**GIT Assignment 3**

**1Q.** What is git stockpile, and how does it work? How do you use it?

Ans.

A **stockpile** is a pile or storage location for bulk materials, forming part of the [bulk material handling](https://en.wikipedia.org/wiki/Bulk_material_handling) process.

Stockpiles are used in many different areas, such as in a port, refinery or manufacturing facility. The stockpile is normally created by a [stacker](https://en.wikipedia.org/wiki/Stacker). A [reclaimer](https://en.wikipedia.org/wiki/Reclaimer) is used to recover the material. Stockpiles are normally stacked in stockyards in refineries, ports and mine sites.

A simple stockpile is formed by machinery dumping coal into a pile, either from dump trucks, pushed into heaps with bulldozers or from conveyor booms. More controlled stockpiles are formed using stackers to form piles along the length of a conveyor, and reclaimers to retrieve the coal when required for product loading, etc.

Individuals may also choose to stockpile certain commodities (e.g. food, medical supplies), that they fear may not be available to purchase in the future. For example, in March 2019, one in ten British shoppers were reported to be stockpiling food prior to [Brexit](https://en.wikipedia.org/wiki/Brexit).[[1]](https://en.wikipedia.org/wiki/Stockpile#cite_note-1)

In the construction field stockpile volume measurement is a monthly work program. We can calculate volume of a stockpile manually or by using different types of software. Calculating the volume of a stockpile manually does not require any software. Software used to calculate stockpile volumes can either be proprietary, such as Microsoft Excel and Autocad, or Libre, such as Libre Office Calc and OpenSCAD.

2Q. Describe the various branching strategies?

Ans.

**Master branch**

This is the **main** branch of the repository in which we have the latest stable code of production.

General rules:

- Access to direct merge is restricted

- Best practice is to create a CI/CD pipeline to merge code into this branch after deployment is done in Production

- It should always have the latest stable version of production server

- Allow access to only CD tools like [Jenkins](https://www.engati.com/blog/jenkins-pipeline) to make commits to this branch

The master branch should never have unreleased code, i.e., commits made but not yet released).

‍

**Integration branch**

This is the most**important and active** branch of the repository from which we make releases to the production server.

General rules:

- Access to direct merge is restricted

- Code is merged into this branch when it becomes eligible for production deployment

- Code in this branch should always be in a deployable state to production

- QA tested code should be into this branch using CI/CD tools

Ideally, the Integration branch should also never have unreleased code, some time on the deployment day when the code is merged a few hours before deployment time.

**Dev-deploy branch**

This branch will be used primarily for deploying on-going development work to the Dev environment. Since multiple teams may work on different features, projects and bugs at the same time, all of them need a Dev environment to test the changes before moving to QA. The idea of the Dev-deploy branch is to merge multiple feature branches to a common branch and deploy the same.

This is so that everyone can use a shared development environment at the same time to validate changes.

‍

3Q. How do you remove data from Git without being removed from your system?

Ans.

We've mentioned that ***git rm FILE*will remove files from the index and local working tree by default**.

However, the *git rm* command provides the *–cached* option to allow us only to remove files from the repository's index and keep the local file untouched.

Next, let's try it with the *user-list.txt* file:

$ git rm --cached user-list.txt

rm 'user-list.txt'

Copy

As the output above shows, the *user-list.txt* file has been removed. So now, let's execute the *git status* command to verify it:

$ git status

On branch master

Your branch is up to date with 'origin/master'.

Changes to be committed:

  (use "git restore --staged <file>..." to unstage)

deleted:    user-list.txt

Untracked files:

  (use "git add <file>..." to include **in** what will be committed)

user-list.txt

Copy

As we can see, the *user-list.txt* is “*deleted*“. Further, since its local copy is still there, it has been marked as “untracked”.

We can similarly remove the *logs* directory. However, since it's a directory, we need to additionally pass the *-r (recursively)* option to the *git rm* command:

$ git rm --cached -r logs

rm 'logs/server.log'Copy

Now, let's commit our changes:

$ git commit -m 'remove user-list.txt and logs'

[master ee8cfe8] remove user-list.txt and logs

 2 files changed, 4 deletions(-)

 delete mode 100644 logs/server.log

 delete mode 100644 user-list.txt

Copy

Then, let's check the currently staged files using the [*git ls-files*](https://git-scm.com/docs/git-ls-files)command:

$ git ls-files -c

.gitignore

README.md

some-file.txt

Copy

As the output shows, the target file and directory are not there anymore. Also, the local copies are kept. Therefore, we've solved the problem.

If we like, we can add them to the *.gitignore* file to prevent Git from tracking them again.

4Q. In Git, what does 'index' or 'hosting location' mean?

The Git index is a critical data structure in Git. It serves as the “staging area” between the files you have on your filesystem and your commit history. When you run git add, the files from your working directory are hashed and stored as objects in the index, leading them to be “staged changes”. When you run git commit, the staged changes as stored in the index are used to create that new commit. When you run git checkout, Git takes the data from a commit and writes it to the working directory *and* the index.

In addition to storing your staged changes, the index also stores filesystem information about your working directory. This helps Git report changed files more quickly. One problem is that the index stores this information for every file at HEAD, even if those files are outside of the sparse-checkout definition. This means that the index can be much larger in a monorepo than it would be if your important subset of files was in its own repository.

Throughout the past year, the Git Fundamentals team at GitHub contributed a new feature to Git called the *sparse index*, which allows the index to focus on the files within the sparse-checkout cone. If you are in a repository that can use sparse-checkout, then you can enable the sparse index using these commands:

git sparse-checkout init --cone --sparse-index

5Q. What is the difference between 'git remote' and 'git duplicate'?

Ans.

GIT REMOTE add just creates an entry in your git configuring  that specifies a name for a particular URL. You must have an existing git report  to use this.

GIT CLONE creates a new git repository by copying an existing one located at the URL you specify.

**The differences between git clone and git remote:**

**git clone:**

Will physically download the files into your computer. It will take space from your computer. If the repo is 200Mb, then it will download that all and place it in the directory you cloned.

**git remote add:**

Won't take space! It's more like a pointer! It doesn't increase your disk consumption. It just gets a snapshot of what branches are available and their git commit history I believe. It doesn't contain the actual file/folders of your project.

If you do:

git remote add TechLeadRepo  git://github.com/user/test.git

then you haven't added anything to your computer. After you've added it in your remote branches then you're able to get a list of all branches on that remote by doing:

git fetch --all

upon fetching (or pulling), you download the files And then if you wanted to do get your colleague's feature22 branch into your local, you'd just do

git checkout -b myLocalFeature22 TechLeadRepo/feature22

Had you cloned his repo then you would have to go into that local repository's directory and simply just checkout to your desired branch

i  hope this information may help you

6Q. What is Git Remote's purpose?

Ans.

The git remote command lets you create, view, and delete connections to other repositories. Remote connections are more like bookmarks rather than direct links into other repositories. Instead of providing real-time access to another repository, they serve as convenient names that can be used to reference a not-so-convenient URL.

For example, the following diagram shows two remote connections from your repo into the central repo and another developer’s repo. Instead of referencing them by their full URLs, you can pass the origin and john shortcuts to other Git commands.

The git remote command is essentially an interface for managing a list of remote entries that are stored in the repository's ./.git/config file. The following commands are used to view the current state of the remote list.

8Q. How can you distinguish between the git pull and git bring commands?

GIT PULL :-

The git pull command is used to fetch and download content from a remote repository and immediately update the local repository to match that content. Merging remote upstream changes into your local repository is a common task in Git-based collaboration work flows. The git pull command is actually a combination of two other commands, git fetch followed by git merge. In the first stage of operation git pull will execute a git fetch scoped to the local branch that HEAD is pointed at. Once the content is downloaded, git pull will enter a merge workflow. A new merge commit will be-created and HEAD updated to point at the new commit.