Git/Github Notes

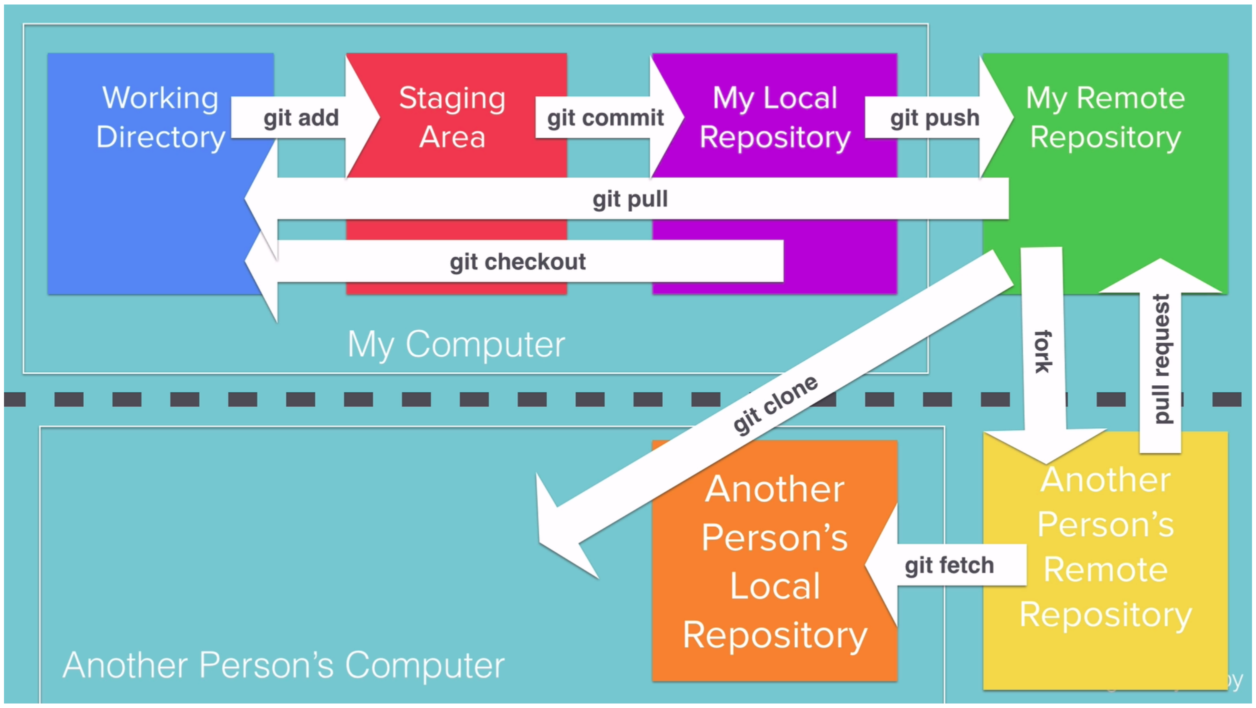

Table of Contents :

1. **Version Control System (VCS)**
    a. Centralized version control system (CVCS).
    b. Distributed/Decentralized version control system (DVCS)
2. **Git**
3. **Initialize Git**
4. **Git Adding New Files**
5. **Git Staging Environment**
6. **Git Commit**
    a. Git commit log
7. **Git Help**
8. **Git Branch**
9. **Git checkout**
10. **Git Pull from GitHub**
    a. fetch
    b. merge
11. **Fork a Repository**
12. **Clone a Fork from GitHub**
13. **References**

Version Control System (VCS) is a software that helps software developers to work together and maintain a complete history of their work.

Listed below are the **functions of a VCS** :

- Allows developers to work simultaneously.
- Does not allow overwriting each other's changes.
- Maintains a history of every version.

Following are the **types of VCS** :

- Centralized version control system (CVCS).
- Distributed/Decentralized version control system (DVCS)

Centralized version control system (CVCS)	**Decentralized version control system (DVCS)**
• Uses a central server to store all files and enables team collaboration. • But the major drawback of CVCS is its single point of failure, i.e., failure of the central server. • Unfortunately, if the central server goes down for an hour, then during that hour, no one can collaborate at all. • And even in the worst case, if the disk of the central server gets corrupted and proper backup has not been taken, then we will lose the entire history of the project.	• DVCS clients not only check out the latest snapshot of the directory but they also fully mirror the repository. • If the server goes down, then the repository from any client can be copied back to the server to restore it. • Every checkout is a full backup of the repository. • Git does not rely on the central server and that is why we can perform many operations when offline.

Git :

- a version control system.
- helps to keep track of code changes.
- used to collaborate on code.

Working with Git

- Initialize Git on a folder, making it a Repository.
- Git now creates a hidden folder to keep track of changes in that folder.
- When a file is changed, added or deleted, it is considered modified.
- Select the modified files we want to Stage.
- The Staged files are Committed, which prompts Git to store a permanent snapshot of the files.

- Git allows us to see the full history of every commit.
- We can revert back to any previous commit.
- Git does not store a separate copy of every file in every commit, but keeps track of changes made in each commit!

Initialize Git

```
git init
Initialized empty Git repository in /Users/user/myproject/.git/
```

Git Adding New Files

Files in the Git repository folder can be in one of 2 states:

Tracked - files that Git knows about and are added to the repository
Untracked - files that are in working directory, but not added to the repository

When we first add files to an empty repository, they are all untracked. To get Git to track them, we need to stage them, or add them to the staging environment.

```
git status
```

```
On branch master

No commits yet

Untracked files:
  (use "git add ..." to include in what will be committed)
    index.html

nothing added to commit but untracked files present (use "git add" to
track)
```

Git Staging Environment

- One of the core functions of Git is the concepts of the Staging Environment, and the Commit.
- As we are working, we may be adding, editing and removing files. But whenever we hit a milestone or finish a part of the work, we should add the files to a Staging Environment.
- Staged files are files that are ready to be committed to the repository we are working on.

```
git add index.html
```

```
git status
```

```
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached ..." to unstage)
    new file: index.html
```

```
git add --all
```

```
git status
```

```
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached ..." to unstage)
        new file:   README.md
        new file:   bluestyle.css
        new file:   index.html
```

Git Commit

- Since we have finished our work, we are ready to move from stage to commit for our repo.
- Adding commits keep track of our progress and changes as we work. Git considers each commit change point or "save point". It is a point in the project we can go back to if we find a bug, or want to make a change.
- When we commit, we should always include a message.
- By adding clear messages to each commit, it is easy for us (and others) to see what has changed and when.

```
git commit -m "First release of Hello World!"
```

```
[master (root-commit) 221ec6e] First release of Hello World!
 3 files changed, 26 insertions(+)
 create mode 100644 README.md
 create mode 100644 bluestyle.css
 create mode 100644 index.html
```

Git Commit Log
To view the history of commits for a repository, we can use the log command:

Git Help

If we are having trouble remembering commands or options for commands, we can use Git help.
There are a couple of different ways we can use the help command in command line:

`git command -help` - See all the available options for the specific command
`git help --all` - See all possible commands

Git Branch

In Git, a branch is a new/separate version of the main repository.

```
git branch hello-world-images
```

```
git branch
  hello-world-images
* master
```

We can see the new branch with the name "hello-world-images", but the * beside the master specifies that we are currently on that branch.

Git checkout:

checkout is the command used to check out a branch. Moving us from the current branch, to the one specified at the end of the command:

```
git checkout hello-world-images
```

```
Switched to branch 'hello-world-images'
```

Git Pull from GitHub

Pulling to Keep up-to-date with Changes
When working as a team on a project, it is important that everyone stays up to date.
Any time we start working on a project, we should get the most recent changes to the local copy.
With Git, we can do that with pull.

pull is a combination of 2 different commands:

- fetch
- merge

Git Fetch
fetch gets all the change history of a tracked branch/repo.

Git Merge
merge combines the current branch, with a specified branch.

Fork a Repository

A fork is a copy of a repository. This is useful when we want to contribute to someone else's project or start our own project based on theirs.

Clone a Fork from GitHub

Now we have our own fork, but only on GitHub. We also want a clone on our local Git to keep working on it.

A clone is a full copy of a repository, including all logging and versions of files.

References :

- Git Tutorial
- Git - Basic Concepts

