

Распределенная система контроля версий Bazaar



Дмитрий Васильев

Когда активно работаешь с файлами, будь то исходные тексты программ, файлы конфигурации или статья, то в большинстве случаев нужно использовать какую-либо систему контроля версий. Например, в данный момент я переписываю это введение, и так как я использую систему контроля версий, то смогу в любой момент вернуться к одному из предыдущих вариантов, если мне не понравится этот. Так как основная работа с файлами – это их изменение, то наличие возможности вернуться к предыдущей версии дает огромную уверенность, что, в свою очередь, положительно влияет на производительность. Если работа идет в команде, то система контроля версий необходима, в противном случае в какой-то момент времени работа команды может застопориться.

От централизованных систем к распределенным

Рассмотрим, что сейчас происходит в области систем контроля версий с открытым исходным кодом. Пока это может быть не так очевидно, но просматривается постепенный рост количества распределенных систем и идет яв-

ное движение в эту сторону. Например, из централизованных систем большинство могут назвать следующие:

- **CVS** (<http://www.nongnu.org/cvs/>) – постепенно исчезающая, но все еще популярная система, разработанная поверх формата RCS файлов;

- **Subversion** (<http://subversion.tigris.org>) – на данный момент наиболее популярная централизованная система контроля версий, изначально разработанная как «лучший CVS» и в итоге исправившая многие ошибки в дизайне CVS;

А что же распределенные системы? Вот наиболее популярные и активно развивающиеся системы:

■ **Git** (<http://git-scm.com>) – распределенная система контроля версий, разработанная Линусом Торвальдсом. Изначально Git предназначалась для использования в процессе разработки ядра Linux, но позже стала использоваться и во многих других проектах – таких, как, например, X.org и Ruby on Rails. На данный момент Git является самой быстрой распределенной системой, использующей самое компактное хранилище ревизий. Но в то же время для пользователей, переходящих, например, с Subversion интерфейс Git может показаться сложным;

■ **Mercurial** (<http://www.selenic.com/mercurial>) – распределенная система, написанная на языке Python с несколькими расширениями на C. Из использующих Mercurial проектов можно назвать такие, как Mozilla и MoinMoin.

■ **Bazaar** (<http://bazaar-vcs.org>) – система, разработка которой поддерживается компанией Canonical – известной своими дистрибутивами Ubuntu и сайтом <https://launchpad.net>. Система в основном написана на языке Python и используется такими проектами, как, например, MySQL и Drupal.

Рассмотрим одну из наиболее гибких, на мой взгляд, распределенных систем контроля версий – Bazaar.

Одним из примеров гибкости Bazaar может служить возможность использования как централизованной модели, так и распределенной и даже смешивание этих моделей контроля версий.

Даже если вы не согласны со мной по вопросу выбора конкретной системы, эта статья поможет вам понять общие принципы работы распределенных систем и затем остановить свой выбор на одной из перечисленных выше.

Зачем нужны распределенные системы?

Как следует из названия, одна из основных идей распределенных систем – это отсутствие четко выделенного цен-

трального хранилища версий – репозитория. В случае распределенных систем набор версий может быть полностью или частично распределен между различными хранилищами, в том числе и удаленными.

Такая модель отлично вписывается в работу распределенных команд, например, распределенной по всему миру команды разработчиков, работающих над одним проектом с открытым исходным кодом. Разработчик такой команды может скачать себе всю информацию по версиям и после этого работать только на локальной машине.

Как только будет достигнут результат одного из этапов работы, изменения могут быть залиты в один из центральных репозиториях или опубликованы для просмотра на сайте разработчика или в почтовой рассылке.

Другие участники проекта, в свою очередь, смогут обновить свою копию хранилища версий новыми изменениями или попробовать опубликованные изменения на отдельной, тестовой ветке разработки.

К сожалению, без хорошей организации проекта отсутствие одного центрального хранилища может быть минусом распределенных систем. Если в случае централизованных систем всегда есть один общий репозиторий, откуда можно получить последнюю версию проекта, то в случае распределенных систем нужно организационно решить, какая из веток проекта будет основной.

Почему распределенная система контроля версий может быть интересна кому-то, кто уже использует централизованную систему – такую как Subversion?

Любая работа подразумевает принятие решений, и в большинстве случаев необходимо пробовать различные варианты: при работе с системами контроля версий для рассмотрения различных вариантов и работы над большими изменениями служат ветки разработки.

И хотя это естественная концепция, пользоваться ею в Subversion непросто. Тем более все усложняется в случае множественных последовательных объединений с одной ветки на другую – в этом случае нужно безошибочно указывать начальные и конечные версии

каждого изменения, чтобы избежать конфликтов и ошибок. Для распределенных систем контроля версий ветки разработки являются одной из основополагающих концепций – в большинстве случаев каждая копия хранилища версий является веткой разработки.

Таким образом, механизм объединения изменений с одной ветки на другую в случае распределенных систем является одним из основных, что позволяет пользователям прикладывать меньше усилий при использовании системы.

Основные концепции Bazaar

Итак, начнем ближе рассматривать Bazaar. На момент написания этой статьи последняя версия Bazaar 1.6.

Но прежде чем рассматривать работу с этой системой контроля версий, сначала обратим внимание на основные концепции, лежащие в ее основе. Если эти термины уже знакомы вам по централизованным системам, они могут иметь немного другое значение в приложении к распределенной системе Bazaar, и их понимание важно для дальнейшего рассмотрения.

■ **Ревизия** – это сохраненное состояние файлов и директорий, включая их содержимое и иерархию в заданный момент времени. С ревизией также связана мета-информация, например:

- ☑ автор изменения;
- ☑ дата изменения;
- ☑ комментарий, связанный с изменением;
- ☑ родительские ревизии, от которых произведена данная ревизия.

После создания, ревизии не меняются и могут быть идентифицированы глобально уникальным номером ревизии (revision-id), например таким: «pqm@pqm.ubuntu.com-20071129184101-u9506rihe4zbzyyz».

Идентификаторы ревизий создаются в момент фиксации (commit) изменений или в момент импорта ревизий из других систем. Идентификаторы ревизий необходимы для функционирования системы, но вряд ли кто-то сможет использовать идентификатор такого типа в разговоре. Специ-

ально для упрощения идентификации ревизий разработаны специфичные для ветки номера ревизий.

Номера ревизий генерируются «на лету» в момент исполнения команд и представляют собой последовательный увеличивающийся номер. Таким образом первая ревизия имеет номер 1, следующая за ней, более поздняя, ревизия – номер 2 и так далее. При объединении изменений с других веток номер ревизии представляется тремя числами, разделенными точками (например: 3112.1.5):

1. Номер ревизии для данной ветки, от которой были произведены объединенные изменения.
2. Порядковый номер (счетчик) объединенной ветки. Так как объединение может затрагивать сразу несколько веток, этот номер служит их счетчиком.
3. Порядковый номер ревизии с момента создания объединяемой ветки.

■ **Рабочее дерево** – это директория под контролем версий, содержащая файлы, которые может редактировать пользователь. Рабочее дерево ассоциировано с веткой. Многие из команд Bazaar используют рабочее дерево для своей работы. Например, команда `commit` (зафиксировать изменения) создает новую ревизию на основе текущего состояния рабочего дерева.

■ **Ветка** – в простейшем случае это упорядоченная серия ревизий, где последняя ревизия называется верхушкой. Ветки могут разделяться на несколько веток и затем, позже, объединяться обратно, формируя граф ревизий. При этом в рамках ветки может быть основная линия разработки, ревизии в которой помечаются обычными номерами ревизий. Также ветка может иметь другие линии разработки, помечаемые номерами ревизий через точку, как описано выше.

■ **Репозиторий** – это хранилище ревизий. Обычно ветка имеет свой собственный репозиторий, но также несколько веток могут разделять один репозиторий, чтобы уменьшить используемое ветками место на диске.

Такое разделение концепций позволяет более гибко использовать Bazaar, в том числе и как централизованную систему.

Рассмотрим набор сценариев, в основе которых лежат описанные выше концепции:

■ **Разделяемые репозитории (Shared repositories)** – в этом случае рабочее дерево и ветка находятся в одной директории, но репозиторий находится на одну директорию выше, что позволяет хранить информацию о ревизиях только в одном месте. Такой сценарий позволяет не тратить место на полные копии ревизий, если ветки, находящиеся под репозиторием, отличаются незначительно, например, относятся к одному проекту.

■ **Легковесные рабочие копии (Lightweight checkouts)** – ветка и рабочее дерево находятся в разных местах. Этот сценарий похож на использование централизованной системы контроля версий, когда удаленный репозиторий хранит всю информацию о ревизиях, а рабочая копия представляет собой только рабочие файлы и директорию.

Мы рассмотрели все основные концепции, лежащие в основе Bazaar, и теперь можно приступить к изучению интерфейса командной строки.

Начинаем работать с Bazaar

Во многих UNIX-совместимых операционных системах Bazaar может быть установлен штатными средствами системы. Для Windows и других ОС, в которых Bazaar недоступен для установки штатными средствами, его можно скачать с официального сайта: <http://bazaar-vcs.org/Download>.

Хотя для Bazaar есть графические интерфейсы, которые также могут быть установлены, и вместе с основной программой мы будем рассматривать только интерфейс командной строки, являющийся основным интерфейсом работы с Bazaar.

Интерфейс командной строки представлен командой `bzr`, и первое, что можно сделать, это проверить установку следующей командой:

```
bzr version
```

В ответ будет выведена текущая версия Bazaar, пути используемых системой файлов и директорий, а также лицензионная информация (Bazaar использует GPL2). Для получения списка основных команд можно использовать команду `help`:

```
bzr help
```

После команды `help` можно указывать любую другую команду, по которой нужно получить подробную информацию, например:

```
bzr help version
```

Или можно вывести все команды Bazaar, в том числе и определенные подключаемыми модулями, командой:

```
bzr help commands
```

Первое, что нужно сделать перед началом работы Bazaar, это указать свое имя и e-mail, которыми будут подписываться все ревизии. Это делается командой `whoami`, например:

```
bzr whoami "John Doe <john@nowhere>"
```

Эта же команда без параметров выводит текущее установленное имя:

```
bzr whoami
```

```
John Doe <john@nowhere>
```

Если вы хотите использовать разные имена для разных веток, то можно воспользоваться опцией `--branch` для установки имени только для текущей ветки:

```
bzr whoami --branch „
"John Doe <john@nowhere>"
```

Конфигурационные файлы Bazaar хранятся в директории `$HOME/.bazaar` для UNIX-подобных операционных систем и в директории `C:\Documents and Settings\<username>\Application Data\Bazaar\2.0` для Windows. В этой директории хранятся три основных файла конфигурации:

■ **bazaar.conf** – описывает конфигурационные параметры по умолчанию;

- **locations.conf** – описывает конфигурационные параметры для отдельных веток;
- **authentication.conf** – описывает идентификационную информацию для доступа к удаленным серверам.

Каждая ветка также может содержать файл конфигурации, в этом случае он находится в каталоге `.bzzr/branch/branch.conf` внутри ветки.

После того, как мы настроили свое имя и e-mail командой `whoami`, эта информация записывается в файл `bazaar.conf`, и в простейшем случае он будет выглядеть так:

```
[DEFAULT]
email = John Doe <john@nowhere>
```

Работаем самостоятельно

Даже если вы работаете в команде, в какой-то момент времени все равно приходится работать одному, и с точки зрения многих команд Bazaar в этом случае мало что меняется.

Если у вас уже есть готовый проект и его надо поставить под контроль версий, нужно начать со следующей цепочки команд:

```
$ cd project-directory
$ bzzr init
$ bzzr add
$ bzzr commit -m "Импортирование проекта"
```

Здесь первая команда `init` создает в рабочей директории `project-directory` служебную директорию `.bzzr`, в которой хранятся репозиторий и ветка проекта.

Следующая команда, `add`, рекурсивно добавляет все файлы и директории в рабочем каталоге под контроль версий. Если нет необходимости в рекурсивном добавлении, то можно использовать опцию `--no-recurse` или перечислить необходимые для добавления объекты сразу вслед за командой.

Bazaar может контролировать и, соответственно, добавлять командой `add`, файлы, директории и символические ссылки (для ссылок хранится значение ссылки, а не объект, на который она ссылается).

Если в процессе работы над проектом создаются новые файлы и директории, которые нужно хранить под контролем версий, они также должны быть добавлены командой `add` или внесены в список игнорируемых файлов, описанной ниже, командой `ignore`. Если этого не сделать, файлы будут показываться Bazaar как неизвестные.

И последняя команда, `commit`, фиксирует изменения на ветке и в репозитории, создавая новую ревизию. Опция `-m` позволяет указать комментарий к ревизии прямо из командной строки. Без этой опции для создания комментария будет вызван текстовый редактор.

На данный момент в Bazaar нет жесткого ограничения на размер комментария (единственное ограничение – это достаточно большой, максимальный размер строки в Python), но рекомендуется делать их разумной длины, достаточной для описания сделанных изменений.

По умолчанию команда `commit` фиксирует все изменения на ветке, даже если запущена из-под директории вет-

ки. Если требуется зафиксировать только отдельную директорию или набор файлов, их можно указать вслед за командой, как здесь:

```
bzzr commit -m "Изменения" README.txt src
```

Если при создании нового проекта планируется создавать несколько локальных веток, то можно воспользоваться описанным выше разделяемым репозиторием для экономии места:

```
$ bzzr init-repo repo
$ cd repo
$ bzzr init trunk
$ cd trunk
```

Создание файлов

```
$ bzzr add
$ bzzr commit -m "Импортирование проекта"
```

Первая команда `init-repo` создает в директории `repo` разделяемый репозиторий, и под директорией `repo` могут храниться различные ветки проекта. Основную ветку мы создаем следующей командой `init`, которая уже была описана выше. В случае создания разделяемого репозитория в директории `repo/.bzzr` хранится репозиторий и в директории `repo/trunk/.bzzr` ветка.

Если при добавлении мы хотим игнорировать какие-либо объекты, можно воспользоваться командой `ignore`, ее синтаксис:

```
bzzr ignore имена объектов
```

Эта команда создаст в корневой директории ветки файл `.bzzrignore`, в котором будут записаны имена или шаблоны для игнорирования. Если удобно, то данный файл также можно редактировать в текстовом редакторе.

Глобальные правила игнорирования можно задать в конфигурационном файле `ignore` в каталоге с конфигурацией, `~/.bazaar/ignore` для UNIX-подобных систем.

Кроме задания полных имен и шаблонов, используемых командной оболочкой, таких как `*.py[co]`, `*.o`, можно использовать регулярные выражения:

```
bzzr ignore "RE:lib/*\.o"
```

Просмотр изменений

Когда закончен очередной этап изменений, хорошей практикой, прежде чем фиксировать эти изменения в системе контроля версий, может быть их просмотр.

Команда `status` показывает изменения в рабочем дереве с момента последней ревизии:

```
$ bzzr status
```

```
modified:
  README.txt
unknown:
  db.tmp
```

Команда `diff` показывает изменения в текстовых файлах в стандартной унифицированной форме:

```
$ bzip diff
```

```
== added file 'README.txt'
--- README.txt      1970-01-01 00:00:00 +0000
+++ README.txt      2008-01-20 14:23:29 +0000
@@ -0,0 +1,1 @@
+файл README
```

Просматривать изменения между заданными ревизиями можно с использованием опции `-r`:

```
bzip diff -r 100..
bzip diff -r 100..120
```

Первая команда показывает все изменения, начиная с ревизии 100, а вторая – между ревизией 100 и 120.

Для просмотра истории ревизий используется команда `log`. Bazaar использует иерархическую историю, где объединенные с данной веткой ревизии показываются с отступами в виде дерева. В этом случае визуально проще отделить объединенные изменения от изменений на основной ветке:

```
$ bzip log
```

```
-----
revno: 100
committer: John Doe <john@nowhere>
branch nick: trunk
timestamp: Tue 2008-08-19 16:25:36 +0100
message:
  Объединены изменения с работы
  -----
  revno: 100.1.1
  committer: John Doe <john@nowhere>
  branch nick: work
  timestamp: Tue 2008-08-19 09:54:08 -0500
  message:
    Добавлен файл README.txt
```

С командой `log` так же, как и с командой `diff`, можно использовать опцию `-r` для указания необходимых ревизий для просмотра.

Для просмотра краткой истории изменений, не включающей описание ревизий с объединенных веток, можно воспользоваться опцией `--short`:

```
$ bzip log --short
```

```
100 John Doe 2008-08-19 [merge]
  Объединены изменения с работы
```

Кроме того, историю ревизий можно просматривать для отдельного файла или директории, добавив его имя после команды:

```
bzip log README.txt
```

Команда `cat` выводит содержимое файла для ревизии заданной опцией `-r`:

```
$ bzip cat -r 100 README.txt
```

```
файл README
```

После просмотра изменений их можно зафиксировать командой `commit`, которая была описана выше.

Выпуск проекта

После некоторого количества ревизий наступает долгожданный момент выпустить очередную версию проекта. Для упаковки новой версии можно воспользоваться командой `export`, которая упаковывает проект в зависимости от заданного расширения файла. Например, следующая команда создаст ZIP-архив проекта:

```
bzip export ../releases/project-1.0.zip
```

При выпуске каждой версии имеет смысл пометить ее, чтобы в будущем можно было использовать более простые символические имена для работы с этой ревизией. Это можно сделать командой `tag`:

```
bzip tag version-1.0
```

После этого созданную метку можно использовать в других командах следующим образом:

```
bzip diff -r tag:version-1.0
```

Исправление ошибок

К сожалению, ошибок невозможно избежать, они случаются, и это надо принимать и быть к этому готовым. Bazaar разработан таким образом, что большинство ошибок можно исправлять, в том числе и с использованием отдельных команд. Рассмотрим различные ситуации и их разрешение.

Если вы случайно поставили под контроль версий не то рабочее дерево, то можно удалить служебный каталог `.bzr`, хотя стоит быть внимательным, чтобы не удалить что-то нужное.

Если случайно под контроль версий был поставлен файл, который нет необходимости хранить под контролем версий, то его можно удалить командой `remove`, но без использования опций команда не будет удалять измененный файл:

```
bzip add file.txt
bzip remove file.txt
```

```
bzip: ERROR: Can't safely remove modified or unknown files:
added:
  file.txt
Use --keep to not delete them, or --force to delete them
regardless.
```

Соответственно, можно использовать опцию `--keep`, чтобы оставить файл на диске, но удалить его регистрацию или опцию `--force`, чтобы удалить и регистрацию, и файл. Надо заметить, что если вы удалите файл, не используя Bazaar, то Bazaar будет считать, что необходимо удалить и регистрацию файла.

Команда `revert` возвращает все файлы и директории в состояние на момент последней фиксации, но при этом сохраняет измененные файлы под другими именами. Она действует рекурсивно, и если нужно вернуть только несколько файлов, то надо быть внимательным и указывать в командной строке только их:

```
bzr revert
```

Если вы случайно сделали фиксацию, то можно использовать команду `uncommit`, чтобы убрать последнюю ревизию и вернуть рабочее дерево в состояние до момента фиксации. Эта команда позволяет также исправлять и комментарии к ревизии:

```
bzr commit -m "Исправлен файл README"
bzr uncommit
bzr commit -m "Исправлен файл README.txt"
```

Кроме того, команда `uncommit` позволяет отменить несколько последних ревизий, например, три последние:

```
bzr uncommit -r -3
```

Если нет необходимости удалять ревизии, а нужно только вернуть рабочее дерево в прежнее состояние, то можно использовать команду `revert` с опцией `-r`:

```
bzr revert -r -3
```

Исправить неправильно поставленные метки можно с помощью команды `tag`, например, перенести метку на другую ревизию можно с помощью опции `--force`, а удалить с помощью опции `--delete`:

```
bzr tag --delete version-1.0
bzr tag --force version-2.0
```

Работаем в команде

После того как мы рассмотрели работу с Bazaar в одиночестве, рассмотрим распределенные возможности при работе в команде (или по такому же сценарию вы можете работать из нескольких мест, например, с работы и из дома). В простейшем случае, если вы работаете в команде, кто-то из разработчиков уже мог создать центральную ветку Bazaar, или при работе в паре это может быть одна из локальных веток другого разработчика, которую вам необходимо скопировать к себе. Кроме работы с файловой системой, Bazaar может работать со следующими протоколами: FTP, HTTP, HTTPS, SFTP и собственным оптимизированным протоколом `bzr`. Одним из простейших способов открыть к ветке доступ для чтения может быть использование HTTP/HTTPS. В этом случае достаточно в конфигурации веб-сервера открыть, с учетом необходимых ограничений, директорию с веткой на доступ, как и любую другую директорию. Для доступа на запись рекомендуется использовать соединение через SSH, которое можно установить, используя схему `bzr+ssh://`. В этом случае автоматически будет использоваться собственный протокол Bazaar через SSH туннель. К сожалению, на данный момент собственный протокол сам по себе не имеет никаких ограничений на доступ.

Для копирования ветки используется команда `branch`, которая по умолчанию копирует все ревизии из указанного места в локальный каталог, создавая новую ветку. В примере ниже мы создаем разделяемый репозиторий для экологии места и затем копируем удаленную ветку, создавая

ветку в каталоге `remote`, и в итоге создаем еще одну копию, копируя локальную ветку в каталог `local`:

```
$ cd projects
$ bzr init-repo project
$ cd project
$ bzr branch bzr+ssh://some.host/some/project remote
```

```
Branched 10 revision(s).
```

```
$ bzr branch remote local
```

```
Branched 10 revision(s).
```

Если теперь в каталоге `remote` мы посмотрим информацию о ветке командой `info`, то увидим ассоциированную родительскую ветку:

```
$ cd remote
$ bzr info
```

```
Standalone tree (format: pack-0.92)
```

```
Location:
```

```
branch root: .
```

```
Related branches:
```

```
parent branch: bzr+ssh://some.host/some/project
```

Хотя возможно большое количество сценариев работы, в нашем примере все локальные изменения будут делаться на ветке в каталоге `local`, а ветка в каталоге `remote` будет отслеживать все изменения, сделанные на удаленной ветке. Но в простейшем случае можно использовать и только одну локальную ветку.

Итак, представим, что мы проделали некоторую работу и создали несколько ревизий на ветке `local`. Теперь мы обновим копию удаленной ветки изменениями, сделанными другими разработчиками, и потом объединим наши изменения. Для обновления ветки нужно выполнить команду `pull` в каталоге `remote`, при этом будут скачаны и добавлены все ревизии, которые были сделаны на родительской ветке после момента последнего обновления:

```
bzr pull
```

Так как изменения на ветке `local` происходили параллельно с изменениями на удаленной ветке, то мы не можем просто сделать `pull`, а должны объединить изменения с помощью команды `merge` в каталоге `remote`:

```
bzr merge ../local
```

Если путь не указан, то `merge` как и `pull` будет использовать путь к родительской ветке. При объединении изменяется только рабочее дерево, чтобы сохранить объединенные изменения, нужно сделать фиксацию командой `commit`. Bazaar использует качественный алгоритм объединения, учитывающий возможные прошлые объединения, и во многих случаях не надо заботиться о таких тонкостях, как начальная и конечная ревизии для объединения.

В случае если в параллельных ветках было изменено одно и то же место в файле, автоматическое объединение может не сработать, и требуется устранить конфликты вручную. При объединении бинарных файлов по умолчанию любые параллельные изменения всегда будут конфликтовать,

но есть возможность подключения модулей, поддерживающих объединение конкретных бинарных форматов. Файлы, в которых не удалось сделать автоматическое объединение, можно посмотреть по команде `status`, или `conflicts`. При обнаружении конфликта Bazaar создаст дополнительный 3 файла для каждого файла с конфликтами:

- **имя.BASE** – файл с содержимым из предыдущей или последней ревизии текущей ветки;
- **имя.THIS** – файл с содержимым из последней ревизии или рабочая копия из текущей ветки;
- **имя.OTHER** – файл с содержимым из другой ветки.

После того, как конфликт был исправлен вручную, нужно использовать команду `resolve`, чтобы сказать Bazaar об этом, при этом ей можно указать только имена исправленных файлов:

```
bzr resolve
```

Также часто бывает полезно определить, кто поменял конкретные строчки в файле, для этого в Bazaar можно использовать команду `annotate`:

```
bzr annotate
```

После того как все изменения сделаны на ветке `remote`, нужно их перенести на удаленную ветку, для этого используется команда `push`, которая переносит все недостающие ревизии на родительскую ветку. Если родительская ветка также была изменена, то Bazaar скажет о расхождении веток и нужно будет сначала сделать их объединение и только затем `push`:

```
bzr push
```

Иногда нужно просто передать изменения для просмотра, не обновляя родительскую ветку. В каком-то случае можно опубликовать свою ветку с изменениями через один из протоколов, поддерживаемых Bazaar, или можно послать изменения по e-mail. Для этого в Bazaar есть команды `send` и `bundle`. Команда `bundle` может быть использована для создания пакета с изменениями, который можно приложить к e-mail-сообщению. Команда `send` позволяет послать сообщение с изменениями прямо из командной строки.

Работа в централизованном стиле

Кроме всего прочего Bazaar позволяет работать в централизованном стиле, когда все изменения в момент фиксации сразу идут и на одну центральную ветку. Рекомендуется все центральные ветки располагать в разделяемом репозитории, что может соответствовать корню репозитория в Subversion. Так как в центральном репозитории есть смысл хранить только историю изменений, его можно создать без рабочего дерева. Рассмотрим процесс создания основной центральной ветки, копирование ее на локальную ветку и добавление файлов:

```
$ bzr init-repo --no-trees sftp://central.host/bzr/repo
```

```
$ bzr init sftp://central.host/bzr/repo/trunk
$ bzr checkout sftp://central.host/bzr/repo/trunk
$ cd trunk
$ cp -ar ../some-files/ .
$ bzr add
$ bzr commit -m "Импорт проекта"
```

Здесь мы использовали команду `checkout`, чтобы скопировать и связать удаленную ветку с локальной. Кроме использования этой команды можно связать любые ветки с помощью команды `bind`. В этом случае в момент фиксации все изменения сразу пойдут и на связанную ветку. Если до момента фиксации на связанной ветке произошли изменения, то команда `commit` скажет о них и нужно будет прежде выполнить описанную ниже команду `update`:

```
bzr bind sftp://central.host/bzr/repo/trunk
```

Также в любой момент можно отвязать локальную ветку командой `unbind`, чтобы вернуться к режиму «локальные фиксации и публикация»:

```
bzr unbind
```

Надо заметить, что команда `checkout` копирует всю историю локально. Если в этом нет необходимости, то можно использовать ее с ключом `--lightweight`:

```
bzr checkout --lightweight sftp://central.host/bzr/repo/trunk
```

После того как мы сделали `checkout` обновления из центральной ветки, можно получать командой `update` как и в централизованных системах:

```
bzr update
```

Если при использовании центральной ветки вам в какой-то момент необходимо сделать фиксацию локальной ветки, можно использовать опцию `--local` команды `commit`:

```
bzr commit --local
```

Что дальше?

К сожалению, не удалось включить в эту статью другие возможности или принципы работы Bazaar, о которых хотелось бы рассказать. За кадром остались, например, такие темы:

- подключаемые модули, позволяющие расширять возможности Bazaar;
- как создавать псевдонимы для команд;
- примеры использования Bazaar в различных ситуациях;
- импорт и экспорт ревизий из других систем контроля версий;
- организация веток в виде стека.

Таким образом, эти темы остаются для самостоятельного изучения или могут быть темой одной из будущих статей. В любом случае я уверен, что изучение одной из распределенных систем контроля версий поможет вам увеличить продуктивность вашей работы. 