Лекция 4. Git на сервере

Лекция 4. Git на сервере

Git на сервере

К этому моменту вы уже должны уметь делать большую часть повседневных задач, для которых вы будете использовать Git. Однако, для совместной работы в Git, вам необходим удаленный репозиторий. Несмотря на то, что технически вы можете отправлять и забирать изменения непосредственно из личных репозиториев, делать это не рекомендуется. Вы легко можете испортить то, над чем работают другие, если не будете аккуратны. К тому же, вам бы наверняка хотелось, чтобы остальные имели доступ к репозиторию даже если ваш компьютер выключен, поэтому наличие более надежного репозитория обычно весьма полезно. Поэтому предпочтительный метод взаимодействия с кем-либо — это создание промежуточного репозитория, к которому вы оба будете иметь доступ, и отправка и получение изменений через него. Мы будем называть этот репозиторий «сервер Git», но обычно размещение репозитория Git требует очень небольшого количества ресурсов, поэтому вряд ли вам для этого будет нужен весь сервер.

Запустить Git-сервер просто. Для начала вам следует выбрать протокол, который вы будете использовать для связи с сервером. Первая часть этой главы описывает доступные протоколы и их достоинства и недостатки. Следующие части освещают базовые конфигурации с использованием этих протоколов, а также настройку вашего сервера для работы с ними. Наконец, мы рассмотрим несколько вариантов готового хостинга, если вы не против разместить ваш код на чьем-то сервере и вы не хотите мучиться с настройками и поддержкой вашего собственного сервера.

Если вас не интересует настройка собственного сервера, вы можете перейти сразу к последней части этой главы для настройки аккаунта на Git-хостинге, и затем перейти к следующей главе,

где мы обсудим различные аспекты работы с распределенной системой контроля версий.

Удаленный репозиторий — это обычно *голый (чистый, bare) репозиторий* — репозиторий Git, не имеющий рабочего каталога. Поскольку этот репозиторий используется только для обмена, нет причин создавать рабочую копию на диске, и он содержит только данные Git. Проще говоря, голый репозиторий содержит только каталог .git вашего проекта и ничего больше.

Протоколы

Git умеет работать с четырьмя сетевыми протоколами для передачи данных: локальный, Secure Shell (SSH), Git и HTTP. В этой части мы обсудим каждый из них и в каких случаях стоит (или не стоит) их использовать.

Важно понимать, что за исключением протокола HTTP, все эти протоколы требуют, чтобы Git был установлен и работал на сервере.

Локальный протокол

Базовым протоколом является *Локальный протокол*, при использовании которого удаленный репозиторий — другой каталог на диске. Наиболее часто он используется, если все члены команды имеют доступ к общей файловой системе, например к NFS, или, что менее вероятно, когда все работают на одном компьютере. Последний вариант не столь хорош, поскольку все копии вашего репозитория находятся на одном компьютере, делая возможность потерять все более вероятной.

Если у вас смонтирована общая файловая система, вы можете клонировать, отправлять и получать изменения из локального репозитория. Чтобы склонировать такой репозиторий или добавить его в качестве удаленного в существующий проект, используйте путь к репозиторию в качестве URL. Например, для клонирования локального репозитория вы можете выполнить что-то вроде этого:

```
$ git clone /opt/git/project.git
```

Или этого:

```
$ git clone file:///opt/git/project.git
```

Git работает немного по-другому если вы укажете префикс file:// для вашего URL. Когда вы просто указываете путь, Git пытается использовать жесткие ссылки и копировать файлы, когда это нужно. Если вы указываете file://, Git работает с данными так же как при использовании сетевых протоколов, что в целом менее эффективный способ передачи данных. Причиной для использования file:// может быть необходимость создания чистой копии репозитория без лишних внешних ссылок и объектов, обычно после импорта из другой СУВ или чего-то похожего (см. главу 9 о задачах поддержки). Мы будем использовать обычные пути, поскольку это практически всегда быстрее.

Чтобы добавить локальный репозиторий в существующий проект, вы можете воспользоваться командой:

```
$ git remote add local_proj /opt/git/project.git
```

Теперь вы можете отправлять и получать изменения из этого репозитория так, как вы это делали по сети.

Преимущества Преимущества основанных на файлах хранилищ в том, что они просты и используют существующие разграничения прав на файлы и сетевой доступ. Если у вас уже есть общая файловая система, доступ к которой имеет вся команда, настройка репозитория очень проста. Вы помещаете голый репозиторий туда, куда все имеют доступ, и выставляете права на чтение и запись, как вы бы это сделали для любого другого общего каталога. Мы обсудим, как экспортировать голую копию репозитория для этой цели в следующем разделе,

«Установка Git на сервер». Также это хорошая возможность быстро получить наработки из чьего-то рабочего репозитория.

Если вы и ваш коллега работаете над одним и тем же проектом и он хочет, чтобы вы проверили что-то, то запуск команды вроде git pull

/home/john/project зачастую проще, чем если бы он отправил на удалённый сервер, а вы забрали бы оттуда.

Недостатки Недостаток этого метода в том, что общий доступ обычно сложнее настроить и получить из разных мест, чем простой сетевой доступ. Если вы хотите отправлять со своего ноутбука, когда вы дома, вы должны смонтировать удалённый диск, что может быть сложно и медленно по сравнению с сетевым доступом.

Также важно упомянуть, что не всегда использование общей точки монтирования является быстрейшим вариантом. Локальный репозиторий быстрый, только если вы имеете быстрый доступ к данным. Репозиторий на NFS часто медленнее, чем репозиторий через SSH на том же сервере, позволяющий Git'у использовать на полную локальные диски на каждой системе.

Протокол SSH

Наверное, наиболее часто используемый транспортный протокол — это SSH. Причина этого в том, что доступ по SSH уже есть на многих серверах, а если его нет, то его очень легко настроить. Кроме того, SSH — единственный из сетевых протоколов, предоставляющий доступ и на чтение, и на запись. Два других сетевых протокола (HTTP и Git) в большинстве случаев дают доступ только на чтение, поэтому даже если они вам доступны, вам все равно понадобится SSH для записи. К тому же SSH протокол с аутентификацией, и благодаря его распространенности обычно его легко настроить и использовать.

Чтобы склонировать Git-репозиторий по SSH, вы можете указать префикс ssh:// в URL, например:

```
$ git clone ssh://user@server:project.git
```

Или вы можете не указывать протокол, Git подразумевает использование SSH, если вы не указали протокол явно:

```
$ git clone user@server:project.git
```

Также вы можете не указывать имя пользователя, Git будет использовать то, под которым вы вошли в систему.

Достоинства SSH имеет множество достоинств. Во-первых, вы по сути вынуждены его использовать, когда нужен авторизованный доступ на запись к репозиторию через сеть. Во- вторых, SSH достаточно легко настроить — демоны SSH распространены, многие системные администраторы имеют опыт работы с ними, и во многих дистрибутивах они уже настроены или есть утилиты для управления ими. Также доступ по SSH безопасен — данные передаются зашифрованными по авторизованным каналам. Наконец, так же как и Git-протокол и локальный протокол, SSH эффективен, делая данные перед передачей максимально компактными.

Недостатки Недостаток SSH в том, что, используя его, вы не можете обеспечить анонимный доступ к репозиторию. Клиенты должны иметь доступ к машине по SSH, даже для работы в режиме только на чтение, что делает SSH неподходящим для проектов с открытым исходным кодом. Если вы используете Git только внутри корпоративной сети, то возможно SSH единственный протокол, с которым вам придется иметь дело. Если же вам нужен анонимный доступ на чтение

для ваших проектов, вам придется настроить SSH для себя, чтобы выкладывать изменения, и что-нибудь другое для других, для скачивания.

Git-протокол

Следующий протокол — Git-протокол. Вместе с Git поставляется специальный демон, который слушает порт 9418 и предоставляет сервис, схожий с протоколом ssh, но абсолютно без аутентификации. Чтобы использовать Git-протокол для репозитория, вы должны создать файл git-export-daemon-ok, иначе демон не будет работать с этим репозиторием, но следует помнить, что в протоколе отсутствуют средства безопасности. Соответственно, любой репозиторий в Git может быть либо доступен для клонирования всем,

либо не доступен никому. Как следствие, обычно вы не можете отправлять изменения по этому протоколу. Вы можете открыть доступ на запись, но из-за отсутствия авторизации в этом случае кто угодно, зная URL вашего проекта, сможет его изменить. В общем, это редко используемая возможность.

Достоинства Git-протокол — самый быстрый из доступных протоколов. Если у вас проект с публичным доступом и большой трафик или у вас очень большой проект, для которого не требуется авторизация пользователей для чтения, вам стоит настроить демон Git для вашего

проекта. Он использует тот же механизм передачи данных, что и протокол SSH, но без дополнительных затрат на кодирование и аутентификацию.

Недостатки Недостатком Git-протокола является отсутствие аутентификации. Поэтому обычно не следует использовать этот протокол как единственный способ доступа к вашему проекту.

Обычно он используется в паре с SSH для разработчиков, имеющих доступ на запись, тогда как все остальные используют git:// с доступом только на чтение. Кроме того, это, вероятно, самый сложный для настройки протокол. Вы должны запустить собственно демон, не являющийся стандартным. Мы рассмотрим его настройку в разделе «Gitosis» этой главы. К тому же, ему необходим сервис xinetd или ему подобный, что не всегда легко сделать. Также необходимо, чтобы сетевой экран позволял доступ на порт 9418, который не является стандартным портом, всегда разрешённым в корпоративных брандмауэрах. За сетевыми экранами крупных корпораций этот неизвестный порт обычно заблокирован.

Протокол HTTP/S

Последний доступный протокол — HTTP. Прелесть протоколов HTTP и HTTPS в простоте их настройки. По сути, всё, что необходимо сделать — поместить голый репозиторий внутрь каталога с HTTP документами, установить обработчик post-update и всё (подробнее об обработчиках

рассказывается в главе 7). Теперь каждый, имеющий доступ к веб-серверу, на котором был размещен репозиторий, может его склонировать. Таким образом, чтобы открыть доступ к вашему репозиторию на чтение через HTTP, нужно сделать что-то наподобие этого:

```
$ cd /var/www/htdocs/

$ git clone --bare /path/to/git_project gitproject.git

$ cd gitproject.git

$ mv hooks/post-update.sample hooks/post-update

$ chmod a+x hooks/post-update
```

Вот и всё. Обработчик post-update, входящий в состав Git по умолчанию, выполняет необходимую команду (git update-server-info), чтобы извлечение (fetch) и клонирование (clone) по HTTP работали правильно. Эта команда выполняется, когда вы отправляете изменения в репозиторий по SSH. Затем остальные могут склонировать его командой:

```
$ git clone http://example.com/gitproject.git
```

В рассмотренном примере, мы использовали каталог /var/www/htdocs, обычно используемый сервером Apache, но вы можете использовать любой веб-сервер, отдающий статические данные, расположив голый репозиторий в нужном каталоге. Данные Git представляют собой обычные файлы (в главе 9 предоставление данных рассматривается более подробно).

Также возможна настройка Git для доступа на запись через HTTP, однако этот способ мало распространен и требует от вас настройки WebDAV. Поскольку этот способ редко используется, мы не будем рассматривать его в рамках этой книги. Положительным моментом настройки Git для записи через HTTP является то, что вы можете использовать любой WebDAV сервер, без поддержки каких-либо специфичных для Git возможностей. Таким образом если ваш хостинг предоставляет WebDAV, вы можете обеспечить запись обновлений репозитория на ваш веб-сайт.

Достоинства Положительным аспектом использования протокола HTTP является простота настройки. Запуск всего нескольких команд дает вам возможность предоставить миру доступ к вашему Git-репозиторию. Вам

понадобится всего несколько минут, чтобы сделать это. Кроме того, использование протокола HTTP не потребует много ресурсов вашего сервера. Поскольку

в основном используется статический HTTP сервер, обычный сервер Apache может обрабатывать в среднем тысячи файлов в секунду — трудно перегрузить даже небольшой сервер.

Также вы можете выставлять ваши репозитории в режиме только для чтения через HTTPS, т.е. вы можете шифровать трафик, или вы даже можете авторизовать клиентов по SSL сертификату. Обычно для этих целей легче использовать открытые ключи SSH, но в некоторых конкретных случаях лучшим решением может оказаться использование подписанных SSL сертификатов или других методов аутентификации основанных на HTTP, для доступа на чтение через HTTPS.

Другим плюсом является то, что HTTP — настолько широко используемый протокол, что корпоративные сетевые экраны часто настроены на пропускание трафика, проходящего через этот порт.

Недостатки Обратной стороной использования протокола HTTP является его относительно низкая эффективность для клиента. Обычно клонирование или извлечение изменений из репозитория при использовании HTTP гораздо продолжительнее, а объем данных и нагрузка на сеть намного больше, чем у любого другого имеющегося сетевого протокола. Поскольку он не заботится о том, чтобы передавались только необходимые вам данные — никакой динамической обработки на стороне сервера в этом случае не происходит — протокол HTTP часто называют *тупым* (dumb) протоколом. Более подробно о разнице в эффективности протокола HTTP и других протоколов рассказывается в главе 9.

Установка Git на сервер

Для того чтобы приступить к установке любого сервера Git, вы должны экспортировать существующий репозиторий в новый «голый» репозиторий, т.е.

репозиторий без рабочего каталога. Обычно это несложно сделать. Чтобы склонировать ваш репозиторий и создать новый «голый» репозиторий, выполните команду clone с параметром --bare. По существующему соглашению, каталоги с голыми репозиториями заканчиваются на .git, например:

```
$ git clone --bare my_project my_project.git

Initialized empty Git repository in /opt/projects/my_project.git/
```

Вывод этой команды слегка обескураживает. Поскольку clone по сути это git init, а затем git fetch, мы видим вывод от git init, который создает пустой каталог. Реальное перемещение объектов не имеет вывода, однако оно происходит. Теперь у вас должна быть копия данных из каталога Git в каталоге my_project.git.

Грубо говоря, это что-то наподобие этого:

```
$ cp -Rf my_project/.git my_project.git
```

Тут есть пара небольших различий в файле конфигурации, но в вашем случае эту разницу можно считать несущественной. Можно считать, что в этом случае берется собственно репозиторий Git без рабочего каталога и создается каталог только для него.

Размещение «голого» репозитория на сервере

Теперь, когда у вас есть голая копия вашего репозитория, всё, что вам нужно сделать, это поместить ее на сервер и настроить протоколы. Условимся, что вы уже настроили сервер git.example.com, имеете к нему доступ по SSH и хотите размещать все ваши репозитории Git в каталоге /opt/git. Вы можете добавить ваш новый репозиторий копированием голого репозитория:

```
$ scp -r my_project.git user@git.example.com:/opt/git
```

Теперь другие пользователи, имеющие доступ к серверу по SSH и право на чтение к каталогу /opt/git, могут склонировать ваш репозиторий, выполнив:

```
$ git clone user@git.example.com:/opt/git/my_project.git
```

Если у пользователя сервера есть право на запись в каталог /opt/git/my_project.git, он автоматически получает возможность отправки изменений в репозиторий. Git автоматически добавит право на запись в репозиторий для группы, если вы запустите команду git init с параметром --shared.

```
$ ssh user@git.example.com
$ cd /opt/git/my_project.git
$ git init --bare --shared
```

Видите, это просто взять репозиторий Git, создать «голую» версию и поместить ее на сервер, к которому вы и ваши коллеги имеете доступ по SSH. Теперь вы готовы работать вместе над одним проектом.

Важно отметить, что это практически всё, что вам нужно сделать, чтобы получить рабочий Git-сервер, к которому несколько человек имеют доступ — просто добавьте учетные записи SSH на сервер, и положите голый репозиторий в место, к которому эти пользователи имеют доступ на чтение и запись. И всё.

Из нескольких последующих разделов вы узнаете, как получить более сложные конфигурации.

В том числе как не создавать учетные записи для каждого пользователя, как сделать публичный доступ на чтение репозитория, как установить веб-интерфейс, как использовать Gitosis, и др. Однако, помните, что для совместной работы пары человек на закрытом проекте, всё, что вам *нужно* — это SSH-сервер и «голый» репозиторий.

Малые установки

Если вы небольшая фирма или вы только пробуете Git в вашей организации и у вас мало разработчиков, то всё достаточно просто. Один из наиболее сложных аспектов настройки сервера Git — управление пользователями. Если вы хотите, чтобы некоторые репозитории были доступны некоторым пользователям только на чтение, а другие и на чтение, и на запись, вам может быть не очень просто привести права доступа в порядок.

SSH доступ Если у вас уже есть сервер, к которому все ваши разработчики имеют доступ по SSH, проще всего разместить ваш первый репозиторий там, поскольку вам не нужно практически ничего делать (как мы уже обсудили в предыдущем разделе). Если вы хотите более сложного управления правами доступа к вашим репозиториям, вы можете сделать это обычными правами файловой системы, предоставляемыми операционной системой вашего сервера.

Если вы хотите разместить ваши репозитории на сервер, на котором нет учетных записей для каждого в вашей команде, кому нужен доступ на запись, вы должны настроить доступ по SSH для них. Будем считать, что если у вас есть сервер, на котором вы хотите это сделать, то SSH-сервер на нем уже установлен, и через него вы имеете доступ к серверу.

Есть несколько способов дать доступ каждому в вашей команде. Первый — настроить учетные записи для каждого. Это просто, но может быть весьма обременительно. Вероятно, вы не захотите для каждого пользователя выполнять adduser и задавать временные пароли.

Второй способ — создать на машине одного пользователя ‘git’, попросить каждого пользователя, кому нужен доступ на запись, прислать вам открытый ключ SSH, и добавить эти ключи в файл ~/.ssh/authorized_keys вашего нового пользователя ‘git’. Теперь все будут иметь доступ к этой машине через пользователя ‘git’. Это никак не повлияет на данные коммита пользователь, под которым вы соединяетесь с сервером по SSH, не затрагивает сделанные вами коммиты.

Другой способ сделать это — использовать SSH-сервер, аутентифицирующий по LDAP- серверу или любому другому централизованному источнику, который у вас может быть уже настроен. Любой способ аутентификации по SSH, какой вы только сможете придумать, должен работать, если пользователь может получить доступ к консоли.

Создание открытого SSH-ключа

Как было уже сказано, многие Git-серверы используют аутентификацию по открытым SSH-ключам. Для того чтобы предоставить открытый ключ, пользователь должен его сгенерировать, если только это не было сделано ранее. Этот процесс похож во всех операционных системах.

Сначала вам стоит убедиться, что у вас ещё нет ключа. По умолчанию пользовательские SSH- ключи хранятся в каталоге ~/.ssh этого пользователя. Вы можете легко проверить, есть ли у вас ключ, зайдя в этот каталог и посмотрев его содержимое:

```
$ cd ~/.ssh

$ ls
                          known hosts
config            id dsa.pub
```

Ищите пару файлов с именами «что-нибудь» и «что-нибудь.pub», где «что-нибудь» — обычно id_dsa или id_rsa. Файл с расширением .pub — это ваш открытый ключ, а второй файл — ваш секретный ключ. Если у вас нет этих файлов (или даже нет каталога .ssh), вы можете создать их, запустив программу ssh-keygen, которая входит в состав пакета SSH в системах Linux/Mac, а также поставляется в составе MSysGit для Windows:

```
$ ssh-keygen

Generating public/private rsa key pair.

Enter file in which to save the key (/Users/schacon/.ssh/id_rsa):
          Enter passphrase (empty for no passphrase):

Enter same passphrase again:

Your identification has been saved in /Users/schacon/.ssh/id_rsa.
Your public key has been saved in /Users/schacon/.ssh/id_rsa.pub.
The key fingerprint is:

43:c5:5b:5f:b1:f1:50:43:ad:20:a6:92:6a:1f:9a:3a    schacon@agadorlaptop.local
```

Сначала необходимо ввести расположение, для сохранения ключа (.ssh/id_rsa), затем дважды ввести пароль, который вы можете оставить пустым, если не хотите его вводить каждый раз, когда используете ключ.Теперь каждый пользователь должен послать свой открытый ключ вам или тому, кто

администрирует Git-сервер (предположим, что ваш SSH-сервер уже настроен на работу с открытыми ключами).

Для этого им нужно скопировать всё содержимое файла с расширением .pub и отправить его по электронной почте. Открытый ключ выглядит как-то так:

```
$ cat ~/.ssh/id_rsa.pub

ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAQEAklOUpkDHrfHY17SbrmTIpNLTGK9Tjom/BWDSU
GPl+nafzlHDTYW7hdI4yZ5ew18JH4JW9jbhUFrviQzM7xlELEVf4h9lFX5QVkbPppSwg0cda3
Pbv7kOdJ/MTyBlWXFCR+HAo3FXRitBqxiX1nKhXpHAZsMciLq8V6RjsNAQwdsdMFvSlVK/7XA
t3FaoJoAsncM1Q9x5+3V0Ww68/eIFmb1zuUFljQJKprrX88XypNDvjYNby6vw/Pb0rwert/En
mZ+AW4OZPnTPI89ZPmVMLuayrD2cE86Z/il8b+gw3r3+1nKatmIkjn2so1d01QraTlMqVSsbx
NrRFi9wrf+M7Q== schacon@agadorlaptop.local
```

Настраиваем сервер

Давайте рассмотрим настройку доступа по SSH на стороне сервера. В этом примере мы будем использовать метод authorized_keys для аутентификации пользователей. Мы подразумеваем, что вы используете стандартный дистрибутив Linux типа Ubuntu. Для начала создадим пользователя ‘git’ и каталог .ssh для этого пользователя:

```
$ sudo adduser git

$ su git

$ cd

$ mkdir .ssh
```

Затем вам нужно добавить открытый SSH-ключ некоторого разработчика в файл au- thorized_keys этого пользователя. Предположим, вы уже получили несколько ключей по электронной почте и сохранили их во временные файлы. Напомню, открытый ключ выглядит как-то так:

```
$ cat /tmp/id_rsa.john.pub

ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQCB007n/ww+ouN4gSLKssMxXnBOvf9LGt4L
ojG6rs6hPB09j9R/T17/x4lhJA0F3FR1rP6kYBRsWj2aThGw6HXLm9/5zytK6Ztg3RPKK+4k
Yjh6541NYsnEAZuXz0jTTyAUfrtU3Z5E003C4oxOj6H0rfIF1kKI9MAQLMdpGW1GYEIgS9Ez

Sdfd8AcCIicTDWbqLAcU4UpkaX8KyGlLwsNuuGztobF8m72ALC/nLF6JLtPofwFBlgc+myiv
O7TCUSBdLQlgMVOFq1I2uPWQOkOWQAHukEOmfjy2jctxSDBQ220ymjaNsHT4kgtZg2AYYgPq
dAv8JggJICUvax2T9va5 gsg-keypair
```

Вы просто добавляете их в ваш файл authorized_keys:

```
$ cat /tmp/id_rsa.john.pub >> ~/.ssh/authorized_keys

$ cat /tmp/id_rsa.josie.pub >> ~/.ssh/authorized_keys

$ cat /tmp/id_rsa.jessica.pub >> ~/.ssh/authorized_keys
```

Теперь вы можете создать пустой репозиторий для них, запустив git init с параметром --bare, что инициализирует репозиторий без рабочего каталога:

```
$ cd /opt/git
$ mkdir project.git
$ cd project.git
$ git --bare init
```

Затем Джон, Джози или Джессика могут отправить первую версию своего проекта в этот репозиторий, добавив его как удаленный и отправив ветку. Заметьте, что кто-то должен заходить на сервер и создавать голый репозиторий каждый раз, когда вы хотите добавить проект. Пусть gitserver — имя хоста сервера, на котором вы создали пользователя ‘git’ и репозиторий. Если он находится в вашей внутренней сети, вы можете настроить запись DNS для git-server, ссылающуюся на этот сервер, и использовать эти команды:

```
# на компьютере Джона
$ cd myproject
$ git init
$ git add .
$ git commit -m 'initial commit'
$ git remote add origin git@gitserver:/opt/git/project.git
```

Теперь остальные могут склонировать его и отправлять (push) туда изменения так же легко:

```
$ git clone git@gitserver:/opt/git/project.git
$ vim README
$ git commit -am 'fix for the README file'
$ git push origin master
```

Этим способом вы можете быстро получить Git-сервер с доступом на чтение/запись для небольшой группы разработчиков.

В качестве дополнительной меры предосторожности вы можете ограничить возможности пользователя ‘git’ только действиями, связанными с Git, с помощью ограниченной оболочки git-shell, поставляемой вместе с Git. Если вы выставите её в качестве командного интерпретатора пользователя ‘git’,

то этот пользователь не сможет получить доступ к обычной командной оболочке на вашем сервере. Чтобы её использовать, укажите git-shell вместо bash или csh в качестве командной оболочки пользователя. Для этого вы должны отредактировать файл /etc/passwd:

```
$ sudo vim /etc/passwd
```

В конце вы должны найти строку, похожую на эту:

```
git:x:1000:1000::/home/git:/bin/sh
```

Замените /bin/sh на /usr/bin/git-shell (или запустите which git-shell, чтобы проверить, куда он установлен). Отредактированная строка должна выглядеть следующим образом:

```
git:x:1000:1000::/home/git:/usr/bin/git-shell
```

Теперь пользователь ‘git’ может использовать SSH соединение только для работы с репозиториями Git и не может зайти на машину. Вы можете попробовать и увидите, что вход в систему отклонен:

```
$ ssh git@gitserver

fatal: What do you think I am? A shell?
    Connection to gitserver closed.
```

4.1 Открытый доступ

Что, если вы хотите иметь анонимный доступ к вашему проекту на чтение? Возможно, вместо размещения внутреннего закрытого проекта, вы хотите разместить проект с открытым исходным кодом. Или, может быть, у вас есть автоматизированные серверы сборки или серверы непрерывной интеграции, которые часто изменяются, и вы не хотите постоянно перегенерировать SSH-ключи, а вы просто хотите добавить анонимный доступ на чтение.

Вероятно, наиболее простой способ для небольших конфигураций — запустить статический веб-сервер, указав в качестве корневого каталога для документов каталог, в котором расположены ваши репозитории Git, и разрешив хук post-update, как было показано в первой части этой главы. Давайте продолжим работу с предыдущего примера. Допустим, ваши репозитории

расположены в каталоге /opt/git, и сервер Apache запущен на вашей машине. Повторюсь, вы можете использовать любой веб-сервер, но в качестве примера мы покажем несколько основных конфигураций Apache, которые покажут основную идею.

Для начала вам следует включить хук:

```
$ cd project.git

$ mv hooks/post-update.sample hooks/post-update

$ chmod a+x hooks/post-update
```

Если вы используете версию Git ниже 1.6, то команда mv не нужна — добавление суффикса .sample к именам примеров хуков началось только недавно.

Что делает хук post-update? Обычно он выглядит так:

```
$ cat .git/hooks/post-update

#!/bin/sh

exec git-update-server-info
```

Это означает, что когда вы отправляете что-то с помощью git push на сервер по SSH, Git будет запускать эту команду, чтобы обновить файлы необходимые для скачивания по HTTP. Затем вы должны добавить запись VirtualHost в конфигурацию вашего Apache с корневым каталогом документов в каталоге с вашими проектами Git. Здесь мы подразумеваем, что ваш DNS сервер настроен на отсылку *.gitserver на ту машину, на которой всё это запущено:

```
<VirtualHost *:80>

ServerName git.gitserver
  DocumentRoot /opt/git

<Directory /opt/git/>
    Order allow, deny
    allow from all

</Directory>

</VirtualHost>
```

Вам также понадобится установить Unix-группу для каталога /opt/git в www-data, чтобы ваш веб-сервер получил доступ на чтение этих каталогов,

поскольку (по умолчанию) Apache запускает CGI-сценарии от имени такого пользователя:

```
$ chgrp -R www-data /opt/git
```

После перезапуска Apache вы должны получить возможность склонировать ваши репозитории из этого каталога указывая их в URL:

```
$ git clone http://git.gitserver/project.git
```

Таким образом, вы можете настроить доступ на чтение по HTTP к любому из ваших проектов для значительного количества пользователей за несколько минут. Другой простой способ дать открытый неаутентифицируемый доступ — использовать Git-демон, однако это требует запуска процесса демона. Мы рассмотрим этот вариант в следующем разделе, если вы предпочитаете этот вариант.

GitWeb

Теперь, когда у вас есть основной доступ на чтение и запись и доступ только на чтение к вашему проекту, вероятно, вы захотите настроить простой веб-визуализатор. Git поставляется в комплекте с CGI-сценарием, называющимся GitWeb, который обычно используется для этого.

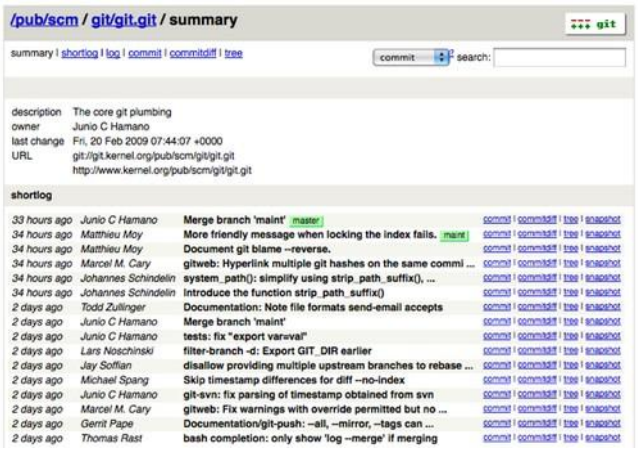

Рисунок 4.1: Веб-интерфейс GitWeb.

Вы можете увидеть GitWeb в действии на таких сайтах как http://git.kernel.org (рис. 4-1).

Если вы хотите проверить, как GitWeb будет выглядеть для вашего проекта, Git поставляется

с командой для быстрой установки временного экземпляра, если в вашей системе есть легковесный веб-сервер, такой как lighttpd или webrick. На машинах с Linux lighttpd часто установлен, поэтому возможно вы сможете его запустить, выполнив git instaweb в каталоге с вашим проектом. Если вы используете Mac, Leopard поставляется с предустановленным Ruby, поэтому webrick может быть лучшим выбором. Чтобы запустить instaweb с не ligttpd, вы можете запустить команду с параметром --httpd.

```
$ git instaweb --httpd=webrick

[2009-02-21 10:02:21] INFO  WEBrick 1.3.1

[2009-02-21 10:02:21] INFO  ruby 1.8.6 (2008-03-03) [universal-darwin9.0]
```

Это запустит сервер HTTPD на порту 1234 и затем запустит веб-браузер, открытый на этой странице. Это очень просто. Когда вы закончили и хотите остановить сервер, вы можете запустить ту же команду с параметром --stop:

```
$ git instaweb --httpd=webrick --stop
```

Если вы хотите иметь постоянно работающий веб-интерфейс на сервере для вашей команды или для проекта с открытым кодом на хостинге, вам необходимо установить CGI-сценарий на вашем веб-сервере. В некоторых дистрибутивах Linux есть пакет gitweb, который вы можете установить, используя apt или yum, так что вы можете попробовать сначала этот способ. Мы рассмотрим установку GitWeb вручную очень вкратце. Для начала вам нужно получить исходный код Git, с которым поставляется GitWeb, и сгенерировать CGI-сценарий под свою систему:

```
$ git clone git://git.kernel.org/pub/scm/git/git.git
```

```
$ cd git/

$ make GITWEB_PROJECTROOT="/opt/git" \
        prefix=/usr gitweb/gitweb.cgi

$ sudo cp -Rf gitweb /var/www/
```

Помните, что вы должны указать команде, где расположены ваши репозитории Git с помощью переменной GITWEB_PROJECTROOT. Теперь вы

должны настроить Apache на использование этого сценария, для чего вы можете добавить виртуальный хост:

```
<VirtualHost *:80>
 ServerName gitserver

                              DocumentRoot /var/www/gitweb

                              <Directory /var/www/gitweb>

    Options ExecCGI +FollowSymLinks +SymLinksIfOwnerMatch
                        AllowOverride All

    order allow,deny
     Allow from all

    AddHandler cgi-script cgi
    DirectoryIndex gitweb.cgi

                                      </Directory>
```

Повторюсь, GitWeb может быть установлен на любой веб-сервер, совместимый с CGI. Если вы предпочитаете использовать что-то другое, настройка не должна стать для вас проблемой. К этому моменту вы должны иметь возможность зайти на http://gitserver/ для просмотра ваших репозиториев онлайн, а также использовать http://git.gitserver для клонирования и извлечения данных для ваших репозиториев по HTTP.

Gitosis

Хранение открытых ключей всех пользователей в authorized_keys для предоставления доступа работает хорошо лишь на время. Когда у вас сотни пользователей, это скорее похоже на пытку. Вы должны заходить на сервер каждый раз, и нет никакого разграничения доступа все перечисленные в файле имеют доступ на чтение и на запись к каждому проекту.

На этой стадии вы можете захотеть обратиться к широко используемому ПО под названием Gitosis. Gitosis — это просто набор сценариев (скриптов), который поможет вам управляться с файлом authorized_keys и реализовать простой контроль доступа. Действительно интересно, что добавление людей и настройка доступа для них осуществляется не через веб-интерфейс, а с помощью специального git-репозитория. Вы настраиваете информацию в этом

проекте и, когда вы отправляете её в репозиторий, Gitosis, исходя из неё, перенастраивает сервер, что круто.

Установка Gitosis — не самая простая задача, хотя и не слишком сложная. Проще всего использовать под него Linux-сервер — в наших примерах используется сервер Ubuntu 8.10 в начальной конфигурации.

Gitosis'у нужны некоторые инструменты для Python, так что первым делом вы должны установить пакет Python'a setuptools, который в Ubuntu называется python-setuptools:

```
$ apt-get install python-setuptools
```

Затем вы клонируете и устанавливаете Gitosis с главного сайта проекта:

```
$ git clone git://eagain.net/gitosis.git

$ cd gitosis

$ sudo python setup.py install
```

Это установит несколько исполняемых файлов, которые Gitosis будет использовать. Затем Gitosis хочет расположить свои репозитории в каталоге /home/git, что неплохо. Но вы уже установили репозитории в /opt/git, так что вместо перенастройки всего на свете вы сделаете символическую ссылку:

```
$ ln -s /opt/git /home/git/repositories
```

Gitosis будет управлять ключами за вас, так что вы должны удалить текущий файл, добавить ключи снова позже и предоставить Gitosis'у управлять файлом authorized_keys автоматически. Сейчас просто уберите этот файл с дороги:

```
$ mv /home/git/.ssh/authorized_keys /home/git/.ssh/ak.bak
```

Затем вы должны вернуть пользователю git его командную оболочку, если вы меняли её на команду git-shell. Люди всё так же не смогут выполнить вход, но для вас это будет контролировать Gitosis. Итак, давайте поменяем эту строку в файле '/etc/passwd'

```
git:x:1000:1000::/home/git:/usr/bin/git-shell
```

обратно на эту:

```
git:x:1000:1000::/home/git:/bin/sh
```

Теперь самое время инициализировать Gitosis. Сделаете это, выполнив команду gitosis- init со своим персональным открытым ключом. Если вашего открытого ключа ещё нет на сервере, вам нужно будет скопировать его туда:

```
$ sudo -H -u git gitosis-init < /tmp/id_dsa.pub

   Initialized empty Git repository in /opt/git/gitosis-admin.git/
Reinitialized existing Git repository in /opt/git/gitosis-admin.git/
```

Это позволит пользователю с таким ключом изменять главный Git-репозиторий, который управляет настройками Gitosis'a. Затем вы должны вручную установить бит исполнения на сценарий post-update в новом управляющем репозитории.

```
$ sudo chmod 755 /opt/git/gitosis-admin.git/hooks/post-update
```

Всё готово. Если вы всё настроили правильно, вы можете попытаться соединиться по SSH с вашим сервером под тем пользователем, для которого вы добавили открытый ключ, чтобы инициализировать Gitosis. Вы должны увидеть что-то вроде этого:

```
$ ssh git@gitserver

PTY allocation request failed on channel 0

fatal: unrecognized command 'gitosis-serve schacon@quaternion'
Connection to gitserver closed.
```

Это означает, что Gitosis узнал вас, но не пустил, потому что вы не пытались выполнить ни одну из команд Git. Ну так давайте выполним настоящую команду Git — вы склонируете управляющий репозиторий Gitosis:

```
# на вашем локальном компьютере

$ git clone git@gitserver:gitosis-admin.git
```

Теперь у вас есть каталог с именем gitosis-admin, в котором есть две главные части:

```
$ cd gitosis-admin

$ find .

./gitosis.conf

./keydir

./keydir/scott.pub
```

Файл gitosis.conf — файл настройки, который используется, чтобы указать пользователей, репозитории и права доступа. В каталоге keydir должны храниться открытые ключи всех пользователей, у которых есть какой-либо доступ к вашим репозиториям — по файлу на пользователя.

Имя файла в keydir (в предыдущем примере scott.pub) у вас будет отличаться — Gitosis берёт это имя из описания в конце открытого ключа, который был импортирован сценарием gitosis-init.

Если вы посмотрите в файл gitosis.conf, там должна быть указана только информация о проекте gitosis-admin, который вы только что склонировали:

```
$ cat gitosis.conf
    [gitosis]
```

```
  [group gitosis-admin]
writable = gitosis-admin
    members = scott
```

Это показывает, что пользователь ‘scott’ — пользователь, чьим открытым ключом вы инициализировали Gitosis — единственный, кто имеет доступ к проекту gitosis-admin.

А теперь давайте добавим новый проект. Добавьте новую секцию с названием mobile и перечислите в ней всех разработчиков из команды, занимающейся мобильными устройствами, а также проекты, к которым этим разработчикам нужно иметь доступ. Поскольку scott — пока что единственный пользователь в системе, добавьте его как единственного члена и создайте новый проект под названием iphone_project, чтобы ему было с чем начать работать:

```
[group mobile]

writable = iphone_project
    members = scott
```

Когда вы вносите изменения в проект gitosis-admin, вы должны зафиксировать изменения и отправить их на сервер, чтобы они возымели эффект:

```
$ git commit -am 'add iphone_project and mobile group'
       [master]: created 8962da8: "changed name"

                  1 files changed, 4 insertions(+), 0 deletions(-)

                                      $ git push

                               Counting objects: 5, done.

   Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 272 bytes, done.
    Total 3 (delta 1), reused 0 (delta 0)

To git@gitserver:/opt/git/gitosis-admin.git
      fb27aec..8962da8  master -> master
```

Вы можете сделать свой первый push в новый проект iphone_project, добавив свой сервер в качестве удалённого (remote) в локальную версию проекта и выполнив git push. Вам больше не нужно вручную создавать голые репозитории на сервере для новых проектов — Gitosis создаёт их сам автоматически, когда видит первый push:

```
               $ git remote add origin git@gitserver:iphone_project.git

                                $ git push origin master

Initialized empty Git repository in /opt/git/iphone_project.git/
                  Counting objects: 3, done.

Writing objects: 100% (3/3), 230 bytes, done.
    Total 3 (delta 0), reused 0 (delta 0)

                         To git@gitserver:iphone_project.git

                         * [new branch]      master -> master
```

Заметьте, что вам не нужно указывать путь (фактически, если вы это сделаете, то оно не сработает), только двоеточие и имя проекта — Gitosis найдёт его за вас. Вы хотите работать над проектом с вашими друзьями, так что вам нужно снова добавить их открытые ключи. Но вместо того, чтобы вручную добавлять их к файлу ~/.ssh/authorized_keys на вашем сервере, добавьте их, один файл на ключ, в каталог keydir. Как вы назовёте ключи определит как вы будете ссылаться на пользователей в gitosis.conf. Давайте по-новому добавим открытые ключи для Джона, Джози и Джессики:

```
$ cp /tmp/id_rsa.john.pub keydir/john.pub

$ cp /tmp/id_rsa.josie.pub keydir/josie.pub

$ cp /tmp/id_rsa.jessica.pub keydir/jessica.pub
```

Теперь вы можете добавить их всех в вашу ‘мобильную’ команду, чтобы они имели доступ на чтение и запись в iphone_project:

```
[group mobile]

writable = iphone_project

members = scott john josie jessica
```

После того, как вы зафиксируете и отправите изменения, все четыре пользователя будут иметь возможность читать и писать в проект.

В Gitosis также есть простой контроль доступа. Если вы хотите, чтобы Джон имел только доступ на чтение к этому проекту, вы можете вместо этого сделать:

```
[group mobile]

  writable = iphone_project
members = scott josie jessica

[group mobile_ro]

readonly = iphone_project
     members = john
```

Теперь Джон может клонировать проект и получать обновления, но Gitosis не позволит ему отправлять изменения обратно в проект. Вы можете создать таких групп сколько хотите, каждую содержащую разные проекты и пользователей. Вы также можете указать другую группу в качестве одного из пользователей (используя @ как префикс), чтобы автоматически добавить всех её членов:

```
   [group mobile_committers]
members = scott josie jessica

[group mobile]

  writable  = iphone_project
members    = @mobile_committers

[group mobile_2]

 writable  = another_iphone_project
members    = @mobile_committers john
```

Если у вас возникли какие-то проблемы, полезным может быть добавить loglevel=DEBUG в секции [gitosis]. Если вы потеряли доступ к отправке, отправив неверную конфигурацию, вы можете вручную поправить файл /home/git/.gitosis.conf на сервере — файл, из которого Gitosis читает свою информацию. Отправка в проект берёт файл gitosis.conf, который вы только что отправили, и помещает его туда. Если вы отредактируете этот файл вручную, он останется таким до следующей успешной отправки в проект gitosis-admin.

4.2 Gitolite

Git начал становиться очень популярным в корпоративных средах, где обычно есть дополнительные требования в плане контроля доступа. Gitolite изначально был создан, чтобы посодействовать

в выполнении таких требований. Но, как оказывается, он также полезен и в мире open source: проект Fedora управляет доступом к своим репозиториям пакетов с помощью gitolite. А ведь этих репозиториев больше 10 000! По видимому, это самая большая установка gitolite где бы то ни было.

Gitolite позволяет указать права доступа не только для репозиториев, но и для веток или имён меток внутри каждого репозитория. То есть вы можете указать, что определённые люди (или группы людей) могут отправлять (push) определённые «ссылки» (ветки или метки), а остальные нет.

Установка

Установить Gitolite очень просто, даже если вы не читали обширную документацию, которая идёт вместе с ним. Вам нужен аккаунт на каком-нибудь Unix сервере; были протестированы различные Linux-ы и Solaris 10. Вам не нужен root-доступ, если git, perl и openssh-совместимый сервер уже настроены. Далее в примерах мы будем использовать аккаунт gitolite на хосте с именем gitserver.

Gitolite несколько необычен, по крайней мере, в сравнении с другим «серверным» ПО доступ осуществляется по ssh, и, следовательно, каждый пользователь на сервере является потенциальным «gitolite-хостом». Поэтому

всё выглядит как «установка» самого ПО и затем «настройка» пользователя как «gitolite-хоста».

Gitolite может быть установлен четырьмя способами. Люди, использующие Fedora'у или Debian, могут получить RPM или DEB и установить его. Те, у кого есть root-доступ, могут сделать установку вручную. В обоих вариантах любой пользователь в системе затем может стать «gitolite-хостом».

Те, у кого нет root-доступа, могут установить его внутрь своих каталогов. И наконец, gitolite может быть установлен с помощью выполнения сценария *на рабочей станции* в bash- шелле. (Если вам интересно, даже тот bash, который идёт с msysgit, достаточен.)

Последний способ мы опишем в этой статье; а остальные методы описаны в документации.

Начните с настройки доступа к вашему серверу с помощью открытого ключа, так, чтобы вы могли войти с вашей рабочей станции на сервер без ввода пароля. Следующий способ работает в Linux; для рабочих станций с другими ОС вам, возможно, нужно будет сделать это вручную. Мы полагаем, что у вас уже есть пара ключей сгенерированных с помощью ssh- keygen.

```
$ ssh-copy-id -i ~/.ssh/id_rsa gitolite@gitserver
```

Эта команда спросит у вас пароль к учётной записи 'gitolite', а затем настроит доступ по открытому ключу. Он **необходим** для сценария установки, так что убедитесь, что вы можете выполнять команды без ввода пароля:

```
$ ssh gitolite@gitserver pwd

/home/gitolite
```

Затем склонируйте Gitolite с главного сайта проекта и выполните сценарий для лёгкой установки (третий аргумент это ваше имя в том виде, в котором вам бы хотелось его видеть в окончательном репозитории gitolite-admin):

```
$ git clone git://github.com/sitaramc/gitolite

$ cd gitolite/src

$ ./gl-easy-install -q gitolite gitserver sitaram
```

Всё готово! Gitolite теперь установлен на сервере, и у вас в домашнем каталоге на рабочей станции теперь есть новый репозиторий, который называется gitolite-admin. Администрирование вашего установленного gitolite осуществляется с помощью внесения изменений в этот репозиторий и их отправки (push).

Та последняя команда выводит довольно большое количество информации, которую может быть интересно прочитать. Также при первом её выполнении создаётся новая пара ключей; вам придётся выбрать пароль или нажать enter, чтобы пароля не было. Зачем нужна вторая пара ключей и как она используется, описано в документе «ssh troubleshooting», поставляемом с Gitolite. (Ну должна же документация быть *хоть для чего-то* хороша!)

По умолчанию на сервере создаются репозитории с именами gitolite-admin и test- ing. Если вы хотите получить локальную копию какого-то из них, наберите (под учётной записью, которая имеет консольный SSH-доступ к gitolite-аккаунту в *authorized_keys*):

```
$ git clone gitolite:gitolite-admin

$ git clone gitolite:testing
```

Чтобы склонировать эти же самые репозитории под любым другим аккаунтом:

```
$ git clone gitolite@servername:gitolite-admin

$ git clone gitolite@servername:testing
```

Изменение параметров установки

Хотя быстрая установка с параметрами по умолчанию подходит для большинства людей, есть несколько способов изменения параметров установки если вам это нужно. Если опустить опцию -q, вы получите «подробную» установку с детальной информацией о том, что происходит на каждом шаге.

Подробный режим также позволяет изменить некоторые параметры на стороне сервера, такие как расположение репозиториев, с помощью редактирования «rc» файла, используемого сервером. Этот «rc» файл содержит развёрнутые комментарии так, чтобы вы легко смогли сделать любые изменения, сохранить их и продолжить. Этот файл также содержит различные настройки, которые вы можете изменить, чтобы активировать или выключить некоторые «продвинутые» функции gitolite.

Конфигурационный файл и правила контроля доступа

Когда установка завершена, вы переходите в репозиторий gitolite-admin (он находится в вашем домашнем каталоге) и осматриваетесь, чтобы выяснить что же вы получили:

```
                    $ cd ~/gitolite-admin/
                             $ ls
                         conf/  keydir/
$ find conf keydir -type f
    conf/gitolite.conf
    keydir/sitaram.pub


repo gitolite-admin
                        = sitaram

repo testing
                        = @all
```

Заметьте, что «sitaram» (это последний аргумент при выполнении gl-easy-install ранее) имеет права на чтение и запись в репозиторий gitolite-admin, а также файл с открытым ключом с таким же именем.

Синтаксис конфигурационного файла для gitolite подробно продокументирован в conf/ example.conf, так что мы рассмотрим здесь только основные моменты.

Вы можете сгруппировать пользователей или репозитории для удобства. Имена групп совсем как макросы; когда вы их определяете, даже неважно,

определяют ли они проекты или пользователей; это различие делается, только когда вы *используете* «макрос».

```
@oss_repos      = linux perl rakudo git gitolite


 @admins        = scott       # Adams, not Chacon, sorry :)
@interns
                = sitaram dilbert wally alice
@engineers
```

Вы можете контролировать права доступа на уровне «ссылок» (то, что находится в .git/ refs/). В следующем примере стажёры (группа @interns) могут отправлять (push) только ветку «int». Инженеры (группа @engineers) могут отправлять любую ветку, чьё имя начинается с «eng-», а также метки, начинающиеся с «rc» и затем содержащие цифры. Администраторы (группа @admins) могут делать всё с любыми ссылками (в том числе откатывать назад).

```
                                repo @oss_repos

    RW  int$                = @interns

    RW  eng-                = @engineers
    RW   refs/tags/rc[0-9]  = @engineers
    RW+                     = @admins
```

Выражение после RW или RW+ — это регулярное выражение (regex), с которым сопоставляется имя отправляемой ссылки (ref). Поэтому мы называем его «refex»! Конечно, «refex» может быть гораздо более сложным, чем показано здесь, так что не переусердствуйте с ними, если вы не очень хорошо знакомы с регулярными выражениями perl.

К тому же, как вы уже, наверное, догадались, Gitolite для удобства дописывает в начале регулярного выражения refs/heads/, если оно не начинается с refs/.

Важной особенностью синтаксиса конфигурационного файла является то, что все правила для репозитория не обязательно должны находиться в одном месте. Вы можете держать все общие вещи вместе, как, например, правила для всех oss_repos выше, а потом добавить уточняющие правила для отдельных случаев следующим образом:

```
repo gitolite
                              = sitaram
```

Это правило будет добавлено к набору правил для репозитория gitolite.

В данный момент вы, возможно, задаётесь вопросом: «Каким образом правила контроля доступа применяются на самом деле?» — так что давайте вкратце рассмотрим это.

В gitolite есть два уровня контроля доступа. Первый — на уровне репозитория; если у вас есть доступ на чтение (или запись) к *любой* ссылке в репозитории, то у вас есть доступ на чтение (или запись) к этому репозиторию.

Второй уровень применим только к доступу на запись и осуществляется по веткам или меткам внутри репозитория. Имя пользователя, запрашиваемый уровень доступа (W или +) и имя ссылки, которая будет обновлена, известны. Правила доступа проверяются в порядке их появления в конфигурационном файле, в поисках совпадений для этой комбинации (но помните, что имя ссылки сопоставляется с регулярным выражением, а не просто строкой). Если совпадение найдено, отправка (push) проходит успешно. При неудачном исходе доступ запрещается.

Продвинутый контроль доступа с запрещающими правилами

До сих пор у нас были только права вида R, RW, или RW+. Однако, в gitolite есть другие права доступа: - означающий «запретить». Это даёт гораздо больше возможностей взамен большей сложности, так как теперь отсутствие разрешающего правила — не *единственный* способ получить запрет доступа, так что *порядок правил теперь имеет значение*!

Ограничение push-ей на основе изменённых файлов

Вдобавок к ограничению веток, в которые пользователю можно отправлять изменения, вы можете также ограничить файлы, которые он может трогать. Например, возможно, Makefile (или какая-то другая программа) не предназначен для изменения кем угодно, так как многие вещи зависят от него

или сломаются, если изменения не будут сделаны *правильно*. Вы можете сказать gitolite-у:

```
repo foo
                        =   @junior devs @senior devs

    RW  NAME/           =   @senior_devs

    -   NAME/Makefile   =   @junior_devs
```

Это мощное средство продокументировано в conf/example.conf.

Персональные ветки

Gitolite также имеет средство, которое называется «персональные ветки» (или даже «персональное пространство имён веток»), которое может быть весьма полезным в корпоративных средах.

Очень часто обмен кодом в мире git происходит через запросы «пожалуйста, заберите (pull)». В корпоративных средах, однако, неаутентифицированный доступ под строгим запретом, и рабочая станция разработчика не может выполнить аутентификацию. Так что вы вынуждены отправить (push) работу на центральный сервер и попросить кого-нибудь забрать (pull) её оттуда.

Это обычно вызывает такой же беспорядок с именами веток, что и в централизованных СУВ, плюс настройка прав доступа для этого становится ежедневной обязанностью админа.

Gitolite позволяет определить «персональный» или «рабочий» префикс пространства имён для каждого разработчика (например, refs/personal/<devname>/*); подробное описание есть в разделе «personal branches» в doc/3-faq-tips-etc.mkd.

«Шаблонные» репозитории

Gitolite позволяет указывать репозитории с помощью шаблонов (на самом деле регулярных выражений perl), таких как, например, assignments/s[0-9][0-

9]/a[0-9][0-9]. Это *очень* мощная функция, которая включается с помощью установки $GL_WILDREPOS = 1; в

rc файле. Она позволяет назначать новый режим доступа («C»), который позволяет пользователям создавать репозитории на основе подобных шаблонов, автоматически назначает владельцем пользователя, который создал репозиторий, позволяет ему раздавать R и RW права другим пользователям и т.п. Эта функция описана в документе doc/4-wildcard-repositories.mkd.

Другие функции

Мы закончим это обсуждение рассмотрением подборки других функций, все они и многие другие описаны в мельчайших подробностях в документе «faqs, tips, etc» и некоторых других.

Логирование: Gitolite регистрирует все успешные действия. Если вы несколько легкомысленно раздали людям права на откатывание изменений (RW+) и кто-то снёс «master», лог-файл спасёт вам жизнь, и вы легко и быстро найдёте потерянный SHA.

Git не в обычном PATH: Одна крайне полезная и удобная функция в gitolite — это поддержка git, установленного вне обычного $PATH (это совсем не такая редкость, как вы думаете; в некоторых корпоративных средах или даже у некоторых хостинг-провайдеров запрещается устанавливать что-либо в систему, и всё заканчивается тем, что вы кладёте всё в свои личные каталоги). Обычно вы вынуждены каким-то образом заставить git *на стороне клиента* учитывать это нестандартное расположение бинарников git-a. С gitolite просто выберите «подробную» установку и задайте $GIT_PATH в «rc» файлах. Никаких изменений на стороне клиента после этого не требуется.

Уведомление о правах доступа: Другая удобная функция проявляется в момент, когда вы просто проверяете и заходите по ssh на сервер. Gitolite показывает, к каким репозиториям у вас есть доступ и какого типа доступ может быть получен. Вот пример:

```
hello sitaram, the gitolite version here is v1.5.4-19-ga3397d4
     the gitolite config gives you the following access:

R       anu-wsd
R       entrans
R       git-notes
R       gitolite
R       gitolite-admin
R       indic_web_input
R       shreelipi_converter
```

Делегирование: При действительно больших установках вы можете делегировать ответственность за группы репозиториев различным людям, которые будут независимо управлять этими частями.

Это уменьшает нагрузку на главного админа и делает его не таким критичным элементом. Эта функция описана в отдельном файле в каталоге doc/.

* *Поддержка Gitweb**: Gitolite имеет поддержку gitweb в нескольких аспектах. Вы можете указать, какие репозитории видны через gitweb. Вы можете назначить «владельца» и «описание» для gitweb из конфигурационного файла для gitolite. В gitweb есть механизм организации контроля доступа через аутентификацию по HTTP, и вы можете заставить его использовать «скомпилированный» конфигурационный файл, сделанный gitolite-ом, что означает действие одинаковых правил контроля доступа (для доступа на чтение) и для gitweb, и для gitolite.

Зеркалирование: Gitolite может помочь вам поддерживать несколько зеркал, и легко переключаться между ними, если основной сервер упадёт.

Git-демон

Для публичного неаутентифицированного доступа на чтение к вашим проектам вы можете захотеть продвинуться дальше, чем протокол HTTP, и начать использовать Git-протокол. Главная причина — скорость. Git-протокол гораздо эффективнее и поэтому быстрее чем HTTP, поэтому, используя его, вы можете сэкономить вашим пользователям время.

Повторимся, это для доступа только на чтение без аутентификации. Если вы запустите его на сервере не за сетевым экраном, он должен использоваться только для проектов, которые публично видны внешнему миру. Если сервер, на котором вы его запускаете, находится за вашим сетевым экраном, вы можете использовать его для проектов, к которым большое число людей или компьютеров (серверов непрерывной интеграции или сборки) должно иметь доступ только на чтение, и если вы не хотите для каждого из них заводить SSH-ключ.

В любом случае, протокол Git относительно просто настроить. Упрощённо, вам нужно запустить следующую команду в демонизированной форме:

```
git daemon --reuseaddr --base-path=/opt/git/ /opt/git/
```

--reuseaddr позволит серверу перезапуститься без ожидания истечения старых подключений,

--base-path позволит людям не указывать полный путь, чтобы склонировать проект, а путь на конце говорит демону Git, где искать экспортируемые репозитории. Если у вас запущен сетевой экран, вы должны проколоть в нём дырочку, открыв порт 9418 на машине, на которой это всё запущено.

Вы можете демонизировать этот процесс несколькими путями, в зависимости от операционной системы. На машине с Ubuntu используйте Upstart-сценарий. Итак, в этом файле

```
/etc/event.d/local-git-daemon
```

поместите такой сценарий:

```
start on startup
stop on shutdown

                                exec /usr/bin/git daemon \

                                   --user=git --group=git \

                                        --reuseaddr \

                                   --base-path=/opt/git/ \

    /opt/git/
   respawn
```

По соображениям безопасности крайне приветствуется, если вы будете запускать этого демона как пользователя с правами только на чтение на репозитории — вы легко можете сделать это, создав пользователя ‘git-ro’ и запустив этого демона из-под него. Для простоты мы запустим его от того же пользователя ‘git’, от которого запущен Gitosis.

Когда вы перезапустите машину, Git-демон запустится автоматически, и возродится, если вдруг завершится. Чтобы запустить его без перезагрузки машины, выполните следующее:

```
                                initctl start local-git-daemon
```

На других системах вы можете использовать xinetd, сценарий вашей системы sysvinit, или что-то другое — главное, чтобы вы могли эту команду как-либо демонизировать и перезапускать в случае завершения.

Затем, вы должны указать Gitosis-серверу, к каким репозиториям предоставить неаутентифицированный доступ через Git-сервер. Если вы добавили по секции для каждого репозитория, вы можете указать, из каких из них Git-демону позволено читать. Если вы хотите предоставить доступ через Git-протокол к вашему проекту iphone, добавьте это в конец вашего файла gitosis.conf:

```
[repo iphone_project]
    daemon = yes
```

Когда это зафиксировано и отправлено, ваш запущенный демон должен начать обслуживать запросы к проекту от всех, у кого есть доступ к порту 9418 на вашем сервере.

Если вы решили не использовать Gitosis, но хотите установить Git-демон, вы должны выполнить следующее в каждом проекте, который должен обслуживаться Git-демоном:

```
$ cd /path/to/project.git

$ touch git-daemon-export-ok
```

Наличие этого файла скажет Git'у, что можно обслуживать этот проект без аутентификации.

Gitosis также может контролировать, какие проекты будет показывать GitWeb. Вам нужно добавить что-то вроде этого в файл /etc/gitweb.conf:

```
$projects_list = "/home/git/gitosis/projects.list";

$projectroot = "/home/git/repositories";

$export_ok = "git-daemon-export-ok";

@git_base_url_list = ('git://gitserver');
```

Вы можете контролировать, какие проекты GitWeb будет позволять просматривать пользователям, добавляя или удаляя пункт настройки gitweb в конфигурационном файле Gitosis. Например,

если вы хотите, чтобы ваш проект iphone просматривался в GitWeb, сделайте, чтобы секция

настроек repo выглядела следующим образом:

```
[repo iphone_project]
    daemon = yes

                                    gitweb = yes
```

Теперь, если вы зафиксируете и отправите изменения, GitWeb автоматически начнёт показывать ваш проект iphone.

Git-хостинг

Если вы не хотите связываться со всей работой по установке собственного Git-сервера, у вас есть несколько вариантов размещения ваших Git-проектов на внешних специальных хостинг сайтах. Это предоставляет множество преимуществ: на хостинг сайте обычно быстро настроить и запустить проект и нет никакого мониторинга или поддержки сервера. Даже

если вы установили и запустили свой собственный внутренний сервер, вы можете захотеть

использовать публичный хостинг сайт для вашего открытого кода — обычно сообществу открытого кода так будет проще вас найти и помочь.

В наши дни у вас есть огромное количество вариантов хостинга на выбор, все со своими преимуществами и недостатками. Чтобы увидеть актуальный список, проверьте страницу GitHosting в главной вики Git: http://git.or.cz/gitwiki/GitHosting

Поскольку мы не можем рассмотреть их все, и поскольку я работаю на один из них, мы в этом разделе рассмотрим процесс создания учётной записи и нового проекта на GitHub. Это даст вам представление о вовлечённых в него вещах.

GitHub — крупнейший на сегодняшний день сайт, предоставляющий Git-хостинг для проектов с открытым исходным кодом, а также один из немногих, предоставляющих одновременно и публичный, и приватный хостинг, так что вы можете хранить ваш открытый и коммерческий

код в одном месте. На самом деле, мы использовали GitHub, чтобы закрыто совместно писать эту книгу (прим. переводчика: и открыто переводить после её издания).

GitHub

GitHub немного отличается от других хостингов кода способом группировки проектов. Вместо того, чтобы брать за основу проекты, GitHub ориентируется на пользователей. Это значит, что если я размещаю свой проект grit на GitHub, вы не найдёте его в github.com/ grit, он будет в github.com/schacon/grit. Здесь нет никакой канонической версии проекта, что позволяет проектам беспрепятственно переходить от одного пользователя к другому, если начальный автор забросил проект.

GitHub — это коммерческая компания, которая взимает плату с учётных записей, использующих приватные репозитории, но любой может хоть сейчас

получить бесплатную учётную запись и разместить сколько ему угодно открытых проектов. Мы быстро рассмотрим, как это делается.

Настройка учётной записи

Первое, что вам нужно сделать, это настроить учётную запись. Если вы посетите страницу Plans & Pricing по адресу http://github.com/plans и нажмёте на кнопку «Create a free account» (см. рисунок 4-2), вы попадёте на страницу регистрации.

Рисунок 4.2: Страница тарифных планов GitHub.

Здесь вы должны выбрать имя пользователя, которое ещё не занято в системе, ввести адрес электронной почты, который будет сопоставлен аккаунту, и пароль (см. рис. 4-3).

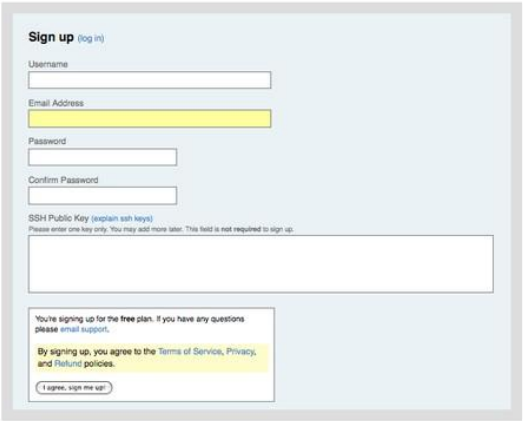

Рисунок 4.3: Страница регистрации пользователя GitHub.

Если есть возможность, сейчас также самое время добавить свой открытый SSH-ключ.

Мы рассмотрели, как создать ключ, ранее, в разделе «Создание открытого SSH-ключа». Возьмите содержимое открытого ключа из своей пары и вставьте в поле для ввода открытого SSH-ключа.

Ссылка «explain ssh keys» направит вас к подробным инструкциям о том, как это сделать на всех основных операционных системах. Нажатие на кнопку «I agree, sign me up» откроет инструментальную панель вашего нового пользователя (см. рис. 4-4).

Рисунок 4.4: Инструментальная панель на GitHub.

После этого вы можете создать новый репозиторий.

Создание нового репозитория

Начните с нажатия на «New repository» рядом с разделом «Your Repositories» на странице инструментальной панели. Вы попадёте к форме для создания нового репозитория (см. рис. 4-5).

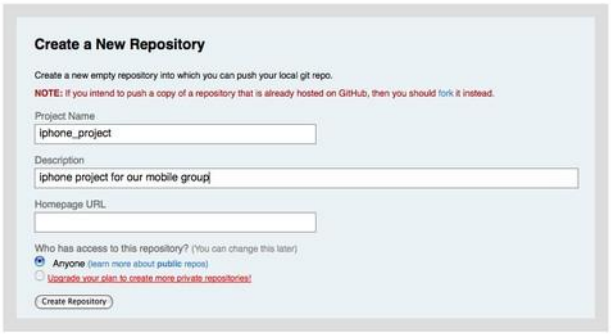

Рисунок 4.5: Создание нового репозитория на GitHub.

Единственное, что вам обязательно нужно сделать, это указать имя проекта, но вы также можете добавить и описание. Когда сделаете это, нажмите на кнопку «Create Repository». Теперь у вас есть новый репозиторий на GitHub (см. рис. 4-6).

Рисунок 4.6: Заглавная информация проекта GitHub.

Поскольку у вас ещё нет кода, GitHub покажет вам инструкцию, как создать совершенно новый проект, отправить существующий или импортировать проект из публичного репозитория Subversion (см. рис. 4-7).

Рисунок 4.7: Инструкции для нового репозитория.

Эти инструкции похожи на то, что мы проходили раньше. Чтобы инициализировать проект, если это ещё не Git-проект, используйте:

```
$ git init

$ git add .

$ git commit -m 'initial commit'
```

Если у вас есть локальный Git-репозиторий, добавьте GitHub как удалённый сервер и отправьте туда свою ветку master:

```
$ git remote add origin git@github.com:testinguser/iphone_project.git

$ git push origin master
```

Теперь ваш проект размещён на GitHub, и вы можете дать ссылку на него любому, с кем вы захотите разделить проект. В этом случае, это http://github.com/testinguser/ iphone_project. Вы также можете видеть в заголовке каждой страницы проекта, что у вас две Git-ссылки (см. рис. 4-8).

Рисунок 4.8: Заголовок проекта с публичной и приватной ссылками.

Ссылка «Public Clone URL» — это публичная ссылка только для чтения, через которую кто угодно может склонировать проект. Можете опубликовать эту ссылку или разместить её на своём сайте — где угодно.

«Your Clone URL» — это SSH-ссылка на чтение и запись, через которую вы можете читать и писать только в том случае, если вы подключаетесь с использованием секретного ключа из пары открытого SSH-ключа, загруженного в вашу учётную запись. Если другие пользователи посетят страницу этого проекта, они не увидят этой ссылки — только публичную.

Импорт из Subversion

Если у вас есть существующий публичный Subversion-проект, который вы хотите импортировать в Git, GitHub часто может сделать это за вас. Внизу страницы инструкций есть ссылка на импорт из Subversion. Если вы кликнете по ней, вы увидите форму с информацией о процессе импорта и текстовое поле, где вы можете вставить ссылку на ваш публичный Subversion-проект (см. рис. 4-9).

Если ваш проект очень большой, нестандартный или приватный, этот процесс, возможно, не сработает для вас. В главе 7 вы узнаете, как делать более сложные импорты вручную.

Добавление участников

Давайте добавим остальную команду. Если Джон, Джози и Джессика зарегистрированы на GitHub, и вы хотите дать им доступ на отправку изменений в свой репозиторий, вы можете добавить их в свой проект как участников. Это позволит им отправлять изменения, используя свои открытые ключи.

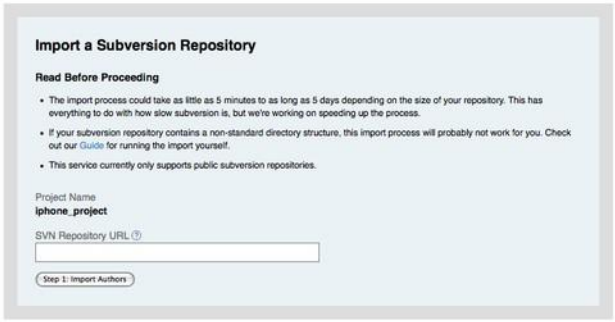

Рисунок 4.9: Интерфейс импорта из Subversion.

Нажмите на кнопку «edit» в заголовке проекта или вкладку «Admin» вверху, чтобы попасть на страницу администратора вашего проекта на GitHub (см. рис. 4-10).

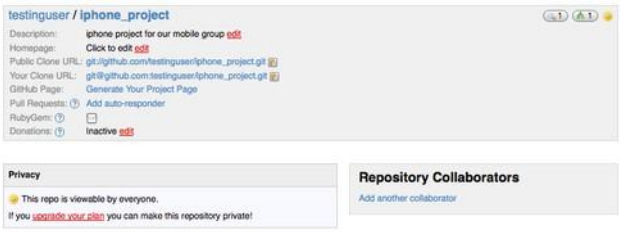

Рисунок 4.10: Страница администратора на GitHub.

Чтобы дать другому пользователю доступ на запись в проект, кликните по ссылке «Add an- other collaborator». Появится новое текстовое поле, в котором вы можете набрать имя пользователя. По мере набора всплывёт подсказка, показывающая возможные совпадения имён. Когда найдёте нужного пользователя, нажмите на кнопку Add, чтобы добавить пользователя как участника вашего проекта (см. рис. 4-11).

Рисунок 4.11: Добавление участника в проект.

Когда закончите добавлять участников, вы должны увидеть их список в разделе Repository Collaborators (см. рис. 4-12).

Если вам нужно отозвать чей-то доступ, можете кликнуть по ссылке «revoke», и его доступ

на отправку будет удалён. Для будущих проектов вы также можете скопировать группы участников, скопировав права доступа из существующего проекта.

Рисунок 4.12: Список участников вашего проекта.

Ваш проект

После того как вы отправили ваш проект или импортировали его из Subversion, у вас есть главная страница проекта, которая выглядит как на рис. 4-13.

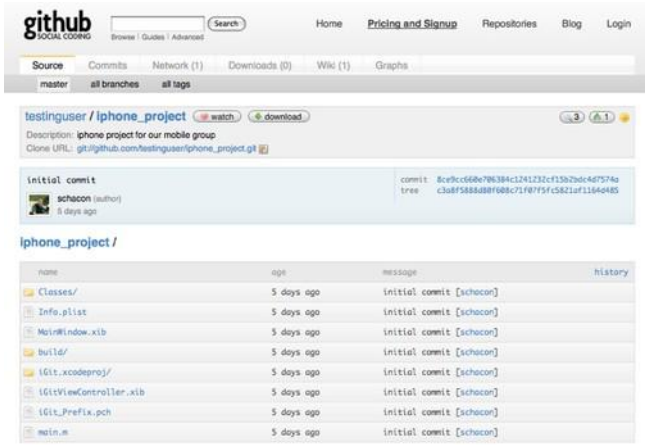

Рисунок 4.13: Главная страница проекта на GitHub.

Когда люди посещают ваш проект, они видят эту страницу. Она содержит вкладки, касающиеся различных аспектов вашего проекта. Вкладка Commits показывает список коммитов в обратном хронологическом порядке наподобие вывода команды git log. Вкладка Network показывает всех людей, отделивших ваш проект и вернувших свои наработки. Вкладка Downloads позволяет выложить бинарные файлы проекта и ссылки на архивы каких-нибудь отмеченных точек проекта. Вкладка Wiki предоставляет вики, где вы можете написать документацию или другую информацию о своём проекте. Вкладка

Graphs показывает некоторую информацию о вкладе участников и статистику проекта. Главная вкладка Source показывает листинг корневого каталога проекта и автоматически выводит под ним содержимое файла README, если он у вас есть. Эта вкладка также показывает информацию о последнем коммите.

Ответвления проектов

Если вы хотите внести вклад в существующий проект, в который у вас нет права на отправку изменений, GitHub приветствует ответвления. Когда вы смотрите на страницу заинтересовавшего вас проекта и хотите немного поработать над ним, вы можете нажать на кнопку «Fork» в заголовке проекта, чтобы GitHub скопировал проект вашему пользователю, и вы смогли отправлять туда свои изменения.

Таким образом, проектам не нужно беспокоиться о добавлении пользователей в качестве участников для предоставления им доступа на отправку изменений. Люди могут ответвить проект и отправлять изменения в свою копию. А мейнтейнер главного проекта может вернуть эти изменения, добавляя форки как удалённые серверы и сливая из них наработки.

Чтобы ответвить проект, посетите страницу проекта (в нашем случае mojombo/chronic) и нажмите на кнопку «Fork» в его заголовке (см. рис. 4-14).

Рисунок 4.14: Получение доступной для записи копии любого репозитория.

Через несколько секунд вы будете направлены на страницу своего нового проекта, на которой указано, что данный проект является ответвлением другого проекта (см. рис. 4-15).

Рисунок 4.15: Вы ответвили проект.

Заключение о GitHub

Это всё, что мы хотели бы рассказать про GitHub, но важно отметить, как быстро вы можете всё сделать. Вы можете создать аккаунт, добавить новый проект и отправить изменения в него за минуты. Если ваш проект с открытым исходным кодом, вы также получите огромное сообщество разработчиков, которые смогут увидеть ваш проект, ответвить его и внести в него свой вклад. Во всяком случае, это хороший способ получить готовую к работе с Git среду, чтобы быстренько испытать его в деле.

Итоги

У вас есть несколько вариантов получения удалённого Git-репозитория так, чтобы вы могли принимать участие в проекте вместе с другими или поделиться работой.

Запуск своего сервера даёт полный контроль и позволяет запускать сервер за вашим сетевым экраном, но такой сервер обычно требует значительной части вашего времени на настройку и поддержку. Если вы разместите ваши данные на хостинге, его просто настроить и поддерживать; однако вам необходимо иметь возможность хранить код на чужом сервере, а некоторые организации этого не позволяют.

Выбор решения или комбинации решений, которые подойдут вам и вашей организации, не должен вызвать затруднений.