SVN 简明教程

www.tutorialspoint.com/svn/index.htm
https://github.com/wuzhouhui/svn

2016 年 11 月 26 日

目录

1 基本概念

1.1 什么是版本控制系统

版本控制系统 (Version Control System, 简称 VCS) 是一种软件, 它可以帮助软件开发人员协同工作, 以及管理产品的完整历史.

版本控制系统的目标包括:

- 支持多人同时操作.
- 不覆盖其他人作出的修改.
- 维护每一个版本的历史.

VCS 可以分成两大类别:

- 集中式的版本控制系统 (Centralized Version Control System, 简称 CVCS);
- 分布式的版本控制系统 (Distributed Version Control System, 简称 DVCS).

本教程只讨论 CVCS, 特别是 Subversion, 它使用中央服务器来存储所有的文件, 并支持团队协作.

1.2 版本控制术语

首先先来介绍本教程将会用到的几个术语.

- 仓库 (Repository): 仓库是所有版本控制系统的核心, 它是开发人员存放所有资料的中心目录. 除了文件, 仓库还会存放历史. 仓库支持网络访问, 相当于一个服务器, 而版本控制工具则是客户端. 客户端可以连接仓库, 然后就可以向仓库提交修改, 或检索修改历史. 通过提交, 其他客户端就可以看到某个客户端作出的修改; 通过检查修改历史, 客户端就可以把其他人的修改作为工作副本.
- 主干 (Trunk): 主干是一个目录, 它是所有主要开发发生的地方, 通常会被开发人员检出, 以便进行项目开发.
- 标签 (Tags): 标签是用于存放项目的命名快照的目录. 通过标签, 开发人员可以给仓库的某个特定版本取一个描述性的, 易于记忆的名字.

比如, `LAST_STABLE_CODE_BEFORE_EMAIL_SUPPORT` 就比 `Repository UUID: 7ceef8cb-3799-40dd-a067-c216ec2e5247` 和 `Revision: 13` 容易记忆.

- 分支 (Branches): 分支用来创建一条新的开发线. 如果开发人员想要把开发过程分裂成两个方向, 就会用到该功能. 例如, 开发人员在发布了 5.0 版本后, 可能会创建一条新的分支, 专门用于开发 6.0 版本, 这样的话, 6.0 的开发就不会与 5.0 的问题修复相互混淆.

- 工作副本 (Working copy): 工作副本是仓库的一个快照. 仓库被团队内的所有人共享, 但人们不能直接修改仓库, 解决办法是每个开发人员都从仓库中检出一份工作副本, 这个工作副本就是他的私有工作区, 开发人员在工作副本中所做的工作并不会影响到团队中的其他人.

- 提交修改 (Commit changes): 把私有工作区的修改存放到中央服务器的过程称为提交. 提交后, 团队中的其他人就可以看到别人作出的修改, 通过检索修改, 开发人员可以把修改更新到他们的工作副本中. 提交是一个原子操作, 要么全部的修改提交成功, 要么全部失败, 不可能出现只提交一半的情况.

2 环境设置

2.1 安装 SVN 客户端工具

Subversion 是一款流行的版本控制工具, 它是开源软件, 可以在因特网上免费获取. 大部分 GNU/Linux 发行版都默认安装了 Subversion, 可以用下面的命令检查:

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

如果系统中已经安装了 Subversion 客户端, 命令就会输出 Subversion 的版本号, 否则的话, 就会输出一条错误信息:

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

如果读者用的是基于 RPM 的 GNU/Linux 发行版, 可以用命令 yum 来安装 Subversion, 安装完成后, 再执行 `svn --version` 检查是否安装成功:

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

[jerry@CentOS ~]$ svn --version
svn, version 1.6.11 (r934486)
   compiled Aug 17 2015, 08:37:43

Copyright (C) 2000-2009 CollabNet.
Subversion is open source software, see http://subversion.tigris.org/
This product includes software developed by CollabNet
   (http://www.Collab.Net/).

The following repository access (RA) modules are available:

* ra_neon : Module for accessing a repository via WebDAV protocol using
   Neon.
  - handles 'http' scheme
```

```
  - handles 'https' scheme
* ra_svn : Module for accessing a repository using the svn network
   protocol.
  - with Cyrus SASL authentication
  - handles 'svn' scheme
* ra_local : Module for accessing a repository on local disk.
  - handles 'file' scheme
```

如果是 Debian 系列的 GNU/Linux 发行版, 就用命令 apt 安装:

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

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

[jerry@Ubuntu]$ svn --version
svn, version 1.8.8 (r1568071)
   compiled Aug 20 2015, 12:51:12 on i686-pc-linux-gnu

Copyright (C) 2013 The Apache Software Foundation.
This software consists of contributions made by many people;
see the NOTICE file for more information.
Subversion is open source software, see http://subversion.apache.org/

The following repository access (RA) modules are available:

* ra_svn : Module for accessing a repository using the svn network
   protocol.
  - with Cyrus SASL authentication
  - handles 'svn' scheme
* ra_local : Module for accessing a repository on local disk.
  - handles 'file' scheme
* ra_serf : Module for accessing a repository via WebDAV protocol using
   serf.
  - using serf 1.3.3
  - handles 'http' scheme
  - handles 'https' scheme
```

2.2 服务器端 Apache 设置

上面介绍了如何在 GNU/Linux 中安装 Subversion 客户端, 现在介绍如何创建一个新的仓库, 并设置访问权限.

在服务器端需要安装 Apache httpd 模块和 svnadmin 工具:

```
[root@CentOS ~]# yum install mod_dav_svn subversion
```

安装了软件包 mod_dav_svn 之后, 用户就可以用 HTTP 来访问仓库, 软件包 subversion 包含了 svnadmin 工具.

/etc/httpd/conf.d/subversion.conf 是 Subversion 的配置文件, 该文件的典型内容如下:

```
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>
```

现在开始创建 Subversion 用户, 并赋予他们访问仓库的权限. 命令 htpasswd 用于创建和更新纯文本文件, 这些文件中存放着用户名和密码. 如果指定了选项 -c, 则命令会创建一个密码文件, 如果密码文件已经存在, 就会被覆盖, 因此选项 -c 只在第一次时用到. 选项 -m 开启密码的 MD5 加密.

假设现在要创建用户 tom:

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

再创建一个用户 jerry:

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

创建 Subversion 父目录, 用于存放所有的工作 (见 /etc/httpd/conf.d/subversion.conf):

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

2.3 服务器端仓库设置

现在创建一个名为 project_repo 的项目仓库. 命令 svnadmin 在创建一个新的仓库时, 会在其中创建几个子目录, 用于存放元数据:

```
[root@CentOS svn]# pwd
/var/www/svn
[root@CentOS svn]# svnadmin create project_repo
[root@CentOS svn]# ls -l project_repo
total 24
drwxr-xr-x. 2 root root 4096 Aug 28 08:43 conf
drwxr-sr-x. 6 root root 4096 Aug 28 08:43 db
-r--r--r--. 1 root root    2 Aug 28 08:43 format
drwxr-xr-x. 2 root root 4096 Aug 28 08:43 hooks
drwxr-xr-x. 2 root root 4096 Aug 28 08:43 locks
-rw-r--r--. 1 root root  229 Aug 28 08:43 README.txt
```

然后, 修改仓库目录的用户与用户组:

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

查看 SELinux 是否开启:

```
[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
```

在笔者的环境中, SELinux 默认是开启的, 因此还需要修改 SELinux 的安全上下文:

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

为了能让开发人员通过 HTTP 进行提交, 执行:

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

和 Apache 相关的配置到这里为止就全部结束了, 每次更新配置都需要重启 Apache 服务:

```
[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...
```

接下来开始配置仓库. 为了实现只有授权的用户才能访问仓库, 并且使用默认的授权文件, 就把下面几行添加到 project_repo/conf/svnserve.conf 的 [general] 部分:

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

传统上, 每一个仓库下面都有 trunk, tags 和 branches 这三个目录.

目录 trunk 是主要开发发生的地方, 通常会被开发人员检出, 以便进行项目开发.

目录 tags 用于存放项目的命名快照. 当需要发布一个产品版本时, 团队就会给代码打一个标签, 然后存放到这个目录中并发布.

目录 branches 用于存放不同的开发线.

在仓库中创建 trunk, tags, 和 branches 这三个目录:

```
[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
[root@CentOS svn]# svn import /tmp/svn-template/ \
> http://127.0.0.1/svn/project_repo \
> -m 'Create trunk, branches, tags directory structure' \
> --username tom
Adding               /tmp/svn-template/trunk
Adding               /tmp/svn-template/branches
Adding               /tmp/svn-template/tags
Committed revision 1.
[root@CentOS svn]#
```

到这里为止, 就已经成功创建了一个仓库, 该仓库允许 tom 和 jerry 访问, 从现在开始, 他们两个就可以对仓库进行操作了.

3 生命周期

本章讨论版本控制系统的生命周期, 再下一章介绍每一种操作对应的 Subversion 命令.

3.1 创建仓库

仓库是开发人员存放所有资料的目录. 除了文件, 仓库还会记录下各个文件的修改历史. 操作 create (创建) 创建一个新的仓库, 在大多数情况下, 这种操作只会做一次. 当开发人员创建一个新的仓库时, VCS 会要求你输入一些信息, 比如仓库的创建位置, 仓库的名字等.

3.2 检出

操作 checkout (检出) 从仓库中创建一个工作副本到本地. 工作副本是开发人员私有的工作空间, 他们在工作副本中作出修改, 然后再提交到仓库上.

3.3 更新

顾名思义, 操作 update (更新) 用来更新工作副本, 它把本地的工作副本和服务器上的仓库同步. 因为仓库是共享的, 所以开发人员会向仓库提交他们的修改, 这时候其他人的工作副本就会变成过时了的.

假设某一项目有 Tom 和 Jerry 两个开发人员, 他们都从仓库中检出了最新版本的代码, 然后各自开始开发. Jerry 的工作效率比较高, 他很快就把修改提交到了仓库上.

这时候, Tom 的工作副本就变成过时了的. 操作 update 会把 Jerry 的修改拉到本地, 然后把 Tom 的工作副本更新到最新版.

3.4 执行修改

有很多操作都可以对工作副本中的文件产生影响, 编辑是其中最常见的操作, 通过编辑, 文件的内容被添加或删除.

开发人员还可以在工作副本中添加文件或目录, 但是它们并不能立即成为仓库的一部分, 而是被添加到未决的修改列表中, 只有在提交之后才会真正变成仓库的一部分.

类似的, 开发人员还可以删除文件或目录, 删除操作会立刻把工作副本中的对应文件删除, 但是被删除的文件其实是被添加到了未决的修改列表中, 只有在提交之后, 仓库中的对应文件才会被删除.

操作 rename (重命名) 修改文件或目录的名字. 操作 move (移动) 把文件或目录从一个位置移动到另一个位置.

3.5 审核修改

当开发人员从仓库检出代码, 或更新本地工作副本时, 他们的工作副本就会和仓库同步. 如果开发人员对工作副本进行了修改, 那它们就会比仓库中的代码要新. 在执行操作 commit (提交) 之前, 最好对修改进行审核.

操作 status (状态) 列出工作副本所发生的变化. 前面已经说过, 无论何时对工作副本中的文件作出修改, 这些修改就会变成未决的修改列表的一部分, 操作 status 可以列出未决的修改列表的内容.

操作 status 只会列出发生变化的文件或目录, 但不会显示具体的修改细节. 操作 diff 用来查看工作副本中的文件内容具体发生了哪些变化.

3.6 修正错误

假设某个开发人员对工作副本进行了一些修改, 但是现在他想撤消这些修改, 这时候, 可以执行操作 revert (撤消).

操作 revert 可以撤消工作副本的所有修改, 也可以针对一个或多个文件/目录进行撤消. 如果是对整个工作副本进行撤消, 操作 revert 就会销毁未决的修改列表, 把工作副本恢复到修改前的状态.

3.7 解决冲突

冲突会在合并时发生. 操作 merge (合并) 会自动处理可以安全合并的情况, 除此之外的情况都会被当作冲突. 例如, 文件 hello.c 在一个分支中被修改, 而在另一个分支中被删除, 因此在合并这两个分支时开发人员就要作出决定. 操作 resolve 可以帮助开发人员解决冲突, 并通知 VCS 如何处理冲突情况.

3.8 提交修改

操作 commit (提交) 把工作副本所发生的变化应用到仓库中. 这个操作会修改仓库中的代码, 其他开发人员可以通过更新来查看新提交的修改.

在提交前, 开发人员要把文件/目录添加到未决的修改列表上, 这是修改等待提交的地方. 在提交时, 开发人员通常要写上提交日志, 在日志中解释为什么这次修改是必要的, 提交日志会成为仓库历史的一部分. 提交是一种原子操作, 要么全部修改都提交上去, 要么一个也没有, 不可能出现只提交一半的修改.

4 检出

Subversion 提供了命令 checkout, 用于从仓库中检出工作副本. 下面的命令会在当前目录下创建一个新目录: project_repo. 不用对命令中仓库的 URL 感到担心, 在大多数情况下, Subversion 管理员知道如何拼写 URL, 而且配置了适当的访问权限控制:

```
[tom@CentOS ~]$ svn checkout http://localhost/svn/project_repo \
> --username=tom
A   project_repo/trunk
A   project_repo/branches
A   project_repo/tags
Checked out revision 1.
```

如果检出成功, 命令就会打印修订号. 如果用户想要知道关于仓库的更多信息, 可以使用命令 info:

```
[tom@CentOS trunk]$ pwd
/home/tom/project_repo/trunk
[tom@CentOS trunk]$ svn info
Path: .
Working Copy Root Path: /home/dell/Documents/svn/project_repo
```

```
URL: http://localhost/svn/project_repo
Relative URL: ^/
Repository Root: http://localhost/svn/project_repo
Repository UUID: e869d833-99ed-4274-8aaa-be9403c940e3
Revision: 1
Node Kind: directory
Schedule: normal
Last Changed Author: user1
Last Changed Rev: 1
Last Changed Date: 2016-08-13 13:26:40 +0800 (Sat, 13 Aug 2016)
```

5 执行修改

假设 Jerry 从仓库中检出了最新的版本, 然后开始对项目进行开发, 他首先在主干目录内创建了一个 array.c 文件:

```
[jerry@CentOS ~]$ cd project_repo/trunk/
[jerry@CentOS trunk]$ cat array.c
#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;
}
```

他想在提交之前先测试一下代码:

```
[jerry@CentOS trunk]$ gcc 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|
```

编译和运行看来都没什么问题, 现在他准备提交代码.

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

如果 Subversion 不知道如何处理某些文件, 就会在文件名的左边打印一个问号 ?.

在提交前, Jerry 需要把文件 array.c 添加到未决的修改列表上:

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

现在再检查一下工作副本的状态:

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

array.c 左边的 A 表示该文件已经成功地添加到了未决的修改列表上.

为了把 array.c 更新到仓库中, 需要执行命令 commit, 并带上参数 -m, 该参数允许开发人员直接在命令行上写提交信息, 如果省略了参数 -m, subversion 就会自动打开一个文本编辑器, 我们也可以在编辑器中写提交信息:

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

现在, 文件 array.c 就已经成功地添加到了仓库中, 修订号也相应地进行了更新.

6 审核修改

Jerry 把 array.c 提交到仓库后, Tom 从仓库中检出了最新的代码, 然后开始工作:

```
[tom@CentOS ~]$ svn co http://svn.server.conm/svn/project_repo
   --username=tom
A    project_repo/tags
A    project_repo/trunk
A    project_repo/branches
A    project_repo/trunk/array.c
Checked out revision 2.
```

Tom 发现已经有人往仓库中提交了代码, 他很好奇这些代码是谁提交上去的, 于是, 他执行下面的命令查看提交历史:

```
[tom@CentOS trunk]$ svn log
------------------------------------------------------------------------
r2 | jerry | 2016-08-13 13:28:07 +0800 (Sat, 13 Aug 2016) | 1 line

Initial commit
------------------------------------------------------------------------
r1 | jerry | 2016-08-13 13:26:40 +0800 (Sat, 13 Aug 2016) | 2 lines
```

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

------------------------------------------------------------------------
```

Tom 发现 Jerry 的代码中有一个问题: 程序没有检查数组越界的情况, 这可能会导致很严重的后果, 于是 Tom 决定自己把这个问题修复掉. 修改完成后, array.c 的内容变成了:

```
#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;
}
```

执行状态命令, 查看未决的修改列表:

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

因为 array.c 的内容发生了变化, 因此它的左边会显示一个 M. Tom 编译并运行程序后, 觉得没什么问题, 但是在提交前, 他还想再检查一下代码发生了哪些变化:

```
[tom@CentOS trunk]$ svn diff
Index: array.c
===================================================================
--- array.c (revision 2)
+++ array.c (working copy)
@@ -9,6 +9,12 @@
    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++)
```

左边的 + 表示这是新增的行. 检查没问题后, 接下来就可以提交了:

```
[tom@CentOS trunk]$ svn commit -m "Fix array overflow problem"
Sending        array.c
Transmitting file data .
Committed revision 3.
```

到这里为止, Tom 所做的修改就已经正式提交到了仓库中.

7 更新

Jerry 曾经提交过 array.c 的第一个版本, 但是他认为应该用两个函数分别负责数据输入和数据打印, 于是, array.c 被修改成:

```
#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;
}
```

编译并简单地测试后, Jerry 准备提交代码, 但是在提交前, 他想要审核一下代码的修改:

```
[jerry@CentOS trunk]$ svn diff
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]);
+   display(arr, n);

-   printf("\n");
-
    return 0;
 }
```

新增的行, 其左边用 + 标记, 被删除的行用 - 标记. 现在 Jerry 试图向仓库中提交代码:

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

结果命令输出:

```
Sending        array.c
Transmitting file data .svn: E155011: Commit failed (details follow):
svn: E155011: File
   '/home/dell/Documents/svn/jerry_project_repo/trunk/array.c' is out of
   date
svn: E170004: Item '/trunk/array.c' is out of date
```

因为 Tom 在此之前已经向仓库提交过修改, 所以 Jerry 的工作副本就变成过时了的, 这时候 Subversion 会禁止 Jerry 提交修改, 否则的话, 其他人的修改可能会被覆盖掉. 为了提交成功, Jerry 必须先更新工作副本:

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

文件名左边的 G 表示该文件被合并了.

```
[jerry@CentOS trunk]$ svn diff
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];
@@ -16,15 +34,10 @@
    }

    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]);
+   display(arr, n);

-   printf("\n");
-
    return 0;
 }
```

上面只显示了 Jerry 的修改, 但是这时候 array.c 被合并过了. 细心的读者可能会发现输出中的修订号变成了 3, 而在上一次的 svn diff 输出中, 修订号是 2. 再查看一下是谁修改了代码:

```
[jerry@CentOS trunk]$ svn log
------------------------------------------------------------------------
r3 | tom | 2016-08-13 16:35:55 +0800 (Sat, 13 Aug 2016) | 1 line

Fix array overflow problem
------------------------------------------------------------------------
r2 | jerry | 2016-08-13 13:28:07 +0800 (Sat, 13 Aug 2016) | 1 line

Initial commit
------------------------------------------------------------------------
r1 | jerry | 2016-08-13 13:26:40 +0800 (Sat, 13 Aug 2016) | 2 lines

Create trunk, branches, tags directory structure.

------------------------------------------------------------------------
```

工作副本更新后, 就可以安全地向仓库提交代码:

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

8 修改错误

假设 Jerry 修改完 array.c 后, 遇到了编译错误, 因此他想要丢弃现在的修改, 这时候就需要执行 revert 操作. 操作 revert 可以撤消本地工作副本中文件或目录的修改, 除此之外, 它还可以用于解决冲突.

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

array.c 被修改了, 如果试图编译它的话:

```
[jerry@CentOS trunk]$ gcc array.c -o array
array.c: In function ‘main’:
```

```
array.c:28:15: error: ‘n’ undeclared (first use in this function)
  scanf("%d", &n);
               ^
array.c:28:15: note: each undeclared identifier is reported only once for
   each function it appears in
array.c:37:15: error: ‘arr’ undeclared (first use in this function)
  accept_input(arr, n);
               ^
```

编译报错了, Jerry 打算撤消 array.c 的修改:

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

再编译一次, 检查还有没有错误:

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

撤消修改后, 工作副本恢复到了修改前的状态. 操作 revert 不仅可以回滚一个文件, 还可以针对整个目录进行回滚. 回滚目录时, 需要带上参数 -R:

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

到现在为止, 我们已经介绍了如何撤消工作副本中未提交的修改, 但是, 如果修改已经提交了, 那这时候又该怎么办. VCS 不允许从仓库中删除提交历史, 我们所能做的只有添加历史. 为了撤消某个修订, 必须撤消该修订中所提交的所有修改, 然后再提交一个新的修订, 这种操作叫作逆向合并 (reverse merge).

假设 Jerry 为 array.c 定义了一个新函数, 用于线性搜索, 审核后, 他提交了代码:

```
[jerry@CentOS trunk]$ svn diff
Index: array.c
===================================================================
--- array.c (revision 5)
+++ array.c (working copy)
@@ -3,6 +3,16 @@

 #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);
+}
+
 static int cmp(const void *a, const void *b)
 {
    return(*(int *)a - *(int *)b);
```

```
[jerry@CentOS trunk]$ svn status
M       array.c
[jerry@CentOS trunk]$ svn commit -m "Added code for linear
search"
Sending        array.c
Transmitting file data .
Committed revision 6.
```

Jerry 想知道 Tom 向仓库提交了哪些修改:

```
[jerry@CentOS trunk]$ svn log
------------------------------------------------------------------------
r5 | tom | 2016-08-14 17:30:06 +0800 (Sun, 14 Aug 2016) | 1 line

Add binary search operation
------------------------------------------------------------------------
r4 | jerry | 2016-08-14 08:51:29 +0800 (Sun, 14 Aug 2016) | 1 line

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

从日志中, Jerry 意识到他犯了一个严重的错误: Tom 已经实现了一个二分搜索, 它比线性搜索要好得多, 所以他添加的代码就变成多余的了. Jerry 决定撤消他的最后一次提交, 也就是把仓库回滚到修订号 5 所处的状态:

```
[jerry@CentOS trunk]$ svn up
Updating '.':
At revision 6.
[jerry@CentOS trunk]$ svn merge -r 6:5 array.c
--- Reverse-merging r6 into 'array.c':
U    array.c
--- Recording mergeinfo for reverse merge of r6 into 'array.c':
 U   array.c
--- Eliding mergeinfo from 'array.c':
 U   array.c
[jerry@CentOS trunk]$ svn commit -m "Reverted to revision 5"
Sending        array.c
Transmitting file data .
Committed revision 7.
```

9 解决冲突

Tom 决定为项目添加一个 README 文件, 文件包含了一个 TODO 列表, 提交后, 仓库的修订号更新到 8:

```
[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         README
Transmitting file data .
Committed revision 8.
```

就在 Tom 提交后, Jerry 从仓库中检出了最新的代码, 然后开始自己的工作. 几个小时后, Tom 再一次更新并提交了 README:

```
[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 operations in README"
Sending        README
Transmitting file data .
Committed revision 9.
```

此时, Jerry 检出的代码已经过时了. Jerry 也更新了 README, 并试图提交:

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

1) array.c Implementation of array operations.
2) README  Instructions for user.
[jerry@CentOS trunk]$ svn status
M       README
[jerry@CentOS trunk]$ svn commit -m "Update README"
Sending        README
Transmitting file data .svn: E155011: Commit failed (details follow):
svn: E155011: File
   '/home/dell/Documents/svn/jerry_project_repo/trunk/README' is out of
   date
svn: E170004: Item '/trunk/README' is out of date
```

9.1 查看冲突

Subversion 发现 README 已经过时了, 因此 Jerry 得先更新一下工作副本:

```
[jerry@CentOS trunk]$ svn up
Updating '.':
C    README
Updated to revision 9.
Conflict discovered in file 'README'.
Select: (p) postpone, (df) show diff, (e) edit file, (m) merge,
        (mc) my side of conflict, (tc) their side of conflict,
        (s) show all options:
```

在更新 README 时发生了冲突, Subversion 不知道如何处理这种情况, 于是, Jerry 输入 df, 查看发生冲突的内容:

```
[jerry@CentOS trunk]$ svn up
Updating '.':
C    README
Updated to revision 9.
Conflict discovered in file 'README'.
Select: (p) postpone, (df) show diff, (e) edit file, (m) merge,
        (mc) my side of conflict, (tc) their side of conflict,
        (s) show all options: df
```

```
--- README.r9   - THEIRS
+++ README  - MERGED
@@ -1,4 +1,11 @@
+<<<<<<< .mine
+* File list
+
+1) array.c Implementation of array operation.
+2) README  Instructions for user.
+=======
 * Supported operations:

 1) Accept input
 2) Display array elements
+>>>>>>> .r9
Select: (p) postpone, (df) show diff, (e) edit file, (m) merge,
        (r) mark resolved, (mc) my side of conflict,
        (tc) their side of conflict, (s) show all options:
```

9.2 推迟处理

接下来, Jerry 选择推迟处理, 即选项 p:

```
Select: (p) postpone, (df) show diff, (e) edit file, (m) merge,
        (r) mark resolved, (mc) my side of conflict,
        (tc) their side of conflict, (s) show all options: p
Summary of conflicts:
  Text conflicts: 1
```

用文本编辑器打开 README, 就可以看到文件同时包含了 Tom 的修改, 以及冲突标记:

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

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

1) Accept input
2) Display array elements
>>>>>>> .r9
```

Jerry 希望同时保留他自己和 Tom 的修改, 所以他只要把冲突标记删除即可, 删除冲突标记后, README 的内容变成了:

```
[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
```

然后, Jerry 开始提交修改:

```
[jerry@CentOS trunk]$ svn commit -m "Updated README"
svn: E155015: Commit failed (details follow):
svn: E155015: Aborting commit:
   '/home/dell/Documents/svn/jerry_project_repo/trunk/README' remains in
   conflict
[jerry@CentOS trunk]$ svn status
C       README
?       README.mine
?       README.r8
?       README.r9
Summary of conflicts:
  Text conflicts: 1
```

9.3 解决冲突

在上面的提交中, 文件名左边的 C 表示该文件还有冲突未被处理. Jerry 虽然已经解决了冲突, 但是 Subversion 并不知道这点, 解决办法是使用相应的命令告诉 Subversion 冲突的处理结果:

```
[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        README
Transmitting file data .
Committed revision 10.
```

10 标签

版本控制系统支持标签操作, 打标签指的是给代码的某个特定版本取一个有意义的名字, 例如, BASIC_ARRAY_OPERATIONS 比修订号 4 更容易记住.

Tom 打算为修订号 4 的代码创建一个标签, 这样他就能更方便地访问代码:

```
[tom@CentOS project_repo]$ svn copy -r 4 trunk/
   tags/basic_array_operations
Updating 'tags/basic_array_operations':
A    tags/basic_array_operations/array.c
Updated to revision 4.
A         tags/basic_array_operations
```

执行成功后, 在 tags/ 下就会出现一个新目录:

```
[tom@CentOS project_repo]$ ls -l tags/
total 4
drwxrwxr-x 2 tom tom 4096 Aug 20 09:16 basic_array_operations
```

检查一下, 如果没什么问题, 就可以提交了:

```
[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 11.
```

11 分支

分支用于创建一条新的开发线, 如果开发人员想让开发过程朝着两个不同的方向发展, 这时候就会用到分支. 比如说, 现在已经发布了版本 1.0, 这时候你可能想创建一条新的分支, 用于版本 2.0 的开发, 这样的话, 版本 1.0 的问题修复就不会和 2.0 的开发相混淆.

这一章介绍如何创建与合并分支.

Jerry 常常被冲突搞得很不开心, 所以他决定创建一条自己的开发分支:

```
[jerry@CentOS project_repo]$ ls
branches
tags
trunk
[jerry@CentOS project_repo]$ svn copy trunk branches/jerry_branche
A         branches/jerry_branch
[jerry@CentOS project_repo]$ svn status
A  +    branches/jerry_branch
[jerry@CentOS project_repo]$ svn commit -m "Jerry's private branche"
Adding         branches/jerry_branch

Committed revision 12.
```

现在, Jerry 就可以安全地在自己的分支里工作了. 他在 array.c 里添加了排序操作, 修改后的 array.c 的内容是:

```
[jerry@CentOS jerry_branch]$ cat array.c
#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, 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");
    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;
}
```

简单的测试后, Jerry 准备提交修改:

```
[jerry@CentOS jerry_branch]$ svn status
M       array.c
[jerry@CentOS jerry_branch]$ svn commit -m "Added sort operation"
Sending        array.c
Transmitting file data .
Committed revision 13.
```

同时, 在主干上, Tom 为 array.c 定义了一个新函数, 用于实现二分查找:

```
[tom@CentOS trunk]$ svn diff
Index: array.c
===================================================================
--- array.c (revision 14)
+++ 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 +60,15 @@
    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;
 }
```

简单的测试后, Tom 提交了修改:

```
[tom@CentOS trunk]$ svn status
M       array.c
[tom@CentOS trunk]$ svn commit -m "Added search operation"
```

```
Sending        array.c
Transmitting file data .
Committed revision 15.
```

Tom 想知道 Jerry 提交了哪些修改:

```
[tom@CentOS trunk]$ cd ../branches/
[tom@CentOS trunk]$ svn up
Updating '.':
A    jerry_branch
A    jerry_branch/README
A    jerry_branch/array.c
Updated to revision 15.
[tom@CentOS trunk]$ svn log
------------------------------------------------------------------------
r13 | jerry | 2016-08-20 13:01:52 +0800 (Sat, 20 Aug 2016) | 1 line

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

从提交历史中可以看到, Jerry 实现了排序操作. Tom 添加的二分搜索操作只能应用在有序的数据上, 所以 Tom 决定把 Jerry 的分支合并到主干上:

```
[tom@CentOS trunk]$ pwd
/home/dell/Documents/svn/tom_project_repo/trunk
[tom@CentOS trunk]$ svn merge ../branches/jerry_branch
--- Merging r12 through r15 into '.':
U    array.c
--- Recording mergeinfo for merge of r12 through r15 into '.':
 U   .
[tom@CentOS trunk]$ svn diff
Index: array.c
===================================================================
--- array.c (revision 15)
+++ array.c (working copy)
@@ -23,6 +23,23 @@
    return -1;
 }

+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;
@@ -60,6 +77,10 @@
    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);

Index: .
===================================================================
--- .   (revision 15)
+++ .   (working copy)

Property changes on: .
___________________________________________________________________
Added: svn:mergeinfo
   Merged /branches/jerry_branch:r12-15
```

现在可以提交了:

```
[tom@CentOS trunk]$ svn commit -m "Merge changes from Jerrys's code"
Sending        .
Sending        array.c
Transmitting file data .
Committed revision 16.
```

A 基本准则

1. 每次提交前, 都要查看一下代码的变化

   在每次提交前, 都要用某种差异比较工具查看一下代码的变化.

2. 查看其他开发人员对代码作出的修改

   在开始每天的工作之前, 使用一款你最喜欢的差异比较工具, 查看其他开发人员在昨天对代码作了哪些修改. 我所知道的最优秀的程序员都把这当成了一种工作习惯.

   在阅读代码的变化时, 开发人员会得到两点好处:

   (a) 代码可能需要改善. 查看代码的变化就像是一次非正式的代码评审, 开发人员可能会发现其中的错误.

   (b) 学到一项新的技术. 同事可能用到了一项你所不知道的技术, 或者是对当前正在开发的项目有了更深的理解.

3. 让仓库尽量的小

   但不能更小.

   分布式版本控制系统要求每一位开发人员都有一份完整的仓库备份, 因此一个仓库中应该包含多少东西需要仔细考虑. 对一个大公司来说, 最不好的做法就是所有的项目都放在一个仓库中.

4. 按逻辑来组织提交

   提交到仓库中的每一次提交, 都应该只对应一个任务. “任务” 可以是修复了某个问题, 或者是添加了某项特性. 代码变化应该是完整的, 且只与该任务有关, 避免在一次提交中修复多个不相关的问题.

5. 完整地解释提交

   每一种版本控制工具都支持在向仓库提交修改时, 包含一段日志, 这种日志非常重要. 如果在每次提交时都能够写上一段完整的日志, 那么仓库不仅包含了代码的每一次修改, 还解释了为什么这次修改是必需的. 这些日志在日后查询时能够起到非常重要的作用.

   我建议开发人员在提交代码时, 尽可能详细地解释每一次修改, 不要写什么 “小修改” 这样的话, 而是要说出这次小修改是什么. 不仅仅要写 “修改问题 1234”, 还要描述问题 1234 具体是什么, 发生的原因, 以及如何修改.

6. 只记录标准文件

   人们有时候会问仓库中可以存放哪些文件, 答案是可以存放任何文件.

   虽然可以在仓库中存放任意文件, 但并不表示这样做就是对的. 最好的做法是只存放手工创建的文件, 我把它们称之为 “标准文件”.

   不要存放自动生成的文件, 比如 `*.exe` 和 `*.dll`. 如果开发人员用到了代码生成工具 (比如编译器), 那就存放输入文件, 而不是输出文件. 如果开发人员要生成几种不同格式的产品文档, 那就记录拥有原始格式的那份文档.

7. 不要破坏代码树

   如果仓库被破坏了, 那么工作副本所带来的优势也会消失殆尽. 在任何时候, 都应该保证仓库处在一种可以让整个团队继续工作的状态. 如果某个开发人员提交了一段无法编译或没有通过测试的代码, 那么整个团队的工作都会受到影响.

   许多开发团队都会对破坏代码树的开发人员进行社交性的惩罚. 这种惩罚并不会给人造成伤害, 仅仅是希望开发人员能够牢记教训. 例如, 要求犯错的人往玻璃罐中放入一美元 (等产品发布后, 就用罐子中的钱请大家看电影). 另一种方式是要求犯错的人为大家磨咖啡. 这些做法的目的都是为了让开发人员牢记教训, 而不是为了惩罚他们.

   总之, 仓库是所有开发人员共享代码的地方, 所以一定要谨慎对待提交到仓库中的每一行代码. 至少应该保证每一次修改都能在本地编译通过.

8. 使用标签

   标签非常廉价, 它们不会消耗太多的资源, 即使使用了大量的标签, 也不会影响版本控制工具的性能. 越多的标签并不会带来越大的责任, 所以开发人员可以尽情地使用它们.

9. 始终在提交之前审核合并

   无论版本控制工具能提供多大的帮助, 它始终比不上开发人员自己的大脑. 需要为合并负责的是开发人员, 而不是工具, 始终把工具当成一个工具来使用, 而不是顾问.

   版本控制工作做完自己能做的工作后, 接下来的事情就必须由开发人员自己来完成: 解决每一个冲突, 确保代码仍然可以编译通过, 执行单元测试, 审核代码的变化等.

   始终在工作副本中完成分支的合并. 在提交合并结果之前审核代码的变化.

10. 不要过多地注释代码

    使用版本控制工具时, 没必要为每一次修改而在代码中写注释, 因为代码的前一版本仍然保存在提交历史中, 所以在需要回溯时总能找到. 这个建议对网页开发人员来说尤其有用, 因为过多的注释会影响网页的加载速度.

11. 少加锁

    只有在必要的时候才加锁, 不要仅仅因为可能需要就对文件加锁; 不要对整个目录加锁, 只对需要的文件加锁; 在不需要锁时要马上释放锁.

12. 在每次提交后构建并测试代码

    安装一个自动构建与测试系统, 使得每当仓库中的代码有更新时, 就触发系统的构建与测试功能, 并将结果反馈给整个团队.

B Ubuntu 搭建 Subversion 服务器

本章介绍 Ubuntu 12.04 搭建 Subversion 服务器的过程[1].

1. 安装必要的软件包

```
~$ sudo apt-get install subversion apache2 libapache2-svn
   apache2-utils
```

2.

```
~$ sudo mkdir -p /svn/repos/
~$ sudo svnadmin create /svn/repos/testrepo
```

   /svn/repos/ 是存放所有仓库的目录, 当然, 你也可以用其他目录, 但是别忘了更新配置文件. 作为演示, 我们创建了一个仓库 testrepo.

3. 更新仓库的用户与用户组, 否则的话, Apache 就无法正常地访问它:

```
~$ sudo chown -R www-data:www-data /svn/repos/testrepo
```

4. 在 Apache 的配置文件目录中, 为 Subversion 创建一个配置文件, 文件的内容是:

```
~$ cat /etc/apache2/sites-available/svn.conf
LoadModule dav_module /usr/lib/apache2/modules/mod_dav.so
LoadModule dav_svn_module /usr/lib/apache2/modules/mod_dav_svn.so
LoadModule authz_svn_module /usr/lib/apache2/modules/mod_authz_svn.so
<Location /svn>
    DAV svn
    SVNParentPath /svn/repos/
    AuthType Basic
    AuthName "Test Repo"
    AuthUserFile /etc/svnpasswd
    Require valid-user
</Location>
~$
```

[1]本章是中文版新增的内容, 英文版中没有 — 译者注

5. 使能站点, 命令的参数是站点的配置文件名:

```
~$ sudo a2ensite svn.conf
Enabling site svn.
To activate the new configuration, you need to run:
  service apache2 reload
~$
```

6. 重启 Apache:

```
~$ sudo service apache2 reload
 * Restarting web server apache2
 AH00558: apache2: Could not reliably determine the server's fully
    qualified
 domain name, using 127.0.1.1. Set the 'ServerName' directive
    globally to
 suppress this message
                                                                 [ OK ]
~$
```

7. 创建 Subversion 的用户及其密码:

```
~$ sudo htpasswd -cm /etc/svnpasswd user1
New password:
Re-type new password:
Adding password for user user1
```

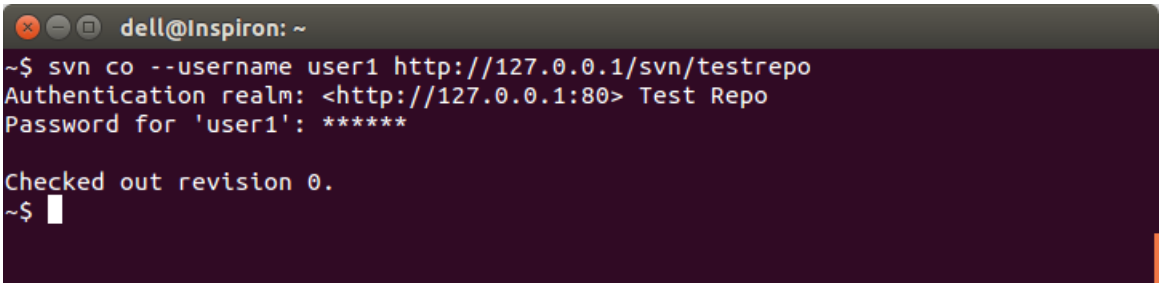
再创建一个用户:

```
~$ sudo htpasswd -m /etc/svnpasswd user2
sudo htpasswd -m /etc/svnpasswd user2
New password:
Re-type new password:
Adding password for user user2
```

在第一次执行 htpasswd 时, 才需要带上参数 -c.

现在, 就可以在网页上输入 http://127.0.0.1/svn/testrepo 来访问仓库:

或者用 Subversion 客户端命令检出工作副本:


dell@Inspiron: ~
~$ svn co --username user1 http://127.0.0.1/svn/testrepo
Authentication realm: <http://127.0.0.1:80> Test Repo
Password for 'user1': ******
Checked out revision 0.
~$

C 索引