**Linkedin Git Assessment Answers**

Git

Q1. How can you check your current git version?

* git –v
* git –version
* git –option
* git –current

Q2. What command lets you create a connection between a local and remote repository?

* git remote add new
* git remote add origin
* git remote new origin
* git remote origin

The command is git remote add. The new added connection can be named origin or new. The only constraints, although it is not documented AFAIK, is that the connection name needs to be acceptable to git-check-ref-format, and it cannot be repeated. If the LinkedIn assessment asks this and you can choose just one option, then leave feedback.

Q3. Describe what the following git commands do to the commit history.

git reset –hard HEAD~5git merge –squash HEAD@{1}

* Reset the HEAD to the 5th commit in the repo, then merges to the master branch
* Reset the commit branch back before the last 5 commits, then squashes them into a single commit
* Delete the last 5 commits
* Merges the last 5 commits into a new branch

git reset –hard HEAD~5 resets the current branch to the commit just before the last 5 (see man gitrevisions for details about this notation and other cool alternatives like HEAD@{2 days ago}). As it is a hard reset, it will also overwrite every change in the working tree as well. See man git-reset.

git merge –squash HEAD@{1} HEAD@{1} is where the branch was just before the previous command (again, see man gitrevisions). This command sets the state of the index to be as it would just after a merge from that commit. This whole operation could be a way to take 5 commits from a branch in which you started a new feature and squash them to a single commit, a meaningful one.

Q4. Your current project has several branches; master, beta, and push-notifications. You’ve just finished the notification feature in the push-notification branch, and you want to commit it to beta branch. How can you accomplish this?

* Checkout the push-notifications branch and run git merge beta
* Checkout the master branch and run git merge beta -> push-notification
* Delete the push-notification branch and it will be committed to the master branch automatically
* Checkout the beta branch and run git merge push-notification

Q5. Which of the following is true you when you use the following command?

git add -A

* All new and updated files are staged
* Files are staged in alphabetical order.
* All new files are staged
* Only updated files are staged

Q6. What will the following command print to the Terminal?

git remote -v

* A list of remote repositories and their URLs
* The current git version you’re running
* An inline editor for modifying remote repositories
* The last 5 git versions you’ve installed

Q7. Looking at the following commands, describe what is happening.

git checkout feature-user-locationgit cherry-pick kj2342134sdf090093f0sdgasdf99sdfo992mmmf9921231

* The commit is being tagged for release on the feature-user-location branch
* A commit is being copied from its original branch over to the feature-user-location branch
* The commit is being cherry picked as the new HEAD of the commit history
* A commit is being copied from the feature-user-location branch to the master branch
* The branch is switched to the feature-user-location branch, and the specified commit is applied to the branch.

Commits aren’t copied when cherry picking, they are cherry picked. The changes introduced by the commit are applied and a new commit is then created. This allow us to get specific changes as if they were patches (in the GIT’s book, this is actually called [Patching](https://git-scm.com/book/en/v2/Appendix-C:-Git-Commands-Patching)). As a new commit is created upon feature-user-location, HEAD also changes to match it. You can see this in cat .git/HEAD and cat .git/refs/heads/feature-user-location for this case. See man git-cherry-pick for details.

**NOTE**: There are two versions of this question so far. The task is always “describe what is happening”, the commands are always a checkout and a cherry-pick, and the correct answer is always the same.

Q8. What does the following command do to the git repository?

git reset –soft HEAD^

* It deletes all previous commits and reset the repository history back to its initial state.
* It resets the working branch to the first commit.
* It keeps the HEAD at the current commit, but clears all previous commits.
* It sets HEAD to the previous commit and leaves changes from the undone commit in the stage/index.

Q9. You find a bug in your project, but can’t locate where it was introduced in the commit history. How would you diagnose this problem?

* Manually backtrack through your commit history.
* Use git search -diff to compare all commits in your repository history.
* Run a git rebase to find the buggy commit.
* Use git bisect to compare the buggy commit to an early commit that works as expected.

Q10. Why would the following command be used?

git rebase -i HEAD~10

* To run a comparative search of the last 10 commits for differences
* To list the last 10 commits and modify them with either the squash or fixup command
* To delete the last 10 commits and reset the HEAD
* In order to locally cache the last 10 commits

Q11. Why would you use a pre-receive hook in your remote repository?

* You wouldn’t, you would use it in the local repository
* To execute a script when a remote receives a push that is triggered before any refs are updated
* To fire a script after updates are made to the remote repository
* To debug all commit tags and release versions

Q12. What option can you use to apply git configurations across your entire git environment?

* –all
* –master
* –global
* –update

Q13. How could you squash multiple commits together without using git merge –squash?

* Caching
* You can’t. git merge –squash is the only git command for that operation.
* Rebasing
* Reflogging

Q14. If you cloned an existing git repository, what would happen?

* A new copy would overwrite the central repository
* A copy of the repository would be created on your local machine
* Nothing, cloning is not a supported git function
* A copy of the repository would be created on the hosting platform

Q15. How can you display a list of files added or modified in a specific commit?

* Find the commit in the remote repository, as that’s the only place that kind of information is stored.
* Use the diff-tree command with the commit hash.
* Run git commit –info with the commit hash.

Q16. What files is this .gitignore programmed to leave out?

#.swiftbuild/ \*.txt\*.metadata

* All files with a .swift, .txt, or metadata file extension, as well as the entire build directory
* Only the build directory
* All files in the build directory, as well as files ending with .txt or .metadata
* Only files with .swift and .txt extensions.

A line starting with # serves as a comment. Hence # .swift does not do anything. See man gitignore.

Q17. After you make changes to a local repository, you run the following command. What will this do?

git commit -a -m “Refactor code base”

* Nothing, you can’t use multiple options in the same command
* Adds all new files to the staging area
* Commits all new files with a message
* Adds all modified files to the staging area, then commits them with a message

Q18. After checking your git status you get the following output, which shows the file beta-notes.js in the commit but also unstaged. How can this situation occur?

Change to be committed: (use “git reset HEAD <file>…” to unstage)modified: beta-notes.jsChanges not staged for commit:(use “git add <file>…” to update what will be committed)(use “git checkout –<file>…” to discard changes in working directory) modified: beta-notes.js

* There were two copies of beta-notes.js but one was deleted
* beta-notes.js was staged, then modified afterwards, creating two different versions of the file
* Two copies of beta-notes.js were created, but only one is being tracked
* There are two tracked copies of beta-notes.js, but one was removed from the commit

Q19. Where are files stored before they are committed to the local repository?

* Saved files
* git documents
* Staging area
* git cache

Q20. What commands would you use to force an overwrite of your local files with the master branch?

·         git pull –allgit reset –hard origin/master ·         git pull -u origin mastergit reset –hard master ·         git pull origin mastergit reset –hard origin/myCurrentBranch ·         git fetch –allgit reset –hard origin/master

The command pull is fetch followed by either merge or rebase (in this case, merge). We don’t want to merge. Merge would be an action to our **repository**. We just want to overwrite our **local files**.

Q21. Which statement is true when you use the git add -A command?

* Only new files in the working directory are staged to the index.
* All new and updated files from the working directory are staged to the index.
* All files in the working directory are staged to the index in alphabetical order.
* Only updated files in the working directory are staged to the index.

Q22. You find that your project has a tag and branch both named push-notifications, which causes confusion when trying to print out given reference. How can you specify which branch you want to look at?

* use git show refs/push-notifications
* use git show push-notifications
* use git show head/refs/push-notifications
* use git show refs/head/push-notifications

[Reference](https://geedew.com/fixing-git-branch-and-tag-name-collision/)

Q23. Your team lead needs a list of all commits that will be moved before you perform a rebase. Which command can you use to access that information?

* git rebase -log
* git rebase -i
* git rebase -verbose
* git rebase -all

Q24. What is the operation doing given the Git commands below?

git bisect startgit bisect bad 5d41402abc4b2a76b9719d911017c592git bisect good 69faab6268350295550de7d587bc323d

* It runs a merge of a good commit that is discovered using a known bad commit and known good commit
* It marks a commit for deletion using a known bad commit and known good commit to determine which commit introduced a bug
* It defines a bad commit and resets the HEAD using a known bad commit and known good commit
* It performs a binary search using a known bad commit and known good commit to determine which commit introduced a bug

Q25. In a situation where you have several commits for a single task, what is the most efficient way to restructure your commit history?

* Cherry pick the related commits to another branch.
* Delete the task commits and recommit with a new message.
* Squash the related commits together into a single coherent commit.
* Stash the related commits under a new hash.

Q26. Which of the following is true of the git push command?

* By default a push doesn’t send tags to the remote repository.
* Commits can only be tagged when they are created.
* Tags are pushed to the remote repository with their respective commits.
* Only annotated tags are automatically pushed to the remote repository with a commit.

[Reference](https://git-scm.com/book/en/v2/Git-Basics-Tagging#_sharing_tags)

Q27. After pushing commits to the remote repository for the first time using the command below, what shorthand command can you use in future?

git push -u origin master

* git push master
* git push origin
* Same as before, git push -u origin master
* git push

Q28. How would you create a custom shortcut or command across your git environment?

* Run git hotfix with the shortcut name.
* Assign a shortcut or command using git options file.
* Use the git custom-key command.
* Create an alias in the git config file.

Q29. What is the status of the beta-notes.js file in the following output?

Changes not staged for commit:(use “git add <file>…” to update what will be committed)(use “git checkout — <file>…” to discard changes in working directory) modified: beta-notes.js

* beta-notes.js is untracked and has been modified.
* beta-notes.js is a tracked file and has been modified, but has not been added to the current commit.
* beta-notes.js is untracked but has been added to the current commit.
* beta-notes.js is tracked, and the modified file has been added to the current commit.

Q30. What command would let you modify your previous commit?

* –fix
* –quickfix
* –modify
* –amend

Q31. What is the best way to characterize the git commit structure?

* Data array
* Data log
* Data snapshot
* Data dictionary

Q32. What change will the following command make to the staging area files?

git rm –cached testfile.js

* testfile.js will be removed from the staging area and its changes no longer tracked.
* testfile.js will be removed from the staging area but its changes will still be tracked.
* A copy of testfile.js will be cached on your desktop.
* The current copy of testfile.js will be saved in the staging area.

Q33. After you’ve successfully merged two branches and committed the changes, what is the next step in keeping your git structure organized?

* Use the git reset –soft HEAD to roll back one commit.
* Run git branch -d <branch name> to delete the merged branch.
* Use git clear-all to clean up any hanging files.
* Run git rebase to move the current commit to its original location.

Q34. While modifying a file, you’re unexpectedly assigned an urgent bug fix on another branch. How can you temporarily save your local work without committing?

* This is not possible, as you cannot save locally without committing.
* Run git hold to save a local copy of what you’re doing to return to later.
* Save your work with git local-cache.
* Use git stash to save your work and come back later and reapply the stashed commit.

Q35. What command would you use to create a new git repository?

* git add
* git start
* git new
* git init

Q36. While working on a feature branch you try to use “git rerere” to solve a recurring merge conflict but nothing is happening. What could be causing this issue?

* The “-all” option isn’t added to the command.
* “rerere.enabled” isn’t enable in the config file.
* The commit hash is missing.
* The filepath isn’t specified.

Q37. Which setting determines what pager is used when Git pages output?

* core.page
* page
* pager
* core.pager

Q38. What does commit object contain?

* A set of files, representing the state of a project at a given point of time.
* Reference to parent commit objects.
* An SHA1 name, a 40-character string that uniquely identifies the commit object.

Q39. Which option enables inclusion of committer name in custom log format?

* %ce
* %cr
* %cd
* %cn

Q40. How many ways are present in Git to integrate changes from one branch into another?

* 3
* 5
* 2
* 4

In Git, there are two main ways to integrate changes from one branch into another: the merge and the rebase. [Reference](https://git-scm.com/book/en/v2/Git-Branching-Rebasing)

Q41. Which user should be created first during setting up of SSH?

* git
* admin
* root
* None of these

Q42. Which command will list tags with the 1.4.2 series?

* git tag ‘v1.4.2’
* git tag -I ‘v1.4.2.\*’
* git tag-list ‘v1.4.2\*’
* git tag ‘v1.4.2\*’

Q43. Which of the following is an integration manager?

* lieutenants
* benevolent dictator
* Depends upon project type
* Depends upon data

Q44. Which Git command begins tracking of a new file?

* add
* addfile
* begin
* track

Q45. Which of the following is called dumb protocol?

* All of these
* SSH
* Git
* HTTP

Q46. Which key press returns a set of suggestions to pick from, when writing a Git command?

* Control
* Shift
* Tab
* Alt

Q47. Which of these terms best describes Git?

* Distributed Version Control System
* Issue Tracking System
* Integrated Development Environment
* Web-Based Repository Hosting Service

Q48. Which command gets a copy of an existing Git repository?

* duplicate
* replicate
* copy
* clone

Q49. How does Git think of its data?

* File
* None of these
* Snapshot
* Folder

Q50. Which option enables inclusion of author name in custom log format?

* %am
* %ad
* %ae
* %an

Q51. Which version onwards did Git offer reversing a file back to what it looked like when last committed?

* 1.7
* 1.6
* 2.0
* 1.8

Q52. Which strategy is used by Git for merging two branches?

* LIFO
* recursive
* FIFO
* octopus

Q53. What does refs store?

* SHA-1 value
* None of these
* Branch name
* Project name

Q54. What Language is used in GIT?

* C
* C++
* C#
* Java

Q55. What is usually the extension of file which has the public key?

* ssh
* pub
* key
* pk

Q56. What is the difference between initializing a normal repo and a bare repo?

* Bare repos store their git history in a .git subfolder.
* Bare repos do not have the .git extension.
* Bare repos do not come with working or checked-out source files.
* Bare repos should be used for local rather than remote repos.

Q57. How many individual commits can a single repository have?

* any number of commits
* only one commit local per repository
* only three commits per branch
* only one commit per HEAD

Q58. What types of tags does Git support?

* heavy and concise
* lightweight and immutable
* heavy and annotated
* lightweight and annotated

Q59. After staging a series of changes to the index, which command could you use to review them prior to a commit?

* git diff –cached
* git diff
* git diff –HEAD
* git status -v -v

Q60. What does the git stash drop command do?

* removes the most recent stash entry
* deletes the stash
* lists everything in the stash
* throws away the oldest entry

Q61. What command creates a new branch from the currently checked-out branch?

* git -b checkout <nameOfBranch>
* git branch
* git checkout <nameOfBranch>
* git checkout -b <nameOfBranch>

Q62. After mistakenly staging a file named myFile to the index, how would you remove it from the index to exclude it from your next commit?

* Use git reset HEAD^.
* Use git reset myFile.txt.
* Use git -rm myFile.txt.
* Use git reset.

Q63. What happens if you run this command from your master branch?

git checkout -b beta-test

* The beta-test branch will be checked out of the current commit.
* The beta-test branch will be checked out and deleted.
* A new branch called beta-test will be created and switched to.
* The beta-test branch will be merged with the master branch.

Q64. How does Git internally manage branches?

* by creating a pointer to the most recent snapshot/commit for the branch.
* by creating a data array of branches in the same repository.
* by creating a data dictionary of code changes.
* be creating a debug log that stores repository changes.

Q65. You want to perform a git reset but cannot recall all of the available options. What command would you use to see a description of them?

* git help reset
* git -h reset
* git options reset
* git reset help

Q66. What is a remote repository?

* a version of the repository that mirrors changes made in the local repository’s master branch for open-source collaboration efforts
* the lead repository elected by the Git arbitrator found within local repositories of collaborating team members
* a read-only version of the repository stored on a backup server in case local repositories become lost or corrupted
* a version of the repository hosted on the internet or network that is pushed to or pulled from by collaborators

Q67. After modifying some existing files in a repository, you decide to discard the changes. What command can you use?

* git restore
* git undo
* git clean
* git checkout

Q68. After starting to merge a feature branch into your master branch, you encounter a merge conflict and decide you do not want to perform the merge. How can you stop the merge and restore to the pre-merge state?

* Use git restore -p.
* Use git merge -u.
* Use git merge –abort.
* Use git merge –undo.

Q69. If you have several commits for a single feature, what is the most efficient way to restructure your commit history?

* Use git stash to consolidate the commits under a new hash.
* Use git squash to consolidate the commits together into a single coherent commit.
* Delete the task commits and recommit with a new message.
* Use git cherry-pick to place the commits in another branch.

Q70. Which command correctly creates a lightweight tag?

* git tag v3.8.1
* git tag –light “v3.8.1”
* git tag v3.8.1 —-annotate -m “<tagMessage>”
* git tag -l v3.8.1

Q71. What is the main issue with using git rebase when working with multiple developers?

* Rebase affects only your repository and creates a diff in the master branch.
* Rebase creates a temporary copy of the master branch in the remote repo.
* Rebase moves the HEAD of the remote master branch one commit forward.
* Rebase deletes all commit history for the new feature branch.

Q72. What Git workflow is used by teams that collaborate on a single branch and avoid creating long-lived development branches?

* Git flow
* Mainline flow
* Trunk-Based Development
* GitHub flow

Q73. Which option on the git log command allows you to limit output to commits made after certain data?

* –since
* –sinceWhen
* -<n>
* –afterDate

Q74. How would you delete unreachable objects older than a specified time from your project database?

* git cache –obsolete <time>
* git branch –rebase <time>
* git delete –inert <time>
* git prune –expire <time>

Q75. What conflicts can occur when forcing a push after rebasing?

* The remote master branch could have existing changes overwritten.
* The origin URL will be reset to its default value.
* The current HEAD will be deleted and can’t be reinstated.
* Nothing, it’s common practice to force a push after rebasing.

Q76. How does this command alter the currently checked-out branch?

git reset –soft HEAD^

* It resets the working branch to the first commit.
* It sets HEAD to previous commit and leaves changes from the undone commit in the stage/index.
* It deletes all previous commits and resets the repository history back to its initial state.
* It keeps the HEAD at the current commit, but clears all previous commits.

Q77. What is the difference between Git and SVN?

* Git works only on Linux, while SVN works on all operating systems.
* SVN works only on Linux, while Git works on all operating systems.
* SVN is a centralized system, while Git is a distributed system.
* Git a centralized system, while SVN is a distributed system.

Q78. This command is an example of what kind of tag?

git tag -a v1.4 -m “ABCD v1.5”

* verbose
* annotated
* lightweight
* deferred

Q79. What is the difference between a soft reset (git reset –soft) and a hard reset (git reset –hard) ?

* A soft reset only changes the commit that HEAD points to, while a hard reset resets the index and working tree to match the specified commit, discarding any changes.
* A soft reset caches the old HEAD pointer, while a hard reset deletes it entirely.
* A hard reset changes only where the HEAD is pointing, while a soft reset changes the HEAD and index.
* A hard reset caches the old HEAD pointer, while a soft reset deletes it entirely.

Q80. Consider the following Git workflow:

 Which of the following options is correct ?

* 1. Develop 2. Release 3. Hotfix 4. Feature 5. Master
* 1. Master 2. Release 3. Hotfix 4. Feature 5. Develop
* 1. Develop 2. Master 3. Hotfix 4. Feature 5. Develop
* 1. Master 2. Hotfix 3. Develop 4. Feature 5. Release

Q81. What information does the git config file store?

* shell scripts and flags
* keychain and account information
* local and global repository options
* pre-compile scripts and settings

[Reference](https://www.atlassian.com/git/tutorials/setting-up-a-repository/git-config#:~:text=The%20git%20config%20command%20is,modify%20a%20configuration%20text%20file.)

Q82. What is version control?

* a type of architecture used to manage large databases
* a system that shows, tracks, and controls changes to a set of files over time
* a programmatic design pattern used to manage code between multiple engineering teams
* a type of software that links a project with a GitHub repository

Q83. What is the difference between using the git stash and git stash pop commands?

* git stash removes a commit from the repo history, while git stash pop saves changes to multiple branches.
* git stash saves changes to multiple branches, while git stash pop removes a commit from the repo history.
* git stash removes the most recent commit, while git stash pop saves current changes.
* git stash creates a stash entry, while git stash pop places the saved state onto the working directory.

Q84. Which command can be used to list the branches that have been merged into the currently checked-out branch?

* git master –status
* git branch –status
* git branch –merged
* git status –merged

Q85. How would you configure Git to abort a commit if a smoke test script fails?

* Create a post-commit shell script that triggers the action.
* Create a post-commit hook to trigger the script.
* Create a pre-commit hook to trigger the script.
* Create a pre-commit shell script that triggers the action.

Q86. Which use case is NOT a good candidate for a Git hook?

* state dependent environment changes
* continuous integration
* increasing code coverage
* enforcing commit rules

Q87. After starting to work on a new feature and creating new files in the working directory related to it, the customer determined the feature was no longer required. What command can be used to remove the untracked files from the working directory ?

* git clean -f
* git rm .
* git reset HEAD
* git checkout .

Q88. What information do Git reflogs (reference logs) store?

* shell script pointers and keychain credentials
* updates to branch tips and other references in the local repository
* release notes and hook script values
* tag and versioning information

Q89. You have just completed rebasing your master branch and need to manually update the remote master, even though there is a merge conflict. How can you accomplish this?

* git push –overwrite
* git push –update
* git push –assert
* git push –force-with-lease

Q90. What is the difference between git fetch amd git pull

* git fetch creates a new branch off the master branch, while git pull creates a new branch off the local repository’s master branch.
* git pull downloads new data from a remote repository without integrating it into local files, while git fetch updates the current HEAD branch with the latest changes from the remote server.
* git fetch updates remote tracking branches with changes from a remote repository, while git pull updates remote tracking branches with changes from a remote repository and merges them into their corresponding local branches.
* git fetch downloads and merges data from the local repository, while git pull informs your colleagues you are about to make changes to the master branch.

Q91. What command displays the difference between the working tree and the stage/index area, as well as files not tracked by Git?

* git current
* git status
* git local
* git context

Q92. Your current repository has three branches: master, beta, and push-notifications. You have just finished the notification feature and commit the changes to the push-notification branch, and you want to include them in the beta branch. How can you accomplish this?

* Check out the beta branch and run git merge push-notifications.
* Check out the push-notifications branch and run git merge beta.
* Check out the master branch and run git merge beta -> push-notifications.
* Delete the push-notifications branch; it will be committed to the master branch automatically.

Q93. You would like to restore some previously stashed work to a new branch. How can you do that?

* Use git branch .
* Add the stashed commits to the current commit, then create a new branch.
* Use git checkout -b.
* Run git stash branch .

Q94. You is the difference between git branch -d and git branch -D?

* -D deletes the local branch, while -d deletes the branch regardless of push and merge status.
* -d deletes the current commit head, while -D deletes the entire branch.
* -d deletes the local branch, while -D deletes the local branch regardless of push and merge status.
* -D deletes the current commit head, while -d deletes the entire branch.