Review Paper on Centralized and Distributed Version Control System.

AMRUTA SUDHIR VATARE[1], PRATIBHA ADKAR[2]
[1,2]MCA Department Modern College of Engineering Pune, India

Abstract- Version Control System is also called Source Code Management System. First Version Control System is developed in the early 1970's when the Source Code Control System (SCCS) was released. Version Control System is a category of the software tools that helps a software team manage changes to source code to over time. Version Control Software keeps track of every modification to the code in a special kind of database. If mistake is made, developers can track back the clock and compare earlier version of code to help fix the mistake while minimizing disruption to all team members. In this research paper we included the details information about the version controls system and their architecture types, advantages of VCS, disadvantages of VCS, and applications of VCS.

Index Terms- Importance of Version Control System, Centralized VCS, Decentralized VCS, Examples of VCS based on both centralized/decentralized architectures (Subversion, Git), Advantages and disadvantages of VCS. Applications of VCS.

I. INTRODUCTION

Version control system are a category of software tools that helps a software team manage changes to source code over time. Version control software keeps track of every modification to the code in a special kind of database. If a mistake is made, developers can track back the clock and compare earlier version of the code to help fix the mistake while minimizing disruption to all team members.

For almost all software projects, the source code is like the crown jewels – a precious asset whose value must be protected. For most software teams, the source code is a repository of the invaluable knowledge and understanding about the problem domain that the developers have collected and refined through careful effort. Version control protects source code from both catastrophe and the casual degradation of human error and unintended consequences.

Software developers working in the team are continually writing new source code and changing existing source code. The code for a project, app or software component is typically organized in a folder structure or "file tree". One developer in a team may be working on a new feature while another developer fixes an unrelated bug by changing code, each developer may make their changes in several parts of the file tree.

Version Control helps teams solves these kinds of problems, tracking every individual change by each contributor and helping prevent concurrent work from conflicting. Changes made in one part of the software can be incompatible with those made by another developer working at the same time. This problem should be discovered and solved in an orderly manner without blocking the work of the rest of the team. Further, in all software development, any changes can introduce new bugs on its own and new software can't be trusted until it's tested. So, testing and development proceed together until a new version is ready.

Good version control software supports a developer's preferred workflow without imposing one way of working. Ideally it also works on any platform, rather than dictate what operating system or tool chain developers must use. Great version control systems facilitate a smooth and continuous flow of changes to the code rather than the frustrating and clumsy mechanism of file locking – giving the green light to one developer at the expense of blocking the progress of others.

Software teams that do not use any form of version control often run into problems like not knowing which changes that have keep made are available to user or the creation of incompatible changes between two unrelated pieces of work that must then be painstakingly untangled and reworked. If you're a developer who has never used version control you may have added versions to your files, perhaps with suffixes like "final" or "latest" and then had to later deal with a new final version. Perhaps you've commented out code blocks because you want to disable certain functionality without deleting the code, fearing that there may be a use for it later. Version control is a way out of these problems.

Version control software is an essential part of the every-day of the modern software team's professional practices. Individuals software developers who are accustomed to working with a capable version control system in their teams typically recognize the incredible value version control also give them even on small solo projects. Once accustomed to the powerful benefits of version control systems, many developers wouldn't consider working without it even for non-software projects.

II. IMPORTANCE OF VERSION CONTROL SYSTEM.

There are many benefits of using a version control system for your projects. These are some of them in detail.

1. Collaboration: -

Without a VCS in place, you're probably working together in a shared folder on the same set of files. Shouting through the office that you are currently working on file "xyz" and that, meanwhile, your teammates should keep their fingers if us bit ab acceptable workflow. It's extremely error prone as you're essentially doing open-heart surgery all the time: sooner or later, someone will overwrite someone else's changes.

With a VCS, everybody on the team can work absolutely freely – on any file at any time. The VCS will later allow you to merge all the changes into a common version. There's no question where the latest version of a file or the whole project is. It's in a common, central place: your version control system. Other benefits of using a VCS are even independent of working in a team or on your own.

2. Storing Versions (Properly): -

Saving a version of your project after making changes is an essential habit. But without a VCS, this become tedious and confusing very quickly:

How much do you save? Only the changed files or the complete project? In the first case, you'll have a hard time viewing the complete project at any point in time – in the latter case, you'll have huge amount of unnecessary data lying on your hard drive.

How do you name these versions? If you're a very organized person, you might be able to stick to an actually comprehensible naming scheme (if you're happy with "acme-inc-redesign-2013-11-12-v23").However, as soon as it comes to variants (say, you need to prepare one version with the header area and one without it), chances are good you'll eventually lose track.

The most important question, however, is probably this one: How do you know what exactly is different in these versions? Very few people take the time to carefully document each important change and include this in a README file in the project folder.

A version control system acknowledges that there is only one project. Therefore, there's only the one version on your disk that you're currently working on. Everything else- all the past versions and variants- are neatly packed up inside the VCS. When you need it, you can request any version at any time you'll have a snapshot of the complete project right at hand.

3. Restoring Previous Versions: -

Being able to restore older versions of a file (or even the whole project) effectively means one thing: you can't mess up! If the changes you've made lately prove to be garbage, you can simply undo them in a few clicks. Knowing this should make you a lot more relaxed when working on important bits of a project.

4. Understanding What Happened: -

Every time you save a new version of your project, your VCS requires you to provide a short description of what was changed. Additionally (if it's a code / text file), you can see what exactly was changed in the file's content. This helps you understand how your project evolved between versions.

5. Backup: -

A side-effect of using a distributed VCS like Git is that it can act as a backup; every team member has a full-blown version of the project on his disk – including the project's complete history. Should your beloved central server break down (and your backup drives fail), all you need for recovery is one of your teammates' local git repository. [1][4]

III. ARCHITECTURE OF VERSION CONTROL SYSTEMS AND EXAMPLES

A) Centralized/client- server version control system architecture.

The historical and most common architecture found today is the centralized repository. In this architecture, users (clients) connect to a central source(server) for access to the repository. Essentially "Developers check out source code from the central repository into a local sandbox and, after making changes, commit it back to the central repository".

Branching is a process where subsections from the main source code are developed independently and then reassembled back when complete. Branching allows developers to edit section (branches) of the repository without posing major changes to the main source and retaining the revision history of the files. Branches can be implemented in this architecture but not as efficiently as the Distributed Architecture.

Traditional VCSs have a centralized server, which acts as a source code repository. In addition to store the code, these tools maintain the revision history for each commit. In below diagram, we have 3 files A, B and C; and project changes are committed in four faces i.e. initial commit, second, third and fourth commit. When there is a change to the file A and C in the second commit, only the difference or delta changes are applied to the initial file and stored. Similarly, when there is another change performed on top of B and C files in third commit, only the delta changes from B and C are stored as part of the commit.[3][6]

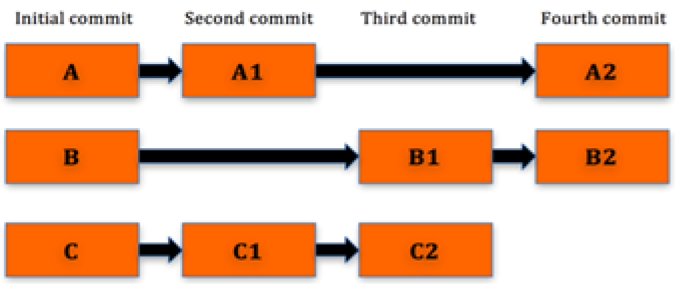

Figure. Working with Centralized VCS.

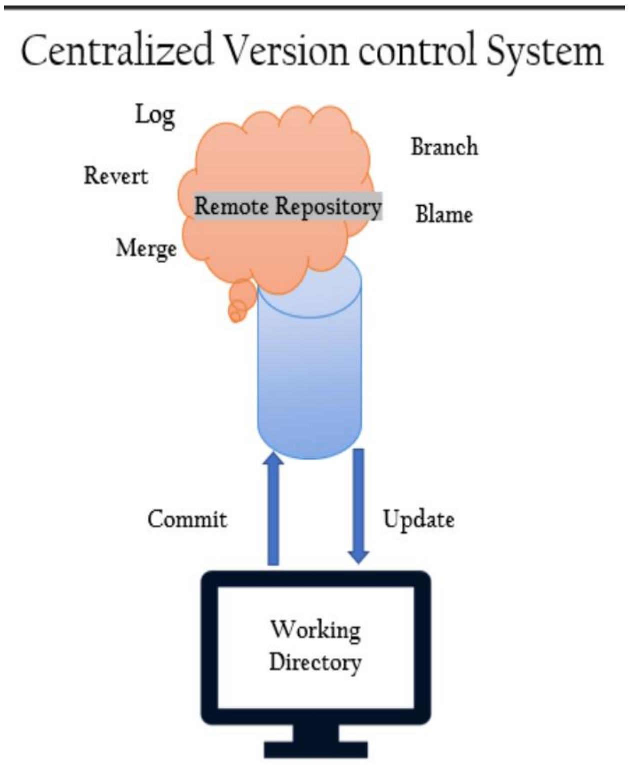

Figure. Centralized VCS Architecture.

Notice that at any given point of time, if we want to get the current state of the entire code base, we will not be able to get since all we can get for a given commit is the delta changes from the last time the files was committed. For example, when we compare two commits, we only get the files that have changes and we can see the actual changes done to the files, which we refer to as delta. You can ask for some previous version of a file(s), but you cannot ask for whole workspace what it was before few commits. If you want to see how the workspace looks at a given commit, you will have to take the base version and path with all the commits done till the desired commit. Hence you will be restricted to either get latest revision; or base revision. Thus, the VCS stores only the differences. [7]

B) Decentralized/Distributed version control systemarchitecture.

This architecture takes a peer-to-peer approach where each peer has a local repository which is similar to the original source repository to which they can commit their changes. The main difference is that each clone is a full-fledged repository not dependent on network access or a central server. Synchronization is conducted by exchanging patches (changesets) via peer to peer, making it possible for independent developers to work asynchronously.

Since Linus Torvalds gave a presentation at Google about a distributed version control system called GIT in May 2007, the adoption and interest for Distributed Version Control Systems has been constantly risin Unlike centralized VCS, where all the heavy lifting is done at the server side and acts as a single point for all the operations and the clients have only the working copy of the code-base; in distributed VCS, each client (referred as collaborator) has its own local repository and will work on its local repository for the most part. [2][3]

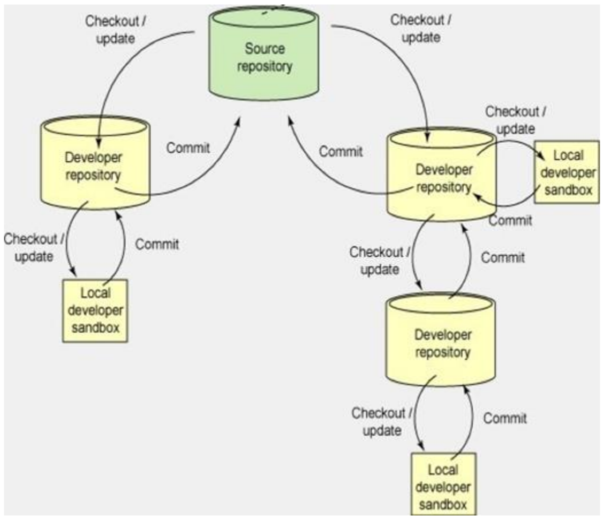

Figure. Working with Decentralized VCS.

The whole approach of central VCS is reversed, and each collaborator will have the complete repository on his local machine i.e. the complete revision history, all the branches, tags, commit information is present on the local machine. We do not have a notion of a central server, but we can configure any repository to be a central repository to treat as a source of truth and to integrate with build and deployment tools like Jenkins, Chef etc.
Below is the block diagram of how different collaborators work with a distributed version control system.

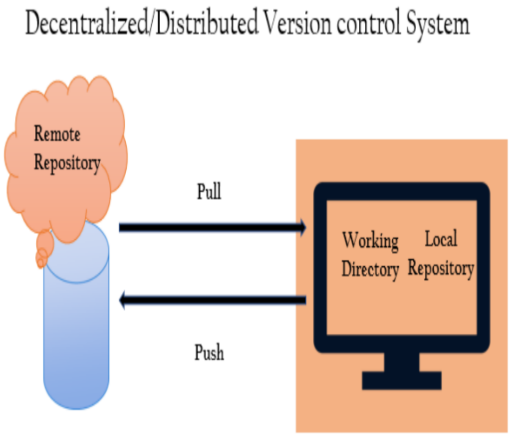

Figure. Decentralized VCS Architecture

From the above diagram, we can see that, unlike traditional VCSs, in distributed VCS, we have collaborators work with other collaborators in a decentralized system.

In distributed VCS, clients don't just check out the latest snapshot of the files; rather 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 clone is really a full backup of all the data. Also note that the terminology used in distributed VCS differs from centralized VCS. Where we use "checkout" and "commit" in central VCS; in distributed VCS, we use "push" and "pull".

1.Push: Send a change to another repository (may require permission)

2. Pull: Grab a change from a repository

There are many distributed version control systems like Git, Mercurial etc. In upcoming series of posts, I will be taking Git and will be referring to Git whenever referring to "distributed version control system".[8]

C) Example of Centralized VCS -Subversion Control System

Subversion is a free/open source version control system (VCS). That is, Subversion manages files and directories, and the changes made to them, over time. This allows you to recover older versions of your data or examine the history of how your data changed. In this regard, many people think of a version control systemas a sort of "time machine."

Subversion can operate across networks, which allows it to be used by people on different computers. At some level, the ability for various people to modify and manage the same set of data from their respective locations fosters collaboration. Progress can occur more quickly without a single conduit through which all modifications must occur. And because the work is versioned, you need not fear that quality is the trade-off for losing that conduit—if some incorrect change is made to the data, just undo that change.

Some version control systems are also software configuration management (SCM) systems. These systems are specifically tailored to manage trees of source code and have many features that are specific to software development—such as natively understanding programming languages or supplying tools for building software. Subversion, however, is not one of these systems. It is a general systemthat can be used to manage any collection of files. For you, those files might be source code—for others, anything from grocery shopping lists to digital video mixdowns and beyond.

On one end is a Subversion repository that holds all your versioned data. On the other end is your Subversion client program, which manages local reflections of portions of that versioned data. Between these extremes are multiple routes through a Repository Access (RA) layer, some of which go across computer networks and through network servers which then access the repository, others of which bypass the network altogether and access the repository directly.[9][7]

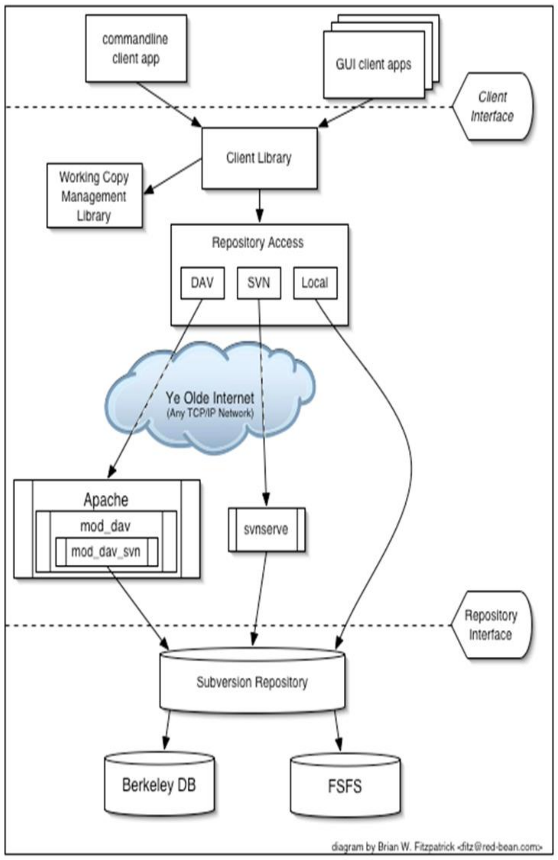

Figure. Subversion VCS Architecture.

Repository types

Subversion offers two types of repository storage.

1. Berkeley DB

The original development of Subversion used the Berkeley DB package. Subversion has some limitations with Berkeley DB usage when a program that accesses the database crashes or terminates forcibly. No data loss or corruption occurs, but the repository remains offline while Berkeley DB replays the journal and cleans up any outstanding locks. The safest way to use Subversion with a Berkeley DB repository involves a single server-process running as one user (instead of through a shared filesystem).

2. FSFS

In 2004, a new storage subsystem was developed and named FSFS. It works faster than the Berkeley DB backend on directories with a large number of files and takes less disk space, due to less logging. Beginning with Subversion 1.2, FSFS became the default data store for new repositories. The etymology of "FSFS" is based on Subversion's use of the term "filesystem" for its repository storage system. FSFS stores its contents directly within the operating system's filesystem, rather than a structured system like Berkeley DB. Thus, it is a "[Subversion] Filesystem atop the Filesystem".[10]

D) Example of Decentralized VCS -Git version Control System

Git is a distributed version-control system for tracking changes in source code during software development. It is designed for coordinating work among programmers, but it can be used to track changes in any set of files. Its goals include speed, data integrity, and support for distributed, non-linear workflows.

Git was created by Linus Torvalds in 2005 for development of the Linux kernel, with other kernel developers contributing to its initial development. Its current maintainer since 2005 is Junio Hamano. As with most other distributed version-control systems, and unlike most client–server systems, every Git directory on every computer is a full-fledged repository with complete history and full version-tracking abilities, independent of network access or a central server. Git is free and open-source software distributed under the terms of the GNU General Public License version 2[11]

1. Role of Git in Development.

Now that you know what Git is, you should know Git is an integral part of Development. Development is the practice of bringing agility to the process of development and operations. It’s an entirely new ideology that has swept IT organizations worldwide, boosting project lifecycles and in turn increasing profits. Development promotes communication between development engineers and operations, participating together in the entire service lifecycle, from design through the development process to production support.

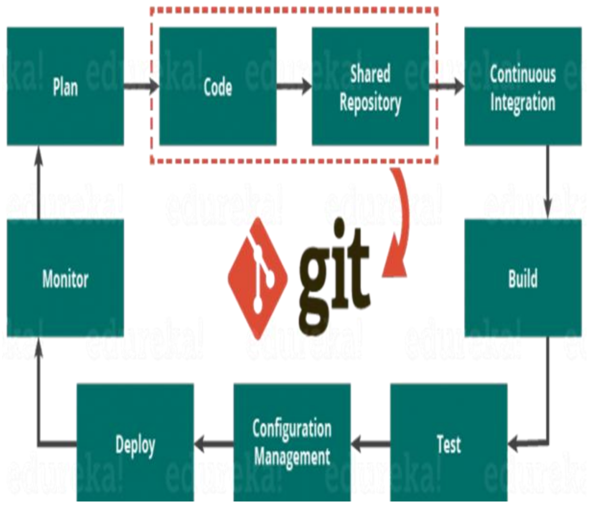

Figure. The diagram shows development life cycle and display how git fits in development.

The diagram above shows the entire life cycle of Developments starting from planning the project to its deployment and monitoring. Git plays a vital role when it comes to managing the code that the collaborators contribute to the shared repository. This code is then extracted for performing continuous integration to create a build and test it on the test server and eventually deploy it on the production.

Tools like Git enable communication between the development and the operations team. When you are developing a large project with a huge number of collaborators, it is very important to have communication between the collaborators while making changes in the project. Commit messages in Git play a very important role in communicating among the team. The bits and pieces that we all deploy lies in the Version Control system like Git. To succeed in Development, you need to have all of the communication in Version Control. Hence, Git plays a vital role in succeeding at Development. [12]

IV. ADVANTAGS AND DISADVANTAGES OF VCS

A) Advantages of Centralized VCS.

1. All the source code is safely stored in a secure place on a centralized server.

2. If the source code on the local machine is lost due to system or hard disk crashes, taking the code from the VCS can restore the source code.

3. Authentication and authorization can be put in place on the VCS.

4. The VCS takes care of managing the versioning and will not allow for commit if there is any conflict.

5. Maintains the commit log for commit information by developers.[1][14]

B) Advantages of Distributed VCS.

Fast

Each collaborator checks out the codebase into his local repository and work on the local repository. Hence all the operations will be fast since there won't be any network call to any server.

Cheap branching and merging

Since, the codebase is on the local hard disk, creating branches and merging is very simple and easy. This is one of the powerful features since working with branches and merging is too complicated if working with centralized repository.

Local branching

The developer can create as many local branches and work on the branches and then merge it back to the main branch. Once the merging is complete, the local branch can be safely deleted. The biggest advantage here is that the branch will not be visible to others unlike the centralized VCS where all the branches resides on one single server and creates lot of confusions when working on a large project.

Snapshots instead of difference

This is one of the key benefits. We can get the complete code repository for each commit that we have perform. Hence, we can easily revert back to any commit without having to deal with applying the changes from the base version manually as in case of central VCS.

Simple and productive tool

Once the developers are comfortable understanding the core concepts and features, developers will be more productive. Developers can also commit the code in modular fashion and collaborate with other developers without impacting other developer's workspace.

Scalable

"Distributed VCS" are highly scalable when compared with "Centralized VCS", especially in open source project where millions of developers contribute and a task which cannot be accomplished by a traditional version control system.

Open source

Git is open source and free. Moreover, developers can work on open source projects on various platforms like GitHub.[8]

C) Disadvantages of Centralized VCS.

- If the main servers are goes down developers can't save versioned changes. This is common scenario that we encounter. Almost all the operation that we performed like checking the file diff, committing, merging etc. all can be performed only VCS is up. For some reasons if the VCS is temporarily down for example network outages, server maintenance, all users accessing the VCS is blocked. Even simple operation like comparing the files with previous versions cannot be done.

- Remote commits are slow.

- Unsolicited changes might ruin development.

- If the central database is corrupted, the entire history could be lost (security issues).

Assumes that there are the 5 developers working with the VCS and they all work on features and commit their change to the repository. Unfortunately, the VCS crash and there is no backup. We are left with restoring the VCS with last stable commit. Now since the revision history and commit log is maintained on the VCS server, and codebase present on the developer's machine in just plain work copy, there is no way we can confidently bring the VCS to the last committed state. Git is open source and free. Moreover, developers can work on open source projects on various platforms like GitHub.

D) Disadvantages of Distributed VCS.

- It may not always be obvious who did the most recent change
- File locking doesn't allow different developers to work on the same piece of code simultaneously. It helps to avoid merge conflicts, but slows down development
- DVCS enables you to clone the repository – this could mean a security issue
- Managing non-mergeable files is contrary to the DVCS concept
- Working with a lot of binary files requires a huge amount of space, and developers can't do diffs
- Learning Git will take some steep learning curve and often confusion and frustrating for developers coming from other traditional VCS tools. But once the core concepts are understood it's a charm to work with.
- Although a lot of tools and plugins are available to be integrated with IDE's not many of them are present as in case of SVN or perforce. Although the git bash of one of the popular tools which can be leveraged if coming from Unix background.

V. APPLICATIONS OF VCS

A. Software Development

The primary use of version control systems has generally been in software development, where the ability to revisit older instances of a file may be useful, if not absolutely essential. There are many circumstances in which having some sort of file versioning is imperative. For example, when a bug appears in a piece of software, the first step to tracking down the bug is to know what changed in the source code since the last working version. If necessary, the changes made can be "rolled back", essentially removing the changes that introduced the error.

If a given piece of source code may be edited by multiple people, version control systems may assist here as well to reduce the number of conversations developers need to have with each other over code management--leaving more time to discuss design, documentation and testing. Some version control systems will lock files so that only one person may edit it at one time. When done, the user releases the lock and the next person may edit the file. Other systems allow multiple developers to work on a given file and when ready merge the changes in a meaningful way.

For software projects in which the development team is scattered over a large area--be it a city, a country, or the globe--managing these changes and meaningful information describing them requires a modern version control system.

B. Documentation

Many documents are "living documents". Since they describe things that change, they must themselves change, or become useless. The ability to revisit historical versions of a document may be used to satisfy curiosity or keep up with those changes or document changes in the subject itself.

If a document describes something relatively complex, the process of writing and editing it may be an interactive one, as with software. As well, multiple people may have multiple roles in the document's development, and the principle of version control may be applied to the document in the same way as they are for larger pieces of software. An example is The Linux Documentation Project, which is a global effort to provide documentation for the Linux operating system.

C. Web Development

Web development is a field that combines both software development and documentation, and many content management systems incorporate version control mechanisms. If yours does not, or if you don't use a content management system, consider using version control.

D. Systems Management

In a large computing environment, typically there are many similar machines whose configurations are modifications to a common base. For example, there might be a several machines with the same operating system, similar start-up and shutdown scripts, and common network configurations, underneath their applications and services. It may be useful to manage their configurations in a central, version-controlled repository, where variations are easily tracked, and common elements are easily replicated. If a change is made to the configuration that affects all machines, it becomes a relatively simple affair to visit each machine and update its configuration with the new one.

E. Home Directory Management

Users with accounts on multiple computer systems may find that they spend more time than they would like tweaking various aspects of their new computing environment, and it would be nice to be able to easily distribute this environment and update it when necessary.

It would be possible to use scp (Secure copy protocol) to copy files securely between machines, but this is less than convenient: every time a change is made, it needs to be manually copied from the source to the target, and of course there is no revision control in case of mistakes or for reference. With Subversion, it is simple when logging onto a different machine to quickly check for updates via a call to svn up. This could be scripted as part of your login script, and if your password is cached, it may even be automatic. [6]

VI. CONCLUSION

This study set out to investigate to what extent the type of VCS affects the choice of CM strategy at software development companies, and what VCS features are desirable for software development companies. The study shows that our initial proposition about the type of VCS having a big impact on the process was turned on its head. Our results indicate that it is the tool that has the biggest effect on the process.

Whether your development team uses Distributed VCS (Git) or Centralised VCS (SVN), you'll benefit from being able to track and review your code for better releases. Just be sure to choose an issue tracking software that supports your choice, so you're able to properly track that work over time.
If you're in the market for issue tracking software, Backlog integrates fully with Git and SVN, providing your team the ability to set up private repositories, propose and compare code changes, leave in-line comments on code, and document your work with wikis.

REFERENCES

[1] Pravin Kumar Sutar., "*Git Workflow, From Development to Deployment Redefined*", International journal of advanced research in computer science and application ISSN 2321-872x ONLINE ISSN 2321-8932.

[2] Emil Tsanov, Asen Bozhikov, *"Version Control in the Cloud"* IEEE Transactions on Software Engineering, SE-1 No.4,4 December 1975.

[3] Candrlic, M. Pavlic, and P. Poscic. "A comparison and the desirable features of version control tools. In Information Technology Interfaces", 2007. ITI 2007. 29th International Conference on, pages 121–126. IEEE, 2007

[4] Brent Laster: - Professional Git 1st Edition, ISBN-13:978-1119284970

[5] https://en.wikipedia.org/wiki/ Version_control

[6] https://homes.cs.washington.edu/~mernst/ advice/version-control.html

[7] https://howtodoinjava.com/vcs/how-version-control-system-vcs-works/

[8] https://howtodoinjava.com/vcs/how-distributed-version-control-system-works/

[9] http://svnbook.red-bean.com/en/1.6/svn-book.html#svn.intro.whatis

[10] https://www.projecthut.com/what-is-svn-repository/

[11] https://www.atlassian.com/git/tutorials/learn-about-code-review-in-bitbucket-cloud

[12] https://www.tutorialspoint.com/git/ git_basic_concepts.htm

[13] https://www.vogella.com/tutorials/Git/ article.html

[14] https://backlog.com/blog/git-vs-svn-version-control-system/