第8章 代码管理

当项目涉及多个人的时候，工作会更加棘手。所有事情的速度都将减慢，也将更难。这可以归结于多个原因，本章将揭示这些原因，并尝试提供与之对抗的一些方法。

本章由两个部分组成，分别介绍：

- 如何使用版本控制系统；
- 如何配置持续集成机制。

首先，代码基准库有了非常大的发展，因此跟踪所有修改就显得更加重要，尤其是涉及许多开发人员的时候。这就是版本控制系统所担当的角色。

其次，几个没有直接相连的大脑为相同的项目工作。他们各有不同的角色，从事不同领域的工作。因此，缺乏全局可见性将产生许多与事情的进展和其他人所完成工作等方面的混乱。这是不可避免的，必须通过一些工具来提供持续的可见性，并减轻这些问题。这可以通过配置一系列用于持续集成（continuous integration）的工具来实现。

现在，我们将详细地讨论这两个领域。

8.1 版本控制系统

版本控制系统（VCS）提供了一种共享、同步和备份任何文件的方法。它们分为两类：

- 集中式系统；
- 分布式系统。

8.1.1 集中式系统

集中式版本控制系统基于单个服务器，这个服务器保存文件并且让客户记录和检查对这

些文件所做的修改。它的原理很简单：每个人都可以获得系统中文件的副本，并且在这份副本上工作。因此，每个用户可以向服务器提交（commit）他们的修改。这些修改将被应用并且提升修订(revision)号。其他用户可以通过一个更新(update)来同步他们的仓库(repository)副本，以获得这些修改。

仓库基于所有的提交来演化，系统将所有的修订归档到一个数据库，用以撤销任何修改或者提供所完成修改的信息，如图 8.1 所示。

在这个集中式配置中，每个用户负责同步自己的本地仓库和主仓库，以便获得其他用户所做的修改。这意味着，当一个本地修改过的文件被其他人修改并检入时，就会发生一些冲突。在这种情况下，将执行一个用户系统上的冲突解决机制，如图 8.2 所示。

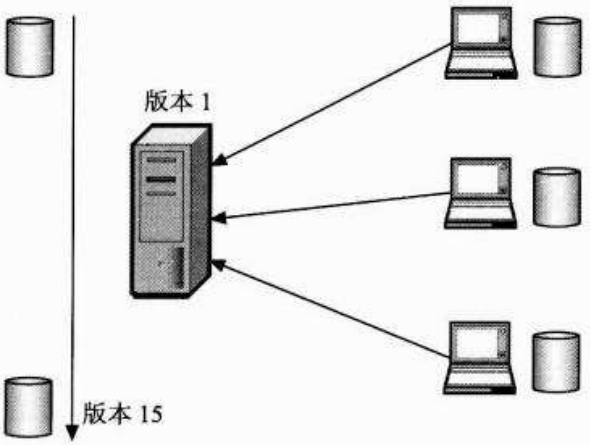

图 8.1

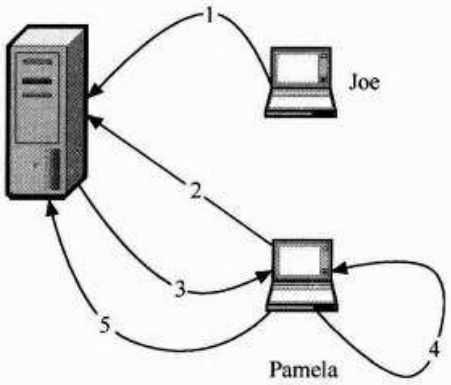

图 8.2

（1）Joe 检入一个修改。

（2）Pamela 试图检入对相同文件的一个修改。

（3）服务器提示她的文件已经过时。

（4）Pamela 更新本地副本。版本控制软件可能（也可能不）可以无缝地合并这两个版本（也就是说，没有冲突）。

（5）Pamela 提交新的版本，它将包含 Joe 和她自己所做的修改。

这个过程在涉及少数开发人员和少量文件的情况下能够很好地运作，但是对于较大的项目而言仍然有问题。例如，一个复杂的修改将涉及许多文件，这种方法很耗费时间，而且在整个工作完成之前一直在本地保持所有文件也是不可行的。

- 这很危险，因为用户可能在他们的计算机上保持没有必要备份的修改。
- 这样，在其检入之前是很难与其他人共享的；如果在完成之前共享则会使仓库处于不稳定状态，这样其他用户将不希望共享。

集中式的 VCS 通过提供“分支”和“合并”来解决这个问题。从主流的修订中找出独立的分支，工作完成之后将其交回到主流中，这样做是有可能的。

第8章 代码管理

当项目涉及多个人的时候，工作会更加棘手。所有事情的速度都将减慢，也将更难。这可以归结于多个原因，本章将揭示这些原因，并尝试提供与之对抗的一些方法。

本章由两个部分组成，分别介绍：

- 如何使用版本控制系统；
- 如何配置持续集成机制。

首先，代码基准库有了非常大的发展，因此跟踪所有修改就显得更加重要，尤其是涉及许多开发人员的时候。这就是版本控制系统所担当的角色。

其次，几个没有直接相连的大脑为相同的项目工作。他们各有不同的角色，从事不同领域的工作。因此，缺乏全局可见性将产生许多与事情的进展和其他人所完成工作等方面的混乱。这是不可避免的，必须通过一些工具来提供持续的可见性，并减轻这些问题。这可以通过配置一系列用于持续集成（continuous integration）的工具来实现。

现在，我们将详细地讨论这两个领域。

8.1 版本控制系统

版本控制系统（VCS）提供了一种共享、同步和备份任何文件的方法。它们分为两类：

- 集中式系统；
- 分布式系统。

8.1.1 集中式系统

集中式版本控制系统基于单个服务器，这个服务器保存文件并且让客户记录和检查对这

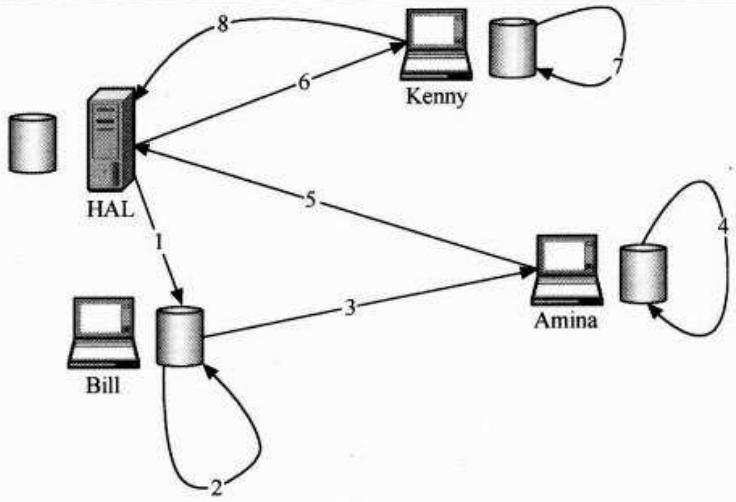

图 8.4

（8）Kenny 照常将修改推送给 HAL。

这里关键的概念是人们推送和拉取其他仓库中的文件，这种行为将根据人们工作和项目管理的方式而变化。因为没有主仓库，所以项目的维护者需要定义人们推和拉取程序修改的策略。

而且，人们在使用多个仓库时必须更加聪明一些。因为修订号对于每个仓库而言都是局部的，没有所有人都可以参考的全局修订号。因此，必须使用标记（tag）来辨别。标记是可以附加到修订中的标签。最后，用户负责备份自己的仓库，这不是集中式架构中管理员负责配置备份策略的方式。

分布式策略

如果供职于每个人都为同一目标工作的公司，那么 DVCS 仍希望拥有集中式服务器。

对此可以应用不同的方法。最简单的方法是设置一个和常规的集中式服务器相似的服务器，每个项目成员都可以向公共流中推送修改。但是这个方法有点过于简单，它无法利用分布式系统的优点，因为人们将会以与集中式系统相同的方法使用推送和拉取命令。

另一种方法是在一个服务器上提供多个不同访问级别的仓库：

- 一个不稳定（unstable）仓库，大家都可以推送修改；
- 一个稳定（stable）仓库，发行管理员之外的所有成员都是只读的，发行管理员有权限从不稳定知识库拉取修改并决定所应合并的内容；
- 各种不同的只读的、对应于发行版本的发行仓库，本章稍后将会介绍。

这允许人们提交修改，管理员可以在它们进入稳定仓库之前进行审阅。

因为 DVCS 提供了无限的组合方式，所以还可以建立其他的策略。例如，Linux Kernel 使用了基于星型模式的 Git（http://git.or.cz），Linus Torvalds 维护官方仓库，并从他所信任的一组开发人员那里拉取修改。在这种模式下，希望修改核心的人尝试将这些修改推送给受信

任的开发人员，这样他们有希望通过这些开发人员到达 Linus 那里。

8.1.3 集中还是分布

要在集中式和分布式方法之间做出选择，更多取决于项目的特性和团队的工作方式。

例如，一个由独立团队开发的应用程序就不需要分布式系统所提供的特性。一切都在开发服务器的控制之下，管理员不需处理外部的代码贡献者，也不用担心开发人员工作的备份。开发人员在需要时可以创建分支，然后尽快回到主干。他们在需要合并修改和离开 Internet 连接时都可能会碰到困难，但是对这样的系统所提供的功能仍然很享受。分支和合并在这种环境下不经常发生。

这就是大部分公司不涉及更广泛的代码贡献者社区的原因。它们自己的雇员大量使用集中的版本控制系统，每个人都在一起工作。

对于拥有大量代码贡献者的项目，集中式方法就显得有些呆板，使用 DVCS 则更有意义。现在有许多开源项目都选用了这种模式。例如，目前正在讨论为 Python 采用一个 DVCS，并且可能很快会发生，因为主要的问题在于建立一组良好的做法，并且培训开发人员使用这种新的代码处理方式。

本书中将使用 DVCS 并解释它在项目管理中的使用，以及一组良好的做法，选择的软件是 Mercurial。

8.1.4 Mercurial

Mercurial（http://www.selenic.com/mercurial/wiki）是用 Python 编写的一个 DVCS，提供了简单但强有力的命令行实用程序来处理代码。

最简单的安装方式是调用 easy_install，命令如下。

```
$ easy_install mercurial
```

在某些 Windows 版本下，在 Python Scripts 目录下生成的脚本是错误的，hg 在提示符下也不可用。这种情况下，可以把它改名为 hg.py，并在提示符下运行 hg.py。

如果仍然遇到问题，可以使用一个特殊的二进制安装程序（参见 http://mercurial.berkwood.com）。

如果在 Debian 或 Ubuntu 这样的系统下，也可以使用它提供的包管理系统，命令如下。

```
$ apt-get install mercurial
```

这样，在提示符下就能够调用 hg 脚本，它拥有一组完备的选项（在此有删节），如下所示。

```
$ hg -h
Mercurial Distributed SCM
list of commands:
    add          add the specified files on the next commit
    clone        make a copy of an existing repository
    commit       commit the specified files
    copy         mark files as copied for the next commit
    diff         diff repository (or selected files)
    incoming     show new changesets found in source
    init         create a new repository in the given directory
    pull         pull changes from the specified source
    push         push changes to the specified destination
    status       show changed files in the working directory
    update        update working directory
use "hg -v help" to show aliases and global options
```

在要创建仓库的文件夹中调用 init 命令，就可以创建出所需的仓库，如下所示。

```
$ cd /tmp/
$ mkdir repo
$ hg init repo
```

现在，可以使用 add 命令来将文件添加到仓库中，如下所示。

```
$ cd repo/
$ touch file.txt
$ hg add file.txt
```

这个文件直到调用 commit（ci）命令时才被检入，如下所示。

```
$ hg status
A file.txt
$ hg commit -m "added file.txt"
No username found, using 'tziade@macziade' instead
$ hg log
changeset:  0:d557683c40bc
tag:            tip
```

```
user:          tziade@macziade
date:          Tue Apr 01 17:56:41 2008 +0200
summary:       added file.txt
```

仓库在其目录中是自包含的，可以使用 clone 命令将其复制到另一个目录，如下所示。

```
$ hg clone . /tmp/repo2
1 files updated, 0 files merged, 0 files removed, 0 files unresolved
```

也可以通过 SSH 在另一个机器上完成这一任务，只要它安装了 SSH 服务器和 Mercurial，如下所示。

```
$ hg clone . ssh://tarek@ziade.org/repo
searching for changes
remote: adding changesets
remote: adding manifests
remote: adding file changes
remote: added 1 changeset with 1 change to 1 files
```

远程仓库可以使用 push 命令来同步，如下所示。

```
$ echo more >> file.txt
$ hg diff
diff -r d557683c40bc file.txt
--- a/file.txt Tue Apr 01 17:56:41 2008 +0200
+++ b/file.txt Tue Apr 01 19:32:59 2008 +0200
@@ -0,0 +1,1 @@
+more
$ hg commit -m 'changing a file'
No username found, using 'tziade@macziade' instead
$ hg push ssh://tarek@ziade.org/repo
pushing to ssh://tarek@ziade.org/repo
searching for changes
remote: adding changesets
remote: adding manifests
remote: adding file changes
remote: added 1 changesets with 1 changes to 1 files
```

diff 命令（di）在这里用来显示所做的修改。

hg 命令提供的另一个很好的特性是 serve，它为当前仓库提供一个小的 Web 服务器：

```
$ hg serve
```

现在，可以在浏览器中访问http://localhost:8000，将会得到一个仓库的视图，如图 8.5 所示。

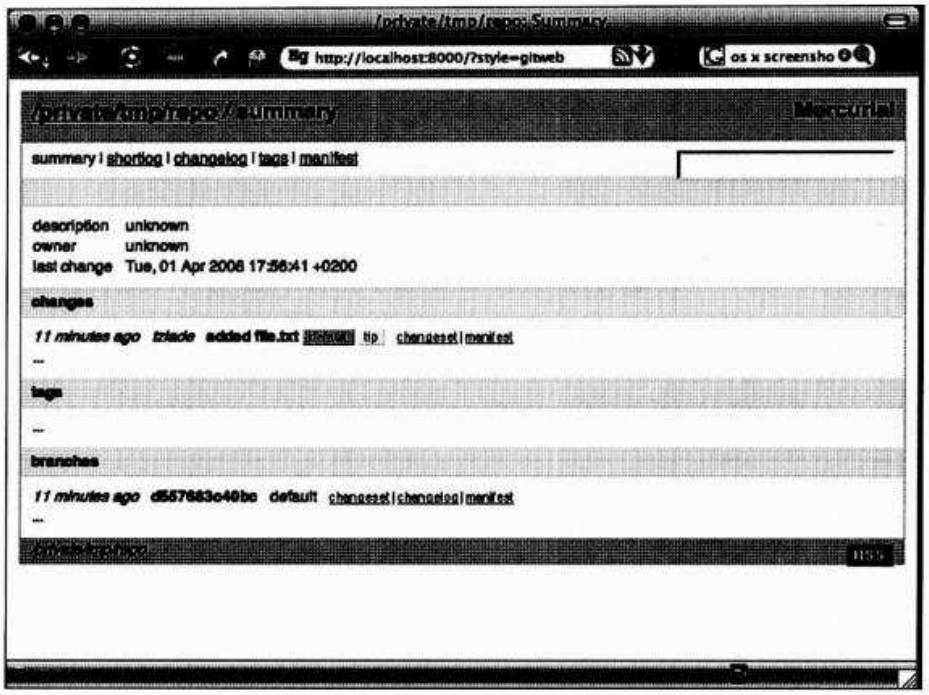

图 8.5

在这个视图之外，hg serve 还能够调用 clone 和 pull 命令，如下所示。

```
$ hg clone http://localhost:8000 copy
requesting all changes
adding changesets
adding manifests
adding file changes
added 1 changesets with 1 changes to 1 files
1 files updated, 0 files merged, 0 files removed, 0 files unresolved
$ cd copy/
$ ls
file.txt
$ touch file2.txt
$ hg add file2.txt
$ hg ci -m "new file"
No username found, using 'tziade@macziade' instead
```

clone 命令可以复制一个仓库来开始工作。

hg serve 不允许人们推送修改，因为这需要建立一个真正的 Web 服务器来处理验证，正如下一小节中所要看到的那样。但是在某些情况下，如果希望暂时共享一个仓库来让别人拉

取修改时，这可能是有用的。

如果想更深入地研究 Mercuria，可以参考 http://hgbook.red-bean.com 上的联机书籍。

8.1.5 使用 Mercurial 进行项目管理

使用 Mercurial 管理仓库的最简单方法是使用其提供的 hgwebdir.cgi 脚本。这是一个 CGI（公共网关接口）脚本，可以用来通过 Web 服务器发布一个仓库，并提供和 hg serve 相同的功能。而且，通过配置一个 mima 文件限制命令的使用，它允许 push 命令以安全的方式运行。

CGI 很健壮而且配置很简单，但是它并非发布一个仓库的最快方式。还有一些基于 fastcgi 或 mod_wsgi 的解决方案。

配置这样一个系统并不困难，但是可能取决于平台相关的部件，所以无法提供一个通用的安装教程。本小节将关注于在 Linux Debian Sarge 和 Apache 2 平台上的配置方法，这是相当常见的配置。

安装这样一个服务器的步骤如下：

- 建立一个专用文件夹；
- 配置 hgwebdir；
- 安装 Apache；
- 设置权限。

1．建立一个专用文件夹

先前描述的多仓库方法用 Mercurial 来配置很简单，因为一个仓库对应于一个系统文件夹。可以建立一个 repositories 文件夹来保存所有的知识库，这个文件夹位于项目专用的文件夹中。这个项目文件夹位于主文件夹中。这里可以使用 mercurial 用户。

为 Atomisator 创建一个 Mercurial 环境，如下所示。

```
$ sudo adduser mercurial
$ sudo su mercurial
$ cd
```

```
$ mkdir atomisator
$ mkdir atomisator/repositories
$ cd atomisator
```

由此，可以使用 hg 创建 stable 和 unstable 仓库，如下所示。

```
$ hg init repositories/stable
$ hg init repositories/unstable
$ ls repositories/
unstable stable
```

一些团队不使用分离的仓库，但是在一个单一的知识库上，他们会使用一个命名的分支来区分稳定版本和开发版本，并且进行一些必需的合并。具体请参见 http://www.selenic.com/mercurial/wiki/index.cgi/Branch。

创建发行版本时，可以通过复制一个 stable 来添加新的仓库。例如，如果发行了 0.1 版本，可以如下这样完成：

```
$ hg clone repositories/stable repositories/release-0.1
```

将前一章中创建的 buildout 和 packages 文件夹复制到 unstable 文件夹中，并且检入它们，以在不稳定仓库中添加 Atomisator 代码。完成这一步之后，unstable 文件夹的内容看上去类似于：

```
$ ls repositories/unstable
buildout packages
```

下一章将介绍与发行相关的知识。

2．配置 hgwebdir

为了提供这些仓库，必须将 hgwebdir.cgi 文件添加到 atomisator 文件夹中。这个脚本将随同安装一起提供。如果找不到，可以在 Mercurial 网站上下载一个源代码分发版本。但是要确定下载的文件和安装的版本严格对应。

```
$ hg --version
Mercurial Distributed SCM (version 0.9.4)
$ locate hgwebdir.cgi
```

```
/usr/share/doc/mercurial/examples/hgwebdir.cgi
$ cp /usr/share/doc/mercurial/examples/hgwebdir.cgi .
```

这个脚本处理一个名为 hgweb.config 的配置文件，它包含仓库文件夹的路径，如下所示。

```
[collections]
repositories/ = repositories/
[web]
style = gitweb
push_ssl = false
```

collections 小节提供指定包含多个仓库的文件夹的通用方法，该脚本将循环访问这些仓库。Web 小节可用来设置一些选项。在本例中，可以设置其中两个：

- style　用来设置网页的外观和风格，gitweb 可能是最好的默认值。注意，Mercurial 使用模板来显示所有页面，而这些模板都可以配置；
- push_ssl　如果设置为 true（默认值），那么用户将不能通过 HTTP 使用推送命令。

3．配置 Apache

下一步要配置的是执行 CGI 脚本的 Web 服务器层。最简单的方法是在 atomisator 文件夹中提供一个配置文件，定义一条 Directory、一条 ScriptAliasMatch 和一条 AddHandler 指令。

添加一个包含以上内容的 apache.conf 文件，如下所示。

```
AddHandler cgi-script .cgi
ScriptAliasMatch        ^/hg(.*)            /home/mercurial/atomisator/
hgwebdir.cgi$1
<Directory /home/mercurial/atomisator>
  Options ExecCGI FollowSymLinks
  AllowOverride None
  AuthType Basic
  AuthName "Mercurial"
  AuthUserFile /home/mercurial/atomisator/passwords
  <LimitExcept GET>
    Require valid-user
  </LimitExcept>
</Directory>
```

注意：

- AddHandler 指令在某些发行版本上可能是不必要的，但在 Debian Sarge 中必须有；

- ScriptAliasMatch 需要启用 mod_alias；
- 当遇到 POST 指令时，意味着用户将向服务器发送数据，需要使用一个密码文件完成一次验证。

如果对 Apache 不熟悉，请参见http://httpd.apache.org/docs。

密码文件由实用程序 htpasswd 生成，并被放在 atomisator 文件夹中，如下所示。

```
$ htpasswd -c passwords tarek
New password:
Re-type new password:
Adding password for user tarek
$ htpasswd passwords rob
New password:
Re-type new password:
Adding password for user rob
```

在 Windows 平台中，如果 htpasswd 在命令提示符下不可用，那么可能需要在 PATH 中手工添加位置。

每当需要添加一位有权限将修改推送到仓库的用户时，都可以用 htpasswd 升级这个文件。

最后，为了使该脚本有权限执行以及确保数据可用于 Apache 进程所使用的用户组，还需要以下几个步骤。

```
sudo chmod +x /home/mercurial/atomisator/hgwebdir.cgi
sudo chown -R mercurial:www-data /home/mercurial/atomisator
sudo chmod -R g+w /home/mercurial/atomisator
```

为了与配置挂钩，可以将这个文件添加到 Apache 所访问的 site-enabled 目录中，如下所示。

```
$ sudo ln -s /home/mercurial/atomisator/apache.conf /etc/apache2/sitesenabled/
007-atomisator
$ sudo apache2ctl restart
```

当 Apache 启动之后，可以通过http://localhost/hg访问该页面（如图 8.6 所示）。

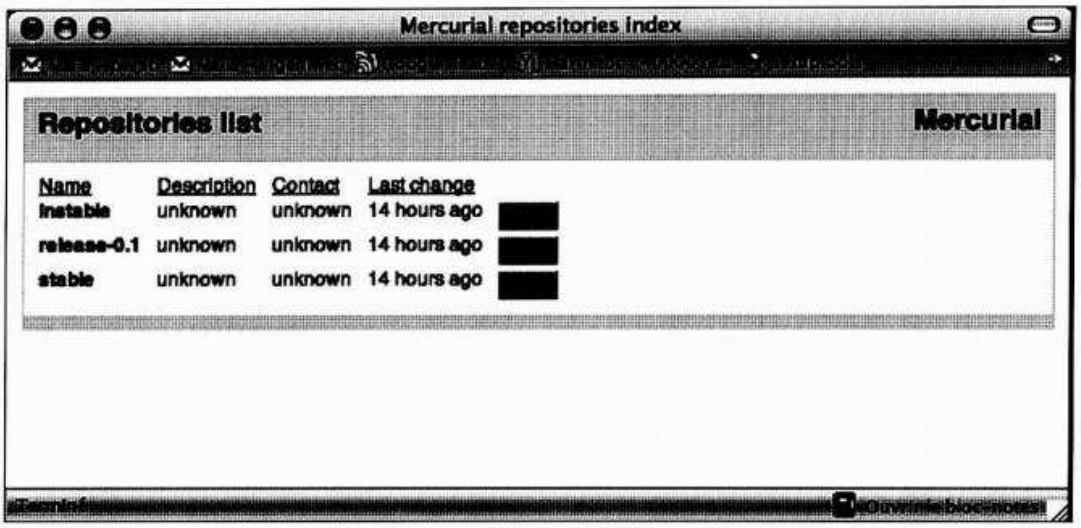

图 8.6

注意，每个仓库都有一个 RSS feed，人们可以通过来它来跟踪修改。每当有人推送一个文件时，就有在 RSS feed 中添加一个新的条目，以及一个指向修改日志的链接。这个修改日志将以一个不同的视图来显示更详细的日志。

如果需要将 Mercurial 仓库作为一个虚拟主机，将需要添加一个特殊的改写规则，这个规则为 hgwebdir 所用的静态文件（诸如样式单）提供服务。

以下是 Web 上用于发行本书的仓库（包含实例的源代码）的 apache.conf 文件，对应于 http://hg-atomisator.ziade.org/。

```
<VirtualHost *:80>
    ServerName hg-atomisator.ziade.org
    CustomLog /home/mercurial/atomisator/access_log combined
    ErrorLog /home/mercurial/atomisator/error.org.log
    AddHandler cgi-script .cgi
    RewriteEngine On
    DocumentRoot /home/mercurial/atomisator
    ScriptAliasMatch ^/(.*) /home/mercurial/atomisator/hgwebdir.cgi/$1
    <Directory /home/mercurial/atomisator>
     Options ExecCGI FollowSymLinks
     AllowOverride None
     AuthType Basic
     AuthName "Mercurial"
     AuthUserFile /home/mercurial/atomisator/passwords
```

```
        <LimitExcept GET>
          Require valid-user
        </LimitExcept>
      </Directory>
    </VirtualHost>
```

每个仓库都可以从这个首页链接到。为了让所有页面都使用相同的样式，每个仓库的.hg 配置目录都将添加一个 hgrc 文件。这个文件能够定义和主 CGI 文件所使用的类似的 Web 小节，如下所示。

```
$ more repositories/stable/.hg/hgrc
[web]
   style = gitweb
   description = Stable branch
   contact = Tarek <tarek@ziade.org>
```

description 和 contact 字段也将用于该网页。

4．设置权限

我们已经看到一个用来过滤允许推送的用户的全局访问文件。这是第一级的权限，因为需要为每个仓库定义 push 策略。先前所定义的策略是：

- 所有注册的开发人员都允许将修改推送到不稳定（unstable）仓库；
- 除了发行管理员之外，其他用户对稳定（stable）知识库有只读权限。

这可以在每个仓库的 hgrc 文件的 allow_push 参数中进行设置。如果用户 tarek 是发行管理员，那么 stable hgrc 文件的内容将类似于：

```
$ more repositories/stable/.hg/hgrc
[web]
style = gitweb
description = Stable branch
contact = Tarek <tarek@ziade.org>
push_ssl = false
allow_push = tarek
```

注意，这里添加的 push_ssl 是用于通过 HTTP 推送的。不稳定（unstable）仓库的 hgrc 文件类似于：

```
$ more repositories/unstable/.hg/hgrc
[web]
```

```
style = gitweb
description = Unstable branch
contact = Tarek <tarek@ziade.org>
push_ssl = false
allow_push = *
```

这意味着，只要将他们添加到密码文件中，这个仓库允许所有人执行推送操作。

本书中为了简单起见没有设置 SSL 配置，但是在实际服务器中，为了使事务更安全，应该使用它。例如，在我们的配置中，是允许 HTTP 嗅探的。

5. 设置客户端

为了避免鉴权提示，以及在提交日志时提供一个易于理解的名称，在客户端的 HOME 目录中可以添加一个.hgrc 文件，如下所示。

```
[ui]
username = Tarek Ziade
[paths]
default = http://tarek:secret@atomisator.ziade.org/hg/unstable
unstable = http://tarek:secret@atomisator .ziade.org/hg/unstable
stable = http://tarek:secret@atomisator.ziade.org/hg/stable
```

ui 小节为服务器提供一个提交者的全名，paths 小节则是一个仓库的 URL 列表。注意，在 URL 中放置用户名和密码，以避免在每次推送完成时出现密码提示。这种做法并不安全，有密码提示会更安全些。最安全的方式是通过 SSH 协议而不是 Web 服务器来工作。

使用这个文件，推送可以这样完成，如下所示。

```
$ hg push # will push to the default repository (unstable)
$ hg push stable # will push to stable
$ hg push unstable # will push to unstable
```

如果需要在其他平台上安装，安装步骤不会有很大的不同。针对特定的平台，http://www.selenic.com/mercurial/wiki/index.cgi/HgWebDirStepByStep 上的内容将会有所帮助。

8.2 持续集成

对于持续集成而言，建立一个仓库只是第一步，持续集成是从极限编程（XP）中发展而来的一组软件方法。在 Wikipedia 上对其原理有清晰的描述（http://en.wikipedia.org/wiki/Continuous_integration#The_Practices），并且定义了确保软件易于构建、测试和交付的方法。

下面总结一下在使用 zc.buildout 和 Mercurial 的、基于 egg 的应用程序环境中的这些方法。

- 维护一个代码仓库　这由 Mercurial 完成。
- 自动化构建　zc.buildout 可以满足这个需求，正如在前一章中所看到的那样。
- 使所构建的成为自测试系统　zc.buildout 提供了启动整个软件测试活动的一种方法。
- 每个人每天提交新修改　Mercurial 为开发人员提供经常修改的工具，但是这更多需要依赖开发人员的行为，只要不破坏构建版本，应该尽可能地经常提交。
- 每个提交都应该被编译　每当对程序做了修改时，软件应该再次编译并运行所有测试，以确认没有导致退化。如果发生了这样的问题，应该向开发人员发出一封警告邮件（本章中尚未介绍）。
- 保持快速的编译　这对于 Python 应用程序而言不是一个真正的问题，因为大部分时候都不需要编译的步骤。在任何情况下，连续编译两次软件时，第二遍应该会更快一些。
- 在一个复制生产环境的临时环境中测试　能在所有生产环境中测试软件是很重要的，本章中尚未介绍这方面的内容。
- 简化获得最新可分发版本的过程　zc.buildout 提供了简单的方法，能够将可分发版本捆绑到档案文件中。
- 每个人都可以了解最新构建的结果　系统应该提供关于构建的反馈信息，本章中尚未介绍这方面的内容。

使用这些方法，能在早期发现问题，甚至是那些与代码或与目标平台相关的问题，从而提升代码质量。

而且，编译和重新执行测试的自动化系统简化了开发人员的工作，因为他们不需要再次启动整个测试。

最后，这些规则将使开发人员对他们提交的修改更负责任。检入有破坏性的代码将产生大家都能看到的反馈。

在我们配置的环境中，还没有涉及的部分只有：

- 在每次提交时编译该系统；
- 在目标系统上编译该系统；
- 提供关于最新构建版本的反馈。

这可以使用自动化构建软件 Buildbot 来实现。

Buildbot

Buildbot（http://buildbot.net/trac）是用 Python 编写的软件，它自动化了任何软件项目的编译和测试周期。它可以配置成每当源代码仓库变化时就生成某些构建版本，并启动某些测试，然后提供一些反馈，如图 8.7 所示。

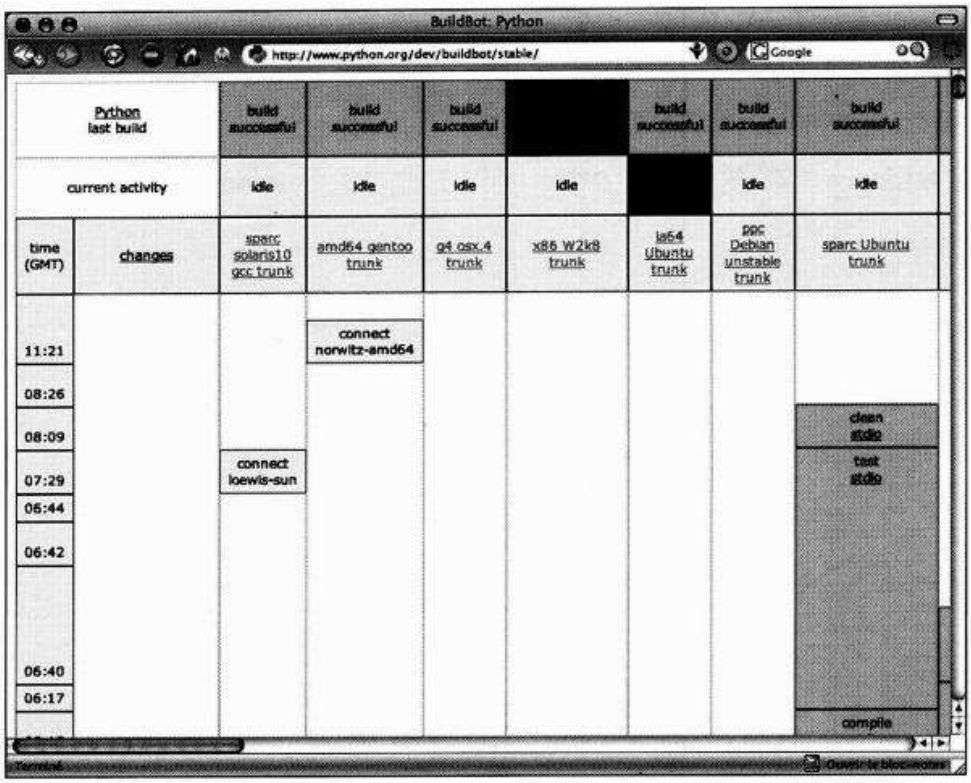

图 8.7

例如，Python 就在核心开发中使用了这个工具，参见 http://www.python.org/dev/buildbot/stable/（别忘了在最后要加上“/”）。

每一列对应一个由步骤(step)组成的一个构建(build)，并与一些构建客户端(build slaves)相关联，整个系统由构建服务器端（build master）驱动。

- 构建服务器端集中并且驱动所有工作。
- 一个构建（build）是用于构建应用程序并且运行测试的步骤序列。
- 步骤（step）是一个原子命令，例如：检查项目文件；构建应用程序；运行测试。

构建客户端是负责执行一个构建的机器，它可以位于任何地方，只要能够访问构建服务器。

1. 安装 Buildbot

要安装 Buildbot，需要安装一系列它所需的软件，并创建一个配置 Buildbot 的 Python 脚

本。可在http://buildbot.net/trac/wiki/UserManual上的联机用户手册中找到相关描述。

另一个选项是使用 collective.buildbot 项目，它提供了一个基于 zc.buildout 的配置工具。换句话说，它可以在一个配置文件中定义 Buildbot，而不需要考虑所有必需软件的安装和 Python 脚本的编写。

在自己的服务器环境中创建这样一个 buildout，此外还有在专用文件夹中的仓库，如下所示。

```
$ cd /home/mercurial/atomisator
$ mkdir buildbot
$ cd buildbot
$ wget http://ziade.org/bootstrap.py
```

然后，在 buildbot 文件夹中添加一个包含以下内容的 buildout.cfg 文件。

```
[buildout]
parts =
    buildmaster
    linux
    atomisator
[buildmaster]
recipe = collective.buildbot:master
project-name = Atomisator project buildbot
project-url = http://atomisator.ziade.org
port = 8999
wport = 9000
url = http://atomisator.ziade.org/buildbot
slaves =
    linux            ty54ddf32
[linux]
recipe = collective.buildbot:slave
host = localhost
port = ${buildmaster:port}
password = ty54ddf32
[atomisator]
recipe = collective.buildbot:project
slave-names = linux
repository=http://hg-atomisator.ziade.org/unstable
vcs = hg
build-sequence =
```

```
    ./build
test-sequence =
    buildout/bin/nosetests
email-notification-sender = tarek@ziade.org
email-notification-recipient = tarek@ziade.org
[poller]
recipe = collective.buildbot:poller
repository=http://hg-atomisator.ziade.org/unstable
vcs = hg
user=anonymous
```

这定义了一个构建服务器，以及一个构建客户端和一个 Atomisator 项目。这个项目定义了一个可供调用的 build 脚本和一个运行项目中的测试运行程序的测试序列。

关于选项的补充信息，可以浏览 http://pypi.python.org/pypi/collective.buildbot。

在 build-sequence 中引用的 build 脚本必须被添加到仓库的根目录下，其内容如下。

```
#!/bin/sh
cd buildout
python bootstrap.py
bin/buildout -v
```

别忘了在推送之前设置执行标志，如下所示。

```
$ chmod +x build
$ hg add build
$ hg commit -m "added build script"
```

现在就可以执行 buildout 了，如下所示。

```
$ python bootstrap.py
$ bin/buildout -v
```

bootstrap.py 是一个小脚本，用以确保系统能够满足构建 buildbot 的要求。

应该在 bin 文件夹中得到两个脚本：一个用来启动构建服务器，另一个用来构建客户端。它们用 buildout 小节来命名，现在来运行它们，如下所示。

```
$ bin/buildmaster.py start
Following twistd.log until startup finished..
2008-04-03 16:06:49+0200 [-] Log opened.
...
2008-04-03 16:06:50+0200 [-] configuration update complete
The buildmaster appears to have (re)started correctly.
$ bin/linux.py start
Following twistd.log until startup finished..
The buildslave appears to have (re)started correctly.
```

现在，在浏览器的地址栏中输入 http://localhost:9000 就能访问 Buildbot。然后，单击 atomisator 链接来强制指定一个构建以控制整个系统。

2．整合 Buildbot 和 Mercurial

完成这个配置还有一步：将仓库提交事件与 Buildbot 挂钩起来。这样，每当有人将文件推送到仓库时会自动重新构建，这可以由 Buildbot 自带的 hgbuildbot.py 脚本来完成。

要将它用作一个命令，只需在 pbp.buildbotenv 之上运行 easy_install。这将安装该脚本并确保同时安装 Buildbot 和 Twisted，如下所示。

```
$ easy_install pbp.buildbotenv
```

然后，在.hg 文件夹中的 unstable hgrc 文件（/path/to/unstable/.hg/hgrc）中添加钩子，如下所示。

```
[web]
style = gitweb
description = Unstable branch
contact = Tarek <tarek@ziade.org>
push_ssl = false
allow_push = *
[hooks]
changegroup.buildbot = python:buildbot.changes.hgbuildbot.hook
[hgbuildbot]
master = atomisator.ziade.org:8999
```

hooks 小节将链接 hgbuildbot 脚本，hgbuildbot 小节则定义主服务器和客户端口。

3．整合 Apach 和 Buildbot

现在，可以在 Apache 中添加一条改写规则，使得无需指定 9000 端口就能够访问 Buildbot。

最简单的方法是创建一个特殊的虚拟主机，并将它添加到 Apache 配置文件集中，如下所示。

```
<VirtualHost *:80>
  ServerName atomisator-buildbot.ziade.org
  CustomLog /var/log/apache2/bot-access_log combined
  ErrorLog /var/log/apache2/bot-error.org.log
  RewriteEngine On
  RewriteRule ^(.*) http://localhost:9000/$1
</VirtualHost>
```

8.3 小　结

本章中介绍了以下内容：

- 集中式和分布式版本控制系统之间的不同；
- Mercurial 的使用方法，这是一个很好的分布式版本控制系统；
- 多仓库策略的设置和使用；
- 持续集成的概念；
- 如何设置 Buildbot 和 Mercurial 以提供持续集成支持。

下一章中将说明如何使用迭代和增量方法来管理软件生命周期。