Git

* source code management, version control system, its working process and advantages.
* advantages over other SCM tools. SVN
* Installed and configured Git and GitHub at organization.
* Used Git extensively in each and every project to store all kinds of code.
* Having good understanding of git terminology.
* Well aware of git stages like work space, stage/index, local repository and central repository.
* Complete understanding of snapshots and commits concepts.
* Attached tags to refer commits as it is very difficult to remember commits ID's.
* Good practice knowledge in using git commands like git pull, git push, git fetch, git clone, git log and other git commands.
* Well aware of git concepts like ignoring git files, git stash.
* Complete hands on experience on dealing with git branching, merging branches, switching branches and resolve merge conflicts.
* Good understanding between git reset and git revert and also cleaning git repository.
* Good knowledge in some advanced concepts like git rebase, git bisect, git squash and git cherry pic.
* Good hands on experience in working with GitHub and BitBucket.

Maven

* Installed and configured Maven.
* Well aware of advantages of using build tool over manual builds.
* Good understanding of architecture of maven.
* Complete depth understanding of maven build life cycle and maven goals.
* Hands on experience in creating maven directory structure and maven local repository to store dependencies.
* Knowledge in using maven central repository.
* Good understanding of maven's main configuration file POM.XML Dealt with multi-module projects by using maven.

Jenkins

* Installed and configured Jenkins in both windows and Linux machines.
* Complete understanding of Jenkins complete work flow as well as its advantages over other CI-CD tools.
* Good hands on experience in integrating many tools with Jenkins like git, maven, selenium, junit, tomcat webservers.
* Installed and configured Java as java is pre-requisite to install Jenkins.
* Installed and configured build tools like Maven and Ant and integrated with Jenkins.
* Good practice experience in creating free style projects and maven projects and dependency projects.
* Established complete Jenkins CI-CD pipeline and complete workflow of build and delivery pipelines.
* Installed various Jenkins plug-ins from Jenkins community as Jenkins is all about plug-in and play.
* Configured many scheduled projects so that they can run frequently without manual trigger.
* Responsible to establish complete pipeline work flow starting from pulling source code from git repository till deploying end product into servers.
* Created many link projects and configured upstream and downstream projects.
* Customizes Jenkins home page and created my own view as well as nested views.
* Created many user accounts in Jenkins and given limited privileges to them to ensure security is there in each and every stage by using Jenkins roles.
* Created many slaves to take the work load from Master Jenkins server.
* Good hands on experience in deploying end product in tomcat web server and other application servers.

**Docker**

* Having good practice experience in installation and configuring of docker.
* Well aware of the advantages of docker over other virtualization technologies in the market.
* Good knowledge in docker concepts like OS level virtualization and layered file system which docker follows.
* Good understanding of docker, its components and docker work flow.
* Created many docker containers form docker images.
* Having good hands on experience in using docker images from docker hub as well as creating our own docker images form docker containers.
* Good understanding in working process of docker file and also created docker images from docker files.
* Created private docker registry to store docker images so that only project members can able to access to make sure that security is at its peaks.
* Created docker volumes to provide high availability to data even if any of the docker container goes down.
* Good hands on experience in sharing docker volumes among containers and between container and host.
* Used docker port mapping to expose port to outside of docker container to access website which is running inside docker container.
* Created own local docker registry server by taking registry image from docker hub.
* Good knowledge in pulling and pushing docker images from and to docker hub.
* Having very good hands on experience in using all kinds of docker instructions in docker file.
* Good knowledge in creating demonized docker containers.

**Kubernetes**

* Responsible to install and configure Kubernetes in physical as well as cloud environments.
* Well aware of advantages, architecture and complete work flow of Kubernetes.
* Good knowledge in each and every component of Kubernetes.
* Installed and configured K8S master and K8S nodes and established communication between them.
* Good understanding of K8S master node components like kube-api server, kube-scheduler and etcd store which forms control plane.
* Knowledge in K8S node components like Kube-proxy, kubelet and container engine.
* Good understanding of single container pod as well as multi container pods.
* Well aware of pod limitations and how to address those issues by using high level K8S abstractions like Replica sets, Deployments, Volumes and Services.
* Good knowledge in achieving auto scaling and auto healing.
* Well aware of upgrading versions that we call rolling updates and as well as roll back concepts.

**Chef**

* Established complete Chef environment in my organization
* Installed & configured Chef Workstation, Chef Server and Bootstrapped Chef nodes.
* Having complete hands on experience in working with managed chef server.
* Having good knowledge in working process of Knife, chef-client, Ohai tool and Idempotency concepts.
* Written so many cookbooks from scratch as well as used many cookbooks from Chef Supermarket.
* Managed dependencies by using Berks file.
* Having good practical experience with ruby scripting. Used extensively in writing recipes to create chef resources.
* Deployed Apache web servers by using community cookbooks.
* Well aware with the concepts like Metadata file and Run lists.
* Created configuration files by using Chef Attributes and deployed in chef nodes.
* Used wrapper cookbooks for calling chef supermarket cookbooks instead of downloading them.
* Well aware of best practices with respect to the usage of chef.
* Completely automated the running of chef-client instead of calling every time manually.
* Having very good knowledge in dealing with advanced concepts in chef like roles. Created many roles in the process of achieving complete automation.
* Well aware of advantages of configuration management process and usage of chef tool..

**Ansible**

* Good hands on experience in installing and configuration of Ansible.
* Well aware of advantages of ansible over other configuration tools.
* Good knowledge in dealing with inventory file and host patterns in adding hosts to inventory file and calling hosts.
* Extensively used Ansible ad-hoc commands instead of using ansible playbooks for each and every small task in ansible.
* Good knowledge in using multiple ansible modules as and when required and their usage.
* Good knowledge in Idempotency concepts of ansible.
* Complete hands on experience in writing YAML script in various playbooks.
* Well aware of using various sections in ansible like target section, task section and etc.
* Used variable section in many playbooks to take the advantage of variables instead of hard coding.
* Used handler section while managing dependencies in ansible playbook.
* Extensively used other ansible concepts like loops to deal with multiple tasks simultaneously and conditionals.
* Used ansible vault to secure information like passwords and secret key files.
* Good complete hands on experience of using Ansible roles by including all sections inside ansible playbook.

**Nagios**

* Installed and configured oldest and latest monitoring tool called Nagios.
* Well aware of complete workflow, architecture and advantages of Nagios monitoring tool.
* Used many Nagios plugins to monitoring different services in different hosts.
* Good understanding of Nagios dashboard and monitoring things.
* Created multiple services groups and multiple services groups.
* Well understanding of Nagios directory structure and capable of managing that directory structure.

**AWS**

Having very good hands on experience on EC2 (Elastic compute cloud), ELB (Elastic Load Balancer), Auto Scaling, S3 (Simple storage service), Glacier, Cloud Front, IAM (Identity and Access Management), VPC (Virtual Private cloud), Route-53, SNS (Simple notification service), Cloud formation, Elastic beanstalk, EFS (Elastic File System), Cloud Watch and Trusted Adviser. Well aware of SQS (simple queue service), SES (simple email service), RDS, Dynamo DB, Red shift, Elasticache, Ops work, white papers, Snow ball, AWS CLI and Elastic transcoder.

**VPC**

* Taken lead in migration process of servers and data from on premise data center to AWS cloud.
* Responsible for complete administration of Cloud infrastructure in my organization.
* Created VPC from the scratch and connected to network by using Internet Gateways, Route tables, and NATs
* Created many public and private subnets for proper segregation of webservers and database servers to provide high level security.
* Defined IP ranges in VPC to have better control over VPC.
* Good Hands on experience in VPC peering to connect multiple VPCs so that all act as one single entity.
* Enabled VPC Flow logs for the auditing purpose to track incoming and outbound traffic to and from VPC.
* Launched Bastion servers/Jump servers in public subnets to have ssh connection to the servers which are present in Private subnets.
* Launched Web servers in Public SN through Auto scaling and connected to load balancer to distribute traffic as well as to provide high availability.
* Launched Database servers in Private SN and provided internet through NAT server.
* Good understanding in dealing with NACLs (Network Access control Lists) and Security Groups to restrict and allow ports to provide security at subnet level and instance level respectively.

**EC2**

* Having very good knowledge wrt EC2s like launching windows and Linux machines and all five types of Elastic Block store (EBS) volumes and their differences.
* Launched all three kinds of load balancers and attached to webservers to distribute traffic as well as to check the health of EC2 instances to make sure that they are always up and running.
* Extensively used Launch configurations and Auto Scaling to provide high availability to EC2 machines and effectively used scaling policies based on web traffic.
* Hands on experience in creating Snapshots to take back up copy of our EBS volumes.
* Created AMIs and Volumes and played with them like attaching, detaching, creating own AMIs for replication of same environment in same/different Availability Zones as well as same/different regions.
* Well understanding of both system status checks and instance status checks and how to trouble shoot if system or instance fails.
* Encrypted volumes to provide security from unauthorized access and misuse of data and provide protection from accidental deletion of machines.

**S3**

* Migrated and stored all kinds of object storage of my organization in to S3 bucket to provide durability and security.
* Enabled versioning on some important data to provide security from accidental deletion and to roll back to previous versions.
* Good knowledge in dealing with Access Control Lists (ACLs) and Bucket policies to restrict unauthorized access in to our own buckets.
* Enabled CRR (Cross Region Replication) to replicate data to other buckets which are present in different regions.
* Launched static web sites for testing purpose by using S3 static website hosting option instead of going always with EC2s, Load balancers and Auto Scaling.
* Good knowledge in dealing with Transfer acceleration to accelerate transfer speed of data into S3 buckets by using AWS globally distributed edge locations/end points.
* Well aware of different storage classes/tiers and effectively used life cycle management policy to transition data to different storage classes automatically after certain period of time that we set.
* Good knowledge in CORS to share resources across buckets without actually coping data. 2/2014 – 9/2015

**Cloud Front**

* Good hands on experience in working with Cloud Front to access the webpage without any network latency through AWS globally distributed edge locations/end points.
* Well aware in setting TTLs for the time period of object to be stored in edge locations.

**IAM**

* Complete hands on experience in managing IAM service to administer AWS resources effectively
* Created many user accounts, placed users into their respective groups and given limited privileges directly to users as well as groups to have better security.
* Well aware of all kinds of policies which are provided by AWS and their usage.
* Good knowledge in working with IAM roles to have password less access to all AWS resources to provide security.
* Used roles mainly to establish password less connection between S3 and EC2 for the data migration to and fro.

**Route-53**

* Good understanding in purchasing domain names form AWS as well as deep knowledge in DNS service.
* Created alias record sets to provide alias names for load balancers DNS name.
* Configured routing policies to provide high availability at region level and protects our infra from regional failure.
* Good hands on experience in working with all kinds of routing policies to route the traffic based on our requirement like Simple, Weighted, Latency, Failover and Geolocation routing policies
* Configured health checks to make sure that failover routing policy is working effectively.

**Databases**

* Well aware of working nature of RDS and establishing secure connection to web servers.
* Configured Multi-AZs to provide high availability without any downtime even if our database server fails.
* Configured Read Replicas to distribute read intensive load across read replicas to avoid load on primary database server.
* Frequently taking backups (both Automated as well as DB snapshots) to have backup copy for standby purposes and also replicating of same database server with same data in other AZs as well as Regions.
* Having knowledge in Dynamo DB and its working process.
* Started learning Redshift and Elasticache as these are very important AWS services wrt Data ware housing and caching engines respectively.

**SNS**

* Configured SNS notification at AutoScaing as well as Route-53 level to get notifications when server and VPC failures respectively.
* Complete hands on experience wrt creating SNS topics, creating subscribers and adding subscriptions.
* Integrated SNS with almost each and every AWS service to get notifications over different protocols like email email json and SMS.

**Cloud watch**

* Hands on experience in working with cloud watch to monitor all AWS services to maintain high availability and reduce downtime.
* Configured cloud watch alarms to get alert whenever any untoward situation arises which helps in addressing issue immediately.
* Aware of both default monitoring and detailed monitoring.
* Good knowledge in effective usage of all metrics which are being provided by AWS.

**Other Services**

* Complete hands on experience in working with EFS to provide shares storage so that each and every member in project will have access to common and centralized storage.
* Good experience in configuring cloud formation templates to create AWS infrastructure form JSON/YAML code.
* Well aware of concepts like converting infrastructure into code so that it is very easy to test, apply version control system as well as replication the environments.
* Good knowledge in handling Elastic Beanstalk to test the code as and when required without worrying about underlying infrastructure
* Taken help from trusted adviser to enhance security, cost control and provide better performance.
* Personally used Elastic Transcoder to convert one media format to another media format.
* Complete understanding of AWS white papers as they are providing many guide lines in terms of security, cost control and how to achieve operational excellence.
* Started working on OpsWork to convert infrastructure into code combining with famous DevOps tool called Chef.
* Having knowledge in data migration concepts like Snowball, Snowball edge and Snow mobile.
* Learned messaging services like SQS (Simple Queue Service) and SES (Simple Email Service)
* Used Glacier service to store objects which are not required for immediate retrieval in the process of cost control
* Good understanding in working with AWS CLI to create and manage AWS infrastructure.

**IAM**

* Complete hands on experience in managing IAM service to administer AWS resources effectively
* Created many user accounts, placed users into their respective groups and given limited privileges to directly to users as well as groups to have better security.
* Well aware of all kinds of policies which are provided by AWS and their usage.
* Good knowledge in working with IAM roles to have password less access to all AWS resources to provide security.
* Used roles mainly to establish password less connection between S3 and EC2 for the data migration to and fro. Launched all three kinds of load balancers and attached to webservers to distribute traffic as well as to check the health of EC2 instances to make sure that they are always up and running.
* Extensively used Launch configurations and Auto Scaling to provide high availability to EC2 machines and effectively used scaling policies based on web traffic.
* Hands on experience in creating Snapshots to take back up copy of our EBS volumes.
* Created AMIs and Volumes and played with them like attaching, detaching, creating own AMIs for replication of same environment in same/different Availability Zones as well as same/different regions.
* Well understanding of both system status checks and instance status checks and how to trouble shoot if system or instance fails.
* Encrypted volumes to provide security from unauthorized access and misuse of data and provide protection from accidental deletion of machines.

**Cloud watch**

* Extensively used cloud monitoring service called Cloud Watch.
* Monitored many things like CPU percentage, RAM space, disk space and many more.
* Created many cloud watch alarms so that will be alerted if any unusual thing happens and take necessary action to make sure in reducing downtime.
* Created different kinds of wizards like line, stacked and numbered wizards.
* Dealt with different kinds of metrics which are been provided by AWS.
* Good knowledge in using both default monitoring and detailed monitoring.
* Integrated cloud watch with so many other AWS services to make sure that there is high availability always..

**GIT**

-DVCS (Distributed version control System) & RCS (revision control System)

-To Prevent overwrite issue & store complete history of source code (.java .html .img .jpg snapshot in Sha value)

**GIT Architecture: Git's End-to-End work flow? : basic imp cmds**

**Source/working Directory=>** git add =>**Staging Area=>**git commit=>**Local Repo=>**git push=>**Remote repo**.

Remote repo/ GitHub (DVCS)

# git init / git init <Repo name> - Create local repository

**diff betw git init and git init - -bare?**

**-git init** create local repo with .git folder to start revision history

**-git init - - bare** it does not have working directory

# **git clone <Git URL>** -clone the project or source code from GITHub project

# **git add ./ <filename>** - file added with modification from source to staging area

# **git status**- to check git operation performed or not

# **git commit <file name> -m “commit”**- To commit the msg or comment on operation

# **git push**- changes may push from local to remote

**How to configure username, email and editor first time in Git?**

# **git config - -global user.name username** -– set user name

# **git config - -global user.email** [**user@gmail.com**](mailto:user@gmail.com) – set user emailID

# **git config - - list / # vim /home/user/.gitconfig** -to check all information about git user

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*# **git remote add origin “GIT URL”** –add git URL to push local repo to remote

# **git push –u origin master -**Whenever we need to push the changes to a remote repository, we use git push along with the remote repository “origin” and “master” branches

# **git remote –v** - to check remote repository is configured or not

# **git push - - set upstream origin master** --set-upstream is used to map a branch in your local to a branch on remote, so that it will know which branch to push/pull from.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**GIT vs SVN, Which VCS you prefer?** **SVN Or Git? Why? What are the advantages of Git over SVN?**

|  |  |
| --- | --- |
| GIT | SVN |
| Distributed VCS | Centralized VCS |
| It create local repo to store everything locally instead of using centralized server | It uses a centralized server to store changes in source code |
| N/W access is not mendatory for Git operation | N/W access is required for almost all SVN operation |
| Local repo and Stagging area | No local Repo and staging area |
| Server goes down still we can work | Server goes down we can’t do any operation |
| Branching Locally and Server side | Branching Server side only |
| Git stores each change as SHA values | SVN uses simple numbers as revisions |

1. Git is distributed VCS. In the sense, when user clones the repository, he will get source code as well as complete repository on his local disk. since repository is distributed across multiple users and each user has their own local repository, user will be able to submit the changes locally without connecting to the master server. (i.e he can also work offline) Where as SVN is a centralised version control system. In svn, when a user checkout’s the repository, he will get only the source code. Repository is not distributed to the user. In SVN multiple users connect to a single/centralized server. If that server goes down, no user will be able to work. 2. Git has staging area. Users can leverage this staging area to stage/save all the changes related to some issue/bug and commit everything at once into local repository. hence, all the changes will be recorded as part of single sha/commit. user's get an opportunity to logically group the changes and commit as single change set. This will enables better tracking. 3. Since git is distributed in nature, almost all transactions happen locally in the client machine and hence less overhead on the server so git is faster. Users are free to create any no. branches in Git. where as in svn, all branches reside in server side, so may cause svn to be overloaded with multiple requests. 4. Git seems to be good with merging. 5. Git stores each change as SHA values but SVN uses simple numbers as revisions.

**1) What is checkin and checkout in git?**

-checkin is putting changes back into the the local or remote repository (from local working directory).

 Checking-in code means to upload code to main branch repository so that its administrator can review the code and finally update the project version

- checkout is getting changes out from the local or remote repository (into local working directory).

checking-out code is the opposite which means to download a copy of code from the repository.

**2) Why we call Git as Distributed VCS?**

- In a DVCS(such as Git, Mercurial, Bazaar or Darcs), clients don’t just check out the latest snapshot of the files: they fully mirror the repository. Thus if any server dies, and these systems were collaborating via it, any of the client repositories can be copied back up to the server to restore it. Every checkout is really a full backup of all the data.

**What is Version Control System(V.C.S.)?**

- GIT is V.C.S. for tracking changes in computer files and coordinating work on those files among multiple people. - It is primarily used for source code management in software development, but it can be keep track of changes in any set of files.

**Why we need any Version Control System (v.C.S)?**

- It allows you to revert files back to a previous state, revert the entire project back to a previous state, compare changes over time, see who last modified something that might be causing a problem, who introduced an issue and when, and more. Using a VCS means that if you screw things up or lose files, you can easily recover.

**3) What is the difference between Commit & Push?** - This command git commit is moving data or files from stage area to local-repo (.git) area. And git push is moving data or files from local-repo(.git) to remote area.

**4) What is the difference betw Push and Pull?** - This command git push is moving data or files from local-repo(.git) to remote area and git pull fetches any new changes/work that has been pushed to that server since you cloned (or last cloned from) it. It’s important to note that pull command pulls the data to your local repository.

**5) Have you ever created Remote repositories in Git? How?**

- We can create repository using two ways:- a) login to github account and click to new repository option and enter reqired name and create it. b) Using git init we can create local repository and push it to central repository.

**6) What happens if I delete .git folder?**

- If we delete .git folder, so we cant perform any git operation, bcoz .git folder is local repository folder.

**7) What is the advantage of STAGE in Git?** - It gives you full control over which files you want to commit when. For that matter, you can use git add -p to control which lines you want to commit. One of the benefits that I like is the ability to commit a portion of a change.

**8)** **Git log -** displays all of the commits in a repository's history in the form of Secure Hash Algorithm (SHA).

# git log **<options>** -**n1, -n2** shows nb of line

**- -auther=“Author name”** -it show log of particular author, **- -grep=“bugfix** –sort and find only bigfix related

**- -oneline** –It show commited msg from 0-9 **- - reverse** –It show commited msg from 9-0

**- -since=“year/month/date”**, **- - until=“year/month/date”**, **- -since=“2022/12/20”** - -follow-complethistory

**9) How do I edit a git commit message after push?**

**-** Changing the latest Git commit message.If the message to be changed is for the latest commit to the repository, **git commit --amend -m "New message"** and **git push --force <repo name> <branch-name>**

**10) What is SHA-1? How Git uses this?** -Everything in Git is check-summed before it is stored and is then referred to by that checksum. The mechanism that Git uses for this check summing is called a SHA-1 hash algorithm. This is a 40-character string composed of hexadecimal characters (0–9 and a–f)

**11) How do you show the content diff of a file which is staged?**

# git diff <options> <filename>

- -staged sha1..sha2value

**12) What is your branching stratogy? OR Can you explain your release process/Stratogy?**

. 24.)\*\* What branching model you suggest for parellel development? - If in between development any new feature is coming, That time we will create new Feature branch from current branch. 25.) Developer fixes a bug. How do you take the change to production? - 1.) create HOTFIX-BRANCH from RELEASE-BRANCH, then fix bug into HOTFIXBRANCH and test HOTFIX-BRANCH branch on all ENVIRONMENT, it is fine then deploy this branch into production. 2.) Then merge HOTFIX-BRANCH into MASTER branch. 26.)\*\* Explain defferent branching models that you have worked-on. - Working only on one branch model. 27.) Did you work on merging the code in Git? - yes 28.)\* How do you merge the code in Git? - git checkout branch and git merge command. 29.)\* What is merge? - Combine two branchs in one branch it is called merge. using git merge command. 30.)\*What is conflict? OR When do we get conflict? - If two users modify the same file in source and target branches and if the same line has different content, - git can't decide which user's code it has to take. we call this situation as conflict. 31.)\* What is fast-forward merge in Git? # FAST-FORWARD MERGE BEFORE MERGE: ------------- 1. In master branch we have ----C1--C2---C3, now we are creating new branch i.e dev\_1.2.3, in this branch we created new files i.e ----C4--C5, now we are merging this files into master branch, means merging dev\_1.2.3 into master branch. It is called FAST-FORWARD MERGE. master | ----C1--C2---C3 | ----------C4----C5 dev\_1.2.3 AFTER MERGE: ------------- master | ----C1----C2---C3------C4----C5 | dev\_1.2.3 32.)\* What is the difference between Merge and Rebase? -Merge = Add all commits from the head branch to the base branch with a merge commit. -Rebase = Change the base of the branch. = Add all commits from the head branch onto the base branch individually. 33.)\* How do you resolve the conflit in Git? - Open the conflict file--> remove conflict markers-->select the right content. - git add - git commit 34.)\* What kind of conflicts you have seen? -Only merge conflicts. 35.) Who resolves the conflicts? - developers. 36.)\*\* What is the difference between branch and tag? -branch is create for development -tag is used after development done in the branch 37.) When do you create a branch and tag? -branch is create for development -tag is used after development done of the branch 38.) How do you create a branch and switch to that using single command? - $ git checkout -b 39.) What is HEAD pointer in Git? Where Git store HEAD info. HEAD is a pointer in git which - Always points to the latest commit in the repository - Always points to tip of the current reposiotory - Always points to parent of the next commit Git store HEAD info - $ cd .git - /.git$ vim HEAD - /.git$ cat refs/heads/ 40.) Can we store binary files in Git? -YES, we can store binary files in Git. 41.) Can skip the staging? How? what are the caveats? -skip the staging using below command: - $ git commit -am "msg" -caveats: - 42.)\* How do you list files/folders modi fied as part of a commit? -$ git show (It show whole info about commit with contents and filenames) -$ git diff-tree --no-commit-id --name-only -r (It shows only files-names) 43.)\* How do you ignore: ex: all files ending with .class all files having alphanumeric all log files but not build.log - Create a .gitignore in the directory where .git is. You can list files in it separated by a newline. - /.git$ vim .gitignore .class etc 44.) How do you add ignore list for all users? - - Create a .gitignore in the directory where .git is. You can list files in it separated by a newline. - /.git$ vim .gitignore .class etc 45.)\* What are the different files you ignore in your project? - only .class file ignore in project. (bcoz file size is big). 46.) How to remove a committed change? Or can we remove? $ git reset --hard HEAD~1 (completely remove changes from local repository). $ git reset --soft HEAD~1 (It will remove from .git to stage) $ git reset --mixed HEAD~1 (It will move from .git to source) 47.) How do you lock the branch? - 48.) How do you clone the particular branch? - $ git clone -b --single-branch 49.) How do you restore a deleted file? Or previous changes of a file? 1.) - $ git log --diff-filter=D --summary (get SHA-value of deleted file using this command.) 2.) - $ git checkout $commit ~1 3.) - $ git status 50.) How do you list the diff. of a file between two different branches. - $ git diff dev\_1.2.4...master -- LoginUser.java 51.) How do you list the changes which are going to be fetched? method:1 $ git fetch $ git log origin/master ^master method:2 $ git fetch && git diff master origin/master --name-only 52.) What is Git Stash? -Often, when you’ve been working on part of your project, things are in a messy state and you want to switch branches for a bit to work on something else. -The problem is, you don’t want to do a commit of half-done work just so you can get back to this point later. The answer to this issue is the git stash command. ------------- 53.) How do you add a new remote to git? Or How do you attach your local repo with remote? - $ git init 54.) What is git ls-tree? -git ls-tree --> Lists files committed as part of a commit. 55.) What is git cherry-pick? - git cherry-pick command that is used to merge specific commit from another branch to current one(i.e master). - $ git cherry-pick 56.) What is git fork? - A fork is a copy of a repository. Forking a repository allows you to freely experiment with changes without affecting the original project. (Note :- we can get any open source project using fork in our repository, exact same copy. only user-name is change.) - Most commonly, forks are used to either propose changes to someone else's project or to use someone else's project as a starting point for your own idea. 57.) What is git cherry-pick ? - If we want to merge particular commit from another branch into master branch using SHA-value of particular commit. 58.) What is git squash ? - Combine all commits from the head branch into a single commit in the base branch. 59.) what is difference between git fetch and pull and clone? git fetch--> It is fetching the modified content from central repository and changes not merge in the source area. git Pull--> It is fetching the modified content from central repository and changes merge in into the source area. git clone--> It is downloading central repository into local repository. 60.) What are Git Hooks? - Git hooks are scripts which trigger when you perform a specific action in Git. They are useful to automate tasks. Example. You create a Git Hook to run a linter every time you commit code. - Git Hooks Are Two Categories: - Client-Side hooks are triggered by operations such as committing and merging. - Server-Side hooks are triggered by network operations such as receiving pushed commits. - Upon initialising a repo, you will have a folder called hooks within your .git folder. - within this folder, you will some example hooks. - These scripts are all shell and perl. - You can write hooks in Ruby, Python, 61.) What are Git TAGs - Tagging in Git or any other VCS refers to creating specific points in history for your repository/data. This is usually done to mark release points. Why should i create TAGs - To mark release points for your code/data - To create historic restore points When To create Tags - When you want to create a release point for a stable version of your code. - When you want to create a historic point for your code/data that you can refer at any future time(to restore your data). Step 1: Checkout the branch where you want to create the tag git checkout "branch name" example : git checkout master \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Step 2: Create tag with some name git tag "tag name" example : git tag v1.0 git tag -a v1.0 -m "ver 1 of .." (to create annotated tags) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Step 3: Display or Show tags git tag git show v1.0 git tag -l “v1.\*” \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Step 4: Push tags to remote git push origin v1.0 git push origin --tags git push --tags (to push all tags at once) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Step 5: Delete tags (if required only) to delete tags from local : git tag -d v1.0 git tag --delete v1.0 to delete tags from remote : git push origin -d v1.0 git push origin --delete v1.0 git push origin :v1.0 to delete multiple tags at once: git tag -d v1.0 v1.1 (local) git push origin -d v1.0 v1.1 (remote) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Checking out TAGS We cannot checkout tags in git We can create a branch from a tag and checkout the branch git checkout -b "branch name" "tag name" example : git checkout -b ReleaseVer1 v1.0 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Creating TAGS from past commits git tag "tag name" "reference of commit" example : git tag v1.2 5fcdb03 62.) What is Git alias? - Git aliases are a powerful workflow tool that create shortcuts to frequently used Git commands. - Using Git aliases will make you a faster and more efficient developer. - Aliases can be used to wrap a sequence of Git commands into new faux Git command. - Git aliases are created through the use of the git config command which essentially modifies local or global Git config files ie.(/.gitconfig) it is in root directory. Example: $ git config --global alias.co checkout $ git co (it is shortcut of checkout) $ git config --global alias.br branch (run this command on terminal for all repository) $ git config --global alias.ci commit $ git config --global alias.st status OR we can configure all alias in .gitconfig file like below: [alias] co = checkout br = branch ci = commit st = status ...... 63.) What is Git gc? - The git gc command is a repository maintenance command. The "gc" stands for garbage collection. - Executing git gc is literally telling Git to clean up the mess it's made in the current repository. - Garbage collection is a concept that originates from interpreted programming languages which do dynamic memory allocation. - Garbage collection in interpreted languages is used to recover memory that has become inaccessible to the executing program. Garbage collection is run automatically on several frequently used commands: - git pull - git merge - git rebase - git commit - git gc vs git prune - git gc is a parent command and git prune is a child. - git gc will internally trigger git prune. - git prune is used to remove Git objects that have been deemed inaccessible by the git gc configuration. 64.) What is git gc --aggressive? - The effects of --aggressive are persistent and only need to be run after a large volume of changes to a repository. 65.) What inside .git folder? - When you run git init in a new or existing directory, Git creates the .git directory, which is where almost everything that Git stores and manipulates is located. - If you want to back up or clone your repository, copying this single directory elsewhere gives you nearly everything you need. Here’s what a newly-initialized .git directory typically looks like: $ ls -a config - The config file contains your project-specific configuration options, description - The description file is used only by the GitWeb program. HEAD - the HEAD file points to the branch you currently have checked out, hooks/ - The hooks directory contains your client- or serverside hook scripts info/ - the info directory keeps a global exclude file for ignored patterns that you don’t want to track in a .gitignore file. objects/ - The objects directory stores all the content for your database, refs/ - the refs directory stores pointers into commit objects in that data (branches, tags, remotes and more), 66.) What is a bare Git repository? - A bare Git repository is a repository that is created without a Working Tree. Go ahead and create one to see. $ git init --bare . - Run ls on that directory and you won't see a Working Tree but just the contents of what is typically in the .git directory. 67.) What is a Git Remote Repository? - A Remote Repository in Git is a special type of repository in that it doesn't have a Working Tree. - This is different than your local repository, which has your project files and then a hidden .git directory. - You can host your own Git remote repository or use one the popular online services, like Github, Gitlab, or Bitbucket. 68.) What is the Working Tree in Git? - Th Working Tree in Git is a directory (and its files and subdirectories) on your file system that is associated with a repository. - It's full of the files you edit, where you add new files, and from which you remove unneeded files. - Any changes to the Working Tree are noted by the Index (see below), and show up as modified files. 69.) Roles and Responsibility in GIT - setting up github environmet - creating repositories and maintaining repositories - creating branches and maintaining branches - creting organization for particular projects, creating teams and creating repositories for organization, add people into that team. - mergaing branches after development is finished. 70.) Whic port is used by git? - The common URL schemes for git repositories are: - ssh:// - default port 22 - git:// - default port 9418 - http:// - default port 80 - https:// - default port 443 ========================= 3.git bi-sect 4.git blob 5.git blame 7.git web-hooks