GIT

1] Tell Git who you are

Configure the author name and email address to be used with your commits.

Note that Git strips some character (for example trailing periods) from user.name.

$ git config --global user.name "Rathod Raja"

$ git config –global user.email rajarathod214@yahoo.com

It is a good idea to introduce yourself to Git with your name and public email address before doing any operation.

2] Create a new repositary

$ git init

3] Checkout repositary

Create a working copy of a local repositary:

$ git clone /path/to/repository

From a remote server

$ git clone username@host:/path/to/repository

4] Add a files

$ git add <filename> or

$ git add *

This snapshot is now stored in a temporary staging area which Git calls the "index". You can permanently store the contents of the index in the repository with git commit:

5] Commit

Commit changes to head (but not yet to the remote repository):

$ git commit -m "Commit message"

Commit any files you've added with git add, and also commit any files you've changed since then:

$ git commit -a

This will prompt you for a commit message. You've now stored the first version of your project in Git.

6] Making changes

Modify some files, then add their updated contents to the index:

$ git add file1 file2 file3

You are now ready to commit. You can see what is about to be committed using git diff with the --cached option:

$ git diff --cached

(Without –cached, git diff will show you any changes that you've made but not yet added to the index.) You can also get a brief summary of the situation with git status:

$ git status

$ git commit

This will again prompt you for a message describing the change, and then record a new version of the project.

Alternatively, instead of running git add beforehand, you can use

$ git commit -a

which will automatically notice any modified (but not new) files, add them to the index, and commit, all in one step.

7] Git tracks content not files

Many revision control systems provide an `add` command that tells the system to start tracking changes to a new file. Git's `add` command does something simpler and more powerful: git add is used both for new and newly modified files, and in both cases it takes a snapshot of the given files and stages that content in the index, ready for inclusion in the next commit.

8] Viewing project history

$ git log

If you also want to see complete diffs at each step, use

$ git log -p

Often the overview of the change is useful to get a feel of each step

$ git log --stat --summary

9] Managing branch

A single Git repository can maintain multiple branches of development. To create a new branch named "experimental", use

$ git branch experimental

to check , current branch or to get list of existing branches,

$ git branch

The asterisk marks the branch you are currently on; type

To change branch,

$ git checkout <branch name>

To switch to the experimental branch. Now edit a file, commit the change, and switch back to the master branch:

$ git commit -a

$ git checkout master

at this point the two branches have diverged, with different changes made in each. To merge the changes made in experimental into master, run

$ git merge <branch name>

If the changes don’t conflict, you’re done. If there are conflicts, markers will be left in the problematic files showing the conflict;

$ git diff

will show this. Once you’ve edited the files to resolve the conflicts,

$ git commit -a

will commit the result of the merge.

10] Connect to a remote repositary

If you haven't connected your local repository to a remote server, add the server to be able to push to it:

$ git remote add origin https://github.com/rajarathod214/votary.git

List all currently configured remote repositories:

$ git remote -v

11] Branches

Create a new branch and switch to it:

$ git checkout -b <branchname>

Switch from one branch to another:

$ git checkout <branchname>

List all the branches in your repo, and also tell you what branch you're currently in:

$ git branch

Delete the feature branch:

$ git branch -d <branchname>

Push the branch to your remote repository, so others can use it:

$ git push origin <branchname>

Push all branches to your remote repository:

$ git push --all origin

Delete a branch on your remote repository:

$ git push origin :<branchname>

12] Update from the remote repositary

Fetch and merge changes on the remote server to your working directory:

$ git pull

To merge a different branch into your active branch:

$ git merge <branchname>

View all the merge conflicts:

$ git diff

View the conflicts against the base file:

$ git diff --base <filename>

Preview changes, before merging:

$ git diff <sourcebranch> <targetbranch>

After you have manually resolved any conflicts, you mark the changed file:

$ git add <filename>

13] Tags

You can use tagging to mark a significant changeset, such as a release:

$ git tag 1.0.0 <commitID>

CommitId is the leading characters of the changeset ID, up to 10, but must be unique. Get the ID using:

$ git log

Push all tags to remote repository:

$ git push --tags origin

14] Undo Local changes

f you mess up, you can replace the changes in your working tree with the last content in head:

Changes already added to the index, as well as new files, will be kept.

$ git checkout -- <filename>

Instead, to drop all your local changes and commits, fetch the latest history from the server and point your local master branch at it, do this:

$ git fetch origin

$ git reset - –hard origin/master

15] Search

Search the working directory for foo():

$ git grep "foo()"

16] Miscellaneous

$ git show c82a22c39cbc32576f64f5c6b3f24b99ea8149c7

$ git show c82a22c39c # the first few characters of the name are usually enough

$ git show head # the tip of the current branch

$ git show <branch name>

$ git show HEAD^ # to see the parent of HEAD

$ git show HEAD^^ # to see the grandparent of HEAD

$ git show HEAD~4 # to see the great-great grandparent of HEAD

$ git show HEAD^1 # show the first parent of HEAD (same as HEAD^)

$ git show HEAD^2 # show the second parent of HEAD

You can also give commits names of your own; after running

$ git tag v2.5 1b2e1d63ff

You can refer to 1b2e1d63ff by the name "v2.5". If you intend to share this name with other people (for example, to identify a release version), you should create a "tag" object, and perhaps sign it;

Any Git command that needs to know a commit can take any of these names. For example:

$ git diff v2.5 HEAD # compare the current HEAD to v2.5
$ git branch stable v2.5 # start a new branch named "stable" based at v2.5
$ git reset --hard HEAD^ # reset your current branch and working directory to its state at HEAD^

Be careful with that last command: in addition to losing any changes in the working directory, it will also remove all later commits from this branch. If this branch is the only branch containing those commits, they will be lost. Also, don't use git reset on a publicly-visible branch that other developers pull from, as it will force needless merges on other developers to clean up the history. If you need to undo changes that you have pushed, use git revert instead.

The git grep command can search for strings in any version of your project, so

$ git grep "hello" v2.5

If you leave out the commit name, git grep will search any of the files it manages in your current directory. So

$ git grep "hello"

is a quick way to search just the files that are tracked by Git.

Many Git commands also take sets of commits, which can be specified in a number of ways. Here are some examples with git log:

$ git log v2.5..v2.6 # commits between v2.5 and v2.6
$ git log v2.5.. # commits since v2.5
$ git log –since="2 week ago" # commit from the last 2 weeks

$ git log v2.5.. Makefile # commit since v2.5 which modify Makefile

You can also give git log a "range" of commits where the first is not necessarily an ancestor of the second; for example, if the tips of the branches "stable" and "master" diverged from a common commit some time ago, then

$ git log stable..master

will list commits made in the master branch but not in the stable branch, while

$ git log master..stable

will show the list of commits made on the stable branch but not the master branch.

The git log command has a weakness: it must present commits in a list. When the history has lines of development that diverged and then merged back together, the order in which git log presents those commits is meaningless.

Most projects with multiple contributors (such as the Linux kernel, or Git itself) have frequent merges, and gitk does a better job of visualizing their history. For example

$ gitk –since=”2 weeks ago” drivers/

allows you to browse any commits from the last 2 weeks of commits that modified files under the "drivers" directory. (Note: you can adjust gitk’s fonts by holding down the control key while pressing "-" or "+".)

Finally, most commands that take filenames will optionally allow you to precede any filename by a commit, to specify a particular version of the file:

$ git diff v2.5:Makefile HEAD:Makefile.in

You can also use git show to see any such file:

$ git show v2.5:Makefile

The object database is the rather elegant system used to store the history of your project—files, directories, and commits.The index file is a cache of the state of a directory tree, used to create commits, check out working directories, and hold the various trees involved in a merge.

$ git tag -l

If you decide that you’d rather see version 2.6.17, you can modify the current branch to point at v2.6.17 instead, with

$ git reset --hard v2.6.17

Every change in the history of a project is represented by a commit. The git-show[1] command shows the most recent commit on the current branch:

$ git show

Every commit has a 40-hexdigit id, sometimes called the "object name" or the "SHA-1 id", shown on the first line of the git show output. You can usually refer to a commit by a shorter name, such as a tag or a branch name, but this longer name can also be useful. Delete the branch , if the branch is

not fully merged in its upstream branch or contained in the current branch, this command will fail with warning:

$ git branch -d <branch name>

Delete the branch irrespective of its merge status:

$ git branch -D <branch>

Create a new branch <new> referencing <start-point> and check it out

$ git checkout -b <new> <start-point>

The special symbol "HEAD" can always be used to refer to the current branch. In fact, Git uses a file named HEAD in the .git directory which branch is current:

$ cat .git/HEAD

$ git branch -r

creating tags:

$ git tag stable-1 1b2e1d63ff

Temporarily setting aside work in progress, While you are in the middle of working on something complicated, you find an unrelated but obvious and trivial bug. You would like to fix it before continuing. You can use git-stash[1] to save the current state of your work, and after fixing the bug (or, optionally after doing so on a different branch and then coming back), unstash the work-in-progress changes.

.. edit and test ...

$git commit -a -m "blorpl: typofix"

$ git stash save "work in progress for foo feature"

This command will save your changes away to the stash and reset your working tree and the index to match the tip of your current branch. Then you can make your fix as usual.

After that, you can go back to what you were working on with:

$ git stash pop

Ensuring reliability, Checking the repository for corruption. The git-fsck[1] command runs a number of self-consistency checks on the repository, and reports on any problems. This may take some time.

$ git fsck

You can run to suppress these messages, and still view real errors.

$ git fsck --no-dangling

Question & Answers

1. How to delete a Git branch both locally and remotely?

To remove a local branch from your local system.

$ git branch -d the_local_branch

To remove a remote branch from the server.

$ git push origin :the_remote_branch

2. How do you undo the last commit?

$ git revert commit-id

3. How to Edit an incorrect commit message in Git?

$ git commit --amend -m "This is your new git message"

4. What are the differences between 'git pull' and 'git fetch'?

Git pull automatically merges the commits without letting you review them first.

Git fetch stores them in your local repository but it not merge them with your current branch. git fetch similar to guit pull but it does not merge the changes.

5. How do you rename the local branch?

$ git branch -m oldBranchName newBranchName

6. How do I remove local files (Not in Repo) from my current Git branch?

$ git clean -f -n

7. How to Checkout remote Git branch?

$ git checkout test

8.How do I remove a Git submodule?

$ git rm the_submodule

$ rm -rf .git/modules/the_submodule

8. How do you create a remote Git branch?

$ git checkout -b your_branch_name

$ git push -u origin your_branch_name

9. How to Change the URL for a remote Git repository?

$ git remote set-url origin git://this.is.new.url

10. How to Change the author of a commit in Git?

$ git filter-branch -f --env-filter "GIT_AUTHOR_NAME='NewAuthorName';

GIT_AUTHOR_EMAIL='authoremail@gmail.com';
GIT_COMMITTER_NAME='CommiterName';
GIT_COMMITTER_EMAIL='committergmail@gmail.com';" HEAD

11. What is GIT?

GIT is a distributed version control system and source code management (SCM) system with an emphasis to handle small and large projects with speed and efficiency.It allows groups of people to work on the same documents and files. In most cases, these files are source code. Using it we can make changes at the same time, and without stepping on each work.

12. What is a repository in GIT?

A repository contains a directory named .git, where git keeps all of its metadata for the repository. The content of the .git directory are private to git.

13. What is the command you can use to write a commit message?

The command that is used to write a commit message is **"git commit –a".** The **–a** on the command line instructs git to commit the new content of all tracked files that have been modified. You can use **"git add<file>"** before git commit **–a** if new files need to be committed for the first time.

14. What is the difference between GIT and SVN?

The difference between GIT and SVN is

a)Git is less preferred for handling extremely large files or frequently changing binary files while SVN can handle multiple projects stored in the same repository.

b)GIT does not support 'commits' across multiple branches or tags. Subversion allows the creation of folders at any location in the repository layout.

c)Gits are unchangeable, while Subversion allows committers to treat a tag as a branch and to create multiple revisions under a tag root.

15. What are the advantages of using GIT?

Data redundancy and replication

High availability

Only one.git directory per repository

Superior disk utilization and network performance

Collaboration friendly

Any sort of projects can use GIT

16. What language is used in GIT?

GIT is fast, and 'C' language makes this possible by reducing the overhead of runtimes associated with higher languages.

17. What is the function of 'GIT PUSH' in GIT?

'GIT PUSH' updates remote refs along with associated objects.

18. Why GIT better than Subversion?

GIT is an open source version control system; it will allow you to run 'versions' of a project, which show the changes that were made to the code overtime also it allows you keep the backtrack if necessary and undo those changes. Multiple developers can checkout, and upload changes and each change can then be attributed to a specific developer.

19. What is "Staging Area" or "Index" in GIT?

Before completing the commits, it can be formatted and reviewed in an intermediate area known as 'Staging Area' or 'Index'.

20. What is GIT stash?

GIT stash takes the current state of the working directory and index and puts in on the stack for later and gives you back a clean working directory. So in case if you are in the middle of something and need to jump over to the other job, and at the same time you don't want to lose your current edits then you can use GIT stash.

21. What is GIT stash drop?

When you are done with the stashed item or want to remove it from the list, run the git 'stash drop' command. It will remove the last added stash item by default, and it can also remove a specific item if you include as an argument.

22. How will you know in GIT if a branch has been already merged into master?

Git branch—merged lists the branches that have been merged into the current branch

Git branch—-no merged lists the branches that have not been merged

23. What is the function of git clone?

The **git clone** command creates a copy of an existing Git repository. To get the copy of a central repository, **'cloning'** is the most common way used by programmers.

24. What is the function of 'git config'?

The **'git config'** command is a convenient way to set configuration options for your Git installation. Behaviour of a repository, user info, preferences etc. can be defined through this command.

25. What does commit object contain?

a) A set of files, representing the state of a project at a given point of time

b) Reference to parent commit objects

c) An SHAI name, a 40 character string that uniquely identifies the commit object.

26. How can you create a repository in Git?

In Git, to create a repository, create a directory for the project if it does not exist, and then run command **"git init"**. By running this command .git directory will be created in the project directory, the directory does not need to be empty.

27. What is 'head' in git and how many heads can be created in a repository?

A 'head' is simply a reference to a commit object. In every repository, there is a default head referred as "Master". A repository can contain any number of heads.

28. What is the purpose of branching in GIT?

The purpose of branching in GIT is that you can create your own branch and jump between those branches. It will allow you to go to your previous work keeping your recent work intact.

29. What is the common branching pattern in GIT?

The common way of creating branch in GIT is to maintain one as "Main" branch and create another branch to implement new features. This pattern is particularly useful when there are multiple developers working on a single project.

30. How can you bring a new feature in the main branch?

To bring a new feature in the main branch, you can use a command **"git merge" or "git pull"** command.

31. What is a 'conflict' in git?

A 'conflict' arises when the commit that has to be merged has some change in one place, and the current commit also has a change at the same place. Git will not be able to predict which change should take precedence.

32. How can conflict in git resolved?

To resolve the conflict in git, edit the files to fix the conflicting changes and then add the resolved files by running **"git add"** after that to commit the repaired merge, run **"git commit".** Git remembers that you are in the middle of a merger, so it sets the parents of the commit correctly.

33. To delete a branch what is the command that is used?

Once your development branch is merged into the main branch, you don't need development branch. To delete a branch use, the command **"git branch –d [head]".**

34. What is another option for merging in git?

"Rebasing" is an alternative to merging in git.

35. What is the syntax for "Rebasing" in ?

The syntax used for rebase is **"git rebase [new-commit] "**

36. What is the difference between 'git remote' and 'git clone'?

'git remote add' just creates an entry in your git config that specifies a name for a particular URL. While, 'git clone' creates a new git repository by copying and existing one located at the URI.

37. What is GIT version control?

With the help of GIT version control, you can track the history of a collection of files and includes the functionality to revert the collection of files to another version. Each version captures a snapshot of the file system at a certain point of time. A collection of files and their complete history are stored in a repository.

38. Mention some of the best graphical GIT client for LINUX?

Some of the best GIT client for LINUX is

a) Git Cola

b) Git-g

c) Smart git

d) Giggle

e) Git GUI

f) qGit

39. What is Subgit? Why to use Subgit?

'Subgit' is a tool for a smooth, stress-free SVN to Git migration. Subgit is a solution for a company -wide migration from SVN to Git that is:

a) It is much better than git-svn

b) No requirement to change the infrastructure that is already placed

c) Allows to use all git and all sub-version features

d) Provides genuine stress –free migration experience.

40. What is the function of 'git diff ' in git?

'git diff' shows the changes between commits, commit and working tree etc.

41. What is ‘git status’ is used for?

As ‘Git Status’ shows you the difference between the working directory and the index, it is helpful in understanding a git more comprehensively.

42. What is the difference between the ‘git diff ’and ‘git status’?

‘git diff’ is similar to **‘git status’**, but it shows the differences between various commits and also between the working directory and index.

43. What is the function of ‘git checkout’ in git?

A **‘git checkout’** command is used to update directories or specific files in your working tree with those from another branch without merging it in the whole branch.

45. What is the function of ‘git rm’?

To remove the file from the staging area and also off your disk **‘git rm’** is used.

46. What is the function of ‘git stash apply’?

When you want to continue working where you have left your work, **‘git stash apply’** command is used to bring back the saved changes onto the working directory.

47. What is the use of ‘git log’?

To find specific commits in your project history- by author, date, content or history **‘git log’** is used.

48. What is ‘git add’ is used for?

‘git add’ adds file changes in your existing directory to your index.

49. What is the function of ‘git reset’?

The function of **‘Git Reset’** is to reset your index as well as the working directory to the state of your last commit.

50. What is git Is-tree?

'git Is-tree' represents a tree object including the mode and the name of each item and the SHA-1 value of the blob or the tree.

51. How git instaweb is used?

'Git Instaweb' automatically directs a web browser and runs webserver with an interface into your local repository.

52. What does 'hooks' consist of in git?

This directory consists of Shell scripts which are activated after running the corresponding Git commands. For example, git will try to execute the post-commit script after you run a commit.

53. Explain what is commit message?

Commit message is a feature of git which appears when you commit a change. Git provides you a text editor where you can enter the modifications made in commits.

54. How can you fix a broken commit?

To fix any broken commit, you will use the command "git commit—amend". By running this command, you can fix the broken commit message in the editor.

55. Why is it advisable to create an additional commit rather than amending an existing commit?

There are couple of reason

a) The amend operation will destroy the state that was previously saved in a commit. If it's just the commit message being changed then that's not an issue. But if the contents are being amended then chances of eliminating something important remains more.

b) Abusing "git commit- amend" can cause a small commit to grow and acquire unrelated changes.

56. What is 'bare repository' in GIT?

To co-ordinate with the distributed development and developers team, especially when you are working on a project from multiple computers **'Bare Repository'** is used. A bare repository comprises of a version history of your code.

57. Name a few Git repository hosting services

Pikacode

Visual Studio Online

GitHub

GitEnterprise

SourceForge.net

58. How do you squash last N commits into a single commit?

Squashing multiple commits into a single commit will overwrite history, and should be done with caution. However, this is useful when working in feature branches. To squash the last N commits of the current branch, run the following command (with {N} replaced with the number of commits that you want to squash):

```
$ git rebase -i HEAD~{N}
```

Upon running this command, an editor will open with a list of these N commit messages, one per line. Each of these lines will begin with the word "pick". Replacing "pick" with "squash" or "s" will tell Git to combine the commit with the commit before it. To combine all N commits into one, set every commit in the list to be squash except the first one. Upon exiting the editor, and if no conflict arises, git rebase will allow you to create a new commit message for the new combined commit.

59. How do you find a list of files that has changed in a particular commit?

```
$ git diff-tree -r {hash}
```

Given the commit hash, this will list all the files that were changed or added in that commit. The -r flag makes the command list individual files, rather than collapsing them into root directory names

only. The output will also include some extra information, which can be easily suppressed by including a couple of flags:

```
$ git diff-tree --no-commit-id --name-only -r {hash}
```

Here –no-commit-id will supress the commit hashes from appearing in the output, and --name-only will only print the file names, instead of their paths.

60. How do you setup a script to run every time a repository receives new commits through push?

To configure a script to run every time a repository receives new commits through push, one needs to define either a pre-receive, update, or a post-receive hook depending on when exactly the script needs to be triggered.

Pre-receive hook in the destination repository is invoked when commits are pushed to it. Any script bound to this hook will be executed before any references are updated. This is a useful hook to run scripts that help enforce development policies.

Update hook works in a similar manner to pre-receive hook, and is also triggered before any updates are actually made. However, the update hook is called once for every commit that has been pushed to the destination repository.

Finally, post-receive hook in the repository is invoked after the updates have been accepted into the destination repository. This is an ideal place to configure simple deployment scripts, invoke some continuous integration systems, dispatch notification emails to repository maintainers, etc.

Hooks are local to every Git repository and are not versioned. Scripts can either be created within the hooks directory inside the ".git" directory, or they can be created elsewhere and links to those scripts can be placed within the directory.

61. What is `git bisect` ? How can you use it to determine the source of a (regression) bug?

Git provides a rather efficient mechanism to find bad commits. Instead of making the user try out every single commit to find out the first one that introduced some particular issue into the code, git bisect allows the user to perform a sort of binary search on the entire history of a repository.

By issuing the command git bisect start, the repository enters bisect mode. After this, all you have to do is identify a bad and a good commit:

```
$ git bisect bad # marks the current version as bad
```

$ git bisect good {hash or tag} # marks the given hash or tag as good, ideally of some earlier commit

Once this is done, Git will then have a range of commits that it needs to explore. At every step, it will checkout a certain commit from this range, and require you to identify it as good or bad. After which the range will be effectively halved, and the whole search will require a lot less number of steps than the actual number of commits involved in the range. Once the first bad commit has been found, or the bisect mode needs to be ended, the following command can be used to exit the mode and reset the bisection state:

$ git bisect reset

62. What are the different ways you can refer to a commit?

In Git each commit is given a unique hash. These hashes can be used to identify the corresponding commits in various scenarios (such as while trying to checkout a particular state of the code using the **git checkout {hash}** command).

Additionally, Git also maintains a number of aliases to certain commits, known as refs. Also, every tag that you create in the repository effectively becomes a ref (and that is exactly why you can use tags instead of commit hashes in various git commands). Git also maintains a number of special aliases that change based on the state of the repository, such as HEAD, FETCH_HEAD, MERGE_HEAD, etc.

Git also allows commits to be referred as relative to one another. For example, HEAD~1 refers to the commit parent to HEAD, HEAD~2 refers to the grandparent of HEAD, and so on. In case of merge commits, where the commit has two parents, ^ can be used to select one of the two parents, e.g. HEAD^2 can be used to follow the second parent.

And finally, refspecs. These are used to map local and remote branches together. However, these can be used to refer to commits that reside on remote branches allowing one to control and manipulate them from a local Git environment.

63. What is git rebase and how can it be used to resolve conflicts in a feature branch before merge?

In simple words, git rebase allows one to move the first commit of a branch to a new starting location. For example, if a feature branch was created from master, and since then the master branch

has received new commits, git rebase can be used to move the feature branch to the tip of master. The command effectively will replay the changes made in the feature branch at the tip of master, allowing conflicts to be resolved in the process. When done with care, this will allow the feature branch to be merged into master with relative ease and sometimes as a simple fast-forward operation.

64. How do you configure a Git repository to run code sanity checking tools right before making commits, and preventing them if the test fails?

This can be done with a simple script bound to the pre-commit hook of the repository. The pre-commit hook is triggered right before a commit is made, even before you are required to enter a commit message. In this script one can run other tools, such as linters and perform sanity checks on the changes being committed into the repository. For example, the following script:

```
#!/bin/sh
files=$(git diff --cached --name-only --diff-filter=ACM | grep '.go$')
if [ -z files ]; then
    exit 0
fi
unfmtd=$(gofmt -l $files)
if [ -z unfmtd ]; then
    exit 0
fi
echo “Some .go files are not fmt’d”
exit 1
```

… checks to see if any .go file that is about to be commited needs to be passed through the standard Go source code formatting tool gofmt. By exiting with a non-zero status, the script effectively prevents the commit from being applied to the repository.

65. One of your teammates accidentally deleted a branch, and has already pushed the changes to the central git repo. There are no other git repos, and none of your other teammates had a local copy. How would you recover this branch?

Check out the latest commit to this branch in the reflog, and then check it out as a new branch.

66. How can you copy a commit made in one branch to another (e.g. a hot fix commit from released branch to current development branch)?

You need to use the `cherry-pick` command. It provides the possibility to play back an existing commit to your current location/branch. So you need to switch to the target branch (e.g. **`git checkout`** development) and call **`git cherry-pick`** {hash of that commit} .

In spite of applying the same changes, it will be a new commit with a new hash because the changes are applied to a different destination.

67. How do you cherry-pick a merge commit?

Cherry-pick uses a diff to find the difference between branches.

As a merge commit belongs to a different branch, it has two parents and two changesets.

For example, if you have merge commt ref `63ad84c` , you have to specify `-m` and use parent `1` as a base:

```
$ git checkout release_branch
$ git cherry-pick -m 1 63ad84c
```

68. What is the difference between `git pull` and `git fetch` ?

`$ git fetch` only downloads new data from a remote repository, but it doesn't integrate any of the downloaded data into your working files. All it does is provide a view of this data.

`$ git pull` downloads as well as merges the data from a remote repository into your local working files. It may also lead to merge conflicts if your local changes are not yet committed. Use the `git stash` command to hide your local changes.

69. What is a conflict in git and how can it be resolved?

A conflict arises when more than one commit that has to be merged has some change in the same place or same line of code. Git will not be able to predict which change should take precedence. This is a git conflict.

To resolve the conflict in git, edit the files to fix the conflicting changes and then add the resolved files by running `git add` . After that, to commit the repaired merge, run `git commit` . Git remembers that you are in the middle of a merge, so it sets the parents of the commit correctly.

70. What is Bash?

Bash stands for Bourne Again Shell (BASH) A Shell is a user interface that allows to interact with the computer resources. Shells can be graphical / command line. Bash is the default command line shell on Mac OS and on most Linux distributions. It is also a simple scripting language for light weight programming tasks.

71. What is Distributed Version Control System?

We can work with, and collaborate with different groups of people in different ways simultaneously within the same project. Distributed version control systems users or committers fully mirror the repository. If server node crashes, any of the client repositories can be copied back up to the server to restore it. Every clone is really a full backup of all the data. Git is one of the example of DVCS.