**About Version Control:** Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later.

**Centralized Version Control System**: The next major issue that people encounter is that they need to collaborate with developers on other systems. To deal with this problem, The CVCS were developed. These systems, such as CVS, Subversion, and Perforce, have a single server that contains all the versioned files, and a number of clients that check out files from that central place. For many years, this has been the standard for version control.

This setup offers many advantages. For example, everyone knows to a certain degree what everyone else on the projects is doing. Admin have fine-grained control over who can do what; and it’s far easier to administer a CVCS than it is to deal with local databases on every client. However, this setup also has some serious downsides. The most obvious is the single point of failure that the centralized server represents. If that server goes down for an hour, then during that hour nobody can collaborate at all or save versioned changes to anything they’re working on.

**Distributed version control system:** This is where Distributed Version Control Systems (DVCSs) step in. In a DVCS (such as GIT, Mercurial), clients don’t just check out the latest snapshot of the files: they fully mirror the repository. Thus if any server dies, and these systems were collaborating via it, any of the client repositories can be copied back up to the server to restore it. Every checkout is really a full backup of all the data. Furthermore, many of these systems deal pretty well with having several remote repositories they can work with, so you can collaborate with different groups of people in different ways simultaneously within the same project. This allows you to set up several types of workflows that aren’t possible in centralized systems, such as hierarchical models

Git:

Configure personal information

* git config --global user.name "username"
* git config --global user.email "email name"
* git config –list: lookup configuration information.

GIT command：

1. git init：initializing a repository in an existing directory. This creates a new subdirectory named .git.
2. git add \*.c. If you want to start version-controlling existing files, you can accomplish that with “git add”
   1. git commit -m "message". Commit your track files with a message.
3. git clone "clone url" rename\_dir : if you want to get a copy of an existing Git repository, please type “git clone”. With “rename” parameter to rename directory.
4. git status : Checking the status of Your files.
5. In order to begin tracking a new file, please use the command “git add”.
6. After updated tracked file, please use the command “git add”.
7. git commit -a -m " message"
   1. git commit -m "message"
8. git rm: delete file
9. git mv file\_from file\_to : move file
10. git log : view the commit history
    1. git log -p -2 :
11. git commit --amend : Change the last commit.
12. git reset HEAD : Unstaging a staged file；Undo changes
13. git checkout – file: unmodifying a modified file.
14. git remote : show the remotes servers
    1. git remote -v :
15. git fetch： fetch all the information that you don’t yet have in your repository from remote service. Then you can merge it into one of your branches.
16. git push [remote-name] [branch-name]: push change data to remote branch.
17. git branch testing ： create a new branch which called “testing”.
18. git checkout testing: switch to an “testing” branch
    1. git checkout -b testing :
19. git checkout master: switch to master branch.
    1. git merge testing: merge branch. From “testing” to master.
    2. git branch -d testing: delete “testing” branch.
    3. git push origin --delete testing: delete remote “testing” branch.
20. A solution to resolve conflict.
    1. git status
    2. git add
    3. git commit
21. git branch: show all branch name
    1. git branch --no-merged
    2. git branch --merged
    3. git branch -a : show all branch name include hide branch.
    4. git checkout -b branch\_name remote\_branch\_name
22. git push: push data to remote branch.
    1. git push origin. Default master branch.
    2. git push origin testing.
23. git tag -a "v2.0.0" -m "Release version 2.0.0"