清华 Linux 系列讲座之二：常用开发工具

4/7/2000, BluePoint Software, Eric Lee

一，diff / patch - 原代码维护基本命令

①diff 是生成源码补丁的必备工具。其命令格式为：

diff [命令行选项] 原始文件 新文件

常用命令行选项如下：

-r 递归处理目录　　-u 输出统一格式(unified format)

-N patch 里包含新文件　　-a patch 里可以包含二进制文件

它的输出在 stdout 上，所以你可能需要把它重定向到一个文件。diff 的输出有“传统格式”和“统一格式”之分，现在大都使用统一格式：

传统格式示例：

```
[hahalee@builder]$ diff a.txt b.txt
1a2
> here we insert a new line
3d3
< why not this third line?
```

统一格式示例：

```
[hahalee@builder]$ diff -u a.txt b.txt
--- a.txt      Thu Apr  6 15:58:34 2000
```

```
+++ b.txt      Thu Apr  6 15:57:53 2000
@@ -1,3 +1,3 @@
 This is line one
+here we insert a new line
 and this is line two
-why not this third line?
```

通过比较可以看出，传统格式的 patch 文件比较小，除了要删除/插入的行外没有冗余信息。统一格式则保存了上下文（缺省是上下各三行，最少需要两行），这样，patch 的时候可以允许行号不精确匹配的情况出现。另外，在 patch 文件的开头明确地用---和+++标示出原始文件和当前文件，也方便阅读。要选用统一格式，用 -u 开关。

通常，我们需要对整个软件包做修改，并生成一个 patch 文件，下面是典型的操作过程。这里就要用到前面介绍的几个命令行开关了：

```
tar xzvf software.tar.gz              # 展开原始软件包，其目录为 software
cp -a  software software-orig         # 做个修改前的备份
cd software
[修改，测试……]
cd ..
diff -ruNa software-orig software > software-my.patch
```

现在我们就可以保存 software-my.patch 做为这次修改的结果，至于原始软件包，可以不必保存。等到下次需要再修改的时候，可以用 patch 命令把这个补丁打进原始包，再继续工作。比如是在 linux kernel

上做的工作，就不必每次保存几十兆修改后的源码了。这是好处之一，好处之二是维护方便，由于 unified patch 格式有一定的模糊匹配能力，能减少原软件包升级带来的维护工作量(见后)

②patch 命令跟 diff 配合使用，把生成的补丁应用到现有代码上。常用命令行选项：

patch [命令行选项] [待 patch 的文件[patch]]

-pn　　patch level(n 是数字)　　-b[后缀] 生成备份，缺省是.orig

为了说明什么是 patch level，这里看一个 patch 文件的头标记。

```
diff -ruNa xc.orig/config/cf/Imake.cf xc.bsd/config/cf/Imake.cf
--- xc.orig/config/cf/Imake.cf  Fri Jul 30 12:45:47 1999
+++ xc.new/config/cf/Imake.cf   Fri Jan 21 13:48:44 2000
```

这个 patch 如果直接应用，它会去找 xc.orig/config/cf 目录下的 Imake.cf 文件，假如你的源码树的根目录是缺省的 xc 而不是 xc.orig，除了 mv xc xc.orig 之外，有无简单的方法应用此 patch 呢？patch level 就是为此而设：patch 会把目标路径名砍去开头 patch level 个节(由/分开的部分)。在本例中，可以用下述命令：cd xc; patch -p1 < /pathname/xxx.patch 完成操作。注意，由于没有指定 patch 文件，patch 程序默认从 stdin 读入，所以用了输入重定向。

如果 patch 成功，缺省是不建备份文件的(注：FreeBSD 下的 patch 工具缺省是保存备份)，如果你需要，可以加上 -b 开关。这样把修改前的文件以“原文件名.orig”的名字做备份。如果你喜欢其它后缀名，也可以用“-b 后缀”来指定。

如果 patch 失败，patch 会把成功的 patch 行给 patch 上，同时（无条件）生成备份文件和一个.rej 文件。.rej 文件里是没有成功提交的 patch 行，需要手工 patch 上去。这种情况在原码升级的时候有可能会发生。

关于二进制文件的说明:binary 文件可以原始方式存入 patch 文件。diff 可以生成(加-a 选项),patch 也可以识别。如果觉得这样的 patch 文件太难看，解决方法之一是用 uuencode 处理该 binary 文件。

二，RCS - 简单版本控制系统(Revision Control System)

单个文件的版本控制/管理，适合对少量文件进行版本控制，不适合小组进行项目协作开发。优点：使用简便；缺点：功能有限。RCS 常用命令有 ci/co/rcsdiff。

rcs 用一个后缀为“,v”的文件保存一文件的内容和所有修改的历史信息，你可以随时取出任意一个版本，用 rcs 保存程序就不必为不同版本分别备份。下面是一个“,v”文件的例子：

```
（太长，忽略。请看演示或自己找一个样本）
```

rcs 文件里记载了每次更新的日期、作者、还有更新说明（Log）。文件的最新版本内容放在 Log 之后，再后面是历次版本与其后一版本的差别，按 check in 的时间做倒序排列。这么做的原因是因为新版本的 check out 机会大些，倒序排列可优化 check out 时间。

ci - check in，保存新版本

此命令把指定文件添加到 rcs 历史文件中，同时把该文件删除。如果当前目录下有个 RCS 目录，ci 会把历史文件存在此处，否则放在当前目录下。

```
[hahalee@builder]$ mkdir RCS
[hahalee@builder]$ ci wood.txt
RCS/wood.txt,v  <--  wood.txt
enter description, terminated with single '.' or end of file:
NOTE: This is NOT the log message!
```

```
>> initial checkin                    #NOTE: 这里是给本次 checkin 做的说明
>> .
initial revision: 1.1
done
[hahalee@builder]$ ls -l RCS/
总共 4
-r--r--r--   1 hahalee  hahalee      451 Apr  7 07:27 wood.txt,v
```

ci 也有丰富的命令行选项，比如，可以指定 check in 的版本号，甚至可以用字符串来做版本号，请查阅 ci 的 manpage。

co - check out，取出当前（或任意）版本

常用命令行选项：

-r[rev]　指定版本的 checkout　　-l[rev]　指定版本，加锁的 checkout

如不加可选的版本号，缺省是最近的版本。如果只需要一份只读的拷贝，用-r（特殊情况，如需要一份只读的当前拷贝，可以不要任何选项）。如需要对 checkout 的文件进行修改，请用-l 选项。常见的操作流程是：

ci xxx.c; co -l xxx.c; 编辑，测试; ci xxx.c ……

在每次 checkin 的时候，版本号会自动在最低位上加 1。为了在文件中反映版本控制信息，rcs 提供了几个特殊的关键字，这里介绍Id和Log，其它的请参考 info cvs。

Id代表文件的版本标识，由文件名/版本号/最后一次 checkin 时间/最后一次 checkin 的用户这几项组成，比如：

```
$Id: wood.txt,v 1.3 2000/04/07 00:06:52 hahalee Exp $
```

如果需要更详细的信息，可以用Log，Log被扩展为该文件的所有修改日期和备注，例：

```
/* $Log: wood.txt,v $
 * Revision 1.2  2000/04/07 00:29:12  hahalee
 * This is my second checkin
 *
 * Revision 1.1  2000/04/07 00:28:39  hahalee
 * Initial revision
 * /
```

顺便介绍一下 ident 命令。它的功能比较简单，就是从文件中搜索出 RCS 标示符并打印出来。可以用 ident /usr/sbin/sendmail 来看看。不用说，如果想在最终的 binary 文件里找到Id串，得要把它声明到一个字符串里去。很多程序里这么写：

```
#ifndef   lint                                  //这里是为了避免 lint 报告“变量未使用”
static const char rcsid[] =
  "$Id: bin/sh.c,v 1.15 1999/08/27 23:13:43 wp Exp $";   //这是从 $Id$ 扩展出来的
#endif
```

rcsdiff – 比较 revision 之间的差异. 运行 diff 命令，生成 patch 文件

命令行格式：rcsdiff [选项] [-r 版本[-r 版本]] [diff 选项] 文件名

说明：如果没给出版本号，把上次 checkin 的内容同当前工作文件比较；如给出了一个版本号，就把那个版本的内容同当前工作文件比较；若给出了两个版本号，则用第一个来跟第二个比较。由于 rcsdiff

调用 diff 命令，所有的 diff 选项都可用。它的输出也是加了额外信息的 diff 格式内容，可以给 patch 使用。

rcs 里面还有 rcs，rcsclean，rlog，merge，rcsmerge 我们没有提到，有的特别简单有的特别繁琐且用得少。其中 rcs 命令可以用来对 rcs 文件进行各种精细的维护，最为复杂。

三，CVS - 并发版本管理系统（Concurrent Versions System）

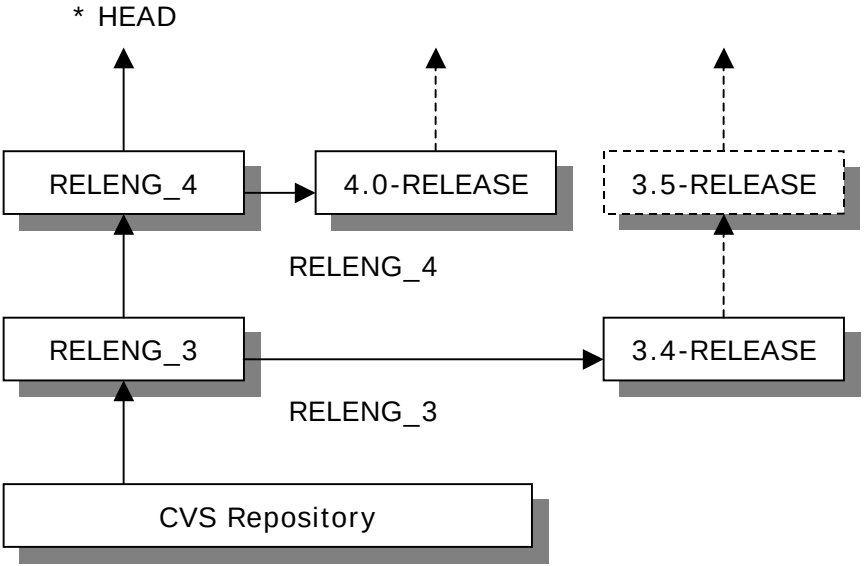
CVS，顾名思义，是个可以用在小组协作环境下的源码版本管理系统。同类的软件有 AT&T 的 SCCS（Source Code Control System），还有 PVCS 等。在 OpenSource 项目里 CVS 用得最为广泛，Linux kernel 不使用 CVS 来维护，下面我们将会参考 FreeBSD 的源码管理来做示例。CVS 是个相当复杂的系统，FreeBSD 甚至设有专门的“CVS 管理员”（CVS “Maester”）来全面负责项目的 CVS repository 维护。

下面介绍与 CVS 相关的若干概念和术语：

Repository	:	“源码仓库”，CVS 存放的项目源码历史档案
CVSROOT	:	该环境变量指明 CVS Repository 存放的目录
Module	:	模块。就是 CVSROOT 下的顶级目录名
Vendor Branch	:	分支。在一套 Repository 里可以存放多个代码分支的历史
Release Tag	:	发行标记。对于每一个版本，可以用符号来做标记

下面是一个 CVS repository 的版本衍生图，大致描绘了 FreeBSD 的版本发行情况。图中的 RELENG_3 和 RELENG_4 表示“Release Engine”，也就是 Vendor Branch，每个 Branch 分头发展，等某个 Branch 的开发到了一定的质量水准，就做个 Release Tag。比如最近的 4.0-RELEASE 的 Release Tag 是 REL_4_0。

这些不同的 Branch 都存放在同一个 Repository Tree 里。



FreeBSD 版本发布示意图（略图）

CVS 是个很复杂的系统，可以参考下面两个 URL 获得进一步的信息：

http://www.cyclic.com

http://www.loria.fr/~molli/cvs-index.html

（在 cvs 软件包里含有详细的文档，应当查阅 info 版本。几个 ps 文件都太老了）

下面介绍 CVS 的基本用法。

① Import 导入/创建 CVS Repository Tree

首先建一个目录作为你的 CVSROOT，然后用 cvs init 命令对其初始化（建立一系列 log，config 文件）。然后到工作目录下使用 cvs import 命令：

```
[hahalee@builder]$ mkdir /home/hahalee/CVS
[hahalee@builder]$ export CVSROOT=/home/hahalee/CVS
[hahalee@builder]$ cvs init
[hahalee@builder]$ cvs import -b 0.5.0 hftpd RELENG_0 REL_0
N hftpd/tar.h
N hftpd/auth.h
[blah...blah...blah...]
N hftpd/docs/rfcs/rfc0959.txt
N hftpd/docs/rfcs/rfc2428.txt
No conflicts created by this import
```

上述操作在$CVSROOT 下生成 hftpd 目录，可以看到里面都是后缀为“,v”的文件，这就是 import 进来的 Repository。RELENG_0 是 vendor-tag，REL_0 是 release-tag。vendor-tag 就是 vendor branch tag，

可以理解为"code name"。

② Checkout 创建私有工作目录/Export

换一个空目录，运行 cvs checkout modules_name 即可：

```
[hahalee@builder]$ cvs checkout hftpd          # hftpd是我们的module name
cvs checkout: Updating hftpd
U hftpd/AUTHORS
U hftpd/COPYING
[blah blah blah]                               # 省略许多
[hahalee@builder t]$ ls -l
总共 0
drwxrwxr-x   5 hahalee  hahalee     1253 Apr  7 20:08 hftpd
[hahalee@builder t]$ find ./ -type d
```

从最后一条命令的输出可看到，checkout 的工作目录里多了 CVS 目录。里面记载了 CVS 相关的信息，可以方便后续的 cvs 操作。如果纯粹是为了拷贝出最新的 source tree，可以用 export，此时不会建立 CVS 目录。

③ Update 更新

当你完成某一部分代码的时候，先不忙提交，可以把别人可能做了的其他修改 update 过来然后统一编译调试无误后再提交，这是 team work 的准则。在 checkout 出来的工作目录下（不管什么子目录），直接 cvsup update 就可以了，当然你要先把 CVSROOT 环境变量设置好。

④ Commit 提交

很简单，cvs commit。但你必须要在 checkout 出来的工作目录里提交才行:

```
[hahalee@builder]$ cvs commit
cvs commit: Examining .
cvs commit: Examining docs
cvs commit: Examining docs/man
cvs commit: Examining docs/rfcs
cvs commit: Examining tools
Checking in AUTHORS;
/home/hahalee/CVS/hftpd/AUTHORS,v  <--  AUTHORS
new revision: 0.6; previous revision: 0.5
done
```

关于并发提交冲突：任何用户可以随意 checkout 他们自己的工作拷贝，commit 也是不受限制的。这样，当用户 a 和 b 分别 checkout 了 1.2 版的 c.c，然后各自对 c.c 做了修改，a 提交了他的修改，然后，当 b 提交的时候，冲突就产生了。

这时候，cvs 会做以下动作：

1， 告诉用户 b，对 c.c 的提交发生冲突

2， 对用户 b 当前的 c.c 做备份文件.#c.c.1.2

3， 试图合并 a 和 b 的修改，生成新的 c.c

然后，用户 b 应当修改 c.c，去掉/合并冲突的行，并以版本 1.4 提交。

⑤ Diff

可以用类似 rcsdiff 的方法用 cvs 生成 patch，命令行语法也类似

```
[hahalee@builder]$ cvs diff -u -r0.5 AUTHORS
Index: AUTHORS
===================================================================
RCS file: /home/hahalee/CVS/hftpd/AUTHORS,v
retrieving revision 0.5
retrieving revision 0.6
diff -u -r0.5 -r0.6
--- AUTHORS    2000/04/07 10:46:02    0.5
+++ AUTHORS    2000/04/07 14:05:57    0.6
@@ -1,3 +1,4 @@
+ah! let me in!
                  So then, who can't spell
                  Devellopppopotamus?
                 Quite a lot of us.
```

还有一个 rdiff，用来生成两个不同的 release 之间的 patch。

⑥ 其他操作

cvs 的其他操作还包括有：

admin　　管理功能

tag　　对某一版本做符号标记

release　　取消 checkout,删除工作目录（release 在这里是“释放”的意思）

add,remove　　往 repository 里添加/删除文件

history　　查看 repository 操作历史记录

⑦ CVS 的多平台特性以及 C/S 扩展

cvs 是多平台的，开发可以在多种平台比如，可以把 linux 上的 CVS Repository 通过 samba export 出来在 Windows 平台上做开发。现在很多软件包里包含有*NIX/Windows/MacOS 等多平台支持代码，cvs 的跨平台特性可提供最好的多平台开发支持。

不过，cvs 的操作是直接基于文件系统的，在需要大量远程协作的场合问题很多，远程的 NFS mount 效率太差，也会有安全问题。新版本的 cvs 自身内建了 Client/Server 支持，也可以利用 Unix 上传统的远程交互手段来通讯。

1，通过 rsh（也可用 ssh 替换）

2，使用 cvs 自带的 C/S 用户认证：pserver（缺省端口 2401）

3，使用 kerberos 的 gserver、kserver

四，RPM 包管理器

(概要,sorry for short of time)

一，RPM 的基本用法

1,安装(-i)，卸载(-e)，查询(-q)，检查(-V,--checksig)

2,生成 rpm 包

二，SPEC 文件的格式，及 build 的步骤

三，讨论：好的包管理器的若干要点

1,基本功能

2,多种介质支持

3,要有 sub package 管理

4,UI 接口/支持