Training outline

- What is code versioning?
- Why is versioning useful?
- Different types of versioning?
- CVCS (Centralized version control system) vs DVCS (Distributed version control system)
- What is git
- Git installation and initial setup
- Local vs remote repositories.
- Bit bucket, github, assembla, beanstalk etc. for managing git source code.
- Three stages in git
- Git commands:
    - git init
    - git clone
    - git config
    - git add
    - git commit
    - git status
    - git log
    - git checkout
    - git revert
    - git reset
    - git clean
    - git branch
    - git merge
    - git commit –amend
    - git rebase
    - git remote
    - git fetch
    - git pull
    - git push
    - git tag
    - git hist
    - git diff
    - git stash
- gitconfig file
- gitignore file
- Git workflows - Branching strategies, branching models.

- Setup choice of editors for commit messages.
- Attaching a commit to a ticket in tool like bitbucket or assembla
- sha1 hash
- Rebasing vs merging
- building custom commands using aliases
- What is bare repository
- Protocols
- SSH Setup and ssh authentication
- gitx, gitk and magit tools
- working with github

Practical (visit http://try.github.io/ for interactive git practical):

- Setup Name and Email for git
- Setup line ending preferences
- Create a program of prepare a source code
- Create the repository using init
- add the program or source code to the repository
- check status of repository
- change the code in one file
- check status again
- Add changes to the staging
- Commit the changes
- Print log of commits using git log in variations like one line, since 5 mins, until 5 mins etc.
- gitk tool
- Tagging a version, tagging previous version, view all tags
- Undoing local changes before staging using checkout command for a file.
- Undoing staged changes before committing.
- Undoing committed changes using git revert command
- Removing a tag
- Amending commits
- Moving files means deleting from one place and adding to other
- Exploring the .git directory: objects, config file, branches and tags, HEAD.
- Creating a branch, adding and committing to the branch
- Switch between branches
- view all branches
- Merging branches
- Creating a conflict

- merging branch to see the conflicts, resolving it and committing the resolved conflict.
- Rebasing vs merging
- Rebasing master with another branch - when to use rebase
- creating github account and its usage
- Clone a remote repository
- git remote for checking what is origin
- list remote branches with git branch –a
- fetching changes using git fetch
- merging fetched changes using local master using git fetch origin master
- pulling changes
- Create a bare repository
- Pushing changes using git push
- difference between two versions of files using diff
- git stash command
- running custom commands using aliases

What is code versioning and how is it useful?

Traditional development process was such that you make changes to the source code or files in an application as you go along and build the application. Imagine you changing a file a few times or making changes to multiple files within a few days. What if you want to see what was changed in the application in last week or say what changes were made by you yesterday? One solution could be to make copies of file (backup) to track that and if you want to see that change just open that file up and see the changes. Imaging your source code having thousands of files and each file having 100s of thousands of lines of code. Managing backups for the files and application will be hectic, right? So what is the solution then? Yes Versioning is the answer to this question.

Versioning or software versioning or revision control means managing different versions of the source code for a program or application. Versioning is useful when you want to track progress of a project or an application. If you manage different versions, you will be able to track different changes that you made during development of an application.

For example there is an application you are developing and say it takes you 7 days to build version 1 of that application. Let's call it v1. After another 7 days version 2 of the application is ready let's call it v2. So if versioning is used you can check v1 even after 15 days and also see the code for the same. For example v1 is well tested and ready and now while v2 is in development there is a huge problem in the application. You know that v1 was well tested

and everything was good ad working well, you can go back to v1 and start developing again and do not need to develop from scratch.

Versioning not only help in versioning of an application as a whole but helps in creating and managing versions of different files. During development, all changes to each and every file are tracked so versions of files are also made. So after a few days you can see what changes are made to a file after you started the development process.

Different Types of Versioning

Where there is a solution, as time goes on there is always more and more solutions that come into picture and things star being more and more simplified or more and more advanced. So is the case with versioning. A version control system can be either a centralized (VCS) or a distributed version control system (DVCS).

An example of Centralized system is SVN and for Distributed it is git, mercurial etc.

Centralized Version Control System (CVCS)

In this type of system a central server is there which keeps all versions of all files (this is called repository) within an application. If multiple developers are working for an application then they can also share their code and also make changes to the application source code. Since it is centralized, everyone working on the project knows what everyone else is working on. Administrators of the server have more control of the versions since there is one centralized location where the versions of the application are stored. Best example for this type of system is SVN.

If for example developers A and B are working for an application they can checkout which means getting the application files from the central server or they can commit which means they can push the changed or source files to the central server. If both have same versions of the application and they start working on their local machines and say developer A commits his changes first and then developer B tries to commit his changes, the system will ask developer B to first checkout the changes that A made which means that B will receive all changes that A made into the application and then B's changes to be applied to that (this is called merging) and that will be then pushed to the central server using the commit. Now, if A wants that latest code which is combination of his changes and the changes B did then he will need to checkout again which will then get the latest source code from the central server.

There can be plenty of questions regarding this. For example what if both of these developers work on the same file and make changes to the same file during development. What happens is first A will make changes to that file and when B tries to commit, system asks him to checkout latest code and in the process after A's changed file is in local of B, it will recognize that and will inform B that they both tried to change the same file and this situation is called conflict. B will then resolve the conflict

by making changes to that file by removing unwanted code which means he can either keep his code in the file or A's code or both and once that is successful, B will then commit the change which ,means that the centralized server will have that code. Now A can checkout that code to get it in his local computer.

Where there is a system, there has to be drawbacks and the main drawback of using centralized version control system is that there is a one point of failure. Which means that if the centralised server where the versions are managed crashes then there is no backup for the same and all developers will have to stop working until that server is up again. Another drawback is that this system can be slow since every command needs to be connected to the central server.

The above drawbacks are enough to bring into picture other version control systems like Distributed version control system which tackles the drawbacks of the traditional version control system.

Distributed Version Control System (DVCS)

Unlike CVCS there is no central server where all versions of the application or the file of the source code is managed. Instead each developer will have the full version control system in their local computers. This means that each user will have a local copy of the repository. The examples for this are git and Mercurial. The main advantage of this system is that it is network of multiple repositories. The question might arise that if there is no central server then how can multiple developers work on the same application or share code with each other.

Here each developer will have their own local repository and still there can be a centralized remote repository which the developers can use to share the code with each other. So when a developer commits the changes, it is reflected into the local repository only. It gets sent to the remote repository when that developer pushes the changes to the remote. After that another developer can get that change by pulling the changes from the remote repository into the local repository. Again question might arise that if here also we have to use a remote repository then again it is single point of failure then why use DVCS. Say there is only one developer working on an application and wants to do version control. If he used DVCS then the versioning will be maintained without the need of a central server which is because DVCS means having a complete repository in local rather than in a central location.

The choice between these two types of version control system depends on your choice and the type of application you are developing plus number of developers building an application. SVN is traditional whereas DVCS like git is latest. So let us look at git in more detail since it is latest.

The three most used DVCS are mercurial, git and bazaar.

A few points to remember:

- Prior to git there were many version control systems like SCCS, RCS, CVS, SVN (Apache), and Bitkeeper's SCM.

- Git is first open source DVCS.
- SVN is two tier architecture whereas git is 3 tier architecture.
- In SVN you have a working copy and a repository whereas in git you have a working copy, a staging index and a repository.
- Once you make a change, you have to first add it to staging index and then commit it to repository so it is three tree architecture.
- To start working with git you have to first install git. Once it is installed, you can start using it to create and manage repositories.
- On windows, once you install git, it comes with a git bash which you use to run git commands. This is similar to ssh terminal.
- To create your first git project open git bash and browse to the directory where you want the project by using "cd " command.
- Once you are in the directory, initialize a git project using git init command. This will create an .init directory here. This is hidden directory by default.
- When a file is added using git add, it puts the file to the staging index ready to be committed. git add . adds all files in current directory. Staging area allows you to pick and choose what you want to commit and what not.
- To remove a file from staging use git rm –cached.
- git commit –a will commit any changed previously committed and not commit untracked (not on staging) files.
- .gitignore file contains what not to track. To not track always a directory use test/*.txt but if you want to always include f1.txt then you can write !test/f1.txt in the .gitignore file.
- **Note:** Repositories on bit bucket and github are bare repositories meaning there is no working directory because no one is working on them.
- **HEAD:** This is reference to the pointer that points to the latest commit on the current branch. Similarly HEAD~ points to commit prior to that, HEAD~1 points to one before that and so on.
- git diff file1 file2 will show the difference between two files.
- git diff file1 will show difference between that file in working directory and staging area.
- git diff --cached file1 or git diff --staged file1 will show difference between staging area and committed (repo).
- Git diff HEAD file1 will show difference between file in working directory and last commit.
- **Note:** All above diff command can be run without file name to get difference of all files. In the difference shown the "-" sign indicates previous version and the "+" sign indicates the latest one.
- git log --stat can be used to see the log with what has been changed, modified etc.
- git log --oneline --graph --decorate can be used to see the log with graphical structure and only one line for each commit.
- To see above for all branches use git log --oneline --graph --decorate --all.
- Use command gitk to open graphical user interface and use gitk --all to view that for all branches.
- If you have added a new file to your working directory (an untracked file) and switch your branch, then git will allow you to switch branch but this new file will be present in the switched branch which will allow committing from this branch too. In this case it doesn't

matter if you have already staged that file. The main fact is if it has been tracked or not (committed previously even once or not.).

- You can create alias for a long command by running alias gl='git log --oneline --graph --all --decorate' . Now if you run gl command, it will identify the alias.
-
-
- **To delete a file:** You can either physically delete it in the working copy and the run git rm filename.txt followed by git commit –m "deleted filename file" . git rm is opposite to git add which is for additions and modifications to be staged whereas git rm is used to stage removal of file.
- You can also delete a file directly from git bash by running git rm filename.txt followed by git commit –m "deleted file".