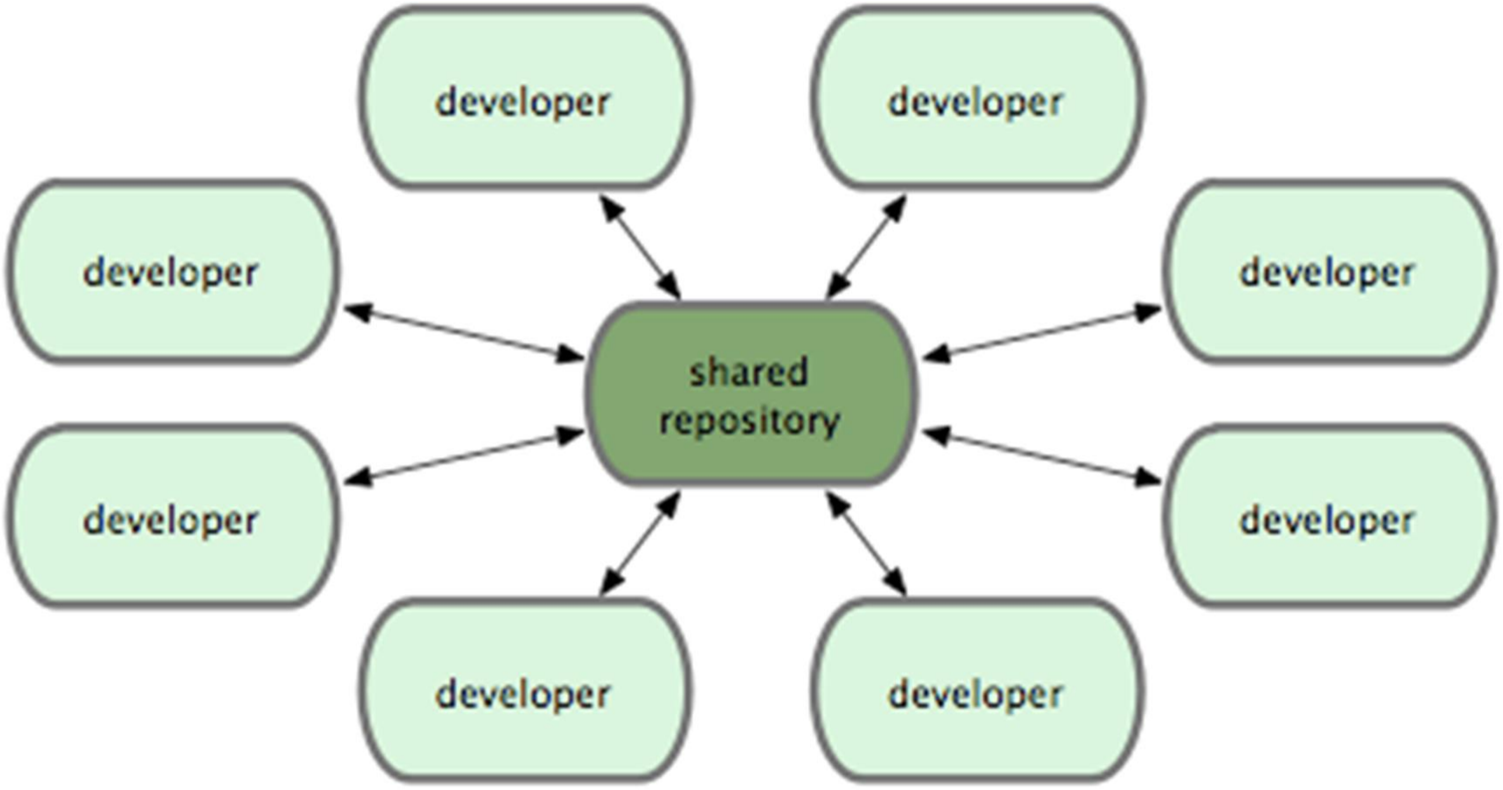
```
>git checkout C3
```

Gitflow

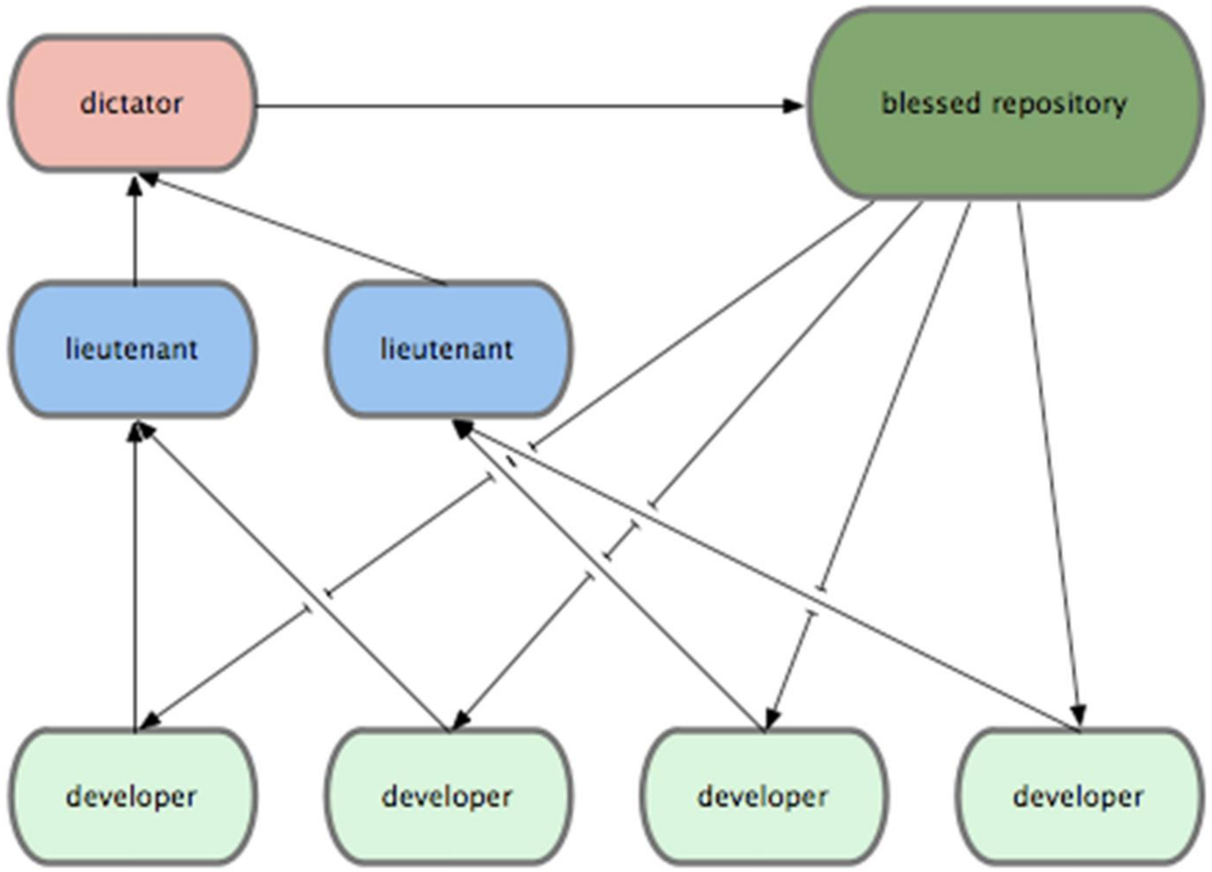
- Recommended Branching Model
- YouTube Video :

https://www.youtube.com/watch?v=1SXpE08hvGs

Git Repo Usage – Centralized workflow



Git Repo Usage – Blessed workflow



Git Cherry Pick

- Pick & Choose commits that you want to apply
- Works across Branches

```
>git cherry-pick SHA1 SHA2
```

- Dependencies??

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

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