PalArch's Journal of Archaeology of Egypt / Egyptology

# A Novel Approach for Pre-Validation, Auto Resiliency & Alert Notification for SVN To Git Migration Using Iot Devices

**Vinay Singh,[1] Alok Aggarwal,[2] Narendra Kumar[3] A. K. Saini**

[1] 1531, Johnson Dr Apt, #732, Buffalo Grove, Illinois 60089, USA, *vsbuild7@gmail.com*
[2] School of Computer Science, University of Petroleum & Energy Studies, Dehradun, India, *alok.aggarwal@ddn.upes.ac.in*
[3] Icfai University, Jaipur, India, *drnk.cse@gmail.com*
[3] Icfai University, Jaipur, India, *aksaini@iujaipur.edu.in*

**Vinay Singh,[1] Alok Aggarwal,[2] Narendra Kumar[3] A. K. Saini: A novel approach for Pre-validation, Auto resiliency & Alert Notification for SVN to Git Migration using IoT devices -- Palarch's Journal of Archaeology Of Egypt/Egyptology 17(9). ISSN 1567-214x.**

**Keywords - Subversion (SVN), Git, Version control system, Trunk, Tag, Translational settings, Author Mapping, IoT, Node MCU, ESP8266 Wi-Fi SoC.**

## Abstract

Software development is getting a transition from centralized version control systems like SVN to decentralized version control systems like Git due to lesser efficiency of former in terms of branching, fusion, time, space, merging, offline commits & builds and repository etc. None of the available SVN-Git migration approaches has following four capabilities; identification and validation of a complete development project structure, SVN & Git SSH connectivity validation, SVN users & Git author mapping validation and remote Git Server space validation. It results in an extensive longer time for migration from SVN to Git along with incomplete migration. In this work a holistic, proactive and novel approach has been proposed for pre-migration validation from SVN to Git using IoT devices which covers all these four major limitations of the available SVN to Git migration approaches. Many scripts have been developed and executed for pre-migration validation and migration preparation which overcomes the problem of incomplete migration. Ten sample software projects have been taken for analysis for SVN to Git migration and results show that with the proposed approach the time consumed during migration from SVN to Git came down to about six times.

**Keywords** - Subversion (SVN), Git, Version control system, Trunk, Tag, Translational settings, Author Mapping, IoT, Node MCU, ESP8266 Wi-Fi SoC

## 1. Introduction

There are various version control systems in use like Subversion (SVN), Mercurial, CVS, Git and many more. Out of these Git is most widely used version control system (VCS) [1]-[4]. Git has a variety of advance features like distributed nature, offline commits facility, fast processing, staging feature and many more. In recent few years, Git has been the first choice as version control of small to large IT corporate world due to its better features [5]-[8].

Git is having a share of 77% of total VCS followed by SVN with a share of 13.5% [9]. Majority of software industries are getting a migration from SVN to Git. Only few migration tools are available in software industry but these too lack in many features like lack of identifying the empty directories as pre-migration check, failover capabilities during migration due to network failure or disk space issue and detailed report generation as post migration steps. Issues observed in SVN to Git migration for about 50,000 software project migrated from 2013 to 2019 are shown in figure 1. It was observed that majority of these were never been successful in first step due to missing project structure validation, weak SSH connectivity issue or sometimes improper mapping between SVN user and Git author.

None of the available SVN-Git migration approaches has following four capabilities; identification and validation of a complete development project structure, SVN & Git SSH connectivity validation, SVN users & Git author mapping validation and remote Git Server space validation. It results in an extensive longer time for migration from SVN to Git along with incomplete migration. In this work a holistic, proactive and novel approach has been proposed using IoT devices for pre-migration validation from SVN to Git which covers all these four major limitations of the available SVN to Git migration approaches. Many scripts have been developed and executed for pre-migration validation and migration preparation which overcomes the problem of incomplete migration. Ten sample software projects have been taken for analysis for SVN to Git migration.

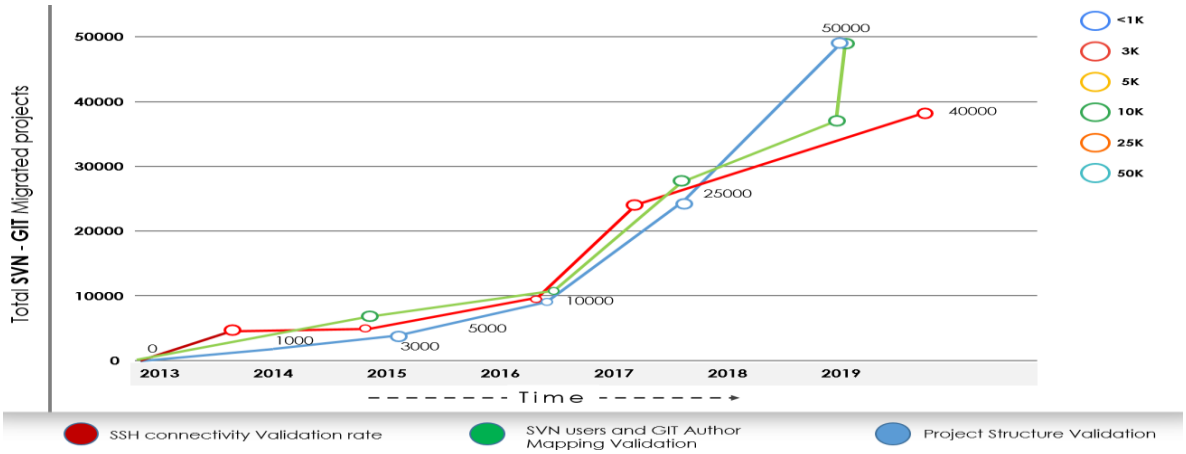


Figure 1. SVN-Git Migration rate and issues found over time [1]-[9]

Rest of the paper is organized is follows. Section 2 gives in brief the major work done by earlier researchers in the migration process of SVN to Git VCS. Proposed approach is given in detail in section 3 covering various steps of pre-validation methodology, pre-migration validation and migration preparation, SVN & Git SSH connectivity validation and space validation, and SVN users and Git author mapping validation & detailed report generation. Results of the proposed approach are given in section 4 along with conclusion.

## 2. Related Work

In SVN, many earlier studies focused on commit size distribution. Commit size can follow power laws [10]-[14], Pareto model [15], Petro-distribution [16] etc. Commits have been categorized based on various features mainly size and comment [17]-[19]. Dynamics of commit behavior with respect to open source software developer's have been investigated for SVN in [20]. Four open source software projects on Apache.org have been considered for the above investigation. An empirical study on inter-commit times in Subversion on two projects written in Java has been performed in [10]. It is observed that in both POI and Tomcat distribution of commit intervals follows power laws. Further, two major factors that cause very long period of inactivity are active committer's individual behavior of long vacations.

For a centralized VCS, a commit signing mechanism has been proposed in [21]. Proposed mechanism support various features like it allows working over a subset of repositories and working on disjoint set of files. Collaborative vocabulary development has been focused in many earlier works [22]-[26]. Git's applicability for collaborative vocabulary development has been investigated in [22]. Git4Voc has been proposed which explores the adoptability of Git to vocabulary development. It is shown that by using vocabulary-specific features, implementation of Git hooks can go beyond the plain functionality of Git. LHCb migration from SVN to Git has been presented in [27]. Issues related to specific requirements of LHCb have been addressed. Technical details of migration of large non-standard SVN repositories have also been addressed. It has been claimed that this migration from SVN to Git has resulted in increased productivity in terms of new projects & number of contribution and code quality in terms of testing and reviews. A mechanism for classification and extraction of changes is proposed in [28]. Various challenges and confusions related to Git have been reported in [29] followed by recommendations for dealing the same. General concepts of centralized and distributed VCS have been discussed in [30] and how these concepts are implemented by SVN and Git VCS.

## 3. Proposed Approach

None of the available SVN-Git migration tools has following four capabilities; identification and validation of a complete development project structure, SVN & Git

SSH connectivity validation, SVN users & Git author mapping validation and remote Git Server space validation. As a part of project structure SVN supports empty repositories which may be essentially required as part of software development, but Git does not like lib (libraries), dist. and target folders. In case of Java, .Net or any project structure having a source code repository with few empty folder structures then these empty folders will never be migrated to Git. These empty directories might have essential part of the project development structure which are needed to download the maven-based dependencies at compile or run time. For this purpose in this work many shell scripts have been developed for identifying all directories/subdirectories which are empty and need a place holder for complete SVN-Git migration and generates an email and SMS to project administrator about the total numbers of repositories and branches that are auto recovered and rectified.

SSH communication need to be established to pre-validate the connection between SVN & Git and also to confirm the same to the root user. For this purpose in this work NodeMCU is used. For migration of revision histories as meta-data, SVN authors should be mapped with Git authors before migration take place. Proposed work gives a mechanism by which SVN authors are mapped with Git authors before migration. By this way revision histories are not lost. Migration usually requires terabytes of disk space. Proposed work measures the total space acquired by SVN projects including various repositories, revisions, software code, SVN logs and meta-data. It makes it possible to login to remote Git server with same user and identifies the sufficient space availability else it generates an alert and email notification to the project administrator to take necessary action well in advance. Holistic view of SVN-Git migration is shown in figure 2.

*Pre-requisites:*

Pre-requisites for pre-migration validation and migration preparation are as follows.

- Install Oracle JRE 1.8 or newer (LTR release)
- Install SVN Mirror add-on migration tool in Git BitBucket
- Sufficient space should be available on the BitBucket server as per the size of SVN repositories
- SVN repo URLs must be accessible from the BitBucket server
- Install NodeMCU compatible with ESP8266 Wi-Fi SoC

*Dependencies:*

Dependencies for pre-migration validation and migration preparation are as follows.

- License procurement for SVN Mirror add-on migration tool from SubGit organization
- Count of SVN users to be migrated to Git

*Steps of Pre-validation Methodology:*

**Step 1:** Environment preparation

**:** JDK installation

**:** SubGit Tool Installation and configuration settings

**:** SSH connection setup between SVN & Git

: NodeMCU ESP8266 Wi-Fi SoC installation

**Step 2:** Project structure mapping validation and correction

**Step 3:** Branch level mapping validation and auto-correction

**Step 4:** Translation settings validation & auto-correction

**Step 5:** Adjust author level mapping validation & auto-correction

**Step 6:** SSH connection validation & alert generation using NodeMCU (ESP8266 Wi-Fi SoC)

**Step 7:** Exit with email notification

**Step 8:** Git server space validation using NodeMCU (ESP8266 Wi-Fi SoC)

**Step 9:** Alert/Notification generation

**Step 10:** Exit with email notification

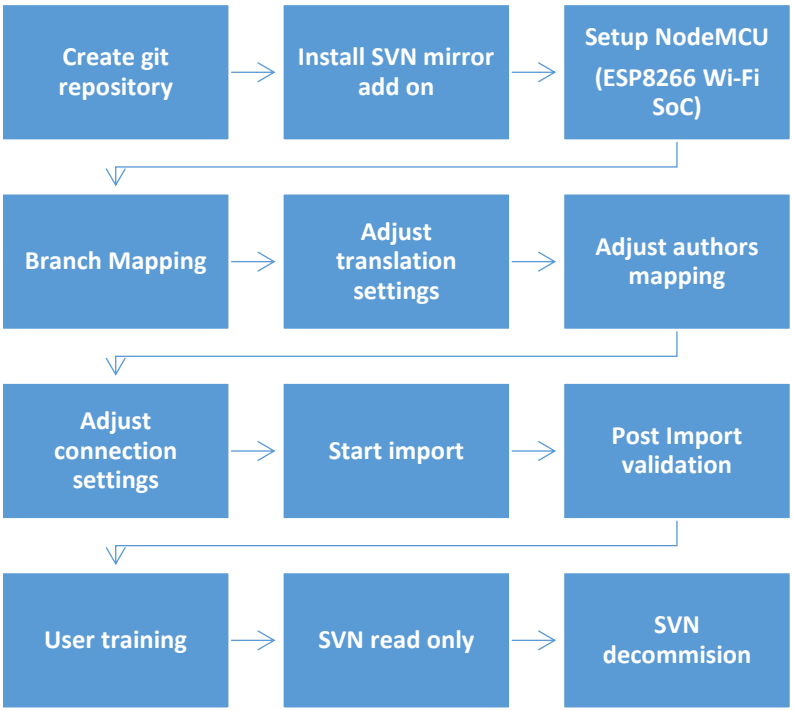


Figure 2. A holistic view of SVN-Git migration

### 3.1 Pre-migration validation and migration preparation

The project structure need to be validated in SVN to identify if there is any empty directory in all repositories. Next SVN repositories need to be validated for empty project structures. It is also required to check and calculate the total revision histories in SVN & total size of SVN and compare the space available in Git server. The admin user must have SSH connectivity to transfer the data from SVN to Git server. For this all repositories in SVN need to be checked out for finding the empty folders. Methodology used in this work for SVN to Git migration is shown in figure 3.

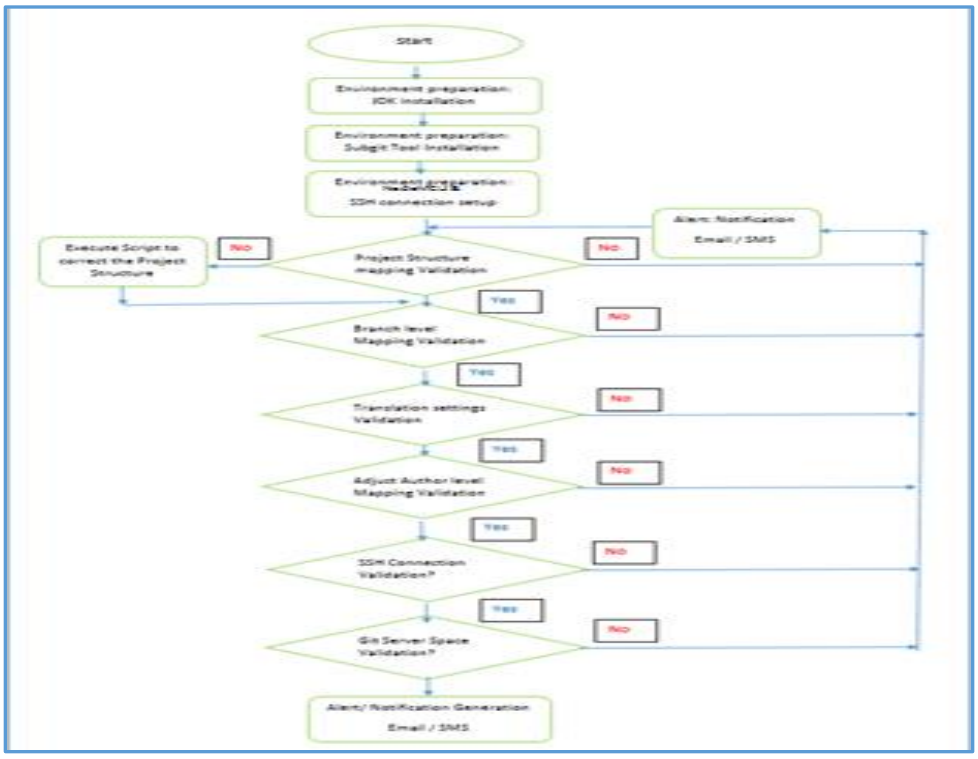


Figure 3. Methodology of SVN to Git migration

**Algorithm:** Project structure identification and correction in SVN project

**for** every repo in *ssh user@host* **do**
List Down *all repos in /var/svn-repos directory*
Goto the *Working* directory
Add all repositories and Branches into url.txt
Repeat above Step for all repositories
**Done**
**for** url in *urls.txt* **do**
Check out ***https : //domain/svn/branches/Master repo***
Goto the *Working* directory
*Search* empty *directories* in */.svn/*
Create *.gitkeep* in every empty sub-directories
*Commit* SVN into empty directory
**Done**

**for** every repo in *ssh user@host* **do**
Check out ***https: //domain/svn/branches/Develop repo***
Goto the *Working* directory
*Search* empty *directories* in */.svn/*
Create *.gitkeep* in every empty sub-directories
*Commit* SVN into empty directory
**Done**
SMS Alert user about SVN Project Structure

***Results:***

An email notification shall be sent by the Script and also results in command line.

*Installation steps for SVN Mirror add on:*

Firstly a fresh Git project need to be created in Git Bitbucket. Once the project is created then it need to be identified if there is any empty repository. SVN-Mirror add on tool is to be installed to migrate the project. For this purpose following command is executed.

*subgit.bat configure --layout auto --trunk TRUNK SVN_URL GIT_REPO*

Figure 4 shows the results of SVN Mirror (SubGit) tool installation and configuration.

```
> subgit.bat configure --layout auto --trunk trunk http://example.com/svn/repository/project C:\repo.git

    SubGit version 3.2.4 ('Bobique') build #3670

    Configuring writable Git mirror of remote Subversion repository:
        Subversion repository URL : http://example.com/svn/repository/project
        Git repository location   : C:\repo.git

    Detecting peg location...
    Authentication realm: <http://example.com:80> Subversion Repository
    Username [user]: user
    Password for 'user':
    Peg location detected: r10248 project/trunk
    Fetching SVN history... Done.
    Growing trees... Done.
    Project origin detected: r1 project/trunk
    Building branches layouts... Done.
    Combing beards... Done.
    Generating SVN to Git mapping... Done.

    CONFIGURATION SUCCESSFUL

    To complete SubGit installation do the following:

    1) Adjust Subversion to Git branches mapping if necessary:
         C:\repo.git\subgit\config
    2) Define at least one Subversion credentials in default SubGit passwd file at:
         C:\repo.git\subgit\passwd
        OR configure SSH or SSL credentials in the [auth] section of:
         C:\repo.git\subgit\config
    3) Optionally, add custom authors mapping to the authors.txt file(s) at:
         C:\repo.git\subgit\authors.txt
    4) Run SubGit 'install' command:
         subgit install "C:\repo.git"
```

Figure 4. Results of SVN Mirror (SubGit) tool installation and configuration

*Configure SVN Mirror settings:*
Now mirror Git repository need to be configured. For this purpose following command is executed to configure Git repository to mirror SVN project:

*$ subgit configure --layout auto --trunk trunk SVN_PROJECT_URL repos.git*

Above command detects branches layout in the SVN project and then creates empty Git repository ready to mirror SVN project.

*Author mappings, Adjust Translation & Initial Translation:*
Repositories names are to be provided to be used by Subgit tool to copy from SVN. Finally, repositories shall be imported by executing import command. It will start initial translation and will import all repositories from SVN to Git.

edit repos.git/subgit/config
$ edit repos.git/subgit/authors.txt
$subgit install repos.git
$ subgit import repos.git

### 3.2 SVN & Git SSH connectivity validation and space validation

During migration process if Git count is not increasing then most probably it is due to network failure. In this situation during migration, migration process should get a restart from the last commit point. For this purpose a script is developed in Python for making a check if Git count is not increasing and to restart the migration process in case of failure. System diagram between SVN & Git using NodeMCU (ESP8266 Wi-Fi SoC) is shown in figure 5. Flow chart for checking the SSH & Network connectivity using NodeMCU (ESP8266 Wi-Fi SoC) is shown in figure 6.

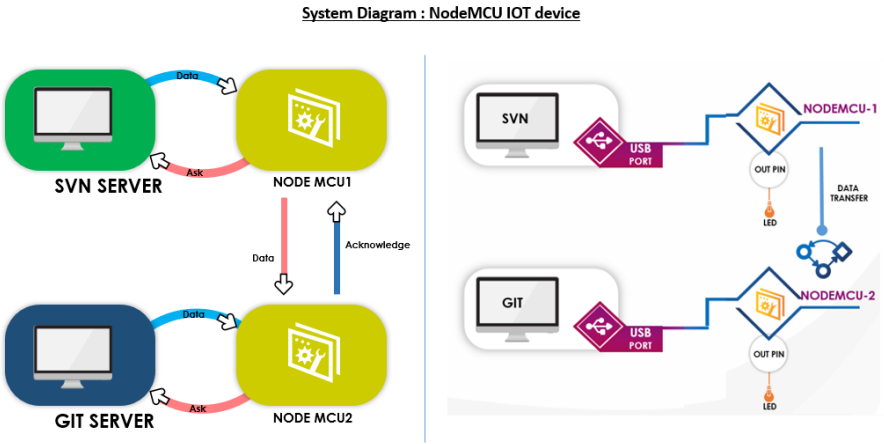


Figure 5. System Diagram between SVN & Git using NodeMCU (ESP8266 Wi-Fi SoC)

### 3.3 SVN users and Git author mapping validation & detailed report generation

One of the biggest problem in SVN-Git migration is the missing author information. A proper and stable mapping from the SVN users to the Git authors is needed for a better mapping of Git author data. In SVN, the author is being stored as an un-versioned revision property, namely svn:author. Every time a SVN user makes a commit, SVN creates a new revision and sets this revision svn:author property to be equal to that exact user's name whereas Git, in turn, also stores author's name along with every commit. But this name differs from that in SVN. SVN stores actual username but Git stores a name that is set by user.name Git directive, e.g., by global setting.

Hence at the time of migration start phase the SVN user are not properly mapped which results in an incomplete migration resulting in waste of time, space and resources. For addressing this problem few algorithms have been written and executed in SVN-Git migration discussed below. Algorithm for translation settings is shown in figure 7 and figure 8 shows the script for checking the authors mappings of SVN & Git. Algorithm for comparing the Git users with SVN users is given in figure 9 and final detailed report generated on migration is shown in figure 10.

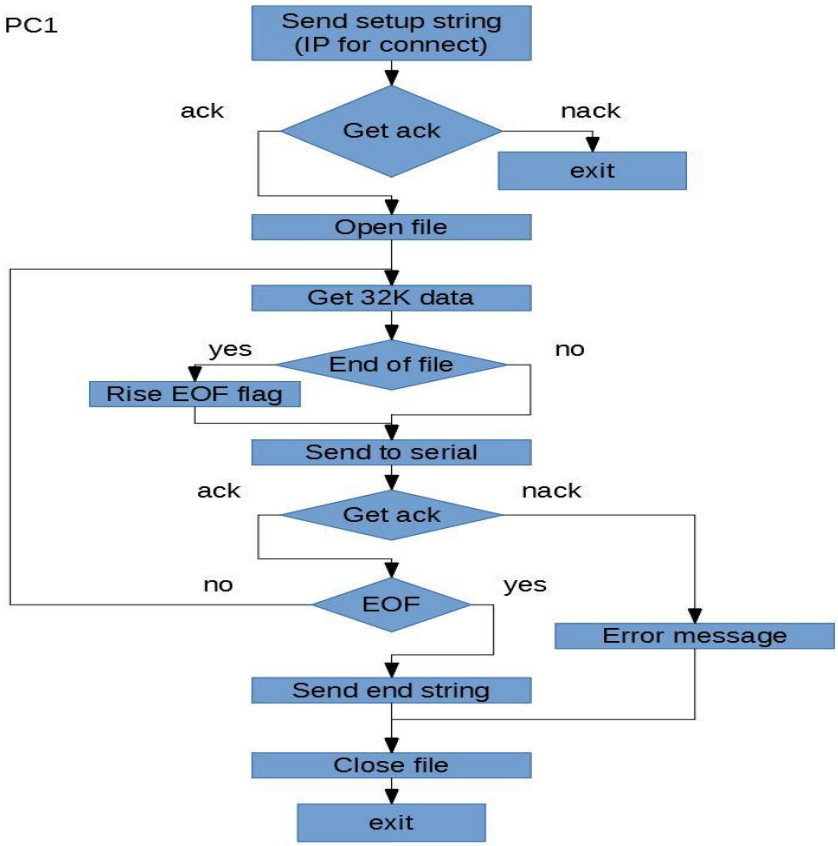


Figure 6. Flow chart for checking the SSH & Network connectivity using NodeMCU (ESP8266 Wi-Fi SoC)

**Algorithm:** For pre migration validation and the generation of a detailed report of pre-migration validation

*gitdir* ← Input a git directory.
*gitdeveloperdir* ← Input a developer directory.
*svnMasterdir* ← Input a svn master directory.
*svndeveloperdir* ← Input a svn developer directory.
*svnRepodir* ← Input a svn Repo directory.
Change directory to *gitdir* .
*PRINT* total count of author.
Change directory to *svnMasterdir* .
***PRINT*** total count of author in SVN.
***PRINT*** all author's name in SVN.
Change directory to *gitdir* .
Get into *gitdir* and *svnmasterdir* get all Author.
Get first and last commits in that git by Date.
Get first and last commits in that svn by Date.
Create branch using above commits.
*From Addr* ← *SenderJsEmailAddress*
*toAddr* ← *ReceiverJsEmailAddress*
*Sub* ← *args.var*[*1*] && *TEXT* ←
a*rgs.var[2]*
*msg* ← *Sub + TEXT*
***Username*** ← $UserJsEmailAddress$
***Password*** ← $UserJsPassward$
Create a *SMTP* Client server for gmail.
Start a secure connection using SSL/TLS cryptographic protocol.
Login to server using *Username* and *Password*
Input *fromAddr* and *toAddr* to the server.
Using *SMTP.sendmail* input *msg* to send the mail.
*Quit* the server after sending the mail using *server.quit*().
*svnRepoURL* ← Input a svn Repo URL.
*gitRepoURL* ← Input a git Repo URL.
*SVNAuthors* ← Get all Author's name from SVN Repo.
*GITAuthors* ← Get all Author's name from GIT Repo.
**if** *SVNAuthors* is available **then**:
    Remove *SVNAuthor* using $rm\ SVNAuthor$
**if** *GITAuthors* is available **then**:
    Remove *GITAuthor* using $rm\ GITAuthor$
Change directory to *svnRepoURL*
add *SVNAuthor* to *SVNAuthors* file.
*PRINT* all SVN author's names are added to file.
add *GITAuthor* to *GITAuthors* file.
*PRINT* all GIT author's names are added to file.

*compValue* ← Compare *SVNAuthors* and *GITAuthors* file using *diff −q SVNAuthors GITAuthors*.

**if** *compValue* equals to 1 **then**:

*PRINT SVNAuthors* and *GITAuthors* names do not match

Send Email to *abc@gmail.com* using *mailX*

**else**:

*PRINT SVNAuthors* and *GITAuthors* names matches.

Send Email to *abc@gmail.com* using *mailX*

```
1.  Begin
2.      Define :gitdir
3.      Define :git_developer_dir
4.      Define svn_master_dir
5.      Define svn_developer_dir
6.      Define svnrepo: Path
7.      Set :gitdir
8.      Set :git_developer_dir
9.      Set :svn_master_dir
10.     Set :svn_developer_dir
11.     Set :svnrepo: Path
12.     Change Directory :gitdir
13.     Print : total Authors in git
14.     Change Directory :svn_master_dir
15.     Print : total count of Authors in SVN
16.     Change Directory : gitdir
17.     Print : total count of  Authors in Git
18.     Change Directory :svn_master_dir
19.     Print :The name of all  Authors in SVN
20.     Change Directory : gitdir
21.     Print :The name of all  Authors in Git
22.     Print :The First commit in Git,SVN
23.     Print :The First commit  date in Git,SVN
24.     Print :The last commit in Git,SVN
25.     Print :The last  commit date in Git,SVN
26.     End
```

Figure 7. Algorithm for Translation Settings

```
#This script would do following
#    Get project name as an argument and navigate to respective SVN and GIT locations
#    Get list of authors.
#    Compare list of authors and send an email.

#!/bin/bash

svn_repo_url = "/Users/shashibisht/svn_tutorial/$1"
git_repo_url = "/Users/shashibisht/git_tutorial/$1"
echo "Removing files from previous runs." > logfile
if [ -f SvnAuthors ]
   then rm SvnAuthors
fi

if [ -f GitAuthors ]
   then rm GitAuthors
fi

cd svn_repo_url
svn log --quiet | grep "^r" | awk '{print $3}' | sort | uniq > SvnAuthors
echo "Added list of SVN authors to SvnAuthors file" > logfile
cd git_repo_url
git log --all --format='%aN' | sort -u > GitAuthors
echo "Added list of GIT authors to GitAuthors file" > logfile
diff -q SvnAuthors GitAuthors  >/dev/null
comp_value=$?

if [ $comp_value -eq 1 ]
then
    echo "SVN and GIT Authors list do not match" > logfile
    mailx -s "SVN Author list not same as GIT Author list" abc@gmail.com
else
    echo "SVN and GIT Authors list match!!" > logfile
    mailx -s "SVN Author list is same as GIT Author list!!!" abc@gmail.com
fi
```

Figure 8. Script for checking the Authors Mappings SVN & Git

```
1.  Begin
2.      Define svn_repo_url
3.      Define git_repo_url
4.      Set svn_repo_url
5.      Set git_repo_url
6.      If  svnAuthors File is present
7.           Delete svnAuthors File
8.      End the construct
9.      If GitAuthors File is present
10.         Delete GitAuthors File
11.     End the construct
12.     Print:  Removing files previous runs
13.     Change Directory svn_repo_url
14.     Find the unique Svn Authors
15.     Print:  Added list of svn authors to svnAuthors file >logfile
16.     Change Directory git_repo_url
17.     Find the unique Git Authors
18.      Print Added list of GIT authors to GitAuthors file
19.      If SVNAuthors Not Equal GitAuthors)
20.           Print SVN and GIT Authors lists do not match
21.           Email " Users Don't Match"
22.      Else
23.           Print SVN and GIT Authors list match
24.           Email " Users  Match"
25.     End the construct
26.  End
```

Figure 9. Algorithm for comparing the Git users with SVN users

```
vsinghxz@PTS-LDR-SINGV9 MINGW64 /c/svn_git_repoistories
$ ./git_svn.sh
************ Comprehensive Report of on SVN to Git Migration********************
*********Authors**********
 Total Authors in Git= 54
 Total Authors in SVN=  10
 All Authors name in Git=
Author: Chinmaya Sahoo <chinmaya.sahoo@walgreens.com>
Author: sali4f <syedazam.ali@walgreens.com>
Author: Vinay Singh <vinay.x.singh@walgreens.com>
 All Authors name in SVN=
 VisualSVN Server
Created folder 'develop'.
Created folder 'master'.
Deleted folder '/branches/master'.
Initial structure.
***************First and Last commits in Develop Branch***************
 First commit in Git=
commit e2d93442905afa1a808a8728ad99872cc1675293
 First commit date in Git=
Date:   Mon Apr 22 20:03:33 2019 +0000
 Last commit in Git=
e2d93442905afa1a808a8728ad99872cc1675293 Initial Commit
 Last commit date in Git=
Date:   Sat May 25 12:53:13 2019 -0500
 First commit in SVN=
r1 | VisualSVN Server | 2019-06-22 23:03:20 -0500 (Sat, 22 Jun 2019) | 1 line
 First commit date in SVN=
 2019-06-22 23:03:20 -0500 (Sat, 22 Jun 2019)
 Last commit in SVN=
------------------------------------------------------------------------
r5 | VisualSVN Server | 2019-06-22 23:06:41 -0500 (Sat, 22 Jun 2019) | 1 line
Changed paths:
   A /branches/master

Created folder 'master'.
------------------------------------------------------------------------
r6 | vsinghxz | 2019-06-22 23:07:54 -0500 (Sat, 22 Jun 2019) | 1 line
Changed paths:
   A /branches/develop/.classpath
   A /branches/develop/.project
   A /branches/develop/.settings
   A /branches/develop/.settings/org.eclipse.core.resources.prefs
   A /branches/develop/.settings/org.eclipse.jdt.core.prefs
   A /branches/develop/.settings/org.eclipse.m2e.core.prefs
   A /branches/develop/.settings/org.eclipse.wst.common.project.facet.core.xml
   A /branches/develop/.springBeans
   A /branches/develop/README.md
   A /branches/develop/pom.xml
   A /branches/develop/src
   A /branches/develop/src/main
   A /branches/develop/src/main/java
   A /branches/develop/src/main/java/poc
```

Figure 10. Final detailed report generated on migration

## 4. Results and Conclusion

Ten sample software projects have been taken for analysis for SVN to Git migration. Figure 11 shows the time consumed before and after missing project structure validation. Blue colored bar shows migration time of ten projects totaling 228 hours without applying the proposed approach. This longer migration time is due to Git space issue, SSH connectivity issue, missing project structure or other reasons. Green colored bar shows migration time of ten projects totaling 36 hours by applying the proposed approach. Proposed approach covers the issues related to Git space issue, SSH connectivity issue, missing project structure or other reasons. Results show that with the proposed approach time consumed during migration from SVN to Git came down to about six times based on the sample of ten projects.

None of the available SVN-Git migration approaches has following four capabilities; identification and validation of a complete development project structure, SVN & Git SSH connectivity validation, SVN users & Git author mapping validation and remote Git Server space validation. It results in an extensive longer time for migration from SVN to Git along with incomplete migration. Proposed work for pre-migration validation from SVN to Git covers all these four major limitations of the available SVN to Git migration approaches. Many scripts have been developed and executed for pre-migration validation and migration preparation which overcomes the problem of incomplete migration.

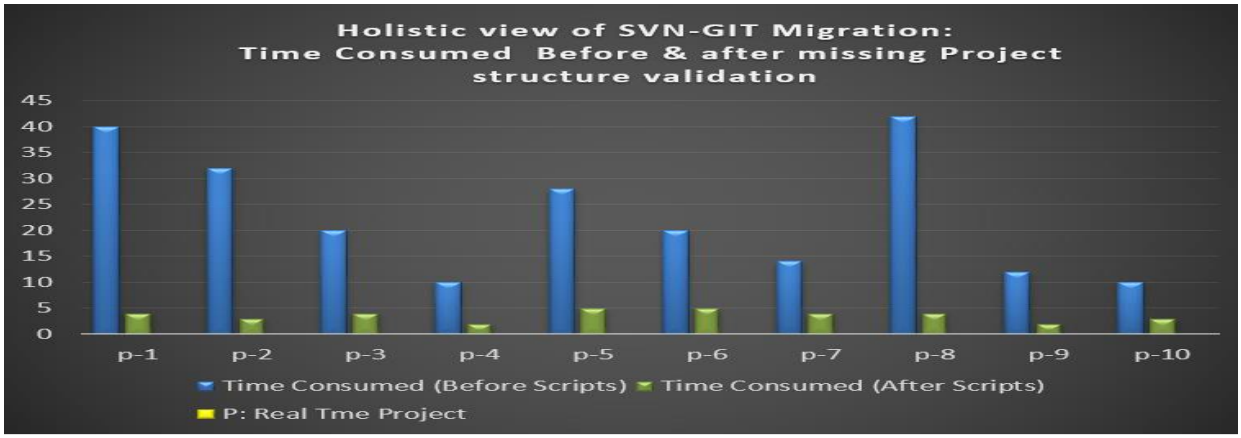


Figure 11. Time consumed before and after missing project structure validation over the sample of ten software projects

With all above mentioned experimentation done in the SRLC Software Research Lab, Chicago it could be stated that it is very much possible and feasible that any version control migration tool can be made more capable and dynamic by providing a holistic approach for the migration for all repositories from SVN to Git. With the help of automation and scripts it can be ensured that SVN repositories have proper structure compatible to the Git structure before actual migration starts and during migration there is proper notification sent to all stakeholders in the IT business.

**References:**

[1] https://www2.physics.ox.ac.uk/it-services/moving-projects-from-svn-to-git (accessed June 6, 2020)
[2] https://stosb.com/static/talks/case_study_git_efl_linuxcon_eu_13.pdf (accessed July 16, 2020)
[3] https://docs.microsoft.com/en-us/vsts/articles/perform-migration-from-svn-to-git?view=vsts (accessed July 9, 2020)
[4] http://iopscience.iop.org/article/10.1088/1742-6596/898/7/072024 (accessed June 7, 2020)
[5] https://www2.physics.ox.ac.uk/it-services/moving-projects-from-svn-to-git (accessed July 26, 2020)
[6] https://homes.cs.washington.edu/~mernst/advice/version-control.html (accessed June 8, 2020)
[7] https://uwaterloo.ca/web-resources/web-developers/developing-wcms/svn-git (accessed July 10, 2020)
[8] https://www.kiva.org/blog/how-we-moved-from-Subversion-to-git-github (accessed July 3, 2020)
[9] https://computing.llnl.gov/newsroom/great-migration-visit-moves-Subversion-github (accessed July 16, 2020)
[10] Q. Hou, Y. Ma, J. Chen, Y. Xu, An Empirical Study on Inter-Commit Times in SVN, Int. Conf. on Software Eng. and Knowledge Eng. (2014) 132-137.
[11] O. Arafat, D. Riehle, The Commit Size Distribution of Open Source Software, Proc. the 42nd Hawaii Int. Conf. Syst. Sci. (HICSS'09), USA. (2009) 1-8.
[12] C. Kolassa, D. Riehle, M. Salim, A Model of the Commit Size Distribution of Open Source, Proc. the 39th Int. Conf. Current Trends in Theory and Practice of Comput. Sci. (SOFSEM'13), Czech Republic. (2013) 52-66.
[13] L. Hattori, M. Lanza, On the nature of commits, Proc. 4th Int. ERCIM Wksp. Softw. Evol. and Evolvability (EVOL'08), Italy. (2008) 63-71.
[14] P. Hofmann, D. Riehle, Estimating Commit Sizes Efficiently, Proc. 5th IFIP WG 2.13 Int. Conf. Open Source Systems (OSS'09), Sweden. (2009) 105-115.
[15] C. Kolassa, D. Riehle, M. Salim, A Model of the Commit Size Distribution of Open Source, In: van Emde Boas P., Groen F.C.A., Italiano G.F., Nawrocki J., Sack H. (eds) SOFSEM 2013: Theory and Practice of Computer Science. SOFSEM 2013. Lecture Notes in Computer Science, Springer, Berlin, Heidelberg. 7741 (2013) 52-66, https://doi.org/10.1007/978-3-642-35843-2_6.
[16] O. Arafat, D. Riehle, The Commit Size Distribution of Open Source Software, Proc. 42nd Hawaii Inter. Conf. Systems Science (HICSS'09), IEEE Computer Society Press, New York, NY. (2009) 1-8.
[17] R. Purushothaman, D.E. Perry, Toward Understanding the Rhetoric of Small Source Code Changes, IEEE Transactions on Software Engineering. 31 (2005) 511-526.
[18] A. Alali, H. Kagdi, J. Maletic, What's a Typical Commit? A Characterization of Open Source Software Repositories, Proc. the 16th IEEE Int. Conf. Program Comprehension (ICPC'08), Netherlands. (2008) 182-191.

[19] A. Hindle, D. Germán, R. Holt, What do large commits tell us?: a taxonomical study of large commits, Proc. 5th Int. Working Conf. Mining Softw. Repos. (MSR'08), Germany. (2008) 99-108.

[20] Y. Ma, Y. Wu, Y. Xu, Dynamics of Open-Source Software Developer's Commit Behavior: An Empirical Investigation of Subversion, Proc. 29th Annual ACM Symposium on Applied Computing (SAC'14). (2014) 1171-1173, https://doi: 10.1145/2554850.2555079.

[21] S. Vaidya, S. Torres-Arias, R. Curtmola, J. Cappos, Commit Signatures for Centralized Version Control Systems, In Dhillon G., Karlsson F., Hedström K., Zúquete A. (eds) ICT Systems Security and Privacy Protection. SEC 2019, IFIP Advances in Information and Communication Technology, Springer, Cham. 562 (2019) 359-373, https://doi.org/10.1007/978-3-030-22312-0_25.

[22] Lavdim Halilaj, Irlán Grangel-González, Gökhan Coskun, Steffen Lohmann, Sören Auer, Git4Voc: Collaborative Vocabulary Development Based on Git, International Journal of Semantic Computing. 10 (2016) 167-191.

[23] J. P. Diane, I. Hillmann, Gordon Dunsire, Versioning vocabularies in a linked data world, IFLA Lion, 2014.

[24] M. Luczak-R¨osch, G. Coskun, A. Paschke, M. Rothe, R. Tolksdorf, Svont-version control of owl ontologies on the concept level. 176 (2010) 79-84.

[25] E. Jim´enez-Ruiz, B.C. Grau, I. Horrocks, R.B. Llavori, Contentcvs: A cvs-based collaborative ontology engineering tool, in SWAT4LS, Citeseer. (2009).

[26] I. Zaikin, A. Tuzovsky, Owl2vcs: Tools for distributed ontology development, in OWLED, Citeseer. (2013).

[27] M Clemencic, B Couturier, J Closier, M Cattaneo, LHCb migration from Subversion to Git, Journal of Physics: Conference Series, Track 5: Software Development. 898 (2017) 1-4.

[28] A. Kaur, D. Chopra, GCC-Git Change Classifier for Extraction and Classification of Changes in Software Systems, In: Hu YC., Tiwari S., Mishra K., Trivedi M. (eds) Intelligent Communication and Computational Technologies. Lecture Notes in Networks and Systems, Springer, Singapore. 19 (2018) 259-267.

[29] V. Isomöttönen, M. Cochez, Challenges and Confusions in Learning Version Control with Git, In: Ermolayev V., Mayr H., Nikitchenko M., Spivakovsky A., Zholtkevych G. (eds) Information and Communication Technologies in Education, Research, and Industrial Applications. ICTERI, Communications in Computer and Information Science, Springer, Cham. 469 (2014) 178-193.

[30] Stefan Otte, Version Control Systems, Open Access paper. (2009)1-12.