Investigation of the use of Git/Github and approaches which may be taken to working with GitHub in Student teams

1. Differences between Git and other version control systems.[1]

a) 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.

b) 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.

c) 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.

d) 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.

[1] http://git-scm.com/book/en/Getting-Started-Git-Basics

2. How does the branch and merge model in Git work?

Any repository starts with a default master branch. By default the master branch points to the last commit made by the user in the series of snapshots committed by Git.

So basically in order to do branch (i.e. to have a new branch) we use the command line $\$\ git\ branch\ [name\ of\ the\ new\ branch]$. The new branch will point to the last commit made just as the master branch. However in this case it is important to know which branch will move forward.

Here it is important to understand the concept of HEAD. It is a pointer to the current branch. By typing the command $\$\ git\ checkout\ [name\ of\ the\ new\ branch]$, the HEAD pointer will move from the master branch to the new branch created by the user.

To make it clearer the following two images show how HEAD, Master and the new Branch –assume it's called testing- look like after each command.

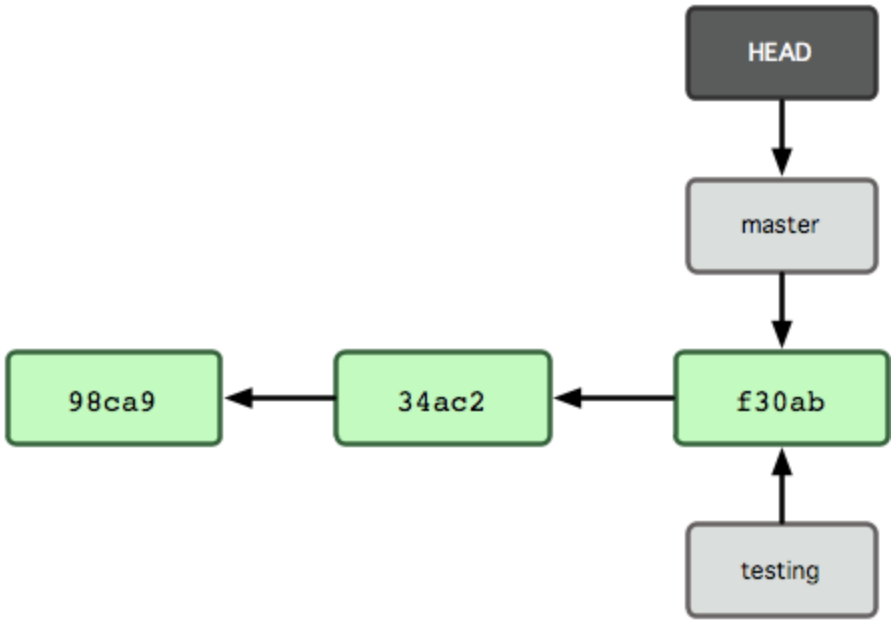

Figure 1: After "$ git branch" command [2]

[2] http://git-scm.com/book/en/Git-Branching-What-a-Branch-Is

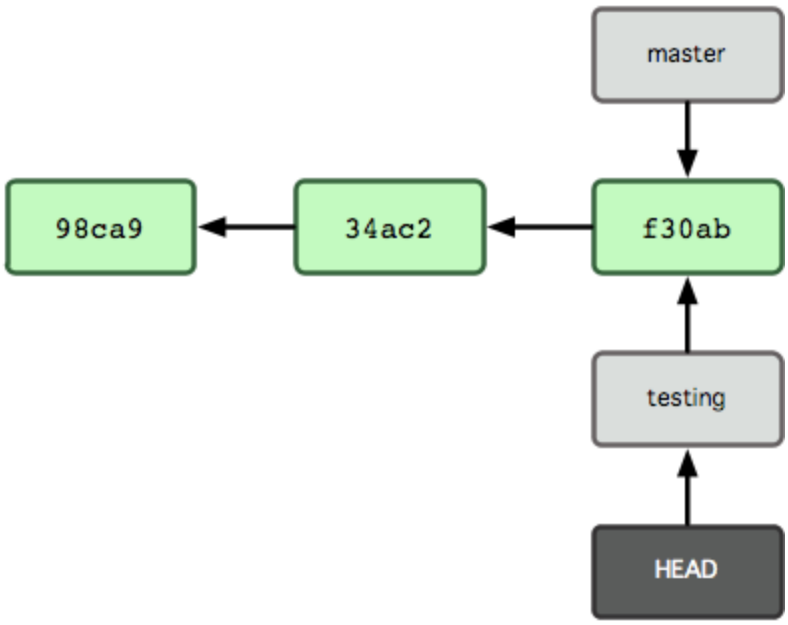

Figure 2: After $ git checkout [3]

So now for any upcoming commits only testing branch –the current branch with HEAD referring to it- will move forward. There other git command that make it possible to change from branch to another, where any changes made at any branch are isolated from each other.

After moving forward the user will indeed need to merge the changes made. In order to do this, two steps will be required:

a) First checking out the branch to merge to - as you may have different branches-. In other word changing the HEAD reference point in order to point to the required branch.
b) Then merging using the merge command

There are two different scenarios of merging:

a) First scenario if the new branch is a direct ancestor of the branch the user wants to merge in. So basically what happens here is a fast forward process. That means the branch you want to merge to will now refer to the same snapshot the ancestor refers to – after committing that snapshot of course. Now both branches are referring to the same snapshot, hence it is a good idea to remove one of them.
b) Second scenario is if the new branch not a direct ancestor. Hence Git needs to three snapshot of the branch to merge in, the branch to merge to and the common ancestor that is identified automatically by Git. Unlike the first scenario a new snapshot will be created and comitted as a result of this three-way merging.

[3] http://git-scm.com/book/en/Git-Branching-What-a-Branch-Is

3. What software tools are available to support teams working with git and GitHub?[4]

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 provides an insight into the collaborators and commits behind each code repository, while Github Network 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 and Pivotal Tracker. 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 (CI) which is an important part of all software development projects that work with teams can be served with Travis CI. 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 can be used.

[4] http://net.tutsplus.com/articles/general/team-collaboration-with-github/

4. What issues / difficulties / solutions have been found by educators using git and GitHub with students?

According to (Xu, 2012), getting familiar with Git and merging the work from multiple members of the group was the main issued he faced while trying to make his student move to the concept of version control using Git.

Some students claimed to face to some problems installing Git on windows. They say it sounds like a second rate citizen on windows, and it is not as easy to learn as other VCS like Mercurial for example.

Another claim is that when you merge two branches “you lose much of the context of those changes”. In other words, after merging you can only know what the last commit was. So you can’t keep tracking of the previous changes made on the merged-in branch before merging. In other VCSs they have what is called named branches, for any named branch you can easily check all of the previous revisions.

One other issue is you can’t do everything with gitgui - Git Hub – pull for example has to be done using command line.