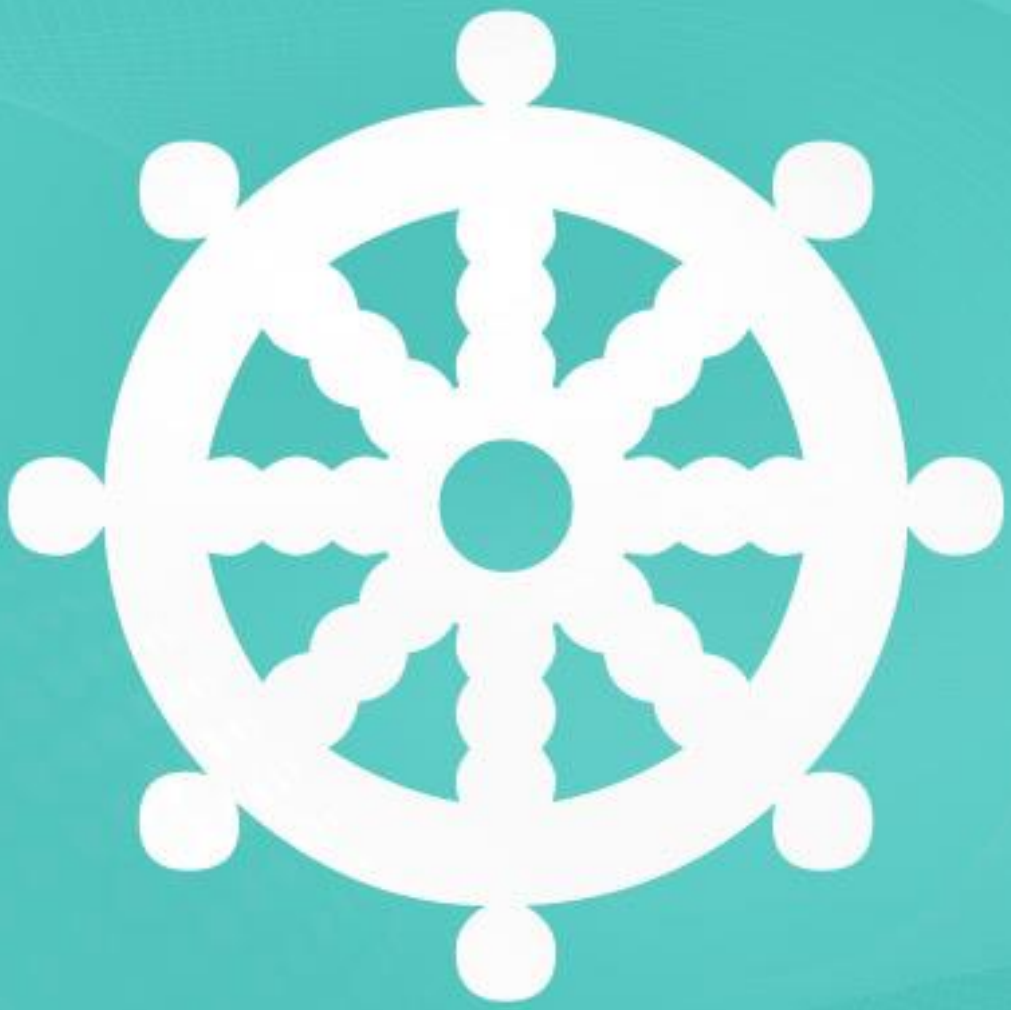

SVN

version control system

tutorialspoint

SIMPLYEASYLEARNING

www.tutorialspoint.com

https://www.facebook.com/tutorialspointindia

https://twitter.com/tutorialspoint

About the Tutorial

Apache Subversion which is often abbreviated as SVN, is a software versioning and revision control system distributed under an open source license. Subversion was created by CollabNet Inc. in 2000, but now it is developed as a project of the Apache Software Foundation, and as such is part of a rich community of developers and users.

This tutorial provides you an understanding on SVN system that is needed to maintain the current and historical versions of files such as source code, web pages, and documentations.

Audience

This tutorial is designed for software professionals interested in learning the concepts of SVN system in simple and easy steps. After completing this tutorial, you will be gain sufficient exposure to SVN from where you can take yourself to higher levels of expertise.

Prerequisites

Before proceeding with this tutorial, you should have a basic understanding on simple terminologies like programming language, source code, documents, etc. Because using SVN to handle all levels of software projects in your organization, it will be good if you have a working knowledge of software development and software testing processes.

Copyright & Disclaimer

Table of Contents

1. BASIC CONCEPTS

What is Version Control System?

Version Control System (VCS) is a software that helps software developers to work together and maintain a complete history of their work.

Following are the goals of a Version Control System.

- Allow developers to work simultaneously.
- Do not overwrite each other's changes.
- Maintain history of every version of everything.

A VCS is divided into two categories.

- Centralized Version Control System (CVCS), and
- Distributed/Decentralized Version Control System (DVCS).

In this tutorial, we will concentrate only on the Centralized Version Control System and especially **Subversion**. Subversion falls under centralized version control system, meaning that it uses central server to store all files and enables team collaboration.

Version Control Terminologies

Let us start by discussing some of the terms that we will be using in this tutorial.

- **Repository**: A repository is the heart of any version control system. It is the central place where developers store all their work. Repository not only stores files but also the history. Repository is accessed over a network, acting as a server and version control tool acting as a client. Clients can connect to the repository, and then they can store/retrieve their changes to/from repository. By storing changes, a client makes these changes available to other people and by retrieving changes, a client takes other people's changes as a working copy.
- **Trunk**: The trunk is a directory where all the main development happens and is usually checked out by developers to work on the project.
- **Tags**: The tags directory is used to store named snapshots of the project. Tag operation allows to give descriptive and memorable names to specific version in the repository.

  For example, LAST_STABLE_CODE_BEFORE_EMAIL_SUPPORT is more memorable than

Repository UUID: 7ceef8cb-3799-40dd-a067-c216ec2e5247 and

Revision: 13

- **Branches**: Branch operation is used to create another line of development. It is useful when you want your development process to fork off into two different directions. For example, when you release version 5.0, you might want to create a branch so that development of 6.0 features can be kept separate from 5.0 bug-fixes.
- **Working copy**: Working copy is a snapshot of the repository. The repository is shared by all the teams, but people do not modify it directly. Instead each developer checks out the working copy. The working copy is a private workplace where developers can do their work remaining isolated from the rest of the team.
- **Commit changes**: Commit is a process of storing changes from private workplace to central server. After commit, changes are made available to all the team. Other developers can retrieve these changes by updating their working copy. Commit is an atomic operation. Either the whole commit succeeds or it is rolled back. Users never see half finished commit.

2. ENVIRONMENT SETUP

SVN Installation

Subversion is a popular open-source version control tool. It is open-source and available for free over the internet. It comes by default with most of the GNU/Linux distributions, so it might be already installed on your system. To check whether it is installed or not use following command.

```
[jerry@CentOS ~]$ svn --version
```

If Subversion client is not installed, then command will report error, otherwise it will display the version of the installed software.

```
[jerry@CentOS ~]$ svn --version
-bash: svn: command not found
```

If you are using RPM-based GNU/Linux, then use ***yum*** command for installation. After successful installation, execute the ***svn --version*** command.

```
[jerry@CentOS ~]$ su -
Password:
[root@CentOS ~]# yum install subversion

[jerry@CentOS ~]$ svn --version
svn, version 1.6.11 (r934486)
compiled Jun 23 2012, 00:44:03
```

And if you are using Debian-based GNU/Linux, then use ***apt*** command for installation.

```
[jerry@Ubuntu]$ sudo apt-get update
[sudo] password for jerry:

[jerry@Ubuntu]$ sudo apt-get install subversion

[jerry@Ubuntu]$ svn --version
svn, version 1.7.5 (r1336830)
```

```
compiled Jun 21 2013, 22:11:49
```

Apache Setup

We have seen how to install Subversion client on GNU/Linux. Let us see how to create a new repository and allow access to the users.

On server we have to install **Apache httpd** module and **svnadmin** tool.

```
[jerry@CentOS ~]$ su -
Password:
[root@CentOS ~]# yum install mod_dav_svn subversion
```

The ***mod_dav_svn*** package allows access to a repository using HTTP, via Apache httpd server and ***subversion*** package installs svnadmin tool.

The subversion reads its configuration from ***/etc/httpd/conf.d/subversion.conf*** file. After adding configuration, ***subversion.conf***, file looks as follows:

```
LoadModule dav_svn_module     modules/mod_dav_svn.so
LoadModule authz_svn_module   modules/mod_authz_svn.so

<Location /svn>
   DAV svn
   SVNParentPath /var/www/svn
   AuthType Basic
   AuthName "Authorization Realm"
   AuthUserFile /etc/svn-users
   Require valid-user
</Location>
```

Let us create Subversion users and grant them access to the repository. *Htpasswd* command is used to create and update the plain-text files which are used to store *usernames* and *passwords* for basic authentication of HTTP users. '-c' options creates *password* file, if *password* file already exists, it is overwritten. That is why use '-c' option only the first time. '-m' option enables MD5 encryption for passwords.

User Setup

Let us create user ***tom***.

```
[root@CentOS ~]# htpasswd -cm /etc/svn-users tom
New password:
Re-type new password:
Adding password for user tom
```

Let us create user ***jerry***

```
[root@CentOS ~]# htpasswd -m /etc/svn-users jerry
New password:
Re-type new password:
Adding password for user jerry
[root@CentOS ~]#
```

Create Subversion parent directory to store all the work (see*/etc/httpd/conf.d/subversion.conf*).

```
[root@CentOS ~]# mkdir /var/www/svn
[root@CentOS ~]# cd /var/www/svn/
```

Repository Setup

Create a project repository named *project_repo*. *svnadmin* command will create a new repository and a few other directories inside that to store the metadata.

```
[root@CentOS svn]# svnadmin create project_repo

[root@CentOS svn]# ls -l project_repo
total 24
drwxr-xr-x. 2 root root 4096 Aug  4 22:30 conf
drwxr-sr-x. 6 root root 4096 Aug  4 22:30 db
-r--r--r--. 1 root root    2 Aug  4 22:30 format
drwxr-xr-x. 2 root root 4096 Aug  4 22:30 hooks
drwxr-xr-x. 2 root root 4096 Aug  4 22:30 locks
-rw-r--r--. 1 root root  229 Aug  4 22:30 README.txt
```

Let us change the user and group ownership of the repository.

```
[root@CentOS svn]# chown -R apache.apache project_repo/
```

Check whether *SELinux* is enabled or not using the SELinux status tool.

```
[root@CentOS svn]# sestatus
SELinux status:                 enabled
SELinuxfs mount:                /selinux
Current mode:                   enforcing
Mode from config file:          enforcing
Policy version:                 24
Policy from config file:        targeted
```

For our server, SELinux is enabled, so we have to change the SELinux security context.

```
[root@CentOS svn]# chcon -R -t httpd_sys_content_t
/var/www/svn/project_repo/
```

To allow commits over HTTP, execute the following command.

```
[root@CentOS svn]# chcon -R -t httpd_sys_rw_content_t
/var/www/svn/project_repo/
```

Restart the Apache server and we are done with the configuration of Apache server.

```
[root@CentOS svn]# service httpd restart
Stopping httpd:                                            [FAILED]
Starting httpd: httpd: apr_sockaddr_info_get() failed for CentOS
httpd: Could not reliably determine the server's fully qualified domain
name, using 127.0.0.1 for ServerName
                                                           [  OK  ]
[root@CentOS svn]# service httpd status
httpd (pid  1372) is running...
[root@CentOS svn]#
```

We have configured the Apache server successfully, now we will configure the repository. To provide repository access to only authentic users and to use the default authorization file; append the following lines to *project_repo/conf/svnserve.conf* file.

```
anon-access = none
authz-db = authz
```

Conventionally, every Subversion project has **trunk, tags,** and **branches** directories directly under the project's root directory.

The *trunk* is a directory where all the main development happens and is usually checked out by the developers to work on the project.

The *tags* directory is used to store named snapshots of the project. When creating a production release, the team will tag the code that goes into the release.

The *branches* directory is used when you want to pursue different lines of development.

Let us create the *trunk, tags,* and *branches* directory structure under the project repository.

```
[root@CentOS svn]# mkdir /tmp/svn-template
[root@CentOS svn]# mkdir /tmp/svn-template/trunk
[root@CentOS svn]# mkdir /tmp/svn-template/branches
[root@CentOS svn]# mkdir /tmp/svn-template/tags
```

Now import the directories from ***/tmp/svn-template*** to the repository.

```
[root@CentOS svn]# svn import -m 'Create trunk, branches, tags directory
structure' /tmp/svn-template/
Adding         /tmp/svn-template/trunk
Adding         /tmp/svn-template/branches
Adding         /tmp/svn-template/tags
Committed revision 1.
[root@CentOS svn]#
```

This is done now! We have successfully created the repository and allowed access to ***Tom*** and ***Jerry***. From now, they can perform all the supported operations to the repository.

3. LIFE CYCLE

The life cycle of a Version Control System is discussed in this chapter. In later chapters, we will see the Subversion command for each operation.

Create Repository

The repository is a central place where developers store all their work. Repository not only stores files, but also the history about changes, which means it maintains a history of the changes made in the files. The 'create' operation is used to create a new repository. Most of the times this operation is done only once. When you create a new repository, your VCS will expect you to say something to identify it, such as where you want it to be created, or what name should be given to the repository.

Checkout

'Checkout' operation is used to create a working copy from the repository. Working copy is a private workplace where developers do their changes, and later on, submit these changes to the repository.

Update

As the name suggests, 'update' operation is used to update working copy. This operation synchronizes the working copy with the repository. As repository is shared by all the teams, other developers can commit their changes and your working copy becomes older.

Let us suppose *Tom* and *Jerry* are the two developers working on a project. Both check out the latest version from the repository and start working. At this point, their working copies are completely synchronized with the repository. *Jerry* completes his work very efficiently and commits his changes to the repository.

Now *Tom's* working copy is out of date. Update operation will pull *Jerry's* latest changes from the repository and will update *Tom's* working copy.

Perform Changes

After the checkout, one can do various operations to perform changes. Edit is the most common operation. One can edit the existing file to add/remove contents from the file.

One can add files/directories. But immediately these files/directories do not become a part of the repository, instead they are added to the pending change-list and become a part of the repository after the commit operation.

Similarly one can delete files/directories. Delete operation immediately deletes file from the working copy, but actual deletion of the file is added to the pending change-list and changes are made to the repository after the commit operation.

'Rename' operation changes the name of the file/directory. 'Move' operation is used to move files/directories from one place to another in a repository tree.

Review Changes

When you check out the working copy or update the working copy, then your working copy is completely synchronized with the repository. But as you do changes to your working copy, it becomes newer than the repository. And it is a good practice to review your changes before the 'commit' operation.

'Status' operation lists the modifications that have been made to the working copy. As we have mentioned before, whenever you do changes in the working copy all these changes become a part of the pending change-list. And the 'status' operation is used to see the pending change-list.

'Status' operation only provides a list of changes but not the details about them. One can use *diff* operation to view the details of the modifications that have been made to the working copy.

Fix Mistakes

Let us suppose one has made changes to his working copy, but now, he wants to throw away these changes. In this situation, 'revert' operation will help.

Revert operation reverts the modifications that have been made to the working copy. It is possible to revert one or more files/directories. Also it is possible to revert the whole working copy. In this case, the 'revert' operation will destroy the pending change-list and will bring the working copy back to its original state.

Resolve Conflicts

Conflicts can occur at the time of merging. 'Merge' operation automatically handles everything that can be done safely. Everything else is considered as conflict. For example, *"hello.c"* file was modified in branch and deleted in another branch. Such a situation requires a person to make the decision. The 'resolve' operation is used to help the user figure out things and to inform VCS about the ways of handling the conflicts.

Commit Changes

'Commit' operation is used to apply changes from the working copy to the repository. This operation modifies the repository and other developers can see these changes by updating their working copy.

Before commit, one has to add files/directories to the pending change-list. This is the place where changes wait to be committed. With commit, we usually provide a log message to explain why someone made changes. This log message becomes a part of the history of the repository. Commit is an atomic operation, which means either the entire commit succeeds or it is rolled back. Users never see half-finished commit.

4. CHECKOUT PROCESS

Subversion provides the *checkout* command to check out a working copy from a repository. Below command will create a new directory in the current working directory with the name *project_repo*. Don't bother about the repository URL, as most of the time, it is already provided by the subversion administrator with appropriate access.

```
[tom@CentOS ~]$ svn checkout http://svn.server.com/svn/project_repo --
username=tom
```

The above command will produce the following result.

```
A    project_repo/trunk
A    project_repo/branches
A    project_repo/tags
Checked out revision 1.
```

After every successful checkout operation, the revision number will get printed. If you want to view more information about the repository, then execute the *info* command.

```
[tom@CentOS trunk]$ pwd
/home/tom/project_repo/trunk

[tom@CentOS trunk]$ svn info
```

The above command will produce the following result.

```
Path: .
URL: http://svn.server.com/svn/project_repo/trunk
Repository Root: http://svn.server.com/svn/project_repo
Repository UUID: 7ceef8cb-3799-40dd-a067-c216ec2e5247
Revision: 1
Node Kind: directory
Schedule: normal
Last Changed Author: jerry
Last Changed Rev: 0
```

```
Last Changed Date: 2013-08-24 18:15:52 +0530 (Sat, 24 Aug 2013)


[tom@CentOS trunk]$
```

5. PERFORM CHANGES

Jerry checks out the latest version of the repository and starts working on a project. He creates *array.c* file inside the trunk directory.

```
[jerry@CentOS ~]$ cd project_repo/trunk/

[jerry@CentOS trunk]$ cat array.c
```

The above command will produce the following result.

```
#include <stdio.h>

#define MAX 16

int main(void)
{
   int i, n, arr[MAX];

   printf("Enter the total number of elements: ");
   scanf("%d", &n);

   printf("Enter the elements\n");

   for (i = 0; i < n; ++i)
      scanf("%d", &arr[i]);

   printf("Array has following elements\n");
   for (i = 0; i < n; ++i)
      printf("|%d| ", arr[i]);

   printf("\n");

   return 0;
```

```
}
```

He wants to test his code before commit.

```
[jerry@CentOS trunk]$ make array
cc     array.c   -o array

[jerry@CentOS trunk]$ ./array
Enter the total number of elements: 5
Enter the elements
1
2
3
4
5
Array has following elements
|1| |2| |3| |4| |5|
```

He compiled and tested his code and everything is working as expected, now it is time to commit changes.

```
[jerry@CentOS trunk]$ svn status
?       array.c
?       array
```

Subversion is showing '*?*' in front of filenames because it doesn't know what to do with these files.

Before commit, *Jerry* needs to add this file to the pending change-list.

```
[jerry@CentOS trunk]$ svn add array.c
A         array.c
```

Let us check it with the 'status' operation. Subversion shows **A** before *array.c*, it means, the file is successfully added to the pending change-list.

```
[jerry@CentOS trunk]$ svn status
?       array
A       array.c
```

To store *array.c* file to the repository, use the commit command with -m option followed by commit message. If you omit -m option Subversion will bring up the text editor where you can type a multi-line message.

```
[jerry@CentOS trunk]$ svn commit -m "Initial commit"
Adding         trunk/array.c
Transmitting file data .
Committed revision 2.
```

Now *array.c* file is successfully added to the repository, and the revision number is incremented by one.

6. REVIEW CHANGES

Jerry already added *array.c* file to the repository. *Tom* also checks out the latest code and starts working.

```
[tom@CentOS ~]$ svn co http://svn.server.com/svn/project_repo --
username=tom
```

Above command will produce the following result.

```
A    project_repo/trunk
A    project_repo/trunk/array.c
A    project_repo/branches
A    project_repo/tags
Checked out revision 2.
```

But, he found that someone has already added the code. So he is curious about who did that and he checks the log message to see more details using the following command:

```
[tom@CentOS trunk]$ svn log
```

Above command will produce the following result.

```
------------------------------------------------------------------------
r2 | jerry | 2013-08-17 20:40:43 +0530 (Sat, 17 Aug 2013) | 1 line

Initial commit
------------------------------------------------------------------------
r1 | jerry | 2013-08-04 23:43:08 +0530 (Sun, 04 Aug 2013) | 1 line

Create trunk, branches, tags directory structure
------------------------------------------------------------------------
```

When *Tom* observes *Jerry's* code, he immediately notices a bug in that. Jerry was not checking for array overflow, which could cause serious problems. So Tom decides to fix this problem. After modification, *array.c* will look like this.

```
#include <stdio.h>
```

```
#define MAX 16

int main(void)
{
   int i, n, arr[MAX];

   printf("Enter the total number of elements: ");
   scanf("%d", &n);

   /* handle array overflow condition */
   if (n > MAX) {
      fprintf(stderr, "Number of elements must be less than %d\n", MAX);
      return 1;
   }

   printf("Enter the elements\n");

   for (i = 0; i < n; ++i)
      scanf("%d", &arr[i]);

   printf("Array has following elements\n");
   for (i = 0; i < n; ++i)
      printf("|%d| ", arr[i]);
      printf("\n");

   return 0;
}
```

Tom wants to use the status operation to see the pending change-list.

```
[tom@CentOS trunk]$ svn status
M       array.c
```

array.c file is modified, that's why Subversion shows **M** letter before file name. Next *Tom* compiles and tests his code and it is working fine. Before committing changes, he wants to double-check it by reviewing the changes that he made.

```
[tom@CentOS trunk]$ svn diff
Index: array.c
===================================================================
--- array.c   (revision 2)
+++ array.c   (working copy)
@@ -9,6 +9,11 @@
    printf("Enter the total number of elements: ");
    scanf("%d", &n);

+   if (n > MAX) {
+      fprintf(stderr, "Number of elements must be less than %d\n",
MAX);
+      return 1;
+   }
+
    printf("Enter the elements\n");

    for (i = 0; i < n; ++i)
```

Tom has added a few lines in the *array.c* file, that's why Subversion shows **+** sign before new lines. Now he is ready to commit his changes.

```
[tom@CentOS trunk]$ svn commit -m "Fix array overflow problem"
```

The above command will produce the following result.

```
Sending        trunk/array.c
Transmitting file data .
Committed revision 3.
```

Tom's changes are successfully committed to the repository.

7. UPDATE PROCESS

Jerry had committed the first version of the code. But he thinks that he should write two functions to accept input and to display array contents. After modification, *array.c* looks as follows.

```
#include <stdio.h>

#define MAX 16

void accept_input(int *arr, int n)
{
   int i;

   for (i = 0; i < n; ++i)
      scanf("%d", &arr[i]);
}

void display(int *arr, int n)
{
   int i;

   for (i = 0; i < n; ++i)
      printf("|%d| ", arr[i]);

   printf("\n");
}

int main(void)
{
   int i, n, arr[MAX];

   printf("Enter the total number of elements: ");
```

```
   scanf("%d", &n);

   printf("Enter the elements\n");
   accept_input(arr, n);

   printf("Array has following elements\n");
   display(arr, n);

   return 0;
}
```

Jerry compiles and tests his code and is ready to commit changes. Before that, he wants to review the changes using the following command.

```
[jerry@CentOS trunk]$ svn diff
```

The above command will produce the following result.

```
Index: array.c
===================================================================
--- array.c   (revision 2)
+++ array.c   (working copy)
@@ -2,6 +2,24 @@

 #define MAX 16

+void accept_input(int *arr, int n)
+{
+   int i;
+
+   for (i = 0; i & n; ++i)
+      scanf("%d", &arr[i]);
+}
+
+void display(int *arr, int n)
+{
```

```
+   int i;
+
+   for (i = 0; i < n; ++i)
+      printf("|%d| ", arr[i]);
+
+   printf("\n");
+}
+
 int main(void)
 {
    int i, n, arr[MAX];
@@ -10,15 +28,10 @@
    scanf("%d", &n);

    printf("Enter the elements\n");
+   accept_input(arr, n);

-   for (i = 0; i < n; ++i)
-      scanf("%d", &arr[i]);
-
    printf("Array has following elements\n");
-   for (i = 0; i < n; ++i)
-      printf("|%d| ", arr[i]);
-
-   printf("\n");
+   display(arr, n);

    return 0;
 }
```

For the new added lines, Subversion shows **+** sign before line and for removed line it shows **-**sign. Now, Jerry tries to commit the changes using the following command:

```
[jerry@CentOS trunk]$ svn commit -m "Add function to accept input and to
display array contents"
```

The above command will produce the following result.

```
Sending        trunk/array.c
svn: Commit failed (details follow):
svn: File or directory 'array.c' is out of date; try updating
svn: resource out of date; try updating
```

Subversion is not allowing to commit *Jerry's* changes, because *Tom* has already modified the repository and *Jerry's* working copy is out of date. To avoid overwriting each other's changes, Subversion fails this operation. *Jerry* must update working copy before committing his changes. So he uses update command as shown below.

```
[jerry@CentOS trunk]$ svn update
G    array.c
Updated to revision 3.
```

Subversion is showing the letter ***G*** before filename, which means this file has been merged.

```
[jerry@CentOS trunk]$ svn diff
```

The above command will produce the following result.

```
Index: array.c
===================================================================
--- array.c   (revision 3)
+++ array.c   (working copy)
@@ -2,6 +2,24 @@

 #define MAX 16

+void accept_input(int *arr, int n)
+{
+   int i;
+
+   for (i = 0; i < n; ++i)
```

```
+      scanf("%d", &arr[i]);
+}
+
+void display(int *arr, int n)
+{
+   int i;
+
+   for (i = 0; i < n; ++i)
+      printf("|%d| ", arr[i]);
+
+   printf("\n");
+}

+
 int main(void)
 {
    int i, n, arr[MAX];
@@ -15,15 +33,10 @@
    }

    printf("Enter the elements\n");
+   accept_input(arr, n);

-   for (i = 0; i < n; ++i)
-      scanf("%d", &arr[i]);
-
    printf("Array has following elements\n");
-   for (i = 0; i < n; ++i)
-      printf("|%d| ", arr[i]);
-
-   printf("\n");
+   display(arr, n);
```

```
    return 0;
 }
```

Subversion is showing only *Jerry's* changes, but *array.c* file is merged. If you observe carefully, Subversion is now showing revision number 3. In the previous output, it was showing revision number 2. Just review who made changes in the file and for what purpose.

```
jerry@CentOS trunk]$ svn log
------------------------------------------------------------------------
r3 | tom | 2013-08-18 20:21:50 +0530 (Sun, 18 Aug 2013) | 1 line

Fix array overflow problem
------------------------------------------------------------------------
r2 | jerry | 2013-08-17 20:40:43 +0530 (Sat, 17 Aug 2013) | 1 line

Initial commit
------------------------------------------------------------------------
r1 | jerry | 2013-08-04 23:43:08 +0530 (Sun, 04 Aug 2013) | 1 line

Create trunk, branches, tags directory structure
------------------------------------------------------------------------
```

Now *Jerry's* working copy is synchronized with the repository and he can safely commit his changes.

```
[jerry@CentOS trunk]$ svn commit -m "Add function to accept input and to
display array contents"
Sending        trunk/array.c
Transmitting file data .
Committed revision 4.
```

8. FIX MISTAKES

Suppose *Jerry* accidently modifies *array.c* file and he is getting compilation errors. Now he wants to throw away the changes. In this situation, 'revert' operation will help. Revert operation will undo any local changes to a file or directory and resolve any conflicted states.

```
[jerry@CentOS trunk]$ svn status
```

Above command will produce the following result.

```
M       array.c
```

Let's try to make array as follows:

```
[jerry@CentOS trunk]$ make array
```

Above command will produce the following result.

```
cc     array.c   -o array
array.c: In function 'main':
array.c:26: error: 'n' undeclared (first use in this function)
array.c:26: error: (Each undeclared identifier is reported only once
array.c:26: error: for each function it appears in.)
array.c:34: error: 'arr' undeclared (first use in this function)
make: *** [array] Error 1
```

Jerry performs 'revert' operation on *array.c* file.

```
[jerry@CentOS trunk]$ svn revert array.c
Reverted 'array.c'

[jerry@CentOS trunk]$ svn status
[jerry@CentOS trunk]$
```

Now compile the code.

```
[jerry@CentOS trunk]$ make array
cc     array.c   -o array
```

After the revert operation, his working copy is back to its original state. Revert operation can revert a single file as well as a complete directory. To revert a directory, use -R option as shown below.

```
[jerry@CentOS project_repo]$ pwd
/home/jerry/project_repo

[jerry@CentOS project_repo]$ svn revert -R trunk
```

Till now, we have seen how to revert changes, which has been made to the working copy. But what if you want to revert a committed revision! Version Control System tool doesn't allow to delete history from the repository. We can only append history. It will happen even if you delete files from the repository. To undo an old revision, we have to reverse whatever changes were made in the old revision and then commit a new revision. This is called a reverse merge.

Let us suppose Jerry adds a code for linear search operation. After verification he commits his changes.

```
[jerry@CentOS trunk]$ svn diff
Index: array.c
===================================================================
--- array.c   (revision 21)
+++ array.c   (working copy)
@@ -2,6 +2,16 @@

 #define MAX 16

+int linear_search(int *arr, int n, int key)
+{
+   int i;
+
+   for (i = 0; i < n; ++i)
+      if (arr[i] == key)
+         return i;
+   return -1;
+}
+
```

```
 void bubble_sort(int *arr, int n)
 {
    int i, j, temp, flag = 1;

[jerry@CentOS trunk]$ svn status
?       array
M       array.c

[jerry@CentOS trunk]$ svn commit -m "Added code for linear search"
Sending        trunk/array.c
Transmitting file data .
Committed revision 22.
```

Jerry is curious about what Tom is doing. So he checks the Subversion log messages.

```
[jerry@CentOS trunk]$ svn log
```

The above command will produce the following result.

```
------------------------------------------------------------------------
r5 | tom | 2013-08-24 17:15:28 +0530 (Sat, 24 Aug 2013) | 1 line

Add binary search operation
------------------------------------------------------------------------
r4 | jerry | 2013-08-18 20:43:25 +0530 (Sun, 18 Aug 2013) | 1 line

Add function to accept input and to display array contents
```

After viewing the log messages, Jerry realizes that he did a serious mistake. Because Tom already implemented binary search operation, which is better than the linear search; his code is redundant, and now Jerry has to revert his changes to the previous revision. So, first find the current revision of the repository. Currently, the repository is at revision 22 and we have to revert it to the previous revision, i.e. revision 21.

```
[jerry@CentOS trunk]$ svn up
At revision 22.
```

```
[jerry@CentOS trunk]$ svn merge -r 22:21 array.c
--- Reverse-merging r22 into 'array.c':
U    array.c

[jerry@CentOS trunk]$ svn commit -m "Reverted to revision 21"
Sending        trunk/array.c
Transmitting file data .
Committed revision 23.
```

9. RESOLVE CONFLICTS

Tom decides to add a README file for their project. So he creates the *README* file and adds TODO list into that. After adding this, the file repository is at revision 6.

```
[tom@CentOS trunk]$ cat README
/* TODO: Add contents in README file */

[tom@CentOS trunk]$ svn status
?       README

[tom@CentOS trunk]$ svn add README
A         README

[tom@CentOS trunk]$ svn commit -m "Added README file. Will update it's
content in future."
Adding         trunk/README
Transmitting file data .
Committed revision 6.
```

Jerry checks out the latest code which is at revision 6. And immediately he starts working. After a few hours, *Tom* updates the README file and commits his changes. The modified README will look like this.

```
[tom@CentOS trunk]$ cat README
* Supported operations:

1) Accept input
2) Display array elements

[tom@CentOS trunk]$ svn status
M       README

[tom@CentOS trunk]$ svn commit -m "Added supported operation in README"
```

```
Sending        trunk/README
Transmitting file data .
Committed revision 7.
```

Now, the repository is at revision 7 and *Jerry's* working copy is out of date. *Jerry* also updates the README file and tries to commit his changes.

Jerry's README file looks like this.

```
[jerry@CentOS trunk]$ cat README
* File list

1) array.c Implementation of array operation.
2) README  Instructions for user.

[jerry@CentOS trunk]$ svn status
M       README

[jerry@CentOS trunk]$ svn commit -m "Updated README"
Sending        trunk/README
svn: Commit failed (details follow):
svn: File or directory 'README' is out of date; try updating
svn: resource out of date; try updating
```

Step 1: View Conflicts

Subversion has detected that the README file has changed since last updated. So, *Jerry* has to update his working copy.

```
[jerry@CentOS trunk]$ svn up
Conflict discovered in 'README'.
Select: (p) postpone, (df) diff-full, (e) edit,
        (mc) mine-conflict, (tc) theirs-conflict,
        (s) show all options:
```

Subversion is complaining that there is a conflict with the README file, and Subversion does not know how to solve this. So *Jerry* chooses the *df* option to review the conflict.

```
[jerry@CentOS trunk]$ svn up
Conflict discovered in 'README'.
Select: (p) postpone, (df) diff-full, (e) edit,
        (mc) mine-conflict, (tc) theirs-conflict,
        (s) show all options: df
--- .svn/text-base/README.svn-base Sat Aug 24 18:07:13 2013
+++ .svn/tmp/README.tmp      Sat Aug 24 18:13:03 2013
@@ -1 +1,11 @@
-/* TODO: Add contents in README file */
+<<<<<<< .mine
+* File list
+
+1) array.c      Implementation of array operation.
+2) README Instructions for user.
+=======
+* Supported operations:
+
+1) Accept input
+2) Display array elements
+>>>>>>> .r7
Select: (p) postpone, (df) diff-full, (e) edit, (r) resolved,
        (mc) mine-conflict, (tc) theirs-conflict,
        (s) show all options:
```

Step 2: Postpone Conflicts

Next *Jerry* chooses the *postpone(p)* options, so that he can resolve the conflict.

```
Select: (p) postpone, (df) diff-full, (e) edit, (r) resolved,
        (mc) mine-conflict, (tc) theirs-conflict,
        (s) show all options: p
C    README
Updated to revision 7.
Summary of conflicts:
```

```
  Text conflicts: 1
```

After opening the README in text editor, he realizes that Subversion has included both *Tom's* code and his code with conflict markers.

```
[jerry@CentOS trunk]$ cat README
<<<<<<< .min
* File list

1) array.c Implementation of array operation.
2) README  Instructions for user.
=======
* Supported operations:

1) Accept input
2) Display array elements
>>>>>>> .r7
```

Jerry wants *Tom's* changes as well as his, so he just removes the lines containing the conflict markers.

So, the modified README will look like this.

```
[jerry@CentOS trunk]$ cat README
* File list

1) array.c Implementation of array operation.
2) README  Instructions for user.

* Supported operations:

1) Accept input
2) Display array elements
```

Jerry resolved the conflict and he retries commit.

```
[jerry@CentOS trunk]$ svn commit -m "Updated README"
svn: Commit failed (details follow):
```

```
svn: Aborting commit: '/home/jerry/project_repo/trunk/README' remains in
conflict


[jerry@CentOS trunk]$ svn status
?       README.r6
?       README.r7
?       README.mine
C       README
```

Step 3: Resolve Conflicts

In the above commit, the letter ***C*** indicates that there is a conflict in the README file. *Jerry* resolved the conflict but didn't tell Subversion that he had resolved the conflict. He uses the resolve command to inform Subversion about the conflict resolution.

```
[jerry@CentOS trunk]$ svn resolve --accept=working README
Resolved conflicted state of 'README'


[jerry@CentOS trunk]$ svn status
M       README


[jerry@CentOS trunk]$ svn commit -m "Updated README"
Sending        trunk/README
Transmitting file data .
Committed revision 8.
```

10. TAGS

Version Control System supports the *tag* operation by using that concept that one can give meaningful name to a specific version of the code. Tag allows to give descriptive and memorable names to specific version of code. For example **BASIC_ARRAY_OPERATIONS** is more memorable than **revision 4**.

Let us see *tag* operation with an example. Tom decides to create a tag so that he can access the code more easily.

```
[tom@CentOS project_repo]$ svn copy --revision=4 trunk/
tags/basic_array_operations
```

Above command will produce the following result.

```
A    tags/basic_array_operations/array.c
Updated to revision 4.
A         tags/basic_array_operations
```

Upon successful completion, the new directory will be created inside the *tags* directory.

```
[tom@CentOS project_repo]$ ls -l tags/
total 4
drwxrwxr-x. 3 tom tom 4096 Aug 24 18:18 basic_array_operations
```

Tom wants to double-check it before commit. Status operation is showing that the tag operation is successful, so he can safely commit his changes.

```
[tom@CentOS project_repo]$ svn status
A  +    tags/basic_array_operations


[tom@CentOS project_repo]$ svn commit -m "Created tag for basic array
operations"
Adding         tags/basic_array_operations


Committed revision 5.
```

11. BRANCHING

Branch operation creates another line of development. It is useful when someone wants the development process to fork off into two different directions. Let us suppose you have released a product of version 1.0, you might want to create new branch so that development of 2.0 can be kept separate from 1.0 bug fixes.

In this section, we will see how to create, traverse and merge branch. Jerry is not happy because of the conflict, so he decides to create a new private branch.

```
[jerry@CentOS project_repo]$ ls
branches  tags  trunk

[jerry@CentOS project_repo]$ svn copy trunk branches/jerry_branch
A         branches/jerry_branch

[jerry@CentOS project_repo]$ svn status
A  +    branches/jerry_branch

[jerry@CentOS project_repo]$ svn commit -m "Jerry's private branch"
Adding         branches/jerry_branch
Adding         branches/jerry_branch/README

Committed revision 9.
[jerry@CentOS project_repo]$
```

Now Jerry is working in his private branch. He adds sort operation for the array. Jerry's modified code looks like this.

```
[jerry@CentOS project_repo]$ cd branches/jerry_branch/

[jerry@CentOS jerry_branch]$ cat array.c
```

The above command will produce the following result.

```
#include <stdio.h>
#define MAX 16

void bubble_sort(int *arr, int n)
{

   int i, j, temp, flag = 1;

   for (i = 1; i < n && flag == 1; ++i) {
      flag = 0;
      for (j = 0; j < n - i; ++j) {
         if (arr[j] > arr[j + 1]) {
            flag = 1;
            temp = arr[j];
            arr[j]  = arr[j + 1];
            arr[j + 1] = temp;
         }
      }
   }
}

void accept_input(int *arr, int n)
{
   int i;

   for (i = 0; i < n; ++i)
      scanf("%d", &arr[i]);
}

void display(int *arr, int n)
{
   int i;
```

```
   for (i = 0; i < n; ++i)
      printf("|%d| ", arr[i]);

   printf("\n");
}

int main(void)
{
   int i, n, key, ret, arr[MAX];

   printf("Enter the total number of elements: ");
   scanf("%d", &n);

   /* Error handling for array overflow */
   if (n >MAX) {
      fprintf(stderr, "Number of elements must be less than %d\n", MAX);
      return 1;
   }

   printf("Enter the elements\n");
   accept_input(arr, n);

   printf("Array has following elements\n");
   display(arr, n);

   printf("Sorted data is\n");
   bubble_sort(arr, n);
   display(arr, n);

   return 0;
}
```

Jerry compiles and tests his code and is ready to commit his changes.

```
[jerry@CentOS jerry_branch]$ make array
cc     array.c   -o array

[jerry@CentOS jerry_branch]$ ./array
```

The above command will produce the following result.

```
Enter the total number of elements: 5
Enter the elements
10
-4
2
7
9
Array has following elements
|10| |-4| |2| |7| |9|
Sorted data is
|-4| |2| |7| |9| |10|

[jerry@CentOS jerry_branch]$ svn status
?       array
M       array.c

[jerry@CentOS jerry_branch]$ svn commit -m "Added sort operation"
Sending        jerry_branch/array.c
Transmitting file data .
Committed revision 10.
```

Meanwhile, over in the trunk, Tom decides to implement search operation. Tom adds code for search operation and his code looks like this.

```
[tom@CentOS trunk]$ svn diff
```

The above command will produce the following result.

```
Index: array.c
===================================================================
--- array.c   (revision 10)
+++ array.c   (working copy)
@@ -2,6 +2,27 @@

 #define MAX 16

+int bin_search(int *arr, int n, int key)
+{
+   int low, high, mid;
+
+   low   = 0;
+   high   = n - 1;
+   mid   = low + (high - low) / 2;
+
+   while (low <= high) {
+      if (arr[mid] == key)
+         return mid;
+      if (arr[mid] > key)
+         high = mid - 1;
+      else
+         low = mid + 1;
+      mid = low + (high - low) / 2;
+   }
+
+   return -1;
+}
+
 void accept_input(int *arr, int n)
 {
    int i;
```

```
@@ -22,7 +43,7 @@

 int main(void)
 {
-   int i, n, arr[MAX];
+   int i, n, ret, key, arr[MAX];

    printf("Enter the total number of elements: ");
    scanf("%d", &n);
@@ -39,5 +60,16 @@
    printf("Array has following elements\n");
    display(arr, n);

+   printf("Enter the element to be searched: ");
+   scanf("%d", &key);
+
+   ret = bin_search(arr, n, key);
+   if (ret < 0) {
+      fprintf(stderr, "%d element not present in array\n", key);
+      return 1;
+   }
+
+   printf("%d element found at location %d\n", key, ret + 1);
+
    return 0;
 }
```

After reviewing, he commits his changes.

```
[tom@CentOS trunk]$ svn status
?       array
M       array.c

[tom@CentOS trunk]$ svn commit -m "Added search operation"
```

```
Sending        trunk/array.c
Transmitting file data .

Committed revision 11.
```

But Tom is curious about what Jerry has been doing in his private branch.

```
[tom@CentOS trunk]$ cd ../branches/
[tom@CentOS branches]$ svn up
A    jerry_branch
A    jerry_branch/array.c
A    jerry_branch/README

[tom@CentOS branches]$ svn log
------------------------------------------------------------------------
r9 | jerry | 2013-08-27 21:56:51 +0530 (Tue, 27 Aug 2013) | 1 line

Added sort operation
------------------------------------------------------------------------
```

By viewing the Subversion's log message, Tom found that Jerry implemented 'sort' operation. Tom implemented search operation using binary search algorithm, it always expects data in sorted order. But what if the user provides data in an unsorted order? In that situation, binary search operation will fail. So he decides to take Jerry's code to sort data before the search operation. So he asks Subversion to merge code from Jerry's branch into trunk.

```
[tom@CentOS trunk]$ pwd
/home/tom/project_repo/trunk

[tom@CentOS trunk]$ svn merge ../branches/jerry_branch/
--- Merging r9 through r11 into '.':
U    array.c
```

After merging, array.c will look like this.

```
[tom@CentOS trunk]$ cat array.c
```

The above command will produce the following result.

```
#include <stdio.h>

#define MAX 16

void bubble_sort(int *arr, int n)
{
   int i, j, temp, flag = 1;

   for (i = 1; i < n && flag == 1; ++i) {
      flag = 0;
      for (j = 0; j < n - i; ++j) {
         if (arr[j] > arr[j + 1]) {
            flag       = 1;
            temp       = arr[j];
            arr[j]       = arr[j + 1];
            arr[j + 1] = temp;
         }
      }
   }
}

int bin_search(int *arr, int n, int key)
{
   int low, high, mid;

   low   = 0;
   high    = n - 1;
   mid   = low + (high - low) / 2;

   while (low <= high) {
      if (arr[mid] == key)
         return mid;
```

```
      if (arr[mid] > key)
         high = mid - 1;
      else
         low = mid + 1;
      mid = low + (high - low) / 2;
   }

   return -1;
}

void accept_input(int *arr, int n)
{
   int i;

   for (i = 0; i < n; ++i)
      scanf("%d", &arr[i]);
}

void display(int *arr, int n)
{
   int i;

   for (i = 0; i < n; ++i)
      printf("|%d| ", arr[i]);

   printf("\n");
}

int main(void)
{
   int i, n, ret, key, arr[MAX];

   printf("Enter the total number of elements: ");
```

```
   scanf("%d", &n);

   /* Error handling for array overflow */
   if (n > MAX) {
      fprintf(stderr, "Number of elements must be less than %d\n", MAX);
      return 1;
   }

   printf("Enter the elements\n");
   accept_input(arr, n);

   printf("Array has following elements\n");
   display(arr, n);

   printf("Sorted data is\n");
   bubble_sort(arr, n);
   display(arr, n);

   printf("Enter the element to be searched: ");
   scanf("%d", &key);

   ret = bin_search(arr, n, key);
   if (ret < 0) {
      fprintf(stderr, "%d element not present in array\n", key);
      return 1;
   }

   printf("%d element found at location %d\n", key, ret + 1);

   return 0;
}
```

After compilation and testing, Tom commits his changes to the repository.

```
[tom@CentOS trunk]$ make array
cc     array.c   -o array

[tom@CentOS trunk]$ ./array
Enter the total number of elements: 5
Enter the elements
10
-2
8
15
3
Array has following elements
|10| |-2| |8| |15| |3|
Sorted data is
|-2| |3| |8| |10| |15|
Enter the element to be searched: -2
-2 element found at location 1

[tom@CentOS trunk]$ svn commit -m "Merge changes from Jerry's code"
Sending        trunk
Sending        trunk/array.c
Transmitting file data .
Committed revision 12.

[tom@CentOS trunk]$
```