Task 1

# Differences between Git and other version control systems.

## Snapshots, Not Differences

The major difference between Git and any other VCS is the way Git thinks about its data. Conceptually, most other systems store information as a list of file-based changes. These think of the information they keep as a set of files and the changes made to each file over time. Git doesn’t think of or store its data this way. Instead, Git thinks of its data more like a set of snapshots of a mini filesystem. Every time commit is being performed, or the state of the project is saved in Git, it basically takes a picture of what all the files look like at that moment and stores a reference to that snapshot. To be efficient, if files have not changed, Git doesn’t store the file again—just a link to the previous identical file it has already stored.

## Almost every operation is local

## Most operations in Git only need local files and resources to operate — generally no information is needed from another computer on the network. Because the entire history of the project exists on the local hard disk most operations seem almost instantaneous. If someone wants to track the changes introduced between the current version of a file and the file a month ago, Git can look up the file a month ago and do a local difference calculation, instead of having to either ask a remote server to do it or pull an older version of the file from the remote server to do it locally. This also means that there is very little that someone can’t do if he is offline or off VPN. If someone gets on an airplane or a train and wants to do a little work, he can easily commit until he gets to a network connection to upload. If someone is at his home and can’t get his VPN client working properly, he can still work. In many other systems, doing so is either impossible or painful.

## Git has Integrity

Everything in Git is check-summed before it is stored and is then referred to by that checksum. This means it’s impossible to change the contents of any file or directory without Git knowing about it. This functionality is built into Git at the lowest levels and is integral to its philosophy. Information can’t be lost in transit or get file corruption without Git being able to detect it.

## Git Generally Only Adds Data

When someone performs actions in Git, he is just only adding data to the Git database. It is very difficult to get the system to do anything that is not undoable or to make it erase data in any way. As in any VCS, changes that haven’t committed yet can be lost or messed up; but after a snapshot is committed into Git, it is very difficult to lose, especially if someone regularly pushes the database to another repository.

# What software tools are available to support teams working with git and GitHub?

For organizing group of people (teams) that participating in many projects then the best way to go is through Organizations. Otherwise if there is only one project then you can go through the collaborators.

There is also the possibility for an individual to contribute through the process of pull requests. There are two ways of a pull request either *Fork and Pull Model* (in case that there is not push access and is used in a public repository) or the *Share* ***Repository Model*. After that the repository owner must serve the pulling request either directly to Github or in the case that conflicts do exist on his local machine. The project owner also has the possibility to track bugs using the feature of Tasks.**

There are two tools that give insight into a repository – Graphs and Network. [Github Graphs](https://github.com/blog/1093-introducing-the-new-github-graphs) provides an insight into the collaborators and commits behind each code repository, while [Github Network](https://github.com/blog/39-say-hello-to-the-network-graph-visualizer) provides visualization on each contributors and their commits across all forked repositories. These analytics and graphs become very powerful, especially when working in teams.

While Github Issues have project management capabilities with Issues and Milestones, some teams might prefer other tools because of other features or existing workflow. Github can be linked with two other popular project management tools – [Trello](https://trello.com/) and [Pivotal Tracker](https://www.pivotaltracker.com). With Github service hooks, updating task can be automated with commits, issues and many other activities. This automation helps in not only saving time, but also increases accuracy in updates for any software development team. The concept of [Continuous Integration](http://en.wikipedia.org/wiki/Continuous_integration) (CI) which is an important part of all software development projects that work with teams can be served with [Travis CI](https://travis-ci.org/). CI ensures that, when a developer checks in their code, an automated build (including tests) detects integration errors as fast as possible. This definitely reduces integration errors and makes rapid iteration much more efficient.

Github offers Code Review. With each commit, Github allows a clean interface for general comments or even specific comments on a line of code. The ability to raise comments or questions on every single line of code is very useful in doing line by line code reviews.

Finally official documentation for the evolving project’s source code can be done using Github Wiki or in case of documenting discussions among team members [Github Hubot](https://github.com/github/hubot) can be used.