1) What is the difference between Git and SVN?

GIT:

Git is a Distributed Version Control system(DVCS). It lets you track changes made to a file and allows you to revert back to any particular change that you wish. It is a distributed architecture that provides many advantages over other Version Control Systems (VCS) like SVN. One of the major advantages is that it does not rely on a central server to store all the versions of a project's files.
Git is a Decentralized Version Control tool
It belongs to the 3rd generation of Version Control tools
Clients can clone entire repositories on their local systems
Push/pull operations are faster
Works are shared automatically by commit

SVN:

SVN is a Centralized Version Control tool
It belongs to the 2nd generation of Version Control tools
Version history is stored on a server-side repository
Only online commits are allowed
Push/pull operations are slower
Nothing is shared automatically

2) What is a distributed VCS?
These are the systems that don't rely on a central server to store a project file and all its versions.
In Distributed VCS, every contributor can get a local copy or "clone" of the main repository.
 every programmer can maintain a local repository which is actually the copy or clone of the central repository which is present on their hard drive. They can commit and update their local repository without any hassles.

With an operation called "pull", they can update their local repositories with new data from the central server and "pull" operation affects changes to the main repository from their local repository.

3) How can you fix a broken commit?
In order to fix any broken commit, use the command "git commit --amend". When you run this command, you can fix the broken commit message in the editor.

4) How can you create a repository in Git?
This is probably the most frequently asked question and the answer to this is really simple.
To create a repository, create a directory for the project if it does not exist, then run the command "git init". By running this command .git directory will be created in the project directory.

5) What is 'bare repository' in Git?
"bare" repository in Git contains information about the version control and no working files (no tree) and it doesn't contain the special .git sub-directory. Instead, it contains all the contents of the .git sub-directory directly in the main directory itself, whereas the working directory consists of :.git subdirectory with all the Git related revision history of your repository.
 working tree, or checked out copies of your project files.
==========git conflict==========
6) What is a 'conflict' in git?
Git can handle on its own most merges by using its automatic merging features. There arises a conflict when two separate branches have made edits to the same line

in a file, or when a file has been deleted in one branch but edited in the other. Conflicts are most likely to happen when working in a team environment.

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>

7) some commands and their uses:

git rm [file] ---deletes the file from your working directory and stages the deletion.

git log --list the version history for the current branch.

git show [commit] --shows the metadata and content changes of the specified commit.

git tag [commitID] --used to give tags to the specified commit.

git checkout [branch name] --used to switch from one branch to another.

git checkout -b [branch name]--creates a new branch and also switches to it.

8) How to resolve a conflict in Git?
The following steps will resolve conflict in Git-

Identify the files that have caused the conflict.

Make the necessary changes in the files so that conflict does not arise again.

Add these files by the command git add.

Finally to commit the changed file using the command git commit

9) In Git how do you revert a commit that has already been pushed and made public?

git revert <name of bad commit>

==========Pulll and Fetch==========
10):What is the difference between git pull and git fetch?
Git pull command pulls new changes or commits from a particular branch from your central repository and updates your target branch in your local repository.

Git fetch is also used for the same purpose but it works in a slightly different way. When you perform a git fetch, it pulls all new commits from the desired branch and stores it in a new branch in your local repository. If you want to reflect these changes in your target branch, git fetch must be followed with a git merge. Your target branch will only be updated after merging the target branch and fetched branch. Just to make it easy for you, remember the equation below:

Git pull = git fetch + git merge

git fetch is the command that tells your local git to retrieve the latest meta-data info from the original (yet doesn’t do any file transferring. It’s more like just checking to see if there are any changes available).
$ git fetch origin

git pull on the other hand does that AND brings (copy) those changes from the

remote repository.
git pull origin master

==========

11) What is the function of ‘git config’?
Git uses your username to associate commits with an identity. The git config command can be used to change your Git configuration, including your username.
Now explain with an example.
Suppose you want to give a username and email id to associate a commit with an identity so that you can know who has made a particular commit. For that I will use:
git config –global user.name “Your Name”: This command will add a username.
git config –global user.email “Your E-mail Address”: This command will add an email id.
12) How will you know in Git if a branch has already been merged into master?
The answer is pretty direct.

To know if a branch has been merged into master or not you can use the below commands:

git branch --merged – It lists the branches that have been merged into the current branch.
git branch --no-merged – It lists the branches that have not been merged.

13) Tell me the difference between HEAD, working tree and index, in Git.
The working tree/working directory/workspace is the directory tree of (source) files that you are able to see and edit.
The index/staging area is a single, large, binary file in <baseOfRepo>/.git/index, which lists all files in the current branch, their SHA-1 checksums, timestamps, and the file name – it is not another directory which contains a copy of files in it.
HEAD is used to refer to the last commit in the currently checked-out branch.

14) What is Git fork? What is the difference between fork, branch, and clone?
A fork is a copy of a repository. Normally you fork a repository so that you are able to freely experiment with changes without affecting the original project. 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

15) Explain the difference between git revert and git reset
Git reset is a powerful command that is used to undo local changes to the state of a Git repository. Git reset operates on “The Three Trees of Git” which are, Commit History (HEAD), the Staging Index, and the Working Directory.
Revert command in Git creates a new commit that undoes the changes from the previous commit. This command adds a new history to the project. It does not modify the existing history.

16) How do you find a list of files that have changed in a particular commit?
git diff-tree -r {hash}

17) What is cherrypick: Cherry picking is the act of picking a commit from a branch and applying it to another. git cherry-pick can be useful for undoing changes, and it will not delete the commit from soure and it will create a new commit to destination repo

Consider:

git cherry-pick <commit-hash>

==========
git init -- to initialize the git repo locally , once you run this
automatically .git folder will create and this is used as git database

vi <filename>   --- this will fall under untrakced area

vi <modifying existing file> this will fall under modified area

to move untracked and modidfied files to stageing area run the bleow

git add . -- this will move to stageing area

to reest from staging aread -- git reset

to check the status -- git status

then commit the changes to local repo with the below

git commit -m "message"

then push this to remote repo

before pushing add remte repo

git remote add origin <githuburl>

git push origin master<branchname>

==========
Adding ssh keys to git hub page

ssh-keygen -t rsa -b 4096 -C "venkva93@in.ibm.com"

eval "$(ssh-agent -s)"

ssh-add ~/.ssh/id_rsa

copy the public key from .ssh/id_rsa.pub and paste it in github ssh keys

==========Tags==========
Git TAGS - What | Why | When | How

What are the tags: Tags are ref's that point to specific points in Git history.
Tagging is generally used to capture a point in history that is used for a marked
version release (i.e. v1. 0.1). A tag is like a branch that doesn't change.

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

================-branching and merging ======================
What is branch : Git branches is much more lightweight than other version control system models. Instead of copying files from directory to directory create a branch and it will stores a branch as a reference to a commit
  branch represents the tip of a series of commits

Step 1 : Create branch
   git branch “branch name”

how to push branch to rempte repo

git push -u origin <branchname>

Step 2 : Checkout branch
   git checkout “branch name”

Step 3 : Merge new branch in master branch

git merge “branch name <this is the source branch>”

git push origin <current branch>

Step 4 : Delete branch

```
   git branch -d “branch name”    – delete from local
   git push origin --delete “branch name”   – delete from remote

To list all the branches inclusing local and remote repo branches

git branch -a

example:

[vardhan@oc2828342533 prac]$ git branch -a
  master
* release
  remotes/origin/HEAD -> origin/master  -- the below 2 from remote repo
  remotes/origin/master
  remotes/origin/release
How to check what branches are merged to branch --> git branch --merged
How to check what branches are not merged to branch --> git branch --no-merged

=====to remove untrack and staging=== difference between git clean and git reset

to delete untracked files and directories -- git clean -df  (d stands for dir f
stands for files)
git reflog

to delete modifed/staging area file -- > git reset

to delete both untracked and stage area ---> git reset && git clean -df

git commit --amend -m "message"

then do git log to check

cherry-pick

Git and GitHub Beginner Tutorial 6 - How to send email from GitHub
==========================================================

Today we will learn
___________
How to trigger notification email from github
whenever there is any change/commit in the project
_____________________________________________

Step 1 : Github - Repository - Settings - integration & services - add email

Step 2 : Test and validate by making some change in the project
====================merge and rebase ==========

In Git there are 2 ways to integrate changes from one branch to another
git merge   <sourecebranchname>
git rebase  <sourecebranchname>

git merge
- Is a non-destructive operation
- Existing branches are not changed in any way
- Creates a new merge commit in the feature branch

git rebase
- Moves the entire feature branch to begin on the tip of the master branch
```

- Re-writes the project history
- We get much cleaner and linear project history

==========Git stash==========

stash is temporary area under git

it stores the temporary files

what are the conditions to store the files in to stash area

 --> the files must and should in the staging area

commands to use to store the data in the stash area

```
git stash
 or
git stash save <"message">
```

statsh0
statsh1

how to see the stash list

```
 git statsh list
   o/p: stash@{0}: <branch name> <message>
```

commands to use copy the data in to branches

```
  git stash apply <stash#>  --- copy paste from stash area
  git stash pop <stash#>       cut paster   ---> it will remove the date from stash
area
```

how to delete the stash area

```
git stash drop <statsh#>   --- to delete the partciular statsh
git stash clear    --- to delete the entire statsh list
```

if you want to stash the untracked files aswell --> git stash -u
if you want to stash all files --> git stash -a

How to create a branch from stash area

```
git stash branch <branch name> <stash hash number>
```

==========Hooks==========

What are Git hooks?
Git hooks are scripts that Git executes before or after events such as: commit, push, and receive. Git hooks are a built-in feature - no need to download anything. Git hooks are run locally.

These hook scripts are only limited by a developer's imagination. Some example hook scripts include:

pre-commit: Check the commit message for spelling errors.
pre-receive: Enforce project coding standards.
post-commit: Email/SMS team members of a new commit.
post-receive: Push the code to production.

Every Git repository has a .git/hooks folder with a script for each hook you can

bind to. You're free to change or update these scripts as necessary, and Git will execute them when those events occur.

https://www.digitalocean.com/community/tutorials/how-to-use-git-hooks-to-automate-development-and-deployment-tasks

below is some real time example:

How do you configure a Git repository to run code sanity checking tools right before making commits, and preventing them if the test fails?
I will suggest you to first give a small introduction to sanity checking.

Sanity or smoke test determines whether it is possible and reasonable to continue testing.

Now explain how to achieve this.

This can be done with a simple script related 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.

Finally, give an example, you can refer the below 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
```

This script checks to see if any .go file that is about to be committed 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.

==========git revert bad commit==========

how to modify the existing commit with out deleting

git commit --amend -m "meassage" -- by default it will take the last commit id

git reset --soft <commit id till you want to retain>   -- it will revert to the commit id you mentioned but the files will be in staging area

git reset <commit id>   -- this will brings you back to the expected commit id but files will be in working tree area

git reset --hard <commmit id>   -- this will revert you back and also it will removes the files from tracking and staging area

to clean all un tracked directories and files -- git clean -df

if you auxidentally deleted the files with hard reset still you can revert back from git reflogs

git ref logs <commithash>

Git revert is uesed to undo the chages to remote repo, git revert will keep the commit history as it is and it will create a new commit id for undoing the changes

git revert <commitid>

==========

Q: What is git reflog?
The ‘reflog’ command keeps a track of every single change made in the references (branches or tags) of a repository and keeps a log history of the branches and tags that were either created locally or checked out. Reference logs such as the commit snapshot of when the branch was created or cloned, checked-out, renamed, or any commits made on the branch are maintained by Git and listed by the ‘reflog’ command.

==========Pull request==========

A pull request (PR) is a request submitted to a GitHub repository to merge code into that project. The PR allows requested reviewers to view and discuss the proposed code. Once the PR passes all standards of reviews and all necessary revisions have been made, it can be merged into the code base.