К.А. Корниенко

FreeBSD 9

Корпоративный Интернет-сервер

Киев - 2013

УДК 004.45
ББК 32.973-018.2
К 60

Корниенко К.А.
К 60 FreeBSD 9. Корпоративный Интернет-сервер. –
К.: ФЛП Декет В.М., 2013 – 176 с.

ISBN 966-8637-57-7

Эта книга – практическое руководство о том, как быстро реализовать эффективную и управляемую систему для совместной работы в сети Интернет на базе операционной системы FreeBSD версии 9. Здесь рассматриваются практические вопросы о том, как проинсталлировать систему, настроить сетевые службы, сконфигурировать защищенный шлюз, почтовый сервер, спам-фильтр, веб-сервер и т.д. Все примеры протестированы и опробованы в реальных условиях.
Для широкого круга пользователей и системных администраторов.

УДК 004.45
ББК 32.973-018.2

ISBN 966-8637-57-7

Оглавление

Введение

Краткая история FreeBSD

У истоков операционной системы UNIX, с которой собственно все и началось, стояла лаборатория Bell Labs компании AT&T. Два человека – Кен Томпсон (Ken Thompson) и Деннис Ритчи (Dennis Ritchie) были главной движущей силой развития UNIX, которая в принципе родилась случайно. В середине 60-х годов AT&T Bell Labs совместно с другими компаниями прикладывала немало усилий для разработки новой операционной системы под названием Multics. Ее предполагалось использовать для крупномасштабных вычислений, которые выполнялись на машине класса мэйнфрейм. Кен Томпсон написал небольшую компьютерную игру, но ему не нравились ни производительность мэйнфрейма, ни стоимость машинного времени. С помощью Денниса Ритчи он переписал эту игру для работы на компьютере DEC PDP-7 и, по ходу дела, написал целую операционную систему. Весной 1969 года Bell Labs вышла из проекта, и программисты остались без вычислительной среды, однако, к этому моменту они уже разработали базовую структуру файловой системы, которая впоследствии превратилась в файловую систему UNIX. Первые версии UNIX были написаны на языке ассемблер, но уже в 1973 году UNIX была переписана на C – совершенно новом языке программирования, разработанном Ритчи. Создание языка программирования C и системы UNIX – две самые важные вехи в истории компьютерной индустрии. Язык C стал первым мультиплатформенным языком, который позволил относительно легко переносить как приложения, написанные на нем, так и саму

систему UNIX между различными компьютерными платформами. Это одна из многих возможностей, благодаря которым система UNIX стала такой популярной.

AT&T в принципе не занималась компьютерным бизнесом отчасти потому, что в то время это было монополией правительства. Поэтому AT&T предложила UNIX в виде исходных кодов правительственным учреждениям и университетам за сравнительно небольшую плату. Вот так система UNIX попала в 80% университетов, имевших компьютерные факультеты. Одной из первых организаций, вплотную занявшихся работой над UNIX, стала группа из Калифорнийского университета в Беркли – Computer Systems Research Group. Этому способствовал и тот факт, что в 1975 году Кен Томпсон оставил Bell Labs и перешел в отдел компьютерных исследований в Беркли. В работе над расширением системы ему активно помогал студент-выпускник Билл Джой (Bill Joy). Измененная и скорректированная в университете версия UNIX была выпущена под названием Berkley Software Distribution, или BSD. А в конце 70-х годов произошло важное событие: Министерство обороны США объявило, что ее подразделение Advanced Research Project Agency будет использовать UNIX и что в качестве базовой принята версия разработчиков из Беркли. Одним из требований, поставленных министерством обороны, была возможность работы в сети и высокая устойчивость системы. Так, благодаря военным, UNIX стала продвигаться вперед по пути совершенствования. В это время Билл Джой оставил университетский городок и основал компанию Sun Microsystems. Рабочие станции Sun использовали версию операционной системы, производную от BSD и известную как SunOS. Большая часть исходного кода BSD была доступна пользователям бесплатно и в 1991 году BSD была портирована на платформу Intel x86, а в Калифорнийском университете образовалась новая группа, которая начала продавать коммерческую версию BSD для платформы x86.

В 1993 году две совершенно разные группы одновременно пришли к выводу, что UNIX заслуживает большего внимания. В результате были созданы два новых проекта. Результатом первого проекта стала операционная система NetBSD. Здесь основное внимание уделялось доступности и универсальности системы. Если существует аппаратная платформа, то наверняка имеется и работающая на ней версия NetBSD. Второй проект породил FreeBSD. В этой разработке внимание было сконцентрировано на том, чтобы система стала проще в использовании. Иначе говоря, эта система была ориентирована на широкий круг пользователей и на платформу Intel x86. Сегодня FreeBSD – самая известная UNIX-система из семейства BSD.

Преимущества FreeBSD

Главным преимуществом FreeBSD мы, безусловно, считаем ее стабильность. По данным компании Netcraft (netcraft.com), изучавшей сайты с самым продолжительным календарным временем беспрерывной работы, из 50 первых в ее списке сайтов 47 функционирует под управлением FreeBSD. С момента последней перезагрузки веб-сервера №1 прошло уже более 10 лет! И, конечно же, он работает под FreeBSD.

Второе неоспоримое преимущество – это то, что FreeBSD бесплатная система, не обременяющая пользователя дорогой лицензией. Вы можете бесплатно установить копию FreeBSD на всех своих компьютерах, сколько бы их ни было. Если вы устанавливаете сервер, вам не потребуется платить за каждое дополнительное сетевое подключение, как в большинстве коммерческих операционных систем.

Третье – это доступность тысяч бесплатных пакетов прикладных программ всегда в режиме онлайн! Эти свободно распространяемые программы элементарно устанавливаются из дерева портов всего двумя командами – система сама найдет нужные дистрибутивы и патчи в сети и скомпилирует все, что нужно на вашу FreeBSD. Требуется только подключение к сети Интернет.

Четвертое – это открытость системы. Доступно все дерево исходного кода FreeBSD, в код можно вносить изменения, выполнять любые проверки и т.д. Ну и наконец, показателем качества является то, что системой FreeBSD пользуются крупнейшие компании и интенсивно используемые веб-сервисы.

Теперь сравним возможности ОС FreeBSD с возможностями Windows и Linux. Microsoft поступила гениально, разработав операционную систему, которой может пользоваться кто угодно. Windows способна выполнять разнообразные задачи, не требуя от пользователя глубоких знаний внутреннего функционирования системы. С одной стороны, современная Windows отвечает самым высоким требованиям, предъявляемым к техническим средствам, однако многим пользователям они совершенно ни к чему. С другой стороны, Windows не имеет интерфейса к целому ряду имеющихся в ней возможностей и тонких настроек. Попросту говоря, Windows предлагает графический интерфейс для решения большинства задач.

Система FreeBSD основана на командной строке. В ней для настройки используются текстовые конфигурационные файлы, редактируя которые можно выполнить необходимые действия быстро и точно. Графический интерфейс – неотъемлемая примета Windows, тогда как в FreeBSD можно и вовсе обойтись без него. Специалисту не нужны окошки для доступа к серверу, который стоит в дальней комнате. FreeBSD позволяет выполнять

абсолютно все необходимые операции по администрированию, используя исключительно командную строку с физической консоли либо с удаленного терминала, даже самого простейшего и базирующегося на любой другой платформе. Хотя Windows тоже можно администрировать удаленно, но для этого нужно специальное программное обеспечение, которое подходит исключительно для Windows, а это значит, что задачи удаленного администрирования Windows-систем можно выполнять только с другой системы Windows. Кроме того, графический интерфейс имеет ряд ограничений, от которых свободна командная строка. В начале может показаться, что работать с командной строкой сложно, но очень скоро вы поймете, что набрать нужную команду гораздо быстрее, чем добиться того же самого эффекта с помощью системы разветвленных меню.

Второе основное отличие – ядро Windows невозможно изменить. Ядро – это сердце операционной системы, оно контролирует все аспекты ее работы. FreeBSD позволяет создать новое ядро, которое будет максимально соответствовать назначению конкретной операционной системы. Благодаря этому возрастает быстродействие и снижаются требования к аппаратным ресурсам. В Windows же предпочтение отдается простоте эксплуатации, а не производительности и эффективному использованию аппаратных средств.

Что касается Linux, то об этой системе сейчас знают все, впрочем, как и о Windows. В последнее время она стала особенно популярной. Фактически Linux – это клон UNIX. Как и FreeBSD, это открытая операционная система, разработанная добровольцами из разных стран мира. У FreeBSD и Linux много общего, однако, для Linux создано больше программ, чем для FreeBSD, но последняя, в свою очередь, позволяет запускать практически все программы, разработанные для Linux. Более того, под FreeBSD они работают даже быстрее. Что касается политики разработки и

поддержки, то у FreeBSD только один дистрибьютор, а у Linux их более 300. FreeBSD будет работать одинаково на любой системе. В случае с Linux это не так. У каждого дистрибьютора свой подход, что нередко вводит пользователей в заблуждение, когда они переходят с одного дистрибутива Linux на другой. FreeBSD является полноценной операционной системой, поддерживаемой основным составом. Linux – это только ядро, поддерживаемое Линусом Торвальдсом (создателем Linux). Компании, занимающиеся распространением Linux, комплектуют свои дистрибутивы целым рядом программ, специально разработанных для Linux и, естественно, каждый дистрибьютор имеет собственное мнение относительно того, что должно входить в дистрибутив. Кроме того, процесс обновления кода FreeBSD отслеживается и координируется намного тщательнее, чем в Linux. Для большинства пользователей это позитивное явление, поскольку они уверены в том, что код был протестирован специалистами на отсутствие проблем. Поскольку в системе FreeBSD поддерживается одно дерево исходного кода, она стабильнее Linux и в большей степени соответствует производственным целям. Основным недостатком FreeBSD, вызванным таким подходом, является то, что нововведения допускаются в систему медленней, чем в Linux. Но есть выбор: либо вы предпочтете стабильность и неприступность операционной системы, либо остановите свой выбор на модных вещичках и новых игровых устройствах, пожертвовав ради этого надежностью.

О чем и для кого эта книга

Эта книга о том, как с минимальными временными затратами построить эффективную и управляемую систему для совместной работы пользователей с Интернет-ресурсами, то есть корпоративный Интернет-сервер, на базе операционной системы FreeBSD версии 9. Здесь рассматриваются вопросы о том, как проинсталлировать систему, настроить сетевые службы, сконфигурировать защищенный шлюз, почтовый сервер, спам-фильтр, веб-сервер и так далее.

Специфика и преимущество этой книги заключается в том, что она ориентирована на практическое решение задач. Она не является подробным описанием возможностей операционной системы. Это – не справочник и не энциклопедия с большой долей «воды» и теории. Это – пошаговое практическое руководство по быстрой реализации тех или иных серверных функций. Именно поэтому вы здесь найдете максимум пользы.

Книгу нужно читать последовательно от одной главы к другой, т.к. часть информации в последующих разделах завязана на предыдущие. Так же в процессе чтения вы будете постепенно набирать опыт, поэтому, перескочив через какой-либо раздел, вы можете пропустить описание очередной новой команды или другую ценную информацию. После прочтения всей книги целиком, вы сможете построить все, что захотите по-своему.

Адресована книга прежде всего системным администраторам, у которых есть базовый опыт администрирования компьютерных сетей, но нет опыта работы с UNIX, либо он невелик. Цель книги – показать практическую реализацию на базе операционной системы FreeBSD функций Интернет-шлюза, прокси-сервера, фаерволла, почтового и веб-серверов, а так же сопутствующих им дополнениям, вплоть до программирования

собственных скриптов. Ваша цель – сделать это все в реальных условиях на вашем собственном Интернет-сервере, в результате чего вы почувствуете себя уверенно при работе с FreeBSD и получите необходимый фундамент для дальнейшего ее изучения. Именно поэтому здесь вы найдете максимум практики.

В процессе работы у вас могут возникнуть дополнительные вопросы, касающиеся работы самой операционной системы FreeBSD. Эту информацию вы всегда можете получить онлайн на официальном сайте в разделе «Руководство FreeBSD»:

http://www.freebsd.org/doc/ru/books/handbook/

Аналогичным образом, при изучении, например, Postfix или Apache, если захотите более тонко настроить сервер, вооружитесь более объемными, подробными и узконаправленными описаниями этих продуктов. Повторим, что здесь – конкретная реализация широкого спектра конкретных задач. Практика, которая работает. Любые шаги влево, вправо, полет фантазии и тонкая настройка – на усмотрение читателя при наличии дополнительных источников информации.

Благодарности

Автор книги выражает свою благодарность:

Всем, кому интересно. Без вас этот труд не имел бы смысла;

Ольге Папановой, которая 12 лет назад помогла автору запустить его первый FreeBSD сервер, когда он, автор, был еще «чайником».

Глава 1
Инсталляция FreeBSD 9

Итак, перед нами стоит глобальная задача: организовать совместную работу пользователей локальной сети в Интернет. При этом локальная сеть должна быть защищена от внешних вторжений, а система должна быть управляемой и гибкой. В перспективе мы захотим себе и прокси-сервер и электронную почту и веб-интерфейсы, но начнем мы с настройки Интернет-шлюза. Будем считать, что операционную систему вы уже выбрали, теперь нужно ее установить и настроить. Таким образом, для решения этой задачи нам понадобятся всего лишь три вещи: железо, софт и книга. Третье перед вами, второе скачаем, найдите первое и вперед.

Исходные данные и постановка задачи

Давайте сразу определим, что мы должны иметь для успешной реализации наших планов. Допустим, на входе мы имеем локальную сеть небольшого офиса, домен и некоторое железо. Опишем их:

1. Локальная компьютерная сеть **10.0.0.0/24**;
2. Подключение к Интернет **22.22.22.20/30**, то есть:
   - IP-address: **22.22.22.22**;
   - Gateway: **22.22.22.21**;
   - DNS: **22.22.0.1**, **22.22.0.2**;
3. Корпоративный Интернет-домен «**example.com**»;
4. Сервер или системный блок с двумя сетевыми картами.

Здесь необходимо отметить, что IP-адрес **22.22.22.22**, который нам якобы выдал провайдер, является вымышленным и любые совпадения случайны. Так же автор надеется на то, что читатель понимает форму записи IP-адресов и сетей, которая была предложена выше.

Мы собираемся установить Интернет-шлюз, одна сетевая карта которого будет подключена к Интернет, а другая – к локальной сети. Таким образом, мы должны получить некий маршрутизатор, который будет пропускать через себя весь внешний трафик.

Пятым пунктом можно было бы дописать «системного администратора, которому не нужно объяснять, как установить операционную систему из образа диска формата ISO», но автор надеется на то, что этот пункт у вас присутствует по умолчанию :)

Приступим же к установке системы.

Выбор инсталляционного пакета

Образы инсталляционных дисков можно скачать с официального сайта FreeBSD:

ftp://ftp.freebsd.org/pub/FreeBSD/releases/ISO-IMAGES/

Далее следует номер стабильного релиза. Мы будем использовать версию 9.1 – это последняя версия на момент написания книги.

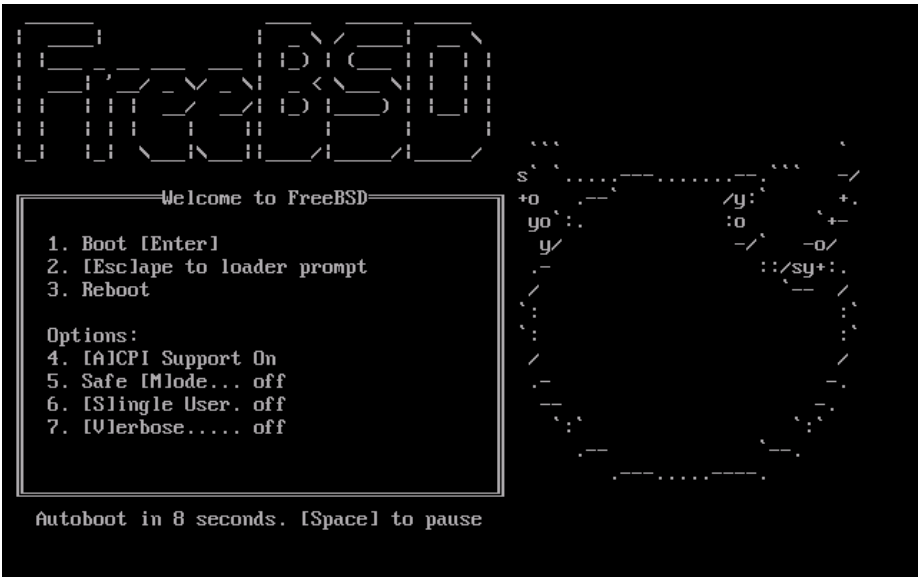
Необходимо правильно выбрать тип инсталляционного пакета, который зависит от платформы, на которую вы будете устанавливать FreeBSD. В основном, нас интересуют два варианта: i386 и amd64.

ПРИМЕЧАНИЕ. Часто упоминание аббревиатуры AMD вводит людей в заблуждение, вплоть до того, что они отказываются использовать родные версии операционных систем, мотивируя это тем, что на процессоре Intel версия для AMD работать не будет. На самом деле распространители ПО используют название amd64 лишь потому, что именно AMD была пионером в разработке этой технологии – так случилось. Так же, часто пользователи путают архитектуру Intel 64 с ia64, которые на самом деле являются совершенно разными и несовместимыми между собой.

Итак, выбираем i386 для 32-разрядной системы или amd64 для 64-разрядной, независимо от типа процессора (AMD или Intel). В книге мы будем устанавливать версию amd64. Далее мы должны определиться с носителем, с которого будем устанавливать нашу ОС (CD, DVD, flash) и скачать нужный нам образ. После этого записываем его на соответствующий носитель, вставляем его в предполагаемый сервер (физический или виртуальный), загружаемся и переходим к следующему разделу.

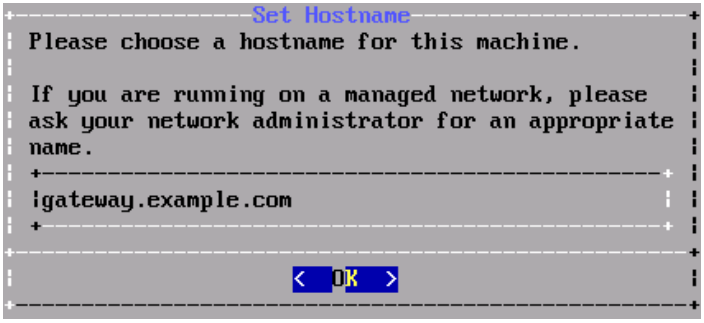
Установка операционной системы

В процессе загрузки мы должны увидеть приветствие и меню для выбора вариантов работы (рис. 1).

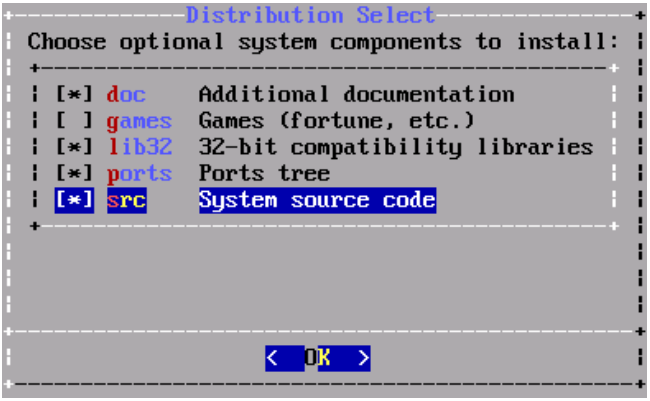


Здесь ничего делать не нужно. Можно просто нажать **[Enter]** и сэкономить 10 секунд времени. Далее мы должны попасть в программу **bsdinstall**, которая используется начиная с версии 9.0 (рис. 2). В более ранних версиях использовался установщик **sysinstall**, но здесь мы его рассматривать не будем, т.к. делаем упор на новую версию системы.

Нажимаем **[Install]**, после чего, на вопрос «хотим ли мы использовать нестандартную раскладку клавиатуры», отвечаем **[No]** и видим окно, где нам предлагают ввести имя нашего сервера. Желательно использовать домен, который у нас есть, ну а имя хоста – дело вкуса (рис. 3).

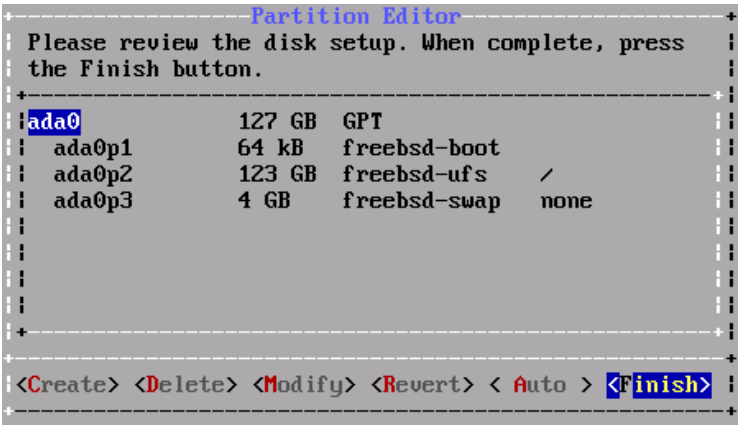


Выбор имени хоста, особенно, если это ваш первый FreeBSD сервер, является важным психологическим моментом, ведь «как вы лодку назовете, так она и поплывет» (с). В книге же мы будем использовать скромное **gateway.example.com**.

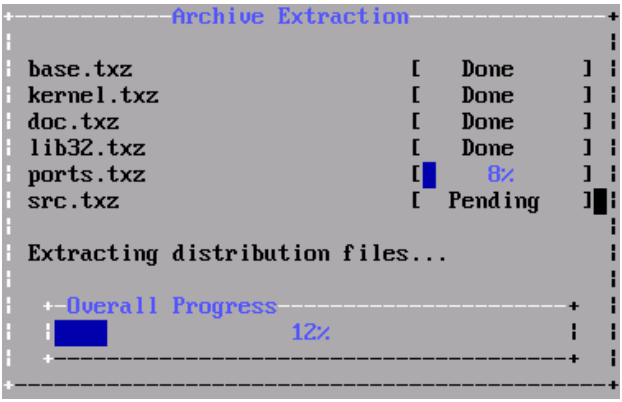


Далее нам предложат выбрать компоненты системы (рис. 4) – выбираем все, кроме игр, после чего переходим к формированию разделов жесткого диска (рис. 5).

Здесь мы пойдем самым коротким и быстрым путем и разрешим установщику все сделать самостоятельно – нажимаем **[Guided]** и на следующий вопрос «использовать ли все дисковое пространство» отвечаем согласием – **[Entire Disk]**. После этого мы должны увидеть план разбивки жесткого диска (рис. 6).



Соглашаемся с предложением, нажимая **[Finish]**, а затем выражаем понимание происходящего и свою твердую уверенность продолжать несмотря на предупреждение о возможной потере текущих данных на винчестере – **[Commit]**. После этого будет сформирована новая таблица разделов и файловая система на вашем жестком диске и начнется непосредственная установка FreeBSD (рис. 7).

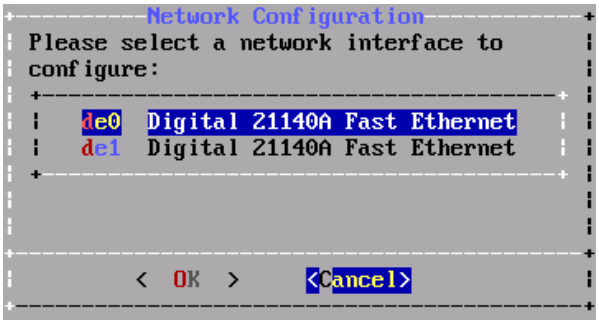


Этот процесс, в зависимости от общей производительности вашей системы, займет около 8 минут. Вы можете зафиксировать свое время, которое потратите на установку и, даже если оно будет отличаться от нашего, то в дальнейшем вы сможете предположить планируемое затратное время для своей системы, когда мы будем о нем писать, например, при сборке ядра или установке различных программ, рассчитывая его пропорциональным образом.

Когда установка будет завершена, вы увидите предложение создать пароль для администратора системы – пользователя **root**. Не создавайте слишком простых паролей для этой учетной записи, однако, пароль не должен быть настолько сложен, чтобы вы его не смогли запомнить и ввести повторно :)

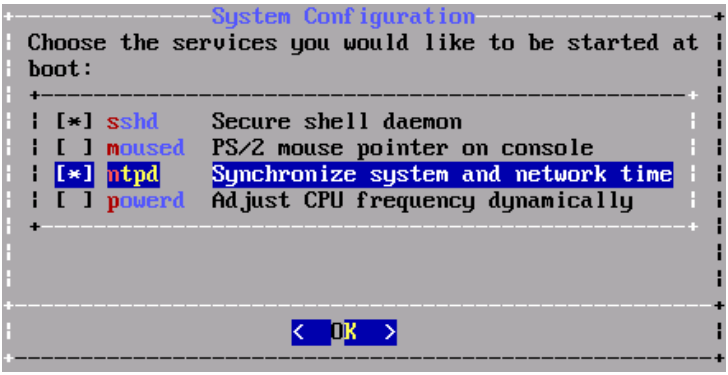
Первичные настройки после установки

Сразу после создания пароля рута мы возвращаемся в **bsdinstall** (рис. 8), где нам предлагают настроить сетевые интерфейсы. В нашем случае система определила две сетевые карты **de0** и **de1**. Можно настроить их сразу, но мы сделаем это позже, чтобы наглядно увидеть, как настраивается сеть из командной строки и где сохраняются эти настройки – **[Cancel]**.

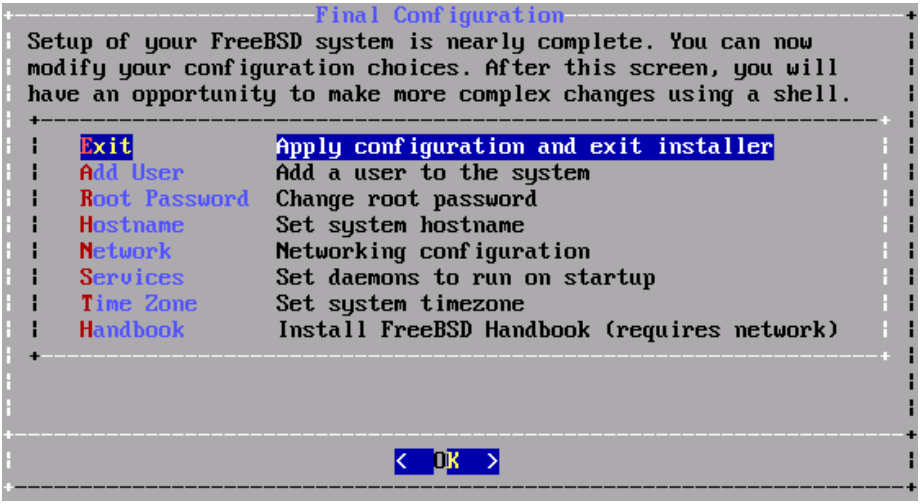


Далее нужно определить свое географическое расположение для того, чтобы правильно зафиксировать часовой пояс. Не стоит пренебрегать этой настройкой, т.к. потом, при работе с электронной почтой, это окажется важным. Итак, на вопрос «синхронизированы ли CMOS часы этого компьютера с UTC» (всемирным координированным временем) отвечаем **[No]**, а затем выбираем свой континент, страну и месторасположение, после чего подтверждаем правильность своего выбора – **[Yes]**.

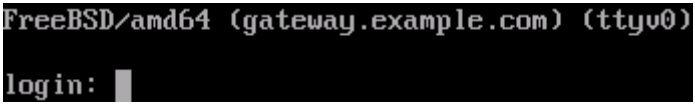
После этого, нам должны предложить установить сразу некоторые службы: **sshd** для удаленного управления, **moused** для подключения мыши, **ntpd** для синхронизации времени и **powerd** для управления процессором. Реально нужны только две – **sshd** и **ntpd**. Отмечаем их и нажимаем **[OK]** (рис. 9).



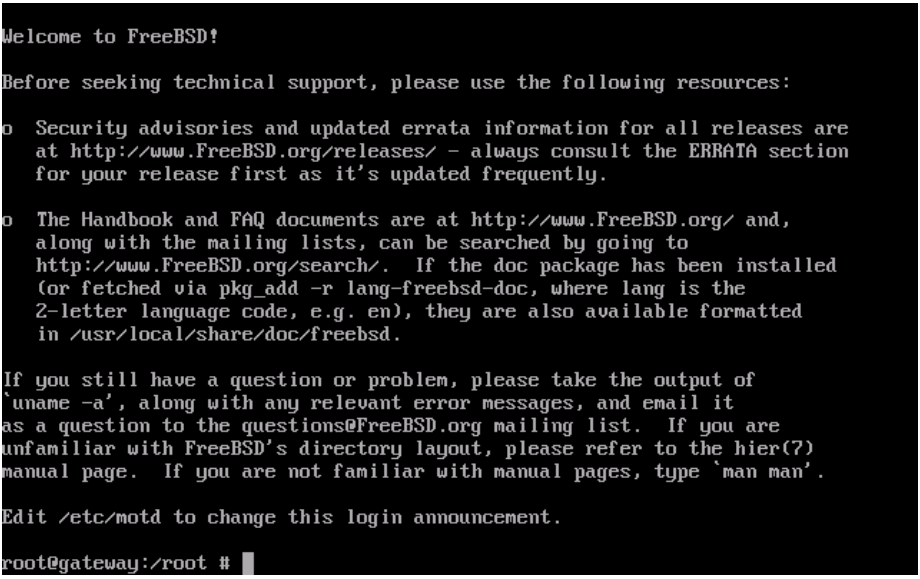
На вопрос «желаем ли мы включить запись аварийных дампов памяти» отвечаем **[No]**, так как они со временем занимают очень много места на жестком диске, а реальная польза от них лично для вас врядли будет. Предложение «создать сейчас в системе учетные записи пользователей» так же вежливо отклоняем – **[No]**. Мы так же сделаем это позже. Затем нам предложат проверить и, при необходимости, исправить все предыдущие настройки (рис. 10).



Вряд ли нам это нужно, поэтому выделяем самый верхний пункт **[Exit]** и нажимаем **[OK]**. На вопрос «хотим ли мы выполнить какие-либо последние изменения из командной строки» отвечаем **[No]**, и на долгожданное предложение перезагрузиться в только что установленную систему отвечаем – **[Reboot]**. Не забываем вытащить установочный диск, чтобы не загружаться с него снова. После перезагрузки мы должны увидеть приглашение для ввода имени пользователя (рис. 11).



Вводим логин **root**, пароль, который создали при установке, и попадаем в систему (рис. 12).



Поздравляем! Вы только что установили операционную систему FreeBSD на свой сервер и подготовили его к дальнейшей настройке!

Глава 2
Подготовка к эффективной работе

А сейчас немного подготовительной практики. Для того, чтобы эффективно работать в операционной системе FreeBSD дальше, нужно понять несколько базовых вещей, а именно: как работать в командной строке, как работать с файлами и правами доступа к ним, как работать с текстовыми файлами и как работать со справочной системой FreeBSD. То есть, эта глава будет посвящена в основном ответам на многочисленные вопросы «Как?». В дальнейшем можете использовать этот раздел в качестве короткого справочника.

Файловая система и поиск объектов

В этом разделе мы опишем не все, но самые основные команды, которыми вы будете пользоваться каждый день. Так же опишем не все, но самые основные ключи и опции этих команд.

Как и большинство современных операционных систем, FreeBSD сохраняет файлы в иерархической древовидной структуре. Такой же тип файловой системы используется в системе Windows. Единственная разница заключается в том, что в Windows используется графическое представление файлов и папок в графическом диспетчере, а в консоли FreeBSD – текстовое. В Windows или DOS при регистрации пользователь, как правило, попадает в корневой каталог. В FreeBSD дело обстоит иначе. После регистрации пользователь попадает в свой начальный «домашний» каталог, который расположен на несколько уровней ниже корневого каталога. Теперь по командам:

`ls [ключи][путь]` – вывод содержимого каталога.
- **`ls -a`** – вывод полностью (со скрытыми файлами и папками);
- **`ls -l`** – вывод подробно (с правами, размером и датами);
- **`ls -G`** – вывод в цвете (различные типы файлов отображаются разными цветами).

`cd [путь]` – перемещение по файловой системе (изменение текущего каталога). Путь может быть абсолютным (начинается с символа «**/**» – корневой каталог в системе) или относительным (считается относительно текущего каталога).

`pwd` – вывод имени текущего каталога (текущий полный путь).

`mkdir [каталог]` – создание каталога.

`rmdir [каталог]` – удаление каталога.

cp [файл][путь] – копирование файлов и каталогов.
cp -r – копирование каталога и всего его содержимого.

mv [файл][путь] – перемещение файлов и каталогов.
mv -r – перемещение каталога и всего его содержимого.

rm [файл][путь] – удаление файлов и каталогов.
rm -r – удаление каталога и всего его содержимого;
rm -f – удаление, не задавая вопросов;
rm -P – физическое удаление (три раза перезаписывает содержимое файла случайной последовательностью байтов);
rm -W – попытка восстановить файл, удаленный командой **rm**.

df [ключи] – вывод информации о свободном месте на диске.

du [ключи] – вывод информации о размере каталогов в текущем.
du –h –d 1 – вывод информации, глубиной в 1 уровень.

Помните, что большинство команд UNIX выполняют указанные действия над файлами и каталогами, не задавая лишних вопросов, даже в тех случаях, когда они могут уничтожить системные файлы. UNIX не ведет пользователей за руку, как Windows. В UNIX по умолчанию предполагается, что пользователь знает, что он делает. Поэтому в тех случаях, когда вы не уверены в своих действиях, используйте опцию **-i**. Тогда большинство команд будет запрашивать подтверждение, если возникнет угроза повреждения существующих файлов.

Все предыдущие команды поддерживают метасимволы и символы-заместители для описания имен файлов по маске:

? – совпадает с любым одиночным символом;
***** – совпадает с любой последовательностью символов;
[ ] – совпадает с диапазоном символов;
[!] – не совпадает с диапазоном символов.

Теперь несколько слов об именах файлов. Хотя технически UNIX позволяет включить в имя файла любой символ (некоторые символы нельзя просто ввести, их следует указывать как двусимвольные последовательности, начинающиеся с косой черты), желательно все-таки ограничиться только буквами, цифрами, точкой, дефисом и символом подчеркивания. Нежелательно начинать имя с дефиса, поскольку в UNIX дефис, как правило, интерпретируется как символ, за которым следует опция (ключ) команды. В имя можно включать и пробелы. Однако в этом случае его необходимо заключить в кавычки, чтобы интерпретатор мог воспринять имя целиком, а не как список нескольких файлов. Мы рекомендуем воспользоваться распространенной практикой: заменять пробелы на символы подчеркивания. Тогда имена легко читаются и не требуют кавычек при работе с ними. Избегайте применения в именах файлов специальных символов. В противном случае, пользуйтесь специальным escape-символом – это символ обратной косой черты «\». Он указывает, что следующий за ним символ следует рассматривать в его первичном значении, а не в специальном.

Искать файлы в системе можно двумя командами:

```
find [путь] -name [шаблон имени файла]
locate [шаблон имени файла]
```

Отличаются они тем, что команда **locate** использует свою собственную индексную базу файловой системы, следовательно поиск происходит намного быстрее, однако есть один минус – индексная база обновляется с некоторой периодичностью, поэтому нельзя рассчитывать на то, что команда **locate** покажет только что созданный, скопированный или перемещенный файл. Индексная база обновляется один раз в неделю автоматически, но ее можно обновить и вручную. Используйте следующую команду

каждый раз, когда хотите что-то найти, например, сразу после установки какой-либо программы в систему:

/etc/periodic/weekly/310.locate

Иногда бывает нужно поработать с архивами:

tar czvf backup.tar.gz /etc/* – заархивировать файлы;
tar xzvf backup.tar.gz –C /bkp.etc/ – разархивировать файлы и поместить их в указанную папку.

Теперь рассмотрим команды для работы с текстовыми файлами. UNIX и FreeBSD включают множество команд обработки текстовых данных в интерпретаторе. Наиболее полезные из них ниже:

wc [файл] – подсчет числа строк, слов и символов.

cat [файл] – вывод содержимого текстового файла.

cut [опции][файл] – вывод строк файла с разделителями.
cut -f[поле] -d'[разделитель]'

sort [файл] – сортировка строк в файле.

grep [шаблон][файл] – поиск строк в файле по шаблону.
grep -i – поиск шаблона без учета регистра;
grep -c – выводить только количество вхождений шаблона;
grep -v – выводить строки, не включающие шаблон.

less [файл] – просмотр текстовых файлов с пролистыванием.

more [файл] – просмотр текстовых файлов с пролистыванием.

И наконец, мы подобрались к самому интересному в этом разделе – как все вышеперечисленные команды комбинируются для выполнения различных операций. Один из элементов, делающих систему UNIX столь мощной, – это возможность использовать вывод одной команды как ввод другой. Кроме того, вывод можно перенаправлять по-разному:

> – перенаправить вывод слева направо;
< – перенаправить вывод справа налево;
| – конвейер.

Примеры:

`ls > listing.txt` – перенаправление вывода – запись листинга текущего каталога в текстовый файл.

`locate filename | grep –v ports` – поиск файла, исключая пути, которые содержат «ports».

`grep word file1.txt > file2.txt` – найти строки, содержащие слово «word» в первом файле, и записать их во второй файл.

`cat file1.txt | grep word > file2.txt` – отфильтровать вывод строк первого файла по слову «word» и записать во второй файл (по сути – то же, что и в предыдущем примере).

`cut -f1 –d' ' file1.txt | sort | uniq -c > file2.txt` – выбрать первое слово из каждой строки первого файла, упорядочить и записать уникальные во второй файл.

Ну и так далее. Поэкспериментировав и потренировавшись, вы без труда овладеете искусством управления текстовыми файлами, что очень и очень пригодится вам в работе, а так же при написании собственных скриптов.

Пользователи, группы и права доступа

Модель пользователей и прав доступа, применяемая в FreeBSD (и большинстве систем UNIX), является одноуровневой. Есть три типа пользователей: суперпользователь **root** – пользователь, который свободен от каких бы то ни было ограничений; пользователи группы **wheel** – пользователи, которые могут получить права **root** (обычно это администраторы системы); все остальные пользователи, права доступа которых так или иначе ограничивают их действия в системе.

Пользователей системы можно так же разделить на реальных людей, подключающихся к системе, и псевдопользователей (таких как bin, operator, daemon, nobody). Последние необходимы системе для того, чтобы управлять процессами.

Список пользователей хранится в файле **/etc/passwd**, список групп пользователей системы – в файле **/etc/group**.

Далее следует разобраться с моделью владения файлами. Все варианты UNIX имеют одинаковую структуру: каждый файл или каталог принадлежит и пользователю, и группе. Тем не менее это не означает, что все пользователи или члены группы имеют одинаковые права доступа. Режим доступа к файлу задается строкой вида **-rwxr-xr-x**. Для каждой разновидности пользователей (пользователь, группа, все остальные) существует набор битов полномочий. Эти биты предоставляют возможность читать файл, модифицировать его и выполнять, иначе говоря, предоставляют три вида доступа: чтение (read), запись (write) и выполнение (execute). Смысл этих битов для файлов следующий:

r – файл можно читать;
w – файл можно модифицировать, удалять, переименовывать;
x – файл можно выполнять.

Например, права на файл определяются записью **-rwxr-xr-x**. Первый дефис означает, что данный объект является файлом, следующая тройка символов **rwx** указывает права владельца файла (можно все), вторая тройка **r-x** определяет права членов группы, к которой принадлежит владелец, и последние три символа **r-x** относятся к правам всех остальных пользователей. Таким образом, в данном случае пользователям группы, как и всем остальным, можно только читать и выполнять файл, но не модифицировать или удалять.

Теперь рассмотрим строки, задающие права доступа к каталогам. Прежде всего, каталоги распознаются по первому биту в строке **d**. Это просто флажок, не связанный с правами доступа. Полномочия на работу в каталоге действуют по тому же принципу, что и полномочия на файлы, но здесь есть отличия:

r – каталог можно читать (выполнить команду **ls**);
w – каталог и файлы внутри него можно модифицировать;
x – в каталоге можно производить поиск файлов.

Так, например, запись **drwxr-xr-x** означает, что данный объект является каталогом, его владелец может выполнять в этом каталоге любые действия, а его группа и все остальные могут только читать содержимое и выполнять поиск.

Теперь рассмотрим команды, модифицирующие собственность и права доступа к каталогам и файлам:

chown [пользователь:группа][файл] – установка пользователя и группы в качестве владельцев файла;
chmod [права][файл] – установка прав доступа к файлу.

Права доступа можно задавать числовыми и символическими аргументами. Числовой способ проще и удобнее. Он заключается в установке трехзначного восьмеричного числа,

которое уникальным образом задает права доступа для каждого типа владельца. Каждая цифра определяет права: пользователя, группы и всех остальных. Число, задающее право, получается путем суммирования чисел, отвечающих битам прав доступа. Возможные значения битов:

4 – доступ на чтение (**r**).
2 – доступ на запись (**w**);
1 – доступ на выполнение (**x**);
0 – нет доступа (**-**);

Таким образом, режиму "чтение и запись" соответствует **6**, режиму "чтение и выполнение" – **5**, а режиму "чтение, запись и выполнение" – **7**. Комбинация цифр формирует трехзначное число, задающее стандартные права доступа к файлу. Например:

0755 – все для владельца, чтение и выполнение для группы и остальных (**-rwxr-xr-x**);
0644 – чтение и запись для владельца, только чтение для группы и остальных (**-r-xr--r--**);
0600 – чтение и запись для владельца, для группы и остальных доступа нет (**-r-x------**).

Самая первая цифра управляет дополнительными свойствами – задает особое поведение файлов и каталогов при определенных обстоятельствах:

0 – обычные права доступа;
1 – бит устойчивости (устанавливается только для каталогов): владелец имеет право удалять или переименовывать только файлы, которыми он владеет, причем лишь при наличии права на запись в этот каталог;

2 – идентификатор группы. Если такой бит установлен для выполняемого файла, то он выполняется с правами группы, владеющей файлом, а не пользователя, запустившего его;
4 – идентификатор пользователя. Если такой бит установлен для выполняемого файла, то он выполняется с правами пользователя, владеющего файлом, а не того, кто его запустил.

Значение дополнительного бита формируется так же суммированием.

Штатные текстовые редакторы (ee, vi)

Основная работа в системе FreeBSD связана с постоянным редактированием текстовых конфигурационных файлов тех или иных служб. Простейший редактор, который можно использовать – **ee** (easy editor). Работать с ним очень просто и интуитивно понятно, кроме того вверху экрана всегда располагается небольшая справка по внутренним командам. Чтобы отредактировать файл, нужно ввести команду:

```
ee [файл]
```

Однако, мы рекомендуем использовать редактор **vi** – один из первых редакторов, разработанных для операционных систем UNIX. Он и по сей день остается одним из самых мощных редакторов и стандартно поставляется практически с каждой операционной системой типа UNIX. К сожалению, среди новичков редактор **vi** пользуется репутацией программы, известной своей загадочностью и трудностью в изучении. В нем отсутствует меню, и все действия осуществляются с помощью клавиш и клавиатурных комбинаций. Понятно, что на их изучение требуется

время. Так зачем же изучать такой редактор? Есть, по меньшей мере, две причины.

Во-первых, он имеется в любой ОС UNIX, с которой вам быть может, придется работать. Рано или поздно вы столкнетесь с ситуацией, когда **vi** окажется единственным редактором в операционной системе…

Во-вторых, когда вы изучите различные комбинации клавиш и команды, в вашем распоряжении окажется очень мощное средство. Редактор **vi** обеспечит возможность очень быстрой работы, так как для набора большинства команд не придется убирать руки с базовых клавиш. Приступим к его описанию.

Сразу после запуска редактор **vi** будет находиться в командном режиме. В этом режиме нажатия клавиш интерпретируются как команды редактору, а не как текст, вводимый в документ. Чтобы переключиться в режим ввода текста, необходимо нажать одну из клавиш:

a – append (присоединить). В этом режиме вводимый текст вставляется после символа, на котором находится курсор.
i – insert (вставить). В этом режиме вводимый текст вставляется перед символом, на котором находится курсор.
o – open (открыть). Это приводит к тому, что после строки, на которой находится курсор, в текст вставляется новая строка, затем курсор перемещается на нее и редактор **vi** переходит в режим insert, разрешая ввод текста на новой строке.

Для возврата в командный режим, нажмите клавишу **Esc**.

Перемещаться по тексту в режиме ввода текста можно с помощью курсорных клавиш или клавиш **Page Up / Page Down**. Однако имеются и другие клавиши, позволяющие перемещаться по документу в командном режиме:

h – перемещает курсор влево на один символ;
j – перемещает курсор вниз на один символ;
k – перемещает курсор вверх на один символ;
l – перемещает курсор вправо на один символ;
w – перемещает курсор вперед на одно слово;
b – перемещает курсор назад на одно слово;
e – перемещает курсор в конец следующего слова;
0 – перемещает курсор в начало строки;
$ – перемещает курсор в конец строки;
) – перемещает курсор в начало следующего предложения;
(– перемещает курсор в начало предыдущего предложения;
} – перемещает курсор в начало следующего абзаца;
{ – перемещает курсор в начало предыдущего абзаца;
G – перемещает курсор в конец текущего документа;
^ – перемещает курсор к первому символу строки, но не пробелу;
H – перемещает курсор на первую строку на экране;
L – перемещает курсор на последнюю строку на экране.

Обратите внимание, что с каждой командой этой таблицы по умолчанию используется число 1. Клавиша **j** перемещает курсор вниз на одну строку, клавиша **k** перемещает его вверх на одну строку, клавиша **w** перемещает вправо на одно слово и так далее. Все эти команды можно модифицировать, вводя перед ними число. Так, например, команда **5j** перемещает курсор вниз не на одну строку, а на пять. Команда **75G** перемещает курсор на 75-ю строку файла, редактируемого в данный момент. А команда **5L** перемещает курсор на пятую снизу строку экрана. Данный синтаксис справедлив для всех команд, кроме **^**, которая перемещает курсор к первому символу строки, не являющемуся пробелом.

В редакторе **vi** клавиши **Backspace** и **Delete** не выполняют тех действий, которых от них можно ожидать. Для удаления текста и тому подобного придется пользоваться различными клавишами в командном режиме. Рассмотрим команды для редактирования текста:

D – удаляет текст от позиции курсора до конца строки;
dd – удаляет всю текущую строку целиком;
R – замещает текущий текст вводимым текстом, начиная с курсора;
S – удаляет текущую строку и начинает ввод текста на новой строке;
x – удаляет символ в позиции курсора и сдвигает символы влево;
X – удаляет символ перед курсором и сдвигает символы влево;
~ – заменяет букву на позиции курсора той же буквой другого регистра;
J – объединяет текущую строку с предыдущей;
yw – помещает в буфер слово, на котором находится курсор;
y$ – помещает в буфер текст от курсора до конца данной строки;
yy – помещает в буфер всю текущую строку;
p – вставляет текст в документ после курсора;
P – вставляет текст перед курсором.

В редакторе **vi** имеется так же ряд команд для выполнения поиска и замены текста:

/текст – поиск текста в прямом направлении до первого совпадения с заданным текстом;
/ – повтор поиска текста в прямом направлении до обнаружения очередного совпадения;
?текст – поиск текста в обратном направлении до первого совпадения с заданным текстом;
? – повтор поиска текста в обратом направлении до обнаружения очередного совпадения;

% – перемещение курсора на соответствующую парную скобку (удобно при программировании);
:s/текст1 /текст2 – замена в текущей строке каждого совпадения текст1 на текст2;
:%s/текст1 /текст2 – замена во всем файле каждого совпадения текст1 на текст2;

И последнее, что нам осталось, – это операции над файлами и выход из редактора:

:w – сохранить изменения в текущем файле;
:w! – сохранить изменения в текущем файле в любом случае;
:q – выйти из редактора;
:q! – выйти из редактора в любом случае;
:e файл – загрузить файл в редактор для редактирования;
:e! – отбросить все изменения и перезагрузить старый вариант;
:wq – сохранить изменения в текущем файле и выйти;

Вот в принципе и все, что вам необходимо знать для эффективной работы с текстовыми файлами в редакторе **vi**. Настоятельно рекомендуем в этом вопросе не идти самым простым путем, а освоить, привыкнуть и работать всегда именно в этом редакторе.

Внутренняя справка FreeBSD (man)

В операционной системе FreeBSD есть очень хорошая и подробная справочная система, правда, на английском языке, но не рассматривайте этот факт, как препятствие или неприятность. В конце концов это возможность немного подучить технический английский – поверьте, он вам понадобится, если будете работать с UNIX. В любом случае в Интернете вы сможете найти перевод тех или иных статей справки. А для того, чтобы посмотреть описание какой-либо команды, а так же описание всех ее ключей и опций, нужно воспользоваться командой **man**. Например:

man mkdir – вывод полной информации о команде для создания каталогов и всех ее опциях.

Если же вы решите довести до совершенства свое умение пользоваться справкой, изучите результат следующей команды:

man man – вывод справки о работе со справкой :)

Глава 3
Базовая настройка основных функций

Теперь, имея только что установленную FreeBSD и некоторые практические навыки, вы готовы заняться конфигурированием вашего сервера. Во избежание недоразумений, сразу проверьте и установите правильную дату и время:

```
# date 201301010900
```

2013 год, январь, 01, 09:00 (здесь и далее символ **#** в командной строке означает приглашение к вводу команды под правами **root**).

ВНИМАНИЕ! С этой главы мы начинаем активно работать с текстовыми конфигурационными файлами. Стиль подачи информации будет следующим: мы не будем показывать в листингах содержимое файлов полностью, т.к. они бывают очень объемными. Мы будем показывать только те строки, которые нужно добавить, либо найти и изменить.

Создание администратора сервера (adduser)

Первое, что необходимо сделать после установки любой операционной системы, это создать себе пользователя с правами администратора, под которым работать далее и вообще всегда. Дело в том, что работать на сервере под пользователем **root**, является изначально грубым нарушением не только компьютерной этики, но и правил безопасности и администрирования в целом. Поэтому создадим себе пользователя для управления нашим сервером:

```
# adduser
```

Последовательно ответьте на все вопросы, а особое внимание уделите имени пользователя (это вы), группе (введите **wheel**), ну и конечно же паролю. Группа **wheel** – это привилегированная группа пользователей, которым разрешено получать права **root**. Мы для себя создадим пользователя **raph** (рис. 13).

```
Full name: Korney A. Kornienko
Uid (Leave empty for default):
Login group [raph]: wheel
Login group is wheel. Invite raph into other groups? []:
Login class [default]:
Shell (sh csh tcsh bash rbash nologin) [sh]: bash
Home directory [/home/raph]:
Home directory permissions (Leave empty for default):
Use password-based authentication? [yes]:
Use an empty password? (yes/no) [no]:
Use a random password? (yes/no) [no]:
Enter password:
Enter password again:
Lock out the account after creation? [no]:
Username   : raph
Password   : *****
Full Name  : Korney A. Kornienko
Uid        : 1001
Class      :
Groups     : wheel
Home       : /home/raph
Home Mode  :
Shell      : /usr/local/bin/bash
Locked     : no
OK? (yes/no): y
```

Само по себе имя **raph** не несет никакой магической силы, например, как **root**. Это просто никнейм автора книги. Мы будем применять его в листингах, чтобы не писать общеизвестных имен, типа **admin**. Вы можете выбрать себе любое другое имя, один из именно ваших личных никнеймов.

После добавления пользователя выйдите из системы командой **exit** или комбинацией клавиш **CTRL-D**, затем войдите под собой, получите права администратора (после команды **su** введите пароль рута) и работайте так дальше:

```
% su
Password:
# _
```

ПРИМЕЧАНИЕ. Если приглашение к вводу команды заканчивается символом **#**, то вы работаете с правами **root**. Если же в конце строки стоит другой символ, например **%** или **$**, то вы работаете с обычными правами пользователя и большинство операций для администрирования вам окажутся недоступны.

Конфигурирование ядра, виртуальные консоли

Начнем, пожалуй, с самого длительного по времени процесса – конфигурирования ядра системы. Сам по себе процесс оптимизации ядра FreeBSD скорее достоин отдельной книги и выходит за рамки данной документации по причине того, что на современном оборудовании оптимизация ядра носит весьма условный характер и была бы полезна скорее для тренировки. Поэтому мы просто включим кое-какие модули, которые нам понадобятся в будущем.

ПРИМЕЧАНИЕ. Система FreeBSD позволяет работать в нескольких консолях одновременно. По умолчанию их 8, а переключиться между ними можно при помощи комбинаций **ALT-F1** … **ALT-F8** соответственно. Причем, как вы уже успели заметить, в первую консоль постоянно попадают системные сообщения – это немного мешает, поэтому, для комфортной работы, переключайтесь сразу из первой консоли в любую другую.

Перейдем во вторую консоль – **ALT-F2**, войдем в систему, зайдем в каталог, где лежат текстовые файлы с описанием параметров ядер, скопируем файл текущего ядра и отредактируем его:

```
# cd /usr/src/sys/amd64/conf/
# cp GENERIC GATEWAY
# vi GATEWAY
   # options    INET6
   options      IPFIREWALL
   options      IPFIREWALL_FORWARD
   options      IPDIVERT
   options      DUMMYNET
```

Первую строку нужно найти и закомментировать – мы отключаем поддержку **IPv6**, так как на данном этапе развития глобальных сетей нам это пока не нужно (символ **#** в текстовом файле означает, что все, что после него до конца строки можно считать комментарием, таким образом, если поставить его в начале, то данная строка, содержащая параметр, закомментирована и неактивна). Остальные опции нужны для работы фаерволла и шлюза – добавьте их в любое место.

ВНИМАНИЕ! Теперь, на протяжении всей книги, при установке программ из портов, выключайте опцию IPV6 в меню выбора параметров там, где она вам будет встречаться. Иначе некоторые программы могут просто не запуститься. В принципе, IPV6 можно и не отключать – это на ваше усмотрение.

Сохраним изменения, после чего скомпилируем новое ядро, перейдем в каталог компиляции и запустим процесс пересборки:

```
# config GATEWAY
# cd ../compile/GATEWAY/
# make cleandepend && make depend && make && make install
```

В последней строке показана запись, которая означает последовательное выполнение команд – они будут выполнены одна за другой автоматически. Т.е. это то же самое, что:

```
# make cleandepend
# make depend
# make
# make install
```

за исключением того, что в последнем случае нам придется дожидаться окончания выполнения текущей команды, а потом вводить следующую.

Этот процесс, в зависимости от общей производительности вашей системы, займет около 32 минут. Поэтому, чтобы не терять времени, переключаемся в третью консоль – **ALT-F3** и продолжаем. Но если вы все-таки решите дождаться окончания процесса сборки нового ядра, тогда после этого, перед перезагрузкой с новым ядром добавьте две строки в **/etc/rc.conf**, иначе не заработает сеть:

```
# vi /etc/rc.conf
   firewall_enable="YES"
   firewall_type="open"

# reboot
```

Настройка сети (ifconfig, route, resolv.conf)

Для начала, давайте еще раз посмотрим, какие сетевые интерфейсы определила система (в первый раз мы их видели при первичной настройке параметров сразу после установки):

```
# ifconfig
de0: flags=8802<BROADCAST,SIMPLEX,MULTICAST> metric 0 mtu 1500
        ether bc:30:5b:ed:f2:53
        media: Ethernet autoselect
de1: flags=8802<BROADCAST,SIMPLEX,MULTICAST> metric 0 mtu 1500
        ether 00:15:5d:00:0c:07
        media: Ethernet autoselect
```

Для настройки сетевых интерфейсов из командной строки, так же используется команда **ifconfig**, только уже с параметрами. Пусть **de0** у нас «смотрит» в мир, а **de1** – в локальную сеть. Тогда пропишем им следующие настройки:

```
# ifconfig de0 inet 22.22.22.22 netmask 255.255.255.252
# ifconfig de1 inet 10.0.0.1 netmask 255.255.255.0
```

Теперь добавим в таблицу маршрутизации шлюз по умолчанию, который выдал провайдер:

```
# route add default 22.22.22.21
```

А DNS серверы провайдера пропишем в специальном конфигурационном файле:

```
# vi /etc/resolv.conf
   search 127.0.0.1
   nameserver 22.22.0.1
   nameserver 22.22.0.2
```

Теперь мы должны «увидеть» и мир и внутреннюю локальную сеть. Посмотрите, что теперь выдаст команда **ifconfig**, а так же проверьте стандартной для всех сетевых систем командой **ping**:

```
# ping freebsd.org
PING freebsd.org (8.8.178.135): 56 data bytes
64 bytes from 8.8.178.135: icmp_seq=0 ttl=57 time=211.055 ms
64 bytes from 8.8.178.135: icmp_seq=1 ttl=57 time=211.115 ms
```

Однако, при перезагрузке сервера, сохранятся только DNS провайдера, так как мы их внесли в специальный текстовый файл. Настройки сети и маршрута по умолчанию будут работать только в текущем сеансе работы системы и при перезагрузке придется их настраивать заново. Чтобы сохранить эти настройки, нужно внести их в параметры загрузки. Об этом немного ниже.

ПРИМЕЧАНИЕ. Теперь вы можете не использовать физическую консоль для работы, а подключиться к серверу удаленно, откуда вам будет удобнее по SSH на порт 22. Поддержку удаленной работы мы обеспечили в самом начале при установке системы, разрешив запуск демона **sshd**. Но наш совет: оставаться возле сервера и работать в консоли до окончания сборки нового ядра и первой перезагрузки системы.

Если провайдер использует ADSL (ppp)

Часто бывает так, что провайдер подает сервис доступа в Интернет по асинхронному каналу связи (ADSL). Причина – неплохие качество и скорость при дешевизне подключения и использования. Конечно, для полноценной работы Интернет-сервера, такая технология не подходит, но иногда просто нет других вариантов. В этом случае необходимо получить у провайдера статический IP-адрес, а модем установить в режиме Bridge. Это нужно для того, чтобы этот самый IP-адрес ссылался именно на сетевой интерфейс сервера, а не на ADSL-модем. Затем:

```
# vi /etc/ppp/ppp.conf
   default:
     set log Phase Chat LCP IPCP CCP tun command
     enable dns

   provider_name:
     set device PPPoE:ed0
     set authname ppp_login
     set authkey ppp_password
     set dial
     set login
     add default HISADDR
```

Естественно, параметры **provider_name**, **ppp_login** и **ppp_password**, нужно изменить в соответствии с параметрами вашего подключения (имя можно выбрать любое, а ppp-логин и ppp-пароль должен выдать Интернет-провайдер). Запускаем **ppp** для установки соединения:

```
# /usr/sbin/ppp -ddial provider_name
```

При этом в системе появится виртуальный сетевой интерфейс **tun0**, и весь трафик логически будет проходить именно через него, а не через **de0**. В рамках данной книги мы будем описывать традиционное подключение к сети Интернет, т.е. работать с существующими физическими интерфейсами **de0** и **de1**. Таким образом, если у вас ADSL, то просто меняйте в дальнейших сетевых настройках **de0** на **tun0.**

Редактирование параметров загрузки (rc.conf)

Все, что мы сделали до этого момента вручную, нужно прописать в файле **/etc/rc.conf** – это основной файл параметров загрузки системы. Он уже содержит некоторые записи, которые мы отметили при установке. На данном этапе приведем его к следующему виду:

```
# vi /etc/rc.conf
   dumpdev="NO"
   hostname="gateway.example.com"
   ifconfig_de0="inet 22.22.22.22 netmask 255.255.255.252
   ifconfig_de1="inet 10.0.0.1 netmask 255.255.255.0
   defaultrouter="22.22.22.21"
   firewall_enable="YES"
   firewall_type="open"
   sshd_enable="YES"
   ntpd_enable="YES"
```

Мы добавили настройки сетевых интерфейсов **de0** и **de1**, информацию о шлюзе по умолчанию, а так же активировали и открыли фаерволл. Если у вас ADSL, тогда уберите настройки интерфейса **de0** и шлюза, так как эти параметры мы получим при запуске **ppp** автоматически. Так же нужно добавить параметры запуска самого **ppp**:

```
# vi /etc/rc.conf
   dumpdev="NO"
   hostname="gateway.example.com"
   ifconfig_de1="inet 10.0.0.1 netmask 255.255.255.0
   ppp_enable="YES"
   ppp_mode="ddial"
   ppp_profile="provider_name"
   firewall_enable="YES"
   firewall_type="open"
   sshd_enable="YES"
   ntpd_enable="YES"
```

Обновление коллекции портов (portsnap, cron)

«Коллекция портов» или «дерево портов» – это технология установки программного обеспечения сторонних разработчиков в ОС FreeBSD. Сам по себе порт – это набор файлов, в которых содержится информация для автоматизации процесса компиляции приложения из исходного кода, то есть: откуда скачать, как собрать и куда положить в системе. Сначала нужно скачать и распаковать текущее дерево портов:

```
# portsnap fetch extract
```

Эта операция займет около 8 минут. Но развитие ПО не стоит на месте, поэтому стоит периодически обновлять дерево портов своей FreeBSD до актуального состояния – один раз в несколько дней, выполнять следующую команду:

```
# portsnap fetch update
```

Будет лучше сразу добавить задание демону-планировщику для выполнения по расписанию, например каждый понедельник в 6:00 утра. Список заданий содержится в файле **/etc/crontab**. Добавим в него строку по аналогии с теми, которые там уже есть, после чего перезапустим демон **cron**:

```
# vi /etc/crontab
   #minute hour mday month wday who command
    ...
   0   6   *   *   1   root   portsnap fetch update

# killall -HUP cron
```

Теперь в дереве портов вы всегда будете иметь актуальную информацию о последних стабильных версиях всех существующих программ, портированных для системы FreeBSD.

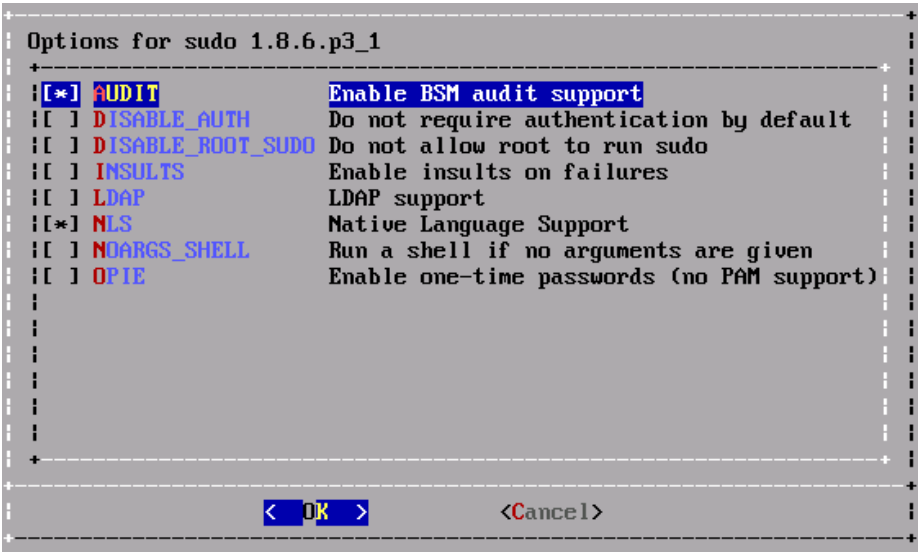
Русификация и удобство консоли (sudo, bash)

Первой программой, которую мы установим из портов, будет программа **sudo**, при помощи которой можно наделять пользователя правами **root**, при этом не передавая его пароль. Это полезно, когда вы хотите дать администраторские права еще кому-то, кроме себя, а пароль рута хотите сохранить в тайне. В принципе это правильно.

Итак, для того, чтобы установить программу из портов, нужно перейти в каталог порта и дать команду к установке:

```
# cd /usr/ports/security/sudo/
# make install clean
```

После ввода команды **make install**, если того требует порт, появится меню с возможностью выбора параметров установки (рис. 14) – вы можете выбрать нужные вам опции на этом этапе. Оставим все по умолчанию, как есть, и просто нажмем **[OK]**.



ПРИМЕЧАНИЕ. Это же меню можно вызвать командой **make config**. В дальнейшем, при установке программ, мы будем часто с этим сталкиваться и, если будет нужно что-то изменить, будем помечать это отдельно. Если отдельной пометки нет – оставляйте все по умолчанию.

После окончания установки найдем и раскомментируем в служебном файле программы **sudo** следующую строку:

```
# vi /usr/local/etc/sudoers
   %wheel ALL=(ALL) NOPASSWD: ALL
```

Это значит, что пользователям из привилегированной группы **wheel**, не нужно будет вводить пароль рута для получения его прав.

Теперь установим альтернативный командный интерпретатор **bash**, который, на наш взгляд, более удобен в использовании, чем штатные **sh** и **csh**. В процессе установки несколько раз будут появляться вопросы о выборе параметров сопутствующих программ – оставляйте все по умолчанию. Этот процесс продлится около 12 минут:

```
# cd /usr/ports/shells/bash/
# make install clean
```

После этого обеспечим поддержку кодировки **UTF-8** в консоли. Найдем файл **/etc/login.conf** и отредактируем его:

```
# vi /etc/login.conf
   russian|Russian Users Accounts:\
     :charset=UTF-8:\
     :lang=ru_RU.UTF-8:\
     :tc=default:
```

Перекомпилируем служебные файлы, а затем назначим себе класс **russian** и изменим командный интерпретатор, прописав полный путь к **bash** в соответствующей строке:

```
# cap_mkdb /etc/login.conf
# chsh raph
   #Changing user information for raph.
   Login: raph
    ...
   Class: russian
    ...
   Shell: /usr/local/bin/bash
    ...
```

Теперь, при входе в систему, мы будем получать консоль, работающую в самой перспективной кодировке **UTF-8** и наиболее удобный командный интерпретатор **bash**, хотя, для новичка все они скорее всего покажутся одинаковыми.

ПРИМЕЧАНИЕ. Последняя команда **chsh** отправит вас в текстовый редактор по умолчанию – **vi**. И такое будет случаться часто, например при вводе команд **vipw** – для редактирования списка учетных записей пользователей системы, или **visudo** – для работы с параметрами программы **sudo**. Поэтому, еще раз рекомендуем вам освоить его команды, описанные в главе 2.

При входе в систему получите права администратора. Как видите, теперь пароль рута не требуется, но для этого пользователь должен быть в группе **wheel**:

```
$ sudo -s
# _
```

Завершение базовой настройки сервера

Иногда бывает полезно прописать имена вашего сервера в файле **/etc/hosts**. Лучше не откладывайте и сделайте это сразу:

```
# vi /etc/hosts
   127.0.0.1      localhost localhost.example.com
   22.22.22.22    gateway.example.com mail.example.com
```

Когда процесс сборки нового ядра во второй консоли завершится, мы будем готовы к тому, чтобы перезагрузить систему и проверить, что же у нас получилось в итоге. Перезагрузиться можно двумя способами – эффект будет один и тот же:

```
# reboot
# shutdown -r now
```

Здесь отметим, что хоть и принято считать FreeBSD одной из самых надежных ОС, но испытывать файловую систему на прочность не стоит. Выключать сервер, как и перезагружать, нужно корректно. Так же два способа остановки системы:

```
# halt
# shutdown -h now
```

Когда сервер загрузится, войдите в систему и посмотрите версию и тип ОС и ядра, а так же проверьте сетевые подключения, фаерволл и запущенные процессы (по очереди запустите следующие команды и посмотрите результат, который они вам покажут):

```
# uname -a
# ifconfig
# ipfw show
# ps -ax
# top
```

Не забывайте о конвейерах. Если вывод команды не помещается на один экран, для удобства используйте **less**. Если

хотите отфильтровать вывод команды по какому-то конкретному слову – используйте **grep**:

```
# ps -ax | less
# ps -ax | grep natd
```

Если возникает необходимость остановить или перезапустить какой-либо процесс, пользуйтесь скриптами из **/etc/rc.d** с опцией **stop** или **restart**. Можно так же воспользоваться командами:

kill [номер] – остановить процесс по номеру;
killall [имя] – остановить процесс по имени;
killall –HUP [имя] – перезапустить процесс по имени.

Так же не забывайте добывать более подробную информацию о процессах и командах, которые мы здесь постепенно изучаем:

```
# man ipfw
```

ПРИМЕЧАНИЕ. Теперь можете переместиться из серверной на свое рабочее место и подключиться к своему серверу по SSH, например, при помощи программы PuTTY для Windows. Она бесплатна и ее легко найти в Интернете. Теперь это ваш основной инструмент для управления вашим сервером.

Базовая настройка сервера завершена. В принципе, это тот необходимый минимум, который должен быть вне зависимости от того, какой тип сервера вы хотите собрать – защищенный шлюз и прокси, почтовый или веб-сервер. Со следующей главы мы начнем активно устанавливать программы из портов, поэтому, если вы работаете в виртуальной среде, сейчас самое время сделать слепок виртуальной машины для быстрого восстановления после возможных экспериментов ;)

Глава 4
Защищенный шлюз и прокси-сервер

В прошлой главе мы провели базовую подготовку сервера. После перезагрузки он заработал с новым ядром и обновленной коллекцией портов. Это то, что нам нужно для того, чтобы развернуть деятельность. В этой главе мы будем устанавливать дополнительные программы и сделаем из него прозрачный шлюз, кеширующий прокси, DNS, DHCP, FTP сервер и межсетевой экран, то есть полноценный граничный шлюз локальной сети.

Базовая настройка кеширующего DNS (named)

Для нормальной работы пользователей в сети обязательно должен быть сервис DNS – для преобразования имен в IP-адреса и наоборот. Так же этот сервис может быть необходим для работы некоторых сетевых программ, поэтому было бы логично организовать его прямо на нашем сервере. Подправим конфигурационный файл демона **named** (следующие строки, если есть и закомментированы – найти и раскомментировать, если же их нет – добавить):

```
# vi /etc/namedb/named.conf
   acl ACCESS { 127.0.0.1; 10.0.0.0/24; };
   options {
    ...
     listen-on  { 127.0.0.1; 10.0.0.1; };
     allow-recursion { ACCESS; };
    ...
     forwarders {
       22.22.0.1;
       22.22.0.2;
     };
   };
```

Теперь наш сервер будет кешировать у себя ответы на все DNS-запросы, а то, что не найдет у себя в кеше, будет спрашивать у DNS серверов провайдера. Причем отвечать он будет только самому себе и нашей локальной сети, в соответствии с созданным нами **acl** (acess list). Для реализации всех наших планов этого будет достаточно. Добавим в **/etc/rc.conf** разрешение на запуск **named** и запустим его скриптом автозапуска:

```
# vi /etc/rc.conf
   named_enable="YES"

# /etc/rc.d/named start
```

Проверим, запустился ли процесс **named** и посмотрим, как он справляется с DNS-запросами:

```
# ps -ax | grep named
 649 ?? Is 0:00,41 /usr/sbin/syslogd -l /var/run/log ...
 735 ?? Is 0:00,09 /usr/sbin/named -t /var/named -u bind

# dig @127.0.0.1 freebsd.org A
; <<>> DiG 9.8.3-P4 <<>> @127.0.0.1 freebsd.org A
 ...
;; QUESTION SECTION:
;freebsd.org.           IN      A

;; ANSWER SECTION:
freebsd.org.    3600    IN      A       8.8.178.135

;; AUTHORITY SECTION:
freebsd.org.    3600    IN      NS      ns3.isc-sns.info.
freebsd.org.    3600    IN      NS      ns2.isc-sns.com.
freebsd.org.    3600    IN      NS      ns1.isc-sns.net.

;; Query time: 99 msec
;; SERVER: 127.0.0.1#53(127.0.0.1)
;; WHEN: Tue Jan 01 10:00:00 2013
;; MSG SIZE  rcvd: 133
```

Так же вы можете использовать **named** для того, чтобы выступать DNS-сервером какого-либо Интернет-домена. Но это довольно-таки специфическая задача и рассматривается она поверхностно в главе 7.

Базовая настройка прозрачного шлюза (natd)

Для организации совместного прозрачного доступа в Интернет из локальной сети нам нужен демон **natd**, фаерволл **ipfw** и возможность передавать пакеты данных с одного сетевого интерфейса на другой. Добавим эти параметры в автозагрузку:

```
# vi /etc/rc.conf
   gateway_enable="YES"
   natd_enable="YES"
   natd_interface="de0"
   firewall_enable="YES"
   firewall_type="/etc/firewall.conf"

# vi /etc/firewall.conf
   add 4000 divert natd ip from any to any via de0
   add 65500 allow ip from any to any

# natd -n de0
# /etc/rc.d/ipfw restart
```

Мы добавили **gateway_enable** – режим шлюза (для передачи сетевых пакетов данных между интерфейсами) и параметры запуска демона **natd** – разрешение и сетевой интерфейс (для преобразования адресов). Обратите внимание на то, что мы изменили тип фаерволла с **open** на **/etc/firewall.conf** – в этом файле будут храниться его правила. Первые два мы сразу и создали – первое нужно для работы **natd**, а второе соответствует типу фаерволла **open**, т.е. все разрешено. Оставим фаерволл пока в таком открытом виде, чтобы было проще двигаться дальше.

В итоге должен получиться прозрачный шлюз, обеспечивающий доступ в Интернет из локальной сети. Для этого при настройке сетевых подключений пользователей Windows укажите IP-адрес вашего сервера 10.0.0.1, как шлюз по умолчанию и DNS сервер.

Установка кеширующего прокси-сервера (squid)

Сейчас мы имеем полностью прозрачный и неконтролируемый доступ пользователей в сеть Интернет. Для реализации функции кеширования HTTP и FTP трафика, а так же для того, чтобы иметь возможность контролировать и ограничивать этот трафик, установим и настроим программу **squid**, после чего немного подправим ее конфигурационный файл. Процесс установки займет около 10 минут:

```
# cd /usr/ports/www/squid/
# make install clean
# vi /usr/local/etc/squid/squid.conf
   acl localnet src 10.0.0.0/24
    ...
   http_access allow localnet
   http_access deny all

# squid -z
# echo squid_enable=\"YES\" >> /etc/rc.conf
# /usr/local/etc/rc.d/squid start
```

В директиве **acl** (access list) мы описали нашу локальную сеть, после чего мы разрешили пользоваться прокси-сервером только из нее, а остальным запретили. Команду **squid -z** нужно выполнить один раз перед первым запуском для того, чтобы система создала структуру каталогов и файлов для хранения кеша.

ПРИМЕЧАНИЕ. Как вы уже заметили, каталог **/etc/rc.d/** содержит скрипты запуска демонов, установленных в системе изначально, а в каталог **/usr/local/etc/rc.d/** помещаются скрипты запуска программ, которые вы устанавливаете вручную. При загрузке компьютера все, что разрешено в **/etc/rc.conf**, запускается автоматически, но ничто не мешает нам пользоваться этими же скриптами в процессе работы и запускать или останавливать процессы вручную.

Это все. Теперь, чтобы пользоваться прокси-сервером, нужно прописать в настройках веб-браузеров пользователей адрес прокси: 10.0.0.1, порт: 3128.

Иногда может возникнуть необходимость ввести ограничения доступа пользователей в Интернет. Например, вы хотите разрешить выход в мир только «избранным» клиентам. Добавьте директиву **acl** с именем **users**, в которой укажите путь к файлу, в котором перечислите тех, кому разрешено пользоваться **squid**:

```
# vi /usr/local/etc/squid/users.txt
   10.0.0.14/32
   10.0.0.28/32

# vi /usr/local/etc/squid/squid.conf
   acl localnet src 10.0.0.0/24
   acl users src "/usr/local/etc/squid/users.txt"
    ...
   http_access allow users
   http_access deny all

# squid -k reconfigure
```

Таким образом, мы разрешили пользоваться **squid** только определенным IP-адресам. Последняя команда перезапускает **squid** с новыми настройками. Можно сделать и по-другому – разрешить всем из локальной сети, кроме списка «избранных»:

```
# vi /usr/local/etc/squid/squid.conf
   acl localnet src 10.0.0.0/24
   acl users src "/usr/local/etc/squid/users.txt"
    ...
   http_access allow localnet !users
   http_access deny all

# squid -k reconfigure
```

Логика работы директивы **http_access** очевидна – разрешить из локальной сети, но не из списка.

Так же можно выборочно запретить доступ к определенным Интернет-ресурсам. Для этого можно применить **acl** четырех типов: **dstdomain** (имя домена назначения), **dstdom_regex** (шаблон для имени домена назначения), **url_regex** (шаблон для адреса) и **urlpath_regex** (шаблон для части адреса, исключая протокол и имя хоста). Рассмотрим следующий пример:

```
# vi /usr/local/etc/squid/squid.conf
   acl localnet src 10.0.0.0/24
   acl dom_deny dstdomain baddomain1.com baddomain2.com
   acl url_deny url_regex "/usr/local/etc/squid/url.txt"
    ...
   http_access allow localnet !dom_deny !url_deny
   http_access deny all

# vi /usr/local/etc/squid/url.txt
   audio
   video
    ...

# squid -k reconfigure
```

Таким образом, мы разрешили пользователям локальной сети ходить на все сайты, кроме двух доменов и тех, у которых в адресе в любом месте встречаются слова из «черного списка».

И последнее – русифицируем сообщения для пользователей, которые они будут видеть, когда **squid** их куда-то не пустит, либо возникнет какая-либо ошибка:

```
# vi /usr/local/etc/squid/squid.conf
   error_directory /usr/local/etc/squid/errors/Russian-1251

# squid -k reconfigure
```

Вы даже можете вручную отредактировать шаблоны этих сообщений, которые лежат в каталоге, который вы только что прописали в качестве параметра **error_directory**.

Аутентификация пользователей SQUID (squid)

Можно еще более усилить ограничение доступа пользователей к нашему прокси-серверу – ввести проверку пароля при попытке выйти в Интернет:

```
# vi /usr/local/etc/squid/squid.conf
   auth_param basic program /usr/local/libexec/squid/ncs
a_auth /usr/local/etc/squid/squid.passwd
   auth_param basic children 4
    ...
   acl localnet src 10.0.0.0/24
   acl auth_users proxy_auth REQUIRED
    ...
   http_access allow localnet auth_users
   http_access deny all
```

Теперь при попытке выйти в Интернет, пользователь получит окно для ввода персонального логина и пароля. В этом случае нам придется вести отдельный файл с паролями в зашифрованном виде, поэтому нам потребуется специальная программа **htpasswd**, которая поставляется с веб-сервером Apache. Но можно немного схитрить и обойтись и без нее. Так как пароли шифруются так же, как и системные, можно просто создавать пользователя в системе и копировать нужные строки из **/etc/master.passwd** следующим образом:

```
# grep raph /etc/master.passwd >> /usr/local/etc/squid/s
quid.passwd
```

После этого пользователя из системы можно удалить. Если с проверкой пароля что-то будет не так, проверьте права доступа к файлу **squid.passwd** – у демона **squid** должны быть права на его чтение. И не забывайте после внесения каких-либо изменений перезапускать **squid**:

```
# squid -k reconfigure
```

Настройка прозрачного прокси (squid, ipfw)

Прозрачный прокси – это когда пользователи ходят в Интернет в любом случае через **squid** даже не настраивая прокси-сервер в своих браузерах. Для этого нам нужно завернуть весь HTTP трафик (порт 80), идущий к нам из локальной сети на наш прокси-сервер (порт 3128). Сейчас мы опять коснемся фаерволла, однако более подробно рассмотрим его несколько позже:

```
# vi /usr/local/etc/squid/squid.conf
   http_port 3128 transparent

# squid -k reconfigure

# vi /etc/firewall.conf
   add 4000 divert natd ip from any to any via de0
   add fwd 127.0.0.1,3128 tcp from any to any 80 via de1
   add 65500 allow ip from any to any

# /etc/rc.d/ipfw restart
```

Теперь, независимо от пользовательских настроек, весь HTTP трафик в любом случае пойдет через наш **squid**. Это нам обеспечит новое правило, которое мы добавили в фаерволл. Посмотрите на него внимательно, вот как оно читается: перенаправить локально на порт 3128 весь TCP-трафик по 80 порту, который проходит через внутренний интерфейс **de1**.

Однако, в этом случае нельзя будет использовать аутентификацию пользователей – так уж устроена система. С другой стороны, нужно задать себе вопрос, а нужна ли она вообще, – ведь любой пароль, сохраненный пользователем в браузере, в принципе теряет смысл своего существования.

Перенаправление трафика SQUID (squidguard)

Если вам окажется мало описанных выше методов ограничения и вы захотите не просто разрешать или блокировать доступ в Интернет, а фильтровать и перенаправлять пользователей более тонко, тогда используйте редиректор трафика прокси-сервера **squidguard**:

```
# cd /usr/ports/www/squidguard/
# make install clean
# vi /usr/local/etc/squid/squidGuard.conf
     # системные параметры
   dbhome /var/db/squidGuard
   logdir /var/log
     # определение рабочего времени (Пн-Сб с 8 до 20)
   time workhours { weekly mtwhfa 08:00 - 20:00 }
     # адреса админов и пользователей
   source admins { ip 10.0.0.10 }
   source users { ip 10.0.0.0/24 }
     # подмена мультимедийных файлов
   rewrite media {
       s@.*\.mp3$@http://10.0.0.1/replace/my.mp3@r
       s@.*\.avi$@http://10.0.0.1/replace/my.avi@r
   }
     # база данных нежелательных сайтов
   dest badsites {
       domainlist badsites/domains
       urllist badsites/urls
   }
     # алгоритм работы списков доступа
   acl {
       admins { pass any }
       users within workhours {
           pass !badsites any
           redirect http://www.example.com
           rewrite media
       } else { pass none }
       default { pass none }
   }
```

```
# vi /usr/local/etc/squid/squid.conf
   url_rewrite_program /usr/local/bin/squidGuard
   url_rewrite_children 4

# squid -k reconfigure
```

Разберем конфигурационный файл **squidGuard.conf** – он в принципе читаемый, все должно быть понятно, но все же. Мы определили рабочее время, а так же две группы пользователей: **admins** и **users**. В секции **rewrite media** описана подмена файлов с расширениями **mp3** и **avi** своими заготовками. Так же мы определили свою секцию **dest badsites**, поэтому теперь нам нужно создать базу данных «плохих сайтов» самостоятельно:

```
# mkdir /var/db/squidGuard/badsites
# touch /var/db/squidGuard/badsites/urls
# vi /var/db/squidGuard/badsites/domains
   baddomain1.com
   baddomain2.com

# chown -R squid:squid /var/db/squidGuard/badsites
# squidGuard -C all
# squid -k reconfigure
```

Далее, в секции **acl**, мы описали принципы работы нашего редиректора: админам можно все; пользователям в рабочее время можно все, кроме «плохих сайтов» (иначе – редирект на корпоративный сайт для повышения лояльности к компании) и кроме скачивания музыки и видео – в этом случае нарушитель сможет получить только наши заготовки. Закачка будет происходить локально с нашего сервера, а значит и внешнего трафика никакого не будет (для реализации этой «фишки» нам понадобится локальный веб-сервер, который мы будем устанавливать несколько позднее). В нерабочее же время доступ в Интернет для пользователей будет закрыт.

Настройка локального FTP сервера (proftpd)

Часто бывает полезно иметь возможность попасть на свой сервер по FTP. Такой сервис можно быстро активировать для пользователей, которым разрешен командный интерпретатор, при этом они получают вход в свой домашний каталог. В этом нам поможет демон **inetd**, который используется для вызова программ или других демонов по требованию. За это его еще называют суперсервер. Перед запуском найдите, раскомментируйте строку **ftp** в его конфигурационном файле:

```
# vi /etc/inetd.conf
   ftp stream tcp nowait root /usr/libexec/ftpd ftpd -l
# /etc/rc.d/inetd start
```

При этом лучше не оставлять без внимания два файла:

/etc/ftpusers – пользователи, которым запрещен FTP доступ;
/etc/ftpchroot – пользователи, для которых изменен корневой каталог, в результате чего им доступен лишь их домашний.

Обязательно добавьте в первый файл всех, кому FTP не нужен, а во второй – всех, кому разрешена командная строка.

В свой домашний каталог мы можем попасть браузером или любым другим FTP клиентом по адресу с вводом своего пароля:

ftp://raph@gateway.example.com/

Либо можно указать пароль прямо в адресной строке:

ftp://raph:password@gateway.example.com/

Однако, этот вариант FTP доступа можно использовать только для своих служебных нужд. Если нужен полноценный многопользовательский FTP сервер для передачи больших объемов информации и, возможно даже анонимный, тогда лучше

установить **proftpd** (при этом не забудьте закомментировать строку **ftp** в **/etc/inetd.conf**):

```
# vi /etc/inetd.conf
   #ftp stream tcp nowait root /usr/libexec/ftpd ftpd -l
# /etc/rc.d/inetd restart

# cd /usr/ports/ftp/proftpd/
# make install clean
# vi /usr/local/etc/proftpd.conf
   <Anonymous ~ftp>
      User          ftp
      Group         ftp
      UserAlias     anonymous ftp
      MaxClients    10
      <Limit WRITE>
        DenyAll
      </Limit>
   </Anonymous>

# pw useradd ftp -s sh
# mkdir /home/ftp
# echo proftpd_enable=\"YES\" >> /etc/rc.conf
# /usr/local/etc/rc.d/proftpd start
```

Теперь вы так же можете заходить в свой домашний каталог по адресу *ftp://raph@gateway.example.com/* с вводом пароля или без, а так же, если не указывать имя пользователя в адресе, то вы получите анонимный доступ только на чтение в общую папку **/home/ftp**:

ftp://gateway.example.com/

ВНИМАНИЕ! Помните, что любой анонимный доступ к вашему серверу – это небезопасно. Поэтому, чтобы временно отключить анонимный FTP, добавьте пользователя **ftp** в файл **/etc/ftpusers**. А чтобы убрать анонимный FTP насовсем – удалите пользователя **ftp** и его домашний каталог.

Настройка локального DHCP сервера (dhcpd)

Полноценный шлюз не может называться таковым, если он не обеспечивает выдачу сетевых параметров хостам, готовым принять их автоматически. Для этого нам нужно запустить DHCP сервер – установим порт **isc-dhcp42-server**:

```
# cd /usr/ports/net/isc-dhcp42-server/
# make install clean
# vi /usr/local/etc/dhcpd.conf
     # имя локального домена
   option domain-name "local.example.com";
     # серверы DNS для клиента
   option domain-name-servers 10.0.0.1;
     # параметры времени действия адресов
   default-lease-time 3600;
   max-lease-time 86400;
     # описание нашей локальной сети
   subnet 10.0.0.0 netmask 255.255.255.0 {
       # диапазон адресов к выдаче клиентам
     range 10.0.0.10 10.0.0.200;
       # шлюз по умолчанию для клиента
     option routers 10.0.0.1;
   }

# vi /etc/rc.conf
   dhcpd_enable="YES"
   dhcpd_ifaces="de1"

# /usr/local/etc/rc.d/isc-dhcpd start
```

Демон DHCP сервера будет работать на интерфейсе **de1**, т.е. теперь компьютеры, подключающиеся к вашей локальной сети, будут иметь возможность автоматически получать сетевые настройки: IP-адрес из диапазона **10.0.0.10 - 10.0.0.200** включительно, шлюз по умолчанию **10.0.0.1** и DNS сервер так же **10.0.0.1**.

Защита сервера и локальной сети (ipfw)

Один из самых важных моментов в настройке сервера – это его защита. Нам необходимо настроить фильтрацию сетевого трафика. В нашей системе мы будем использовать межсетевой экран (фаерволл) IPFW – именно для этого мы собирали новое ядро в прошлой главе. Напомним его параметры:

```
# vi /usr/src/sys/amd64/conf/GATEWAY
     # включение системы IPFW
   options    IPFIREWALL
     # включение средств перенаправления трафика FWD
   options    IPFIREWALL_FORWARD
     # включение средств трансляции IP-адресов NAT
   options    IPDIVERT
     # включение средств ограничения скорости PIPE
   options    DUMMYNET
```

Для активации фаерволла мы добавили в **/etc/rc.conf**:

```
# vi /etc/rc.conf
   firewall_enable="YES"
   firewall_type="open"
```

Параметр **firewall_type** определяет режим работы фаерволла и может принимать следующие значения (два из них вам уже знакомы):

open – пропускать весь трафик (открытый фаерволл);
client – защищать только эту машину (локальный);
simple – защищать всю сеть (простой набор правил);
closed – полностью запретить весь трафик, кроме loopback;
[filename] – абсолютный путь к файлу, содержащему правила.

Это не весь список режимов, а в реальной жизни вам пригодятся только два из перечисленных – первый и последний. В самом начале, пока вы заняты установкой и настройкой

дополнительных программ, пусть будет режим **open**, чтобы ничего не мешало. Затем, при настройке защиты, переходите к файлу с правилами.

ПРИМЕЧАНИЕ. Настроенный фаерволл представляет собой упорядоченный список правил с номерами от 1 до 65535. К каждому пакету, попавшему в систему, поочередно применяются критерии каждого правила до тех пор, пока не будет найдено совпадение, либо пока не будет достигнут конец списка. Если совпадение найдено, выполняется действие правила. Последнее правило с номером 65535 нельзя ни изменить, ни удалить.

Теперь поговорим о самих правилах. В общем виде правило системы IPFW имеет следующий формат (упрощенно):

```
cmd number action proto from src to dst options
```

То есть (слева направо): команда (например **add**, **delete**, **flush** и т.д.), номер правила (число от **1** до **65535**), действие (**allow**, **deny**, **count** и т.д.), протокол (**ip**, **tcp**, **udp**, **icmp** и т.д., см. файл **/etc/protocols**), откуда (**from any**, **me**, **IP-адрес порт**), куда (**to any**, **me**, **IP-адрес порт**), дополнительные параметры (**in**, **out via интерфейс** и т.д.). Например:

```
add 500 deny tcp from 10.0.0.10 to any 110
```

Эта команда добавит в IPFW правило с номером 500, которое запрещает любой **tcp** трафик от хоста 10.0.0.10 на любой другой хост по 110 порту. Т.е. пользователь 10.0.0.10 из нашей локальной сети не сможет подсоединиться ни к одному POP3 серверу в мире. В дополнительных условиях можно указать входящий этот трафик или исходящий и по какому интерфейсу должен пройти. Такую запись вы недавно уже видели:

```
add fwd 127.0.0.1,3128 tcp from any to any 80 via de1
```

Эта команда добавит правило, которое перенаправит любой **tcp** трафик на любой другой хост по 80 порту, проходящий через интерфейс **de1** (наша локальная сеть) локально на порт 3128. Если номер не указывается, то система прибавит 100 к последнему существующему правилу (кроме 65535).

Теперь предлагаем вам проверенную временем и отработанную в реальных условиях готовую базовую модель фаерволла:

```
# vi /etc/firewall.conf
    # трансляция IP-адресов NAT
  add 4000 divert natd ip from any to any via de0
    # перенаправление прямого HTTP на SQUID
  add fwd 127.0.0.1,3128 tcp from any to any 80 via de1
    # разрешение служебных пакетов
  add allow ip from any to any via lo0       # внутренн.
  add allow udp from any to any              # udp
  add allow icmp from any to any             # icmp
  add allow ip from any to any frag          # опоздавш.
  add allow tcp from any to any established # установл.
    # разрешение сетевых служб и сервисов
  add allow tcp from any 20 to any setup     # ftp data
  add allow tcp from any to any 21 setup     # ftp cmd
  add allow tcp from any to any 22 setup     # ssh
  add allow tcp from any to any 25 setup     # smtp
  add allow tcp from any to any 53 setup     # named
  add allow tcp from any to any 110 setup    # pop3
  add allow tcp from any to any 143 setup    # imap
  add allow tcp from any to any 465 setup    # smtps
  add allow tcp from any to any 993 setup    # imaps
  add allow tcp from any to any 995 setup    # pop3s
  add allow tcp from any to me 80 setup      # http in
  add allow tcp from me to any 80 setup      # http out
  add allow tcp from me to any 443 setup     # https out
  add allow tcp from any to me 3128 setup    # squid in
    # разрешение вспомогательных портов верхнего уровня
  add allow tcp from any to any 1025-65535 setup

# /etc/rc.d/ipfw restart
```

Информацию о том, к какому именно протоколу относится тот или иной порт можно получить в файле **/etc/services**. Конечно же у каждого своя специфика работы, однако, эта базовая модель на данном этапе подойдет 90% из вас. По некоторым правилам у вас уже пойдет трафик, а некоторые мы написали здесь на перспективу, чтобы не возвращаться к этому вопросу в следующих главах.

После перезагрузки фаерволла мы увидим, что кроме наших правил есть еще несколько, которые добавляются автоматически: три служебных в самом начале с номерами 100, 200 и 300 и еще одно самое последнее с номером 65535:

```
# ipfw show
00100     0     0   allow ip from any to any via lo0
00200     0     0   deny ip from any to 127.0.0.0/8
00300     0     0   deny ip from 127.0.0.0/8 to any
 ... список наших правил ...
65535   123   456   deny ip from any to any
```

Данный фаерволл является экраном закрытого типа, т.е. он работает по принципу: «открыть все, что нужно, остальное – закрыть». Последнее правило, как раз и определяет его тип. Оно запретит прохождение любого пакета, который не подпадет ни под одно из наших разрешающих правил.

ПРИМЕЧАНИЕ. В нашем фаерволле мы сознательно не используем опции, определяющие направление и интерфейс прохождения пакетов, чтобы не запутывать вас на первых же этапах его изучения. Конечно, такие дополнения повысили бы детальность, а значит надежность и неприступность системы. Оставляем этот вопрос вам для самостоятельного изучения, т.к., повторим, это – базовая модель фаерволла.

Пока вы тестируете какое-либо правило можно добавлять и удалять его непосредственно из командной строки, а когда вы уже будете уверены, добавляйте его в соответствующее место в

/etc/firewall.conf, чтобы не потерять при перезагрузке. В основном вам понадобятся следующие команды:

ipfw add [номер][правило] – добавить правило с номером;
ipfw delete [номер] – удалить правило с номером;
ipfw show – показать все текущие правила;
ipfw zero – обнулить счетчики правил.

Чтобы в реальном времени увидеть все пакеты данных, проходящие через ваш сервер, можно использовать команду **tcpdump**, но читабельность этой программы оставляет желать лучшего – вы можете убедиться сами. Поэтому, если хотите хоть немного наглядности, установите **trafshow**:

```
# tcpdump -i de0
 ...
# cd /usr/ports/net/trafshow/
# make install clean
# trafshow -i de0
 ...
```

И напоследок, пища к размышлению и импульс к запуску вашей фантазии – при помощи системы IPFW можно так же ограничивать скорость трафика определенного вида. Пример:

```
# ipfw add pipe 1008 tcp from any to 10.0.0.8 out via de1
# ipfw pipe 1008 config bw 256Kbit/s
```

То есть весь трафик, предназначенный хосту 10.0.0.8, будет уходить от нашего сервера к нему со скоростью не более 256 Кбит в секунду. Таким образом можно «нарезать» наш Интернет для различных хостов и подсетей, а можно и наказывать провинившихся пользователей, понижая им скорость работы в сети до «безопасной». Мы обязательно используем эту возможность в главе 8.

Простая система подсчета трафика (ipa)

Под конец этой главы опишем, как подсчитать и проконтролировать общий объем вашего входящего и исходящего трафика. За этим необходимо обязательно следить, особенно первое время. Для подсчета трафика мы будем использовать программу **ipa** в связке с **ipfw**. Установим из портов все, что нам нужно и сразу создадим конфигурационные файлы **ipa**:

```
# cd /usr/ports/sysutils/ipa
# make install clean
# cd /usr/ports/net/ipa_ipfw
# make install clean
# cd /usr/ports/databases/ipa_sdb
# make install clean

# vi /usr/local/etc/ipa.conf
     # служебные модули и параметры
   ac_mod "ipa_ipfw.so";
   db_mod "ipa_db_sdb.so";
   global {
      update_time = 1m;
      append_time = 1h;
      ac_list = ipfw;
      db_list = sdb;
      ipfw:maxchunk = 1G;
      sdb:db_group = wheel;
   }
     # правила IPA для получения информации из IPFW
   rule IN { ipfw:rules = 800; info = "IP INCOMING"; }
   rule OUT { ipfw:rules = 900; info = "IP OUTGOING"; }

# vi /usr/local/etc/ipastat.conf
     # служебные модули и параметры
   st_mod "ipa_st_sdb.so";
   dynamic_rules = yes;
   global { st_list = sdb; }
```

Теперь нужно добавить в начало нашего фаерволла два правила-счетчика (одно для входящего трафика, другое – для исходящего) для того, чтобы **ipa** снимала с них информацию. Счетчики **count** будут фиксировать количество и объем пакетов и пропускать их дальше по фаерволлу на проверку соответствия основным правилам:

```
# vi /usr/local/etc/firewall.conf
   add 800 count ip from any to me in via de0
   add 900 count ip from me to any out via de0
# /etc/rc.d/ipfw restart
```

Все готово. Запускаем демона **ipa** и наблюдаем за подсчетом проходящих по правилам фаерволла байтов:

```
# echo ipa_enable=\"YES\" >> /etc/rc.conf
# /usr/local/etc/rc.d/ipa start
# ipastat -q -r IN -r OUT
```

Периодически наблюдая за объемами суммарного трафика, вы сможете вовремя почувствовать и отреагировать, если что-либо в вашей системе пойдет не так, как вы того ожидаете. Кроме того, в этой системе реализованы механизмы запуска определенных команд при достижении лимитов и порогов, т.е. можно многое автоматизировать. Мы обязательно займемся этим в главе 8.

ПРИМЕЧАНИЕ. Когда вы устанавливаете из портов какую-либо программу, вместе с ней, если не поленились разработчики, должна появиться и документация, обычно здесь:

/usr/local/share/doc/имя программы/

Если в процессе настройки программы при ее запуске или отладке что-то идет не так, ищите информацию в логах, которые сосредоточены в каталоге: **/var/log/**

Глава 5
Почтовый сервер и фильтрация спама

В этой главе мы будем заниматься исключительно электронной почтой. Здесь мы настроим штатную почтовую систему **sendmail**, после чего перейдем к более развитому и перспективному на наш взгляд **postfix**. Первый вариант подойдет для серверов с небольшим количеством неприхотливых пользователей, ну а к совершенству мы попытаемся приблизиться оседлав именно второй вариант.

Предварительная подготовка Интернет-домена

Для того, чтобы наш сервер мог принимать и отправлять почту, прежде всего нужно создать MX запись в нашем домене. А для того, чтобы создать MX, нужно создать еще и хост (A запись). Для работы почты, будет достаточно, чтобы записи вашего домена выглядели примерно так (приводим только часть файла-описания доменной зоны):

```
@         MX    10 gateway.example.com.
gateway   A     22.22.22.22
```

Но в идеале лучше для почты создавать отдельный хост **mail**, и еще несколько хостов на будущее, чтобы потом уже не возвращаться к этому вопросу:

```
@         MX    10 mail.example.com.
@         A     22.22.22.22
gateway   A     22.22.22.22
mail      A     22.22.22.22
www       CNAME gateway
```

ВНИМАНИЕ! Многие почтовые системы проверяют корректность имени хоста, который пытается передать им письмо, по его IP-адресу, поэтому нужно, чтобы в мире по IP 22.22.22.22 можно было получить корректное имя, и желательно то, которое мы прописали у себя в MX записи. Для этого нужно прописать у вашего провайдера обратную PTR запись для вашего хоста, т.е. **mail.example.com**.

Эффект от изменений у доменного регистратора и у провайдера будет виден не сразу. Все дело в том, что должны обновиться данные о вашем домене в DNS серверах верхнего уровня в мире. Как быстро это произойдет зависит от TTL (времени жизни) вашего домена, но в общем случае этот процесс длится не более одних суток.

Настройка штатного почтового сервера (sendmail)

Теперь можно поработать с настройками **sendmail**. Для начала разрешим его запуск и сетевую работу:

```
# echo sendmail_enable=\"YES\" >> /etc/rc.conf
```

Теперь перейдем в **/etc/mail** и продолжим. Разрешим отправлять письма в мир только из локальной сети (это обязательно, чтобы не превратиться в открытый релей для спамеров):

```
# cd /etc/mail/
# cp access.sample access
# vi access
   10.0.0   RELAY
```

Создадим файл **local-host-names**, в котором пропишем наш домен **example.com**, чтобы **sendmail** принимал для него электронные сообщения для локальных пользователей:

```
# vi local-host-names
   example.com
```

После выполнения этих самых основных настроек нужно перечитать карты конфигурации и перезапустить **sendmail**:

```
# make maps
# make restart
```

Чтобы создать почтовый ящик, нужно просто создать пользователя в системе. Для этого можно воспользоваться командой **adduser**, но, если этот пользователь создается только ради почты, которую будут забирать по POP3, то можно получить прямой доступ к системной базе пользователей и паролей командой **vipw**. Однако, при этом нужно быть предельно внимательными. База эта имеет формат текстового файла с разделителями – двоеточиями. Здесь каждому пользователю

определена строка, имеющая строго определенный формат. Опишем интересующие нас поля:

login:passwd:uid:gid:class:0:0:fullname:homedir:shell

Значения полей соответствуют их наименованиям и должны быть понятны: логин (имя), пароль, код пользователя, код группы, класс, полное имя пользователя, домашний каталог и последнее поле – оболочка (командный интерпретатор).

Создадим теперь почтовый ящик **test@example.com**, т.е. создадим пользователя **test** путем копирования строки, соответствующей нашему пользователю (**raph**):

```
# vipw
   root:*:0:0::0:0:Charlie &:/root:/bin/csh
    ...
   raph:*:1001:0::0:0:Usr&:/home/raph:/usr/local/bin/bash
   test:*:2001:6::0:0:Usr&:/nonexistent:/sbin/nologin
```

Обратите особое внимание на последние два параметра у пользователей **raph** (мы) и **test** (просто почтовый ящик). Не забудьте так же изменить код пользователя, в нашем случае **2001**, а **6** – это код существующей группы **mail**. Информация о группах пользователей системы содержится в файле **/etc/group**.

Теперь в нашей системе появился новый пользователь **test** без домашнего каталога и возможности войти в консоль, но он будет получать электронную почту в нашем домене, а больше нам от него ничего и не нужно. Кроме того, если вы не изменяли поле с паролем (он все равно зашифрован), то пароль у него будет такой же, как у нас. Это конечно же нужно исправить:

```
# passwd test
Changing local password for test
New Password:
Retype New Password:
# _
```

Но можно, для добавления почтового ящика, обойтись одной командой и не проделывать все эти манипуляции с базой данных пользователей системы. Добавим того же пользователя **test** и не забудем создать ему пароль:

```
# pw useradd -n test -g mail -d /nonexistent -s /sbin/nologin
# passwd test
```

Псевдонимы и дополнительные домены (sendmail)

Если вы хотите назначить псевдонимы почтовым ящикам или группам ящиков, используйте файл **/etc/mail/aliases**. Например:

```
# vi /etc/mail/aliases
   info:   user1, user2
   user3:  user3, username@anotherdomain.com

# newaliases
```

В этом случае письмо для **info@example.com** будет доставлено локальным пользователям **user1** и **user2** (пользователя **info** создавать не нужно), а письмо для **user3@example.com** будет доставлено локальному адресату и переслано на внешний адрес за пределы нашей почтовой системы.

Если вы собираетесь принимать почту для нескольких почтовых доменов, то просто добавьте их в **local-host-names**:

```
# vi local-host-names
   example.com
   example2.com
   example3.com
```

Теперь каждый пользователь будет существовать, как почтовый ящик, и принимать почту во всех трех ваших доменах. Если же вы хотите разделить и не смешивать пользователей этих доменов, используйте файл **/etc/mail/virtusertable**. Например:

```
# vi /etc/mail/virtusertable
   user1@example.com     user1
   user2@example2.com    user2
   @example3.com         user3

# make maps && make restart
```

В этом случае адресат **user1** будет существовать только в домене **example.com**, **user2** – в домене **example2.com**, а вся почта домена **example3.com** отправится в ящик пользователя **user3**.

ВНИМАНИЕ! После редактирования конфигурационных файлов **sendmail**, не забывайте перекомпилировать карты и перезапускать сам **sendmail** для применения новых параметров (вы должны находиться в каталоге **/etc/mail**):

```
# newaliases
# make maps
# make restart
```

Отправка и получение сообщений (mail, cucipop)

Вся входящая почта пользователей системы будет записываться в каталог **/var/mail** в файл, соответствующий имени пользователя. Посмотреть содержимое почтового ящика можно командой **mail**. Ей же можно и отправить письмо из

командной строки. Вообще у нее много возможностей и даже есть более менее нормальная справка – разобраться не составит труда:

mail test@example.com – отправить письмо по адресу;
mail -u test – поработать с ящиком **/var/mail/test**.

Теперь все-таки мы должны отдать почту нашим пользователям в его их почтовые программы по протоколу POP3. Для этого необходимо установить соответствующую службу, например **cucipop**. Тут нам снова поможет система портов:

```
# cd /usr/ports/mail/cucipop/
# make install clean
```

Но **cucipop** – это не демон, постоянно находящийся в памяти, а программа, которую нужно запустить при обращении к серверу по протоколу POP3 (TCP порт 110). В этом нам поможет уже известный вам демон **inetd**. Найдите, раскомментируйте и отредактируйте в его конфигурационном файле строку, которая начинается с **pop3**:

```
# vi /etc/inetd.conf
   pop3 stream tcp nowait root /usr/local/libexec/cucipop cucipop

# echo inetd_enable=\"YES\" >> /etc/rc.conf
# /etc/rc.d/inetd restart
```

Напомним, информация о том, что **pop3** – это 110 порт, содержится в файле **/etc/services** и в этом же файле перечислены и другие порты, которые назначены определенным сетевым протоколам.

Теперь ваш сервер умеет полноценно работать с электронной почтой, т.е. принимать сообщения для пользователей вашего домена от других почтовых систем и сохранять их у себя в каталоге **/var/mail**, отдавать пользователям их сообщения по протоколу POP3, а так же отправлять сообщения из локальной

сети в мир. Если что-то идет не так, вы можете инициировать SMTP сессию и, зная команды протокола, пообщаться со своим **sendmail** вживую. Это выглядит примерно так:

```
# telnet localhost 25
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
220 mail.example.com ESMTP Sendmail
helo me
250 mail.example.com Hello localhost, pleased to meet you
mail from: admin@freebsd.org
250 2.1.0 admin@freebsd.org... Sender ok
rcpt to: test
250 2.1.5 test... Recipient ok
data
354 Enter mail, end with "." on a line by itself
This is a TEST MESSAGE!!!
.
250 2.0.0 r099qYxS001511 Message accepted for delivery
^]
telnet> Connection closed.
```

Это все. Теперь, чтобы пользоваться электронной почтой, нужно прописать в настройках почтовых программ адреса POP3 и SMTP серверов: 10.0.0.1, а так же логин и пароль пользователя системы.

Однако, нужно признать, что данная почтовая система является довольно-таки примитивной. Ее преимущество – простота запуска. Она, конечно, сможет обслужить и большое количество пользователей и справится с серьезной нагрузкой, но управлять и работать с ней каждый день неудобно. Поэтому далее мы будем ее совершенствовать.

Установка альтернативного MTA (postfix)

Первое, что мы сделаем для дальнейшего развития нашей почтовой системы, это заменим фундамент. Вместо штатного **sendmail** установим более развитый и перспективный почтовик **postfix**. Однако, давайте сразу выясним один принципиальный момент – как администрировать и где хранить информацию о доменах, псевдонимах и пользователях. Глобально мы имеем два варианта и выбор нужно сделать сейчас.

Первый – это когда для ящиков используются системные пользователи, а информация о псевдонимах и доменах хранится в текстовых файлах (это мы только что настраивали в **sendmail**). В этом случае мы администрируем систему из командной строки.

И второй – когда вся информация о доменах, пользователях, их паролях и псевдонимах хранится в базе данных. В этом случае систему можно администрировать из консоли, но гораздо удобнее было бы использовать веб-интерфейс.

ПРИМЕЧАНИЕ. Второй вариант предпочтительнее, когда вы разворачиваете почтовую систему на несколько почтовых доменов с большим количеством почтовых ящиков. В этом случае вы не перегружаете ваш сервер системными пользователями, но вам не обойтись без СУБД и веб-сервера. Первый же вариант вполне подойдет для построения небольших почтовых систем.

Мы будем рассматривать оба варианта, причем основным и опорным будет первый, т.к. он немного проще в реализации, а все, что нужно для второго, отметим отдельно. Если же вы хотите испытать работу обоих схем, начните с первой, а потом просто переустановите программы с новыми опциями и добавьте все, что необходимо. Итак:

```
# cd /usr/ports/mail/postfix/
# make install clean
```

В меню выбора параметров установки для варианта без MySQL (первый) выбираем:

```
[*] PCRE   Perl Compatible Regular Expressions
[*] SASL2  Cyrus SASLv2 (Simple Auth.and Sec.Layer)
[*] TLS    Enable SSL and TLS support
```

Если хотите установить связку с MySQL (второй вариант), то:

```
[*] PCRE     Perl Compatible Regular Expressions
[*] DOVECOT2 Dovecot 2.x SASL authentication method
[*] TLS      Enable SSL and TLS support
[*] MYSQL    MySQL maps (uses WITH_MYSQL_VER)
```

То есть мы меняем SASL и добавляем MySQL. В конце установки, независимо от выбранного варианта работы, на вопрос «хотим ли мы активировать Postfix» ответим утвердительно:

```
Would you like to activate Postfix in
/etc/mail/mailer.conf [n]? y
```

Новый почтовый сервер установлен. Теперь, разрешим запуск **postfix** и запретим **sendmail**, чтобы не мешал. Добавим следующие строки в **rc.conf**, а файл **periodic.conf** придется создать, т.к. он пока не существует:

```
# vi /etc/rc.conf
   postfix_enable="YES"
   sendmail_enable="NO"
   sendmail_submit_enable="NO"
   sendmail_outbound_enable="NO"
   sendmail_msp_queue_enable="NO"

# vi /etc/periodic.conf
   daily_clean_hoststat_enable="NO"
   daily_status_mail_rejects_enable="NO"
   daily_status_include_submit_mailq="NO"
   daily_submit_queuerun="NO"
```

Теперь приступим к конфигурированию самого **postfix**:

```
# vi /usr/local/etc/postfix/main.cf
    # параметры для SMTP сессии
    # должно совпадать с PTR у провайдера
  myhostname = mail.example.com
  mydomain = example.com
    # что добавлять к неполному адресу
  myorigin = $mydomain
    # для каких доменов принимать почту локально
  mydestination = $mydomain
    # какие интерфейсы и сети обслуживать
  inet_interfaces = all
  mynetworks_style = subnet
  mynetworks = 10.0.0.0/24, 127.0.0.1/32
    # ограничения размеров письма и почтового ящика
  message_size_limit = 10485760
  mailbox_size_limit = 1073741824
    # ограничения на передачу сообщений
  smtpd_recipient_restrictions =
        # разрешить отправку из локальной сети
      permit_mynetworks,
        # запретить для неизвестных локальных адресов
      reject_unauth_destination
```

Секция **smtpd_recipient_restrictions** очень важна. Именно эти два параметра в ней не дают превратить наш почтовый сервер в открытый релей – они должны присутствовать в конфигурации всегда.

Останавливаем **sendmail** и запускаем в работу **postfix**. Так же необходимо выполнить команду **newaliases**, так как **postfix** будет пытаться найти список псевдонимов и, если не найдет, будет ругаться вплоть до отказа принимать почту:

```
# /etc/rc.d/sendmail stop
# postfix check
# /usr/local/etc/rc.d/postfix start
# newaliases
```

Почтовые ящики – это те же локальные пользователи системы. Хранится почта там же, в **/var/mail**, и забирать ее с сервера можно так же по POP3 при помощи уже установленного **cucipop**, т.е. для пользователей ничего не изменилось. Теперь можно протестировать работу **postfix** так же, как тестировали **sendmail**. Все это подробно описано в предыдущих разделах, посвященных настройке **sendmail**:

```
# telnet localhost 25
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
220 mail.example.com ESMTP Postfix
helo me
250 mail.example.com
mail from: admin@freebsd.org
250 2.1.0 Ok
rcpt to: test
250 2.1.5 Ok
data
354 End data with <CR><LF>.<CR><LF>
This is a TEST MESSAGE!!!
.
250 2.0.0 Ok: queued as 7D37CA2F153
^]
telnet> Connection closed.
```

Если вы собираетесь принимать почту для нескольких доменов, то просто добавьте их в конфигурацию **postfix**:

```
# vi /usr/local/etc/postfix/main.cf
     # для каких доменов принимать почту локально
   mydestination = $mydomain, example2.com, example3.com
```

Теперь каждый пользователь будет существовать, как почтовый ящик, и принимать почту во всех трех ваших доменах. Если же вы хотите разделить и не смешивать пользователей этих доменов, тогда нужно настраивать виртуальные домены и ящики:

```
# vi /usr/local/etc/postfix/main.cf
     # для каких доменов принимать почту локально
   mydestination = $myhostname
     # путь к файлу с виртуальными доменами
   virtual_alias_domains = hash:/usr/local/etc/postfix/v
irtual_alias_domains
     # путь к файлу с виртуальными пользователями
   virtual_alias_maps = hash:/usr/local/etc/postfix/virt
ual_alias_maps

# vi /usr/local/etc/postfix/virtual_alias_domains
   example.com    20130101
   example2.com   20130101
   example3.com   20130101

# vi /usr/local/etc/postfix/virtual_alias_maps
   user1@example.com     user1
   user2@example2.com    user2
   @example3.com         user3

# postmap hash:/usr/local/etc/postfix/virtual_alias_domains
# postmap hash:/usr/local/etc/postfix/virtual_alias_maps
# /usr/local/etc/rc.d/postfix restart
```

ВНИМАНИЕ! Если вы указываете домен в списке виртуальных, не используйте его в качестве параметра **mydestination**, т.к. система не будет знать, как ей поступить: доставлять почту локально или же отправлять ее на виртуальное преобразование и, возможно даже, откажется работать.

Во избежание путаницы мы изменили значение параметра **mydestination** и добавили два списка – виртуальных доменов и пользователей. В первом прописали все наши почтовые домены, а во втором – соответствие почтовых адресов локальным пользователям, т.е. распределили их по доменам. Можно было бы не менять значение **mydestination**, но тогда и не добавлять **example.com** в список виртуальных доменов. Тогда он был бы основным и работал бы для всех локальных пользователей.

Связь POSTFIX с базой данных (postfix + mysql)

Если вы устанавливаете **postfix** в связке с **postfixadmin** и **mysql**, тогда все ваши почтовые домены будут виртуальными и о локальных пользователях системы речь идти не будет. Конфигурация должна будет выглядеть примерно так (выделим только изменения и дополнения):

```
# vi /usr/local/etc/postfix/main.cf
   myhostname = mail.example.com
   mydomain = example.com
   myorigin = $mydomain
     # для каких доменов принимать почту локально
   mydestination = $myhostname
     # путь к файлу с запросом к MySQL о псевдонимах
   virtual_alias_maps = proxy:mysql:/usr/local/etc/postfi
x/mysql_virtual_alias_maps.cf
     # путь к файлу с запросом к MySQL о пользователях
   virtual_mailbox_maps = proxy:mysql:/usr/local/etc/post
fix/mysql_virtual_mailbox_maps.cf
     # путь к файлу с запросом к MySQL о почтовых доменах
   virtual_mailbox_domains = proxy:mysql:/usr/local/etc/p
ostfix/mysql_virtual_domains_maps.cf
     # каталог для хранения всех почтовых ящиков
   virtual_mailbox_base = /usr/mail
     # описание пользователя для работы с почтой
   virtual_minimum_uid = 65534
   virtual_uid_maps = static:65534
   virtual_gid_maps = static:65534

   inet_interfaces = all
   mynetworks_style = subnet
   mynetworks = 10.0.0.0/24, 127.0.0.1/32
   message_size_limit = 10485760
   mailbox_size_limit = 1073741824
   smtpd_recipient_restrictions =
       permit_mynetworks,
       reject_unauth_destination
```

Создадим каталог для хранения почтовых ящиков (параметр **virtual_mailbox_base**) и три файла, в которых опишем запросы к базе данных для получения информации о псевдонимах, пользователях и доменах, настраиваемых в **postfixadmin**:

```
# mkdir /usr/mail
# chown 65534:65534 /usr/mail

# vi /usr/local/etc/postfix/mysql_virtual_alias_maps.cf
   user = postfix
   password = pass
   hosts = localhost
   dbname = postfix
   query = SELECT goto FROM alias WHERE address='%s' AND
active = '1'

# vi /usr/local/etc/postfix/mysql_virtual_mailbox_maps.cf
   user = postfix
   password = pass
   hosts = localhost
   dbname = postfix
   query = SELECT maildir FROM mailbox WHERE username='%s'
AND active = '1'

# vi /usr/local/etc/postfix/mysql_virtual_domains_maps.cf
   user = postfix
   password = pass
   hosts = localhost
   dbname = postfix
   query = SELECT domain FROM domain WHERE domain='%s' AND
active = '1'
```

ВНИМАНИЕ! Чтобы двигаться дальше, нужно обеспечить **postfix** информацией. Вам придется немного забежать вперед и выполнить первые два раздела следующей главы – установите веб-сервер **apache**, СУБД **mysql**, язык **php** и интерфейс **postfixadmin**. Только после этого запускайте **postfix** и продолжайте настройку системы.

Установка POP3/IMAP4 сервера (dovecot)

Теперь заменим установленный нами **cucipop** на более совершенный **dovecot**. Ведь мы хотим предоставлять пользователям не только **POP3**, но и **IMAP4** – это очень удобно при работе с почтой с нескольких устройств, в том числе и мобильных:

```
# cd /usr/ports/mail/dovecot2/
# make install clean
```

В меню выбора параметров установки для варианта без MySQL (первый) оставьте все по умолчанию. Если же хотите установить связку с MySQL (второй вариант), то отметте соответственно **[*] MYSQL**.

Установщик почему-то не создает конфигурационные файлы в каталоге **/usr/local/etc/dovecot**, поэтому после инсталляции перепишите их туда вручную, после чего внесите в них следующие изменения:

```
# cp -r /usr/local/share/doc/dovecot/example-config/* /u
sr/local/etc/dovecot/

# vi /usr/local/etc/dovecot/dovecot.conf
     # работаем с пользователями на всех интерфейсах
   listen = *

# vi /usr/local/etc/dovecot/conf.d/10-auth.conf
     # разрешаем аутентификацию открытым текстом
   disable_plaintext_auth = no
     # аутентификация системных пользователей
   !include auth-system.conf.ext

# vi /usr/local/etc/dovecot/conf.d/10-ssl.conf
     # временно отключаем SSL/TLS
   ssl = no
   #ssl_cert = </etc/ssl/certs/dovecot.pem
   #ssl_key = </etc/ssl/private/dovecot.pem
```

```
# vi /usr/local/etc/dovecot/conf.d/10-mail.conf
     # формат и место хранения почтовых папок
   mail_location = mbox:~/mail:INBOX=/var/mail/%u
     # ограничиваем возможность работы пользователей
   first_valid_uid = 500
   last_valid_uid = 0
   first_valid_gid = 1
   last_valid_gid = 0

# echo dovecot_enable=\"YES\" >> /etc/rc.conf
# /usr/local/etc/rc.d/dovecot start
```

Перед запуском **dovecot** не забудьте закомментировать строку **pop3** в **/etc/inetd.conf** и перезапустить демон **inetd**, иначе может возникнуть конфликт. Обратите так же внимание на то, что пользователи из группы **wheel** не смогут пользоваться **dovecot**, т.к. эта группа имеет код **0**. И это правильно. Не стоит смешивать администраторов системы с почтовыми пользователями.

В принципе все готово для работы с сервером IMAP. Однако, если вы сейчас захотите проверить его работу почтовым клиентом, то у вас, возможно, ничего не получится. А дело в том, что в нашей конфигурации в параметре **mail_location** прописано хранение почты в домашних каталогах пользователей, и, если таковых нет (как вы помните, вместо домашнего каталога мы писали **/nonexistent**), то и входящую почту складывать некуда. Лучше всего удалить эти почтовые ящики при помощи **vipw** и создать пользователей заново командой **adduser** так же без командной строки, но уже вместе с домашними каталогами.

Если же вы занимаетесь миграцией после определенного периода работы и таких ящиков у вас уже много, на помощь приходит **man** и все решается. Команда **adduser** умеет массово создавать сколько угодно пользователей, описанных в текстовом

файле специального формата (такого же, как **/etc/passwd**, только поле с паролем не зашифровано и расположено в конце строки):

```
# vi /usr/local/etc/postfix/newmails.txt
   user1:2001:6:::::/home/user1:/usr/sbin/nologin:pass123
   user2:2002:6:::::/home/user2:/usr/sbin/nologin:pass456

# adduser -f /usr/local/etc/postfix/newmails.txt
```

Теперь ваши пользователи могут работать с почтой, подключаясь к вашему серверу по IMAP4. Параметры для подключения те же, кроме самого протокола (IMAP вместо POP3).

Связь DOVECOT с базой данных (dovecot + mysql)

Если вы двигаетесь по пути №2 и устанавливаете связку с базой данных **mysql**, тогда приведите конфигурацию **dovecot** к следующему виду (выделим так же только изменения и дополнения):

```
# vi /usr/local/etc/dovecot/conf.d/10-auth.conf
     # разрешаем аутентификацию открытым текстом
   disable_plaintext_auth = no
     # аутентификация базы данных SQL
   #!include auth-system.conf.ext
   !include auth-sql.conf.ext

# vi /usr/local/etc/dovecot/conf.d/10-mail.conf
     # формат и место хранения почтовых папок
   mail_location = maildir:/usr/mail/%d/%n
     # ограничиваем возможность работы пользователей
   #first_valid_uid = 500
   #last_valid_uid = 0
   first_valid_gid = 65534
   #last_valid_gid = 0
```

```
# vi /usr/local/etc/dovecot/conf.d/auth-sql.conf.ext
     # путь к файлу с запросом к MySQL о паролях
   passdb {
     driver = sql
     args = /usr/local/etc/dovecot/dovecot-sql.conf.ext
   }
     # путь к файлу с запросом к MySQL о пользователях
   userdb {
     driver = sql
     args = /usr/local/etc/dovecot/dovecot-sql.conf.ext
   }
```

Ну и по аналогии с **postfix**, создадим теперь файл с описанием запросов к базе данных о пользователях и их паролях:

```
# vi /usr/local/etc/dovecot/dovecot-sql.conf.ext
   driver = mysql
   connect = host=localhost dbname=postfix user=postfix pa
ssword=pass
   default_pass_scheme = MD5-CRYPT
   password_query = SELECT username AS user,password FROM
mailbox WHERE username = '%u' AND active='1'
   user_query = SELECT maildir, 65534 AS uid, 65534 AS gid
FROM mailbox WHERE username = '%u' AND active='1'
```

ВНИМАНИЕ! Чтобы двигаться дальше, нужно так же, как и **postfix**, обеспечить информацией и **dovecot**. Если вы до сих пор еще этого не сделали, то обязательно выполните первые два раздела следующей главы прямо сейчас – установите веб-сервер **apache**, СУБД **mysql**, язык **php** и интерфейс **postfixadmin**. Иначе ваш **dovecot** просто не запустится.

Отличия двух вариантов и конвертация (mb2md)

Если вы реализовали сначала первый вариант системы, а потом решили на том же сервере перейти на второй, тогда вам придется переустановить **postfix** и **dovecot** с поддержкой MySQL. Для этого нужно остановить соответствующий процесс и пересобрать программу заново. Например, для **postfix**:

```
# /usr/local/etc/rc.d/postfix stop
# cd /usr/ports/mail/postfix/
# make config reinstall clean
```

Последняя команда в нашем случае должна быть именно такой. Опция **config** нужна для того, чтобы вызвать меню выбора параметров. Если запустить переустановку без нее, то меню не будет, т.к. эти параметры сохранились с прошлого раза в текстовый файл **/var/db/ports/postfix/options**. Поэтому нужно либо удалить этот файл, либо использовать опцию **config** при запуске команды **make**.

При переустановке программы старые конфигурационные файлы обычно остаются на месте. Для **dovecot** действия будут абсолютно аналогичными:

```
# /usr/local/etc/rc.d/dovecot stop
# cd /usr/ports/mail/dovecot2/
# make config reinstall clean
```

С первым вариантом реализации почтовой системы все понятно. Опишем принципиальные отличия второго варианта:

1. Все почтовые домены и ящики – виртуальные. Информация о них хранится в базе данных **mysql**. Для создания почтового адреса не нужно создавать пользователей в системе;

2. Почта хранится не в домашних каталогах пользователей, а в каталоге: **/usr/mail/[домен]/[пользователь]** в более гибком формате **maildir**;

3. Для входа в свой почтовый ящик пользователи должны к логину приписать еще и домен, т.е. в качестве логина использовать полный адрес своей электронной почты;

4. Создавать/удалять почтовые домены, пользователей и псевдонимы нужно в специальном веб-интерфейсе **postfixadmin**. Вы можете передать администрирование почтовой системы кому-либо не допуская его в консоль.

Есть так же и конверторы почтовых ящиков, которые помогут вам перенести уже существующую почту ваших пользователей в новый формат – например **dsync**, который входит в состав **dovecot**, или сторонний скрипт **mb2md**, который устанавливается из портов и работает очень просто, например:

```
# cd /usr/ports/mail/mb2md/
# make install clean
# mb2md -s /home/test/mail/ -R -d /usr/mail/example.com/test/
```

Эта команда перенесет почту пользователя системы **test** из формата **mailbox** в формат **maildir** виртуальному пользователю **test@example.com**. Однако, текущую схему работы с учетом перспектив лучше конечно же настраивать сразу, т.к. любая конвертация данных не дает 100% гарантии того, что все выполнится корректно.

Аутентификация при отправке почты (cyrus-sasl)

Этот раздел выполняйте только в том случае, если настраиваете первый вариант системы (без MySQL). Итак, если вы собираетесь пользоваться почтовым сервером для отправки писем не из локальной сети, а извне, например с мобильных устройств, то для обеспечения безопасности необходимо настроить SMTP аутентификацию:

```
# cd /usr/ports/security/cyrus-sasl2-saslauthd/
# make install clean

# vi /usr/local/lib/sasl2/smtpd.conf
   pwcheck_method: saslauthd
   mech_list: PLAIN LOGIN

# vi /usr/local/etc/postfix/main.cf
   smtpd_recipient_restrictions =
       permit_mynetworks,
         # разрешить отправку аутентифицированным
       permit_sasl_authenticated,
       reject_unauth_destination
     # подключение аутентификации при отправке
   smtpd_sasl_auth_enable = yes
   smtpd_sasl_security_options = noanonymous
   broken_sasl_auth_clients = yes

# echo saslauthd_enable=\"YES\" >> /etc/rc.conf
# /usr/local/etc/rc.d/saslauthd start
# /usr/local/etc/rc.d/postfix restart
```

Теперь, если письмо будет отправляться с устройства не из вашей локальной сети, **postfix** потребует аутентификацию пользователя, т.е. предоставить логин и пароль (в нашем случае мы не храним логины и пароли почтовых аккаунтов в отдельном месте и везде используем аутентификацию системы, т.е. для POP3, IMAP и SMTP они будут одни и те же – системные).

Аутентификация при отправке (dovecot-sasl + mysql)

Этот раздел выполняйте только в том случае, если настраиваете второй вариант системы (с MySQL), иначе – выполните предыдущий.

Традиционный Cyrus SASL, описанный в предыдущем разделе не работает с методом шифрования паролей, применяемым по умолчанию в **postfixadmin**, поэтому мы построим данный сервис на базе Dovecot SASL. Внесите следующие изменения и перезапустите службы:

```
# vi /usr/local/etc/postfix/main.cf
   smtpd_recipient_restrictions =
       permit_mynetworks,
         # разрешить отправку аутентифицированным
       permit_sasl_authenticated,
       reject_unauth_destination
     # подключение аутентификации при отправке
   smtpd_sasl_auth_enable = yes
   smtpd_sasl_type = dovecot
   smtpd_sasl_path = private/auth
   smtpd_sasl_security_options = noanonymous
   broken_sasl_auth_clients = yes

# vi /usr/local/etc/dovecot/conf.d/10-master.conf
   service auth {
     unix_listener /var/spool/postfix/private/auth {
       mode = 0666
       user = postfix
       group = postfix
     }
   }

# /usr/local/etc/rc.d/postfix restart
# /usr/local/etc/rc.d/dovecot restart
```

Шифрование почтового трафика SSL/TLS (openssl)

При активном использовании мобильных устройств, кроме аутентификации, желательно еще и шифровать почтовый трафик между клиентом и сервером. Для этого нужно сначала создать SSL сертификат и ключ сервера, а затем внести изменения в конфигурационные файлы **postfix** и **dovecot**:

```
# cd /etc/ssl
# openssl req -new -x509 -nodes -out cert.pem -keyout
key .pem -days 365
Country Name (2 letter code) []:UA
State or Province Name (full name) []:Ukraine
Locality Name (eg, city) []:Kiev
Organization Name (eg, company) []:EXAMPLE LTD
Organizational Unit Name (eg, section) []:MAIL SERVER
Common Name (e.g. server FQDN) []:mail.example.com
Email Address []:postmaster@example.com

# vi /usr/local/etc/postfix/main.cf
     # подключение шифрования SSL/TLS
   smtpd_use_tls = yes
   smtpd_tls_received_header = yes
   smtpd_tls_cert_file = /etc/ssl/cert.pem
   smtpd_tls_key_file = /etc/ssl/key.pem

# vi /usr/local/etc/postfix/master.cf
   smtps   inet   n   -   n   -   -   smtpd
     -o smtpd_tls_wrappermode=yes

# vi /usr/local/etc/dovecot/conf.d/10-ssl.conf
     # включаем шифрование SSL/TLS
   ssl = yes
   ssl_cert = </etc/ssl/cert.pem
   ssl_key = </etc/ssl/key.pem

# /usr/local/etc/rc.d/postfix restart
# /usr/local/etc/rc.d/dovecot restart
```

ВНИМАНИЕ! Мы впервые коснулись второго конфигурационного файла **postfix** – файла конфигурации транспорта **master.cf**. Редактировать его нужно очень осторожно и внимательно – те две строки нужно просто найти и раскомментировать, причем в начале второй строки должно быть ровно два пробела – не больше и не меньше, иначе не запустится **postfix**.

Необходимые для работы SSL/TLS порты: 465 (smtps), 993 (imaps) и 995 (pop3s) мы открыли в фаерволле заранее – можно тестировать шифрование мобильными и стационарными почтовыми клиентами.

Антивирусный контроль писем (clamav)

Еще один штрих к нашему почтовому серверу – это антивирусный контроль всех проходящих через него писем. Для этого мы будем использовать бесплатный, но достаточно эффективный **clamav**. Установка займет примерно 12 минут:

```
# cd /usr/ports/security/clamav-milter/
# make install clean

# vi /etc/rc.conf
   clamav_clamd_enable="YES"
   clamav_milter_enable="YES"
   clamav_freshclam_enable="YES"

# vi /usr/local/etc/clamav-milter.conf
     # отбой письма при обнаружении вируса
   OnInfected Reject
     # сообщение отправителю при отбое письма
   RejectMsg "VIRUS DETECTED: %v"
     # добавлять информацию в заголовки писем
   AddHeader Replace
```

```
# vi /usr/local/etc/freshclam.conf
     # ближайший сервер обновлений
   DatabaseMirror db.ua.clamav.net

# vi /usr/local/etc/postfix/main.cf
     # подключаем антивирус, как почтовый фильтр
   smtpd_milters = unix:/var/run/clamav/clmilter.sock
   milter_default_action = accept

# /usr/local/etc/rc.d/clamav-freshclam start
# /usr/local/etc/rc.d/clamav-clamd start
# /usr/local/etc/rc.d/clamav-milter start
# /usr/local/etc/rc.d/postfix restart
```

Антивирус готов. Теперь, если заглянуть в заголовок входящего письма, можно увидеть там следующие поля:

```
X-Virus-Scanned: clamav-milter 0.97.6 at mail.example.com
X-Virus-Status: Clean
```

Стоп-спам. Борьба средствами POSTFIX (postfix)

Что ж, теперь мы с вами имеем довольно-таки неплохой почтовик, достаточно надежный и гибкий в управлении и эксплуатации, но это окажется временно. Рано или поздно возникает вопрос: «Откуда берется спам и что с этой ситуацией делать?». Главное – не паниковать и последовательно настроить эффективную защиту от спама.

ПРИМЕЧАНИЕ. Вообще существует четыре глобальных метода борьбы с нежелательной почтой – фильтрация средствами самого MTA, технология «серых списков», использование внешних блок-листов и использование локального спам-фильтра. Здесь мы последовательно рассмотрим все четыре и сделаем выводы о наиболее эффективном их сочетании на практике.

Итак, первое, что нужно сделать – это правильно настроить **postfix** для того, чтобы он сам еще на этапе SMTP сессии выявлял и отсекал ненадежные и подозрительные хосты, явно похожие на спамеров. Для этого мы будем использовать механизмы и возможности уже известной нам секции **smtpd_recipient_restrictions** основной конфигурации:

```
# vi /usr/local/etc/postfix/main.cf
     # пустой отправитель для пробных сообщений
   address_verify_sender = <>
     # отложенная обработка всех ограничений
   smtpd_delay_reject = yes
     # обязательное приветствие HELO/EHLO
   smtpd_helo_required = yes
     # запрет проверки существования получателя
   disable_vrfy_command = yes
     # ограничения на передачу сообщений
   smtpd_recipient_restrictions =
       permit_mynetworks,
       permit_sasl_authenticated,
       reject_unauth_destination,
         # запретить некорректные команды конвейера
       reject_unauth_pipelining,
         # списки для проверки приветствий, имен хостов,
         # адресов отправителей и получателей
       check_helo_access hash:/usr/local/etc/postfix/acc
ess_helo,
       check_client_access hash:/usr/local/etc/postfix/a
ccess_client,
       check_sender_access hash:/usr/local/etc/postfix/a
ccess_sender,
       check_recipient_access hash:/usr/local/etc/postfi
x/access_recipient,
         # запретить, если имя хоста отсутствует в DNS
       reject_unknown_client_hostname,
         # запретить, если приветствие некорректно
       reject_non_fqdn_helo_hostname,
       reject_invalid_helo_hostname,
       reject_unknown_helo_hostname,
```

```
        # запретить, если адрес отправителя некорректен
      reject_non_fqdn_sender,
      reject_unknown_sender_domain,
      reject_unverified_sender,
        # запретить, если адрес получателя некорректен
      reject_non_fqdn_recipient,
      reject_unknown_recipient_domain,
      reject_unverified_recipient

# vi /usr/local/etc/postfix/access_helo
   10                          REJECT Incorrect config.
   172.16                      REJECT Incorrect config.
   192.168                     REJECT Incorrect config.
   127.0.0.1                   REJECT Incorrect config.
   localhost                   REJECT Incorrect config.
   localhost.localdomain       REJECT Incorrect config.
   22.22.22.22                 REJECT You are not me.
   example.com                 REJECT You are not me.
   gateway.example.com         REJECT You are not me.
   localhost.example.com       REJECT You are not me.

# cd /usr/local/etc/postfix
# touch access_client access_sender access_recipient

# postmap hash:/usr/local/etc/postfix/access_helo
# postmap hash:/usr/local/etc/postfix/access_client
# postmap hash:/usr/local/etc/postfix/access_sender
# postmap hash:/usr/local/etc/postfix/access_recipient

# /usr/local/etc/rc.d/postfix restart
```

Подробное описание всех этих директив вы можете найти в документации **postfix**. Вкратце можно сказать, что мы настроили отсечение неправильно настроенных хостов, хостов с некорректными именами, не имеющими обратной PTR записи, несуществующих в DNS хостов, доменов и т.д. Однако, бывает так, что вам пытается передать нормальное письмо нормальный, но неправильно настроенный почтовик – для таких ситуаций мы создали соответствующие списки.

Файл **access_helo** предназначен для описания в нем некорректных приветствий хостов, передающих нам почту. Выше мы описали возможные ошибки в приветствии и решили отклонять такие сессии. Если хотите разрешить какое-либо некорректное приветствие, добавьте его отдельной строкой по аналогии с существующими и замените **REJECT** на **OK**. Аналогично настраиваются списки имен хостов (**access_client**), списки адресов отправителей (**access_sender**) и получателей (**access_recipient**). Пока мы создали их пустыми, но в перспективе вы можете использовать их как белые списки для того, чтобы быстро решать ситуации, когда из-за некорректной настройки удаленного хоста ваш сервер отказывается принимать от него полезные письма. Например:

```
# vi /usr/local/etc/postfix/access_sender
   user@remotedomain.com      OK

# postmap hash:/usr/local/etc/postfix/access_sender
# /usr/local/etc/rc.d/postfix restart
```

Такая запись исключит все проверки хоста на корректность для указанного отправителя, т.к. параметры проверки по спискам в секции **smtpd_recipient_restrictions** находятся выше.

ВНИМАНИЕ! Если в конфигурации **postfix** есть активная ссылка на какой-либо файл, содержащий индексированный список ограничений, то этот файл должен обязательно существовать, можно даже пустой, иначе **postfix** не будет работать. В противном случае необходимо закомментировать параметр ограничения.

Однако, спамеры не стоят на месте и таким образом вы заблокируете всего лишь до 30% спама. Результат не самый плохой и иногда, особенно на первых этапах работы с новым доменом, этого может оказаться достаточно, но нам этого мало.

Стоп-спам. Технология серых списков (postgrey)

Довольно интересной является технология «серых списков». Если какой-либо хост отправляет нам письмо впервые, то наш сервер не будет принимать его сразу, а как бы поставит на паузу выдав при этом хосту-отправителю сообщение «попробуйте немного позже» при этом записав себе имя хоста и адрес отправителя. Если хост «нормальный», он через какое-то время попробует отдать нашему серверу письмо еще раз, а некоторые спамеры часто этим уже не занимаются. После нормальной передачи письма хост заносится в белый список и в дальнейшем его письма принимаются без задержек. Установим программу **postgrey**:

```
# cd /usr/ports/mail/postgrey/
# make install clean

# vi /usr/local/etc/postfix/main.cf
     # ограничения на передачу сообщений
   smtpd_recipient_restrictions =
        ...
       reject_unverified_recipient,
         # передача сообщений фильтру POSTGREY
       check_policy_service inet:127.0.0.1:10023

# echo postgrey_enable=\"YES\" >> /etc/rc.conf
# /usr/local/etc/rc.d/postgrey start
# /usr/local/etc/rc.d/postfix restart
```

Теперь, какое-то время, пока хосты-отправители не пройдут нашу проверку письма будут приходить с небольшой задержкой, а в заголовке пришедших писем вы увидите новое поле X-Greylist, но оно, возможно, будет пустым, т.к. на момент написания книги в скрипте запуска **postgrey** была ошибка, исправить которую вы можете сами. Найдите и приведите строку с описанием текста заголовка к следующему виду:

```
# vi /usr/local/etc/rc.d/postgrey
     --x-greylist-header='X-Greylist: delayed %t seconds
by postgrey-%v at %h; %d'"}

# /usr/local/etc/rc.d/postgrey restart
```

Теперь в заголовках писем вы увидите поле с информацией:

```
X-Greylist: delayed 308 seconds by postgrey-1.3
4 at mail.example.com; Tue, 1 Jan 2013 09:00:00
```

Однако, при помощи этой технологии, вы избавитесь так же не более, чем от 30% спамеров. Кроме того, объединив усилия **postfix** и **postgrey**, вы не получите фильтр, работающий на 60%. В лучшем случае это будет 40%, но не все сразу – двигаемся дальше.

Стоп-спам. Внешние блок-листы – за и против (dnsbl)

Внешние блок-листы использовать очень просто, но здесь необходимо задуматься – а стоит ли? DNS BlackList – это внешний ресурс, «черный список» хостов, которые, как посчитал некто, рассылают спам. Вроде бы и нет ничего проще, чем прописать у себя в конфигурации десяток dnsbl и «не принимать почту от спамеров», ведь «там кто-то знает лучше», но кто этот кто-то? Практика показывает, что со временем пользователи начинают жаловаться, что к ним не доходит чья-то почта и оказывается, что хост ваших партнеров попал в один из черных списков, с которым сверяется ваш **postfix**. Почему он туда попал – уже не важно, ведь вы теряете полезную информацию. Поэтому мы против бездумного использования DNSBL и считаем это технологией для ленивых, но для полноты картины опишем. Добавим необходимые

директивы почти в самом конце все той же секции **smtpd_recipient_restrictions**:

```
# vi /usr/local/etc/postfix/main.cf
     # ограничения на передачу сообщений
   smtpd_recipient_restrictions =
        ...
       reject_unverified_recipient,
         # использование внешних списков DNSBL
       reject_rbl_client bl.spamcop.net,
       reject_rbl_client dnsbl.sorbs.net,
       reject_rbl_client zen.spamhaus.org,
         # передача сообщений фильтру POSTGREY
       check_policy_service inet:127.0.0.1:10023

# /usr/local/etc/rc.d/postfix restart
```

Теперь пропадет до 70% спама. Использовать или нет – дело ваше и покажет практика, однако теперь за работой вашего сервера стоит следить гораздо более внимательно. При первых подозрениях убирайте все **dnsbl**, тем более, что есть более надежное средство.

Стоп-спам. Локальный спам-фильтр (dspam)

Испытав на себе несколько методов блокирования спама, мы поняли, что ни один из них в отдельности, ни даже все три в совокупности не дают нам того эффекта, который мы хотели бы иметь. А хотим мы убрать хотя бы 99% нежелательной почты. А лучше 99,9%. Эта задача по силам локальному обучаемому фильтру, в качестве которого мы установим и настроим **dspam**:

```
# cd /usr/ports/mail/dspam
# make install clean
```

Выбираем следующие опции:

```
[*] SYSLOG       Logs via syslog
[*] DEBUG        Enable debugging logging
[*] DAEMON       Daemonize dspam; speaks LMTP
[*] HASH         Use hash driver
[*] POSTFIX_MBC Dspam as mailbox_command Postfix
```

После установки займемся файлом конфигурации:

```
# vi /usr/local/etc/dspam.conf
    # формат хранения данных (индексированные списки)
  StorageDriver /usr/local/lib/dspam/libhash_drv.so
    # параметры возврата сообщений в MTA
  DeliveryHost    127.0.0.1
  DeliveryPort    24
  DeliveryIdent   localhost
  DeliveryProto   SMTP
    # доверенные пользователи (добавить)
  Trust nobody
  Trust dovecot
    # включение режима самообучения
  Preference "trainingMode=TEFT"
    # только пометчать спам-сообщения
  Preference "spamAction=tag"
    # добавлять в тему спам-сообщений текст
  Preference "spamSubject=[SPAM]"
    # добавлять сигнатуры в заголовки писем
  Preference "signatureLocation=headers"
    # методы парсинга заголовков писем
  TrainPristine off
  ParseToHeaders off
  ChangeModeOnParse off
  ChangeUserOnParse off
    # параметры запуска и работы демона
  ServerPID               /var/run/dspam.pid
  ServerMode              auto
  ServerParameters       "--deliver=innocent,spam -d %u"
  ServerIdent             "mail.example.com"
  ServerDomainSocketPath  "/var/run/dspam.sock"
```

```
# vi /var/db/dspam/group
     # глобальная группа для всех пользователей
   globalgroup:shared:*

# echo dspam_enable=\"YES\" >> /etc/rc.conf
# /usr/local/etc/rc.d/dspam start
```

Теперь настроим связку с **postfix**. Помните, что **master.cf** очень капризный файл и даже лишний пробел может все испортить:

```
# vi /usr/local/etc/postfix/master.cf
   smtp  inet   n   -   n   -   -   smtpd
     -o content_filter=lmtp:unix:/var/run/dspam.sock
    ...
   localhost:24   inet   n   -   n   -   -   smtpd
     -o content_filter=
     -o receive_override_options=no_unknown_recipient_ch
ecks,no_header_body_checks
     -o smtpd_helo_restrictions=
     -o smtpd_client_restrictions=
     -o smtpd_sender_restrictions=
     -o smtpd_recipient_restrictions=permit_mynetworks,reject
     -o mynetworks=127.0.0.0/8
     -o smtpd_authorized_xforward_hosts=127.0.0.0/8

# /usr/local/etc/rc.d/postfix restart
```

ПРИМЕЧАНИЕ. Мы настроили **dspam** таким образом, чтобы он не удалял письма, которые считает спамом, а только определял их и помечал в заголовках и в теме письма. Дело в том, что и **dspam** может ошибаться, поэтому нужно перестраховаться от потерь важной информации. Пользователю будут доставляться все письма, а при возникновении ошибок каждый сможет переобучить фильтр. На наш взгляд такой режим работы является наиболее приемлемым и эффективным, т.к. часть ответственности за работу спам-фильтра возложена и на пользователей.

Теперь все входящие письма будут проходить через ваш **dspam**, о чем будет говорить целая серия полей в заголовках:

```
X-DSPAM-Result:      Innocent
X-DSPAM-Processed:   Tue Jan 1 09:00:00 2013
X-DSPAM-Confidence:  0.9899
X-DSPAM-Probability: 0.0000
X-DSPAM-Signature:   50eff5ca583321172312311
```

Поле X-DSPAM-Result содержит вердикт фильтра: Innocent – полезное письмо, либо Spam – соответственно, спам. Так же, при определении спама должен добавиться текст [SPAM] в тему письма. Но сразу чуда не произойдет, т.к. фильтр нужно первоначально обучить. Для этого создайте следующие алиасы:

```
# vi /etc/mail/aliases
   spam:    "|/usr/local/bin/dspam --user root --class=s
pam --source=error"
   notspam: "|/usr/local/bin/dspam --user root --class=i
nnocent --source=error"

# newaliases
```

Теперь, если ваш фильтр не определит какой-либо спам, то вы можете указать ему на его ошибку перенаправив это письмо по адресу **spam@example.com**. Аналогичным образом запускается процесс переобучения, если фильтр определит полезное письмо, как спам, только в этом случае его нужно перенаправить по адресу **notspam@example.com**. Фильтр начнет достаточно точно определять и помечать спам примерно после 50-100 обучающих писем.

Так как мы определили всех пользователей в одну глобальную группу, то и результаты обучения будут для всех общие. Посмотреть статистику работы фильтра можно так:

```
# dspam_stats -H globalgroup
```

Наибольшего эффекта вы достигнете, если не поленитесь и перенаправите на себя абсолютно всю почту вашего домена, даже ту, которая отправляется для несуществующих абонентов вашего сервера, то есть явный поток спама:

```
# vi /usr/local/etc/postfix/main.cf
   local_recipient_maps =
   luser_relay = test
```

Теперь на ящик **test** пойдет поток писем всего вашего домена, даже для несуществующих адресатов. Обучив фильтр в таких боевых условиях, вы будете приятно удивлены результатом – безошибочная фильтрация 99,8% спама. Это уже то, что нужно, но для удобства работы пользователей и создания поистине мощной почтовой системы осталось еще несколько штрихов.

Оптимизация IMAP сервера (antispam, pigeonhole)

Так как наши пользователи работают по протоколу IMAP4 и все равно все свои письма хранят на сервере, то и отделять полезные письма от спама мы тоже будем на сервере. У каждого пользователя будет отдельная папка **SPAM**, куда весь спам будет попадать автоматически. Если он не получит письмо, которое ожидает, то сможет поискать его в ней – вдруг все-таки фильтр ошибся. Кроме того, мы настроим переобучение фильтра просто перемещая письма по папкам IMAP сервера, т.е. при перемещении письма из папки **SPAM** в любую другую с полезными письмами, фильтр поймет свою ошибку, и наоборот, т.е. заменим процесс отправки писем на ящики **spam** и **notspam** для обучения фильтра простым перемещением писем из одной папки в другую. Для этого нам нужно усовершенствовать **dovecot**:

```
# cd /usr/ports/mail/dovecot2-pigeonhole/
# make install clean
# cd /usr/ports/mail/dovecot2-antispam-plugin/
# make install clean

# vi /usr/local/etc/dovecot/conf.d/15-lda.conf
   protocol lda {
        # плагин сортировки писем по папкам
     mail_plugins = $mail_plugins sieve
   }

# vi /usr/local/etc/dovecot/conf.d/20-imap.conf
   protocol imap {
        # плагины обучения фильтра и автосоздания папок
     mail_plugins = $mail_plugins antispam autocreate
   }

# vi /usr/local/etc/dovecot/conf.d/90-plugin.conf
   plugin {
        # автосоздание папок при входе на сервер
     autocreate = SPAM
     autocreate2 = Sent
     autocreate3 = Trash
       # автоподписка на созданные папки
     autosubscribe = SPAM
     autosubscribe2 = Sent
     autosubscribe3 = Trash
       # запуск скрипта сортировки входящих писем
     sieve_default = /usr/local/etc/dovecot/spam.sieve
     sieve_global_dir = /usr/local/etc/dovecot
       # обучение фильтра при перемещении писем по папкам
     antispam_backend = dspam
     antispam_signature = X-DSPAM-Signature
     antispam_signature_missing = error
     antispam_spam = SPAM
     antispam_trash = Trash
     antispam_dspam_binary = /usr/local/bin/dspam
     antispam_dspam_args = --source=error;--signature=%%s
   }
```

```
# vi /usr/local/etc/dovecot/spam.sieve
   require ["fileinto","imap4flags"];
       # проверка условия результата работы фильтра
   if header :contains "X-DSPAM-Result" "Spam"
   {
       # если спам, то пометить, как прочитанное
           setflag "\\seen";
       # и поместить в папку SPAM
           fileinto "SPAM";
           stop;
   }

# sievec /usr/local/etc/dovecot/spam.sieve
# /usr/local/etc/rc.d/dovecot restart
```

Для того, чтобы все заработало, осталась самая малость – определить **dovecot**, как локальный доставщик, чтобы он сам мог сортировать письма по папкам:

```
# vi /usr/local/etc/postfix/main.cf
   mailbox_command = /usr/local/libexec/dovecot/dovecot-
lda -f "$SENDER" -a "$RECIPIENT"

# /usr/local/etc/rc.d/postfix restart
```

Теперь мы с вами можем быть довольны достигнутым результатом. Хотя, было бы неплохо еще настроить автоматическую очистку папки **SPAM**, например, раз в неделю, но мы специально не будем этого здесь описывать, чтобы вы имели возможность сделать такую задачу самостоятельно.

Этой почтовой системе не хватает только веб-интерфейса…

Глава 6
Веб-сервер для визуализации сервисов

Бесспорно, командная строка является очень мощным инструментом администрирования вашего сервера, но, согласитесь, без визуализации иногда обойтись трудно, особенно, когда нужно дать доступ к какой-либо информации пользователям. Для решения этой задачи лучше всего подходят конечно же веб-технологии. В этой главе мы займемся веб-сервером Apache и настроим несколько полезных веб-интерфейсов, как для администрирования, так и для удобства ваших пользователей – пожалуй, это единственное, чего нам не хватает для того, чтобы гордиться своей работой.

Установка и настройка веб-сервера (apache)

Для того, чтобы пользоваться веб-интерфейсами в принципе и, возможно, поселить у себя несколько сайтов, нам понадобится основа – веб-сервер. Мы установим Apache вместе с языком программирования PHP и системой управления базами данных MySQL – эта связка используется веб-разработчиками довольно часто:

```
# cd /usr/ports/www/apache22/
# make install clean
# cd /usr/ports/lang/php5/
# make install clean
```

Обязательно к опциям, выбранным по умолчанию, добавляем модуль для связи с веб-сервером:

```
[*] APACHE   Build Apache module
```

```
# cd /usr/ports/lang/php5-extensions/
# make install clean
```

Расширения языка можно установить вообще все, но как минимум, к тем, что уже отмечены, нужно добавить эти два:

```
[*] IMAP     IMAP support
[*] MYSQL    MySQL database support
```

```
# cd /usr/ports/databases/mysql55-server/
# make install clean

# vi /usr/local/etc/apache22/httpd.conf
    # подключение модуля PHP
   LoadModule php5_module libexec/apache22/libphp5.so
   AddType application/x-httpd-php .php
   AddType application/x-httpd-php-source .phps
    # определение имени стартового сценария
   <IfModule dir_module>
       DirectoryIndex index.html index.php
   </IfModule>
```

```
# vi /usr/local/etc/php.ini
   date.timezone = Europe/Kiev

# vi /etc/rc.conf
   mysql_enable="YES"
   apache22_enable="YES"

# /usr/local/etc/rc.d/mysql-server start
# /usr/local/etc/rc.d/apache22 start
```

Файл **php.ini** нужно создать – эта строка внутри него нужна для корректного отображения дат и времени. И это, как ни странно, все. Теперь ваш сервер отвечает на HTTP-запросы по 80 порту, к тому же он понимает язык PHP и умеет работать с базами данных MySQL. Зайдите по адресу *http://example.com* и убедитесь в том, что:

It Works!

Если не получится, попробуйте зайти по внешнему или внутреннему IP-адресу – возможно дело в DNS или в фаерволле. В любом случае, прежде чем продолжать, лучше будет, если вы добьетесь правильной работы по имени своего домена.

Теперь проверим, подключился ли модуль PHP. Создадим файл **index.php** с информационной директивой:

```
# vi /usr/local/www/apache22/data/index.php
   <? phpinfo(); ?>
```

Попробуйте открыть у себя в браузере адрес *http://example.com/index.php*. Если вы видите информационную страницу о PHP, значит все ОК.

Здесь необходимо отметить, что мы устанавливаем веб-сервер не для профессионального хостинга, а для собственных

админских нужд, поэтому глубоко копать его настройки и конфигурацию не будем. Однако, некоторые изменения внесем:

```
# vi /usr/local/etc/apache22/httpd.conf
    # поднимем корневой каталог повыше
  DocumentRoot "/usr/local/www"
    # параметры корневого каталога
  <Directory "/usr/local/www">
      Options Indexes FollowSymLinks
      AllowOverride None
      Order allow,deny
      Allow from all
  </Directory>
    # добавим доступ к каталогу документации
  <Directory "/usr/local/share/doc">
      Options Indexes FollowSymLinks
      AllowOverride None
      Order allow,deny
      Allow from 10.0.0.0/24
  </Directory>
    # перенаправление в каталог документации
  Alias /doc "/usr/local/share/doc"

# apachectl restart
```

Мы изменили корневой каталог и дали возможность доступа к каталогу документации, но ограничили его нашей локальной сетью, а так же создали алиас для перехода в этот каталог. Попробуйте:

http://example.com

http://example.com/doc

Управление почтовой системой (postfixadmin)

Почтовой системой **postfix** на уровне создания почтовых ящиков и псевдонимов можно управлять из специального веб-интерфейса. Установим и подготовим к работе **postfixadmin**:

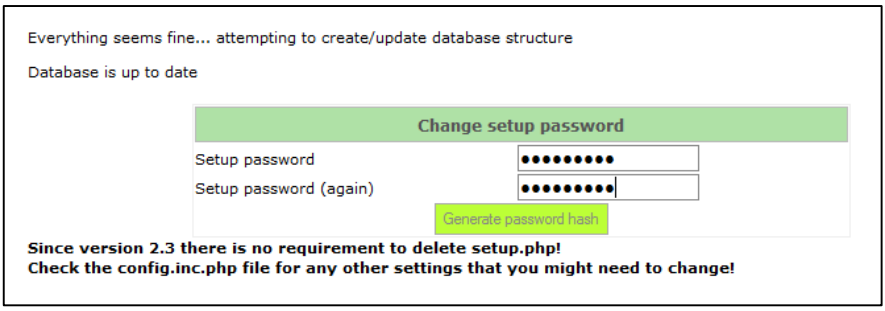
```
# cd /usr/ports/mail/postfixadmin/
# make install clean

# cd /usr/local/www/postfixadmin/
# vi config.inc.php
    # флаг готовности конфигурации
  $CONF['configured'] = true;
    # язык интерфейса по умолчанию
  $CONF['default_language'] = 'ru';
    # параметры соединения с базой данных: тип БД, хост,
    # логин и пароль пользователя, имя базы данных
  $CONF['database_type'] = 'mysql';
  $CONF['database_host'] = 'localhost';
  $CONF['database_user'] = 'postfix';
  $CONF['database_password'] = 'pass';
  $CONF['database_name'] = 'postfix';
    # тип шифрования паролей в базе данных
  $CONF['encrypt'] = 'md5crypt';
    # структура хранения почтовых ящиков:
    # /usr/mail/домен/пользователь
  $CONF['domain_path'] = 'YES';
  $CONF['domain_in_mailbox'] = 'NO';
```

Создадим в **mysql** базу данных **postfix**, в которой будет храниться информация о доменах, пользователях и псевдонимах нашей почтовой системы:

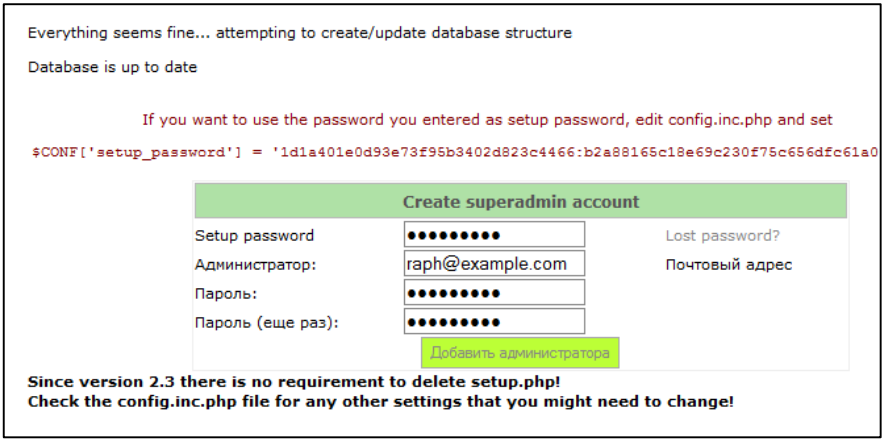
```
# mysql
mysql> create database postfix;
mysql> grant all on postfix.* to postfix@localhost ident
ified by 'pass';
mysql> quit
```

Второй командой мы разрешили работать со всеми таблицами созданной базы данных пользователю **postfix** с паролем **pass**. Все необходимые для работы таблицы **postfixadmin** создаст сам при первом запуске.

Теперь нужно продолжить настройку в вашем веб-браузере: ***http://example.com/postfixadmin/setup.php*** – вы должны увидеть приглашение к созданию setup-пароля (рис. 15).



Этот пароль нужен для создания администраторов системы. Введите его и нажмите **[Generate password hash]**, после чего вы увидите хеш-строку вашего пароля и приглашение к созданию администратора (рис. 16).



Этот хеш нужно скопировать и записать в конфигурационный файл **postfixadmin**, который мы только что редактировали, и только после этого продолжать в браузере:

```
# vi config.inc.php
    # введите сгенерированный хеш setup-пароля
  $CONF['setup_password'] = '1d1a401e0d93e73f95b340...';
```

Теперь в браузере введите ваш setup-пароль и данные создаваемого администратора. После этого можете заходить в систему по адресу: ***http://example.com/postfixadmin***.

Создайте здесь все свои почтовые домены и почтовые ящики в них. Если вы еще не настраивали **postfix** и **dovecot** на работу с **mysql**, сделайте это сейчас. Посмотреть структуру базы данных вы сможете при помощи внутренних команд **mysql**. Подключитесь к СУБД командой **mysql** и попробуйте следующие команды (не забывайте в конце всегда ставить точку с запятой):

show databases; – вывести список существующих баз данных;

use postfix; – выбрать для работы базу данных postfix;

show tables; – вывести список таблиц текущей базы данных;

select * from domain; – вывести все поля таблицы domain (т.е. все наши почтовые домены);

select username from mailbox; – вывести содержимое поля username из таблицы mailbox (т.е. все наши почтовые адреса);

select password from mailbox where username =

'test@example.com'; – вывести содержимое поля password из таблицы mailbox, где username = test@example.com (т.е. хеш пароля от почтового ящика test@example.com);

quit – отключиться и выйти.

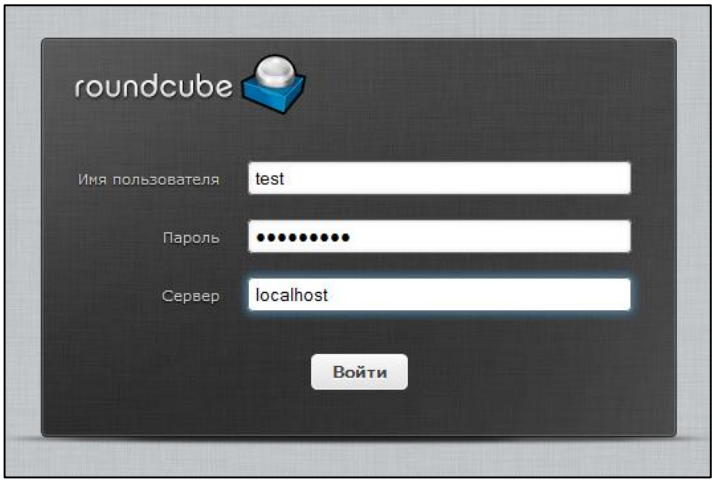
Пользовательский доступ к почте (roundcube)

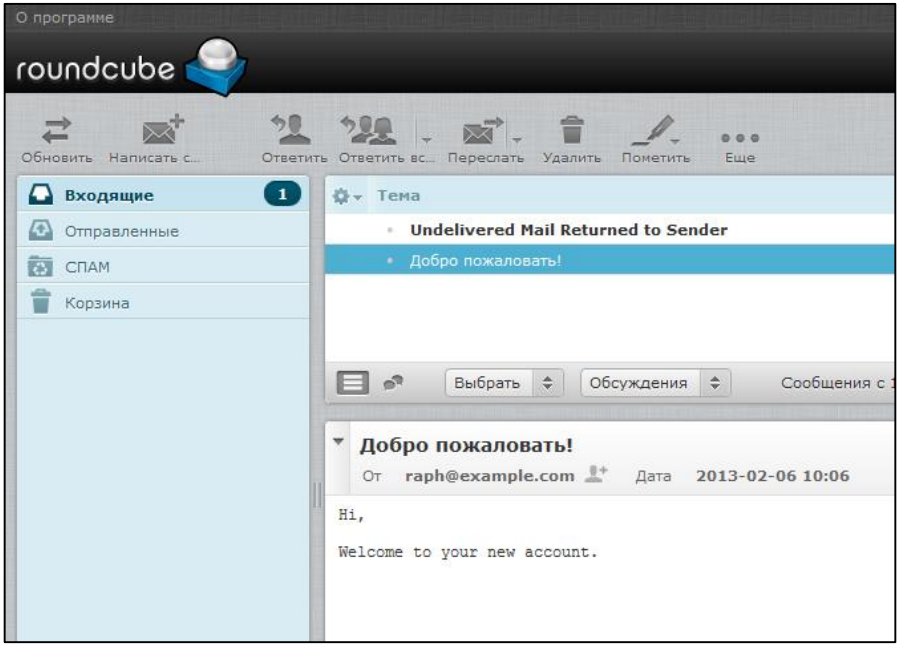
Самым долгожданным и интересным событием, которое сможете оценить не только вы сами, несомненно является установка собственного веб-интерфейса для работы пользователей с электронной почтой. Наиболее интересным из множества существующих, пожалуй, является **roundcube**. Установим его:

```
# cd /usr/ports/mail/roundcube/
# make install clean

# mysql
mysql> create database roundcubemail;
mysql> grant all on roundcubemail.* to roundcube@localho
st identified by 'pass';
mysql> quit

# cd /usr/local/www/roundcube/
# mysql roundcubemail < SQL/mysql.initial.sql
# cp config/main.inc.php.dist config/main.inc.php
# cp config/db.inc.php.dist config/db.inc.php
# vi config/db.inc.php
     # параметры соединения с базой данных
   $rcmail_config['db_dsnw'] = 'mysql://roundcube:pass@l
ocalhost/roundcubemail';
```

После установки мы создали в **mysql** служебную базу данных **roundcubemail**, и разрешили работать со всеми таблицами этой базы данных пользователю **roundcube** с паролем **pass**.

Теперь можно посмотреть, что же у нас с вами получилось: ***http://example.com/roundcube*** – вы должны увидеть предложение ввести логин, пароль и имя почтового сервера (рис. 17). Вводите имя пользователя, его пароль и сервер **localhost**. Изучайте интерфейс и его возможности (рис. 18).


roundcube
Имя пользователя
test
Пароль
Сервер
localhost
Войти

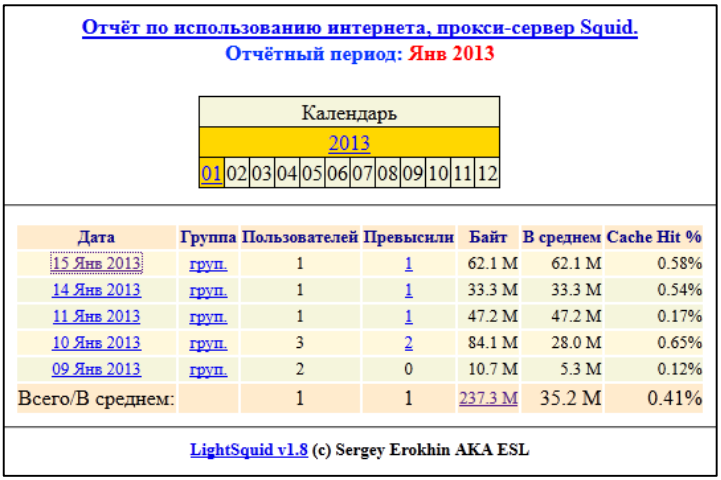

О программе
roundcube
Обновить
Написать с...
Ответить
Ответить вс...
Переслать
Удалить
Пометить
Еще
Входящие
1
Отправленные
СПАМ
Корзина
Тема
Undelivered Mail Returned to Sender
Добро пожаловать!
Выбрать
Обсуждения
Сообщения с
Добро пожаловать!
От raph@example.com
Дата 2013-02-06 10:06
Hi,
Welcome to your new account.

Анализатор лог-файлов SQUID (lightsquid)

Рано или поздно у вас возникнет вопрос о том, кто, когда, куда и сколько ходит в Интернет, и эту информацию нужно будет показать, как минимум самим пользователям для понимания. В этом случае вам поможет анализатор лог-файлов вашего прокси-сервера:

```
# cd /usr/ports/www/lightsquid/
# make install clean

# vi /usr/local/etc/lightsquid/lightsquid.cfg
     # путь к каталогу с логами SQUID
   $logpath      ="/var/squid/logs";
     # используемый формат логов SQUID
   $squidlogtype = 0;
     # язык интерфейса по умолчанию
   $lang         ="ru";

# vi /usr/local/etc/apache22/httpd.conf
     # разрешаем CGI в рабочем каталоге LIGHTSQUID
   <Directory "/usr/local/www/lightsquid">
       AddHandler cgi-script .cgi
       AllowOverride All
   </Directory>

# apachectl restart
# /usr/local/www/lightsquid/check-setup.pl
all check passed, now try access to cgi part in browser

# /usr/local/www/lightsquid/lightparser.pl
```

В зависимости от объема лог-файлов, работа этого скрипта займет определенное время. После окончания обработки зайдите на: ***http://example.com/lightsquid*** и изучите возможности того мощного инструмента, который оказался у вас в руках (рис. 19).



Отчёт по использованию интернета, прокси-сервер Squid.
Отчётный период: Янв 2013

Календарь											
2013											
01	02	03	04	05	06	07	08	09	10	11	12

Дата	Группа	Пользователей	Превысили	Байт	В среднем	Cache Hit %
15 Янв 2013	груп.	1	1	62.1 M	62.1 M	0.58%
14 Янв 2013	груп.	1	1	33.3 M	33.3 M	0.54%
11 Янв 2013	груп.	1	1	47.2 M	47.2 M	0.17%
10 Янв 2013	груп.	3	2	84.1 M	28.0 M	0.65%
09 Янв 2013	груп.	2	0	10.7 M	5.3 M	0.12%
Всего/В среднем:		1	1	237.3 M	35.2 M	0.41%

LightSquid v1.8 (c) Sergey Erokhin AKA ESL

Для автоматического формирования отчетов добавьте команду в планировщик и перезапустите **cron**:

```
# vi /etc/crontab
   0   2   *   *   *   root   /usr/local/www/lightsquid/
lightparser.pl yesterday

# killall -HUP cron
```

В каталоге **/usr/local/etc/lightsquid** вместе с конфигурационным файлом вы найдете еще три, которые помогут усовершенствовать отображение информации:

group.cfg – для распределения пользователей по группам (например по отделам, кабинетам, департаментам и т.д.);

realname.cfg – для назначения имен IP-адресам (например имя компьютера, имя пользователя и т.д.);

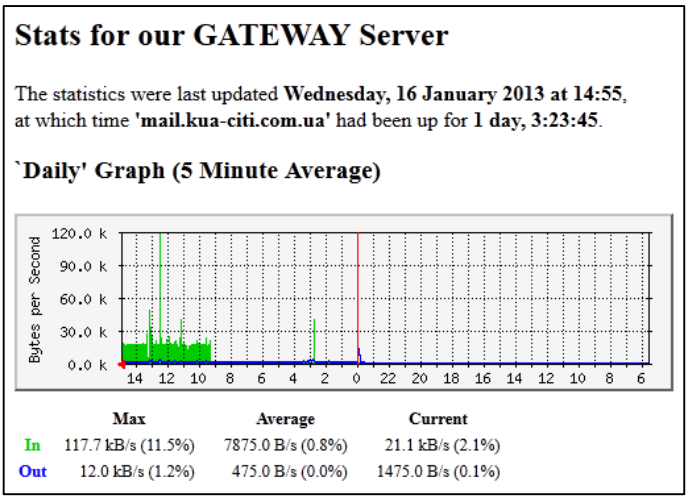
skipuser.cfg – для исключения IP-адресов или имен из статистики.

Графики использования сетевого трафика (mrtg)

Очень полезная программа **mrtg** построит вам любые графики любых параметров, которые можно измерять с течением времени. Мы же будем использовать ее в связке с **snmp** для построения графиков загрузки нашего Интернет-канала:

```
# cd /usr/ports/net-mgmt/net-snmp/
# make install clean
# cd /usr/ports/net-mgmt/mrtg/
# make install clean

# vi /usr/local/share/snmp/snmpd.conf
   rwuser root noauth
   rouser root noauth
   rwcommunity public 22.22.22.22
   rocommunity public 22.22.22.22

# vi /usr/local/etc/mrtg/mrtg.cfg
     # рабочий каталог
   WorkDir: /usr/local/www/mrtg
     # получение информации с SNMP
   Target[gateway]: 1:public@22.22.22.22
     # максимум по вертикальной оси
   MaxBytes[gateway]: 1024000
     # заголовки HTML документа
   Title[gateway]: Traffic Analysis for Gateway
   PageTop[gateway]: <H1>Stats for our GATEWAY Server</H1>

# mkdir /usr/local/www/mrtg
# echo snmpd_enable=\"YES\" >> /etc/rc.conf
# /usr/local/etc/rc.d/snmpd start
# /usr/local/bin/mrtg /usr/local/etc/mrtg/mrtg.cfg

# vi /etc/crontab
   */5   *   *   *   *   root   /usr/local/bin/mrtg /usr
/local/etc/mrtg/mrtg.cfg

# killall -HUP cron
```

В последнем блоке команд мы настроили автоматическое обновление графиков **mrtg** каждые 5 минут. Пройдет некоторое время и вы увидите, как загружается ваш Интернет-канал: *http://example.com/mrtg/gateway.html* (рис. 20).



Может оказаться так, что SNMP перепутает информацию о входящем и исходящем трафике. Тогда измените значение параметра **Target** на **-1**:

```
# vi /usr/local/etc/mrtg/mrtg.cfg
    ...
   Target[gateway]: -1:public@22.22.22.22
```

Виртуальные хосты веб-сервера (httpd.conf)

Последнее, что мы сделаем – это создадим виртуальные хосты в **apache** для красоты и на перспективу, ведь, возможно, у вас когда-нибудь поселится несколько сайтов. Итак:

```
# vi /usr/local/etc/apache22/httpd.conf
   Include etc/apache22/extra/httpd-vhosts.conf

# vi /usr/local/etc/apache22/extra/httpd-vhosts.conf
   NameVirtualHost *:80
   <VirtualHost *:80>
     ServerName    default
   </VirtualHost>

   <VirtualHost *:80>
     ServerAdmin   postmaster@example.com
     DocumentRoot  /usr/local/www/roundcube
     ServerName    mail.example.com
     ErrorLog      /var/log/mail.example.com-error.log
   </VirtualHost>

# apachectl restart
```

Теперь при обращении к нашему веб-серверу по IP-адресам *http://10.0.0.1* и *http://22.22.22.22*, либо по полному имени хоста *http://gateway.example.com* мы будем попадать туда, куда и раньше – в корневой каталог **/usr/local/www** (для этого мы создали первую секцию – ловушку для адресов, которые не определены у нас, как виртуальные хосты) а при обращении по имени виртуального хоста *http://mail.example.com* – мы попадем в его каталог, то есть **/usr/local/www/roundcube**.

В дальнейшем, при необходимости создавать новые виртуальные хосты, просто добавляйте аналогичные секции **VirtualHost** и не забывайте создавать их в вашем домене.

Глава 7
Специфические инструменты

В этой главе мы рассмотрим некоторые довольно популярные узкоспециальные задачи-фишки и их возможные реализации. В основном, конечно же, речь пойдет о сетевых сервисах, объединении сетей, перенаправлении соединений и т.д.

Перенаправление портов внутрь сети (natd, socket)

Так же нередко бывает, что нужно извне, например из дома, подключиться к какому-либо внутреннему ресурсу. Например, у вас есть терминальный сервер на Windows, который не подключен напрямую к сети Интернет (что разумно), но к нему необходимо подключаться извне. Для этих целей используем **socket**:

```
# cd /usr/ports/sysutils/socket
# make install clean

# vi /etc/services
   rdp  3389/tcp

# vi /etc/inetd.conf
   rdp stream tcp nowait root /usr/local/bin/socket -v 10.0.
0.20 3389
```

Теперь, после перезапуска **inetd**, можно подключаться терминальным клиентом (mstsc) к нашему серверу на его внешний IP-адрес 22.22.22.22, порт 3389 – сервер перенаправит входящее соединение внутрь нашей локальной сети на хост 10.0.0.20, порт 3389, если конечно вы не забудете разрешить такой трафик в фаерволле.

Если же ожидается существенная нагрузка, то для этих же целей будет лучше использовать **natd**. Аналогичная задача решается следующим образом:

```
# vi /etc/rc.conf
   natd_enable="YES"
   natd_interface="de0"
   natd_flags="-f /etc/natd.conf"

# vi /etc/natd.conf
   redirect_port tcp 10.0.0.20:3389 3389
```

VPN сервер для удаленных пользователей (mpd)

Если вам необходимо обеспечить доступ в вашу локальную сеть для удаленных пользователей Windows так, как будто бы они находятся в ней, нужно установить и настроить VPN сервер **mpd5**:

```
# cd /usr/ports/net/mpd5/
# make install clean

# cd /usr/local/etc/mpd5/
# cp mpd.conf.sample mpd.conf
# vi mpd.conf
   startup:
        # установить пользователя логин пароль права
     set user admin pass admin
   default:
        # загружаем с параметрами секции pptp_server
     load pptp_server
   pptp_server:
        # диапазон IP-адресов для выдачи клиентам
     set ippool add pool1 10.0.0.210 10.0.0.220
        # диапазон IP-адресов для показа клиентам
     set ipcp ranges 10.0.0.8/32 ippool pool1
        # серверы DNS для клиентов
     set ipcp dns 10.0.0.1
        # сервер WINS для клиентов
     set ipcp nbns 10.0.0.1
        # принимаем соединения на интерфейсе de0
     set pptp self de0

# vi mpd.secret
   ruser1  "pass1"  10.0.0.201
   ruser2  "pass2"  *

# echo mpd_enable=\"YES\" >> /etc/rc.conf
# /usr/local/etc/rc.d/mpd5 start
```

Теперь удаленные пользователи Windows могут создавать у себя «Новую сеть» - «Подключение к рабочему месту», вводить

там ваш внешний IP-адрес 22.22.22.22, логины и пароли из файла **mpd.secret** и входить в вашу локальную сеть. Причем **ruser1** получит фиксированный IP-адрес 10.0.0.201, а **ruser2** – динамический адрес из диапазона 10.0.0.210 – 10.0.0.220, ну а доступ они будут иметь к серверу с адресом 10.0.0.8.

Объединение удаленных локальных сетей (ipsec)

Часто бывает так, что организация имеет территориально распределенную структуру, т.е. центральный офис, филиалы, склады, магазины и т.д. Конечно же в таком случае было бы логично объединить все офисы в одно информационное поле. Подключив каждый из них к сети Интернет можно достаточно быстро и просто организовать VPN (виртуальную частную сеть) средствами **ipsec**.

Пусть второй офис имеет локальную сеть 192.168.0.0/24 и подключение к Интернет с адресом 66.66.66.66. Для организации VPN нам необходим будет FreeBSD сервер, аналогичный нашему, однако, на обоих серверах придется пересобрать ядро с поддержкой **ipsec**:

```
# vi /usr/src/sys/amd64/conf/GATEWAY
    ...
   options  IPSEC
   device   crypto
    ...
   device   gif
```

Поддержка устройства **gif** в ядре уже есть изначально, нужно просто проверить наличие этой строки. После сборки и перезагрузки с новым ядром можно конфигурировать туннель. Выполняем в командной строке на нашем сервере:

```
# ifconfig gif0 create
# ifconfig gif0 10.0.0.1 192.168.0.1 netmask 255.255.255.0
# ifconfig gif0 tunnel 22.22.22.22 66.66.66.66
# route add 192.168.0.0/24 192.168.0.1
# ipfw add 3000 allow ip from any to any via gif0
```

На удаленном сервере выполняем все то же самое, только логически меняем IP-адреса местами. После этого проверяем, как выглядят наши новые интерфейсы **gif0** и пингуем какой-нибудь компьютер из удаленной сети:

```
# ifconfig gif0
gif0: flags=8011<UP,POINTOPOINT,MULTICAST> mtu 1280
      tunnel inet 22.22.22.22 --> 66.66.66.66
      inet 10.0.0.1 --> 192.168.0.1 netmask 0xffffff00
# ping 192.168.0.10
```

Естественно, не забываем добавить все это в системные конфигурационные файлы на обоих серверах:

```
# vi /etc/rc.conf
   gif_interfaces="gif0"
   gifconfig_gif0="22.22.22.22 66.66.66.66"
   ifconfig_gif0="10.0.0.1 192.168.0.1 netmask 255.255.255.0"

# vi /etc/rc.local
   route add 192.168.0.0/24 192.168.0.1

# vi /usr/local/etc/firewall.conf
    ...
   add 3000 allow ip from any to any via gif0
```

Если нужно соединить не две удаленных сети, а больше, проделывайте то же самое и стройте сеть в режиме «звезды», выбрав какой-либо узел в роли центрального (обычно это центральный офис, на сервере которого создаются новые интерфейсы **gif1**, **gif2** и т.д.).

Копирование входящей и исходящей почты (synonym)

Иногда возникает необходимость копировать всю входящую и исходящую почту пользователей в теневом режиме. Оставим вопросы этики за пределами этой книги, ведь цели и принципы работы организаций могут отличаться. Для решения этой задачи на **sendmail** нужно установить и запустить почтовый фильтр **synonym**:

```
# cd /usr/ports/mail/synonym/
# make install clean
# echo synonym_enable=\"YES\" >> /etc/rc.conf
```

Создадим правило, которое будет копировать всю исходящую почту нашего домена на адрес **admin@example.com**:

```
# vi /usr/local/etc/synonym.conf
   <Rules>
     <Rule>
       <Condition>
         <Header>From</Header>
         <Match>.*</Match>
       </Condition>
       <Action>
         <ActionType>Copy</ActionType>
         <Address>admin@example.com</Address>
       </Action>
     </Rule>
   </Rules>
```

Возможности для создания правил очень разнообразны, в документации есть примеры. Теперь, если вы используете **sendmail**, пропишите этот фильтр в его конфигурации:

```
# vi /etc/mail/sendmail.cf
   # Input mail filters
   O InputMailFilters=synonym
   Xsynonym, S=local:/var/run/synonym/synonym.sock, T=C:
10m;S:1s;R:1s;E:5m
```

Если же вы используете **postfix**, тогда выполните привязку следующим образом:

```
# vi /usr/local/etc/postfix/main.cf
   smtpd_milters = unix:/var/run/synonym/synonym.sock
   milter_default_action = accept
```

А вообще, такая задача может быть легко решена средствами самого **postfix**:

```
# vi /usr/local/etc/postfix/main.cf
   recipient_bcc_maps = hash:/usr/local/etc/postfix/bcc_recipient
   sender_bcc_maps = hash:/usr/local/etc/postfix/bcc_sender
```

Не забудьте создать эти файлы и выполнить над ними команду **postmap**. Правда, **postfix** не предоставит вам таких возможностей писать гибкие правила фильтрации, как **synonym**, но, если нужно только теневое копирование всей почты, этого будет достаточно.

Копирование файлов между серверами (ssh, scp)

Протокол SSH позволяет не только удаленно работать на сервере, а и передавать файлы. Для этого используются команды:

ssh [польз]@[сервер] – войти в командную консоль на удаленном сервере;

scp [польз]@[сервер]:[файл 1] [файл 2] – скопировать с сервера файл 1 и сохранить его локально, как файл 2.

scp [файл 1] [польз]@[сервер]:[файл 2] – скопировать локальный файл 1 и сохранить его на сервере, как файл 2.

После ввода пароля операции будут выполнены. Однако, рано или поздно вы захотите копировать файлы с одного сервера на другой не вручную, а в автоматическом режиме без ввода пароля. Для этого нужно сформировать SSH ключ на сервере-источнике файлов (например 66.66.66.66) и передать его на сервер-получатель (наш 22.22.22.22). Выполним три следующих команды на том сервере, с которого хотим передавать информацию в автоматическом режиме (т.е. 66.66.66.66). Сгенерируем SSH ключ:

```
# ssh-keygen -t dsa
```

Подключимся к получателю и выполним в его консоли команду **mkdir** – создадим папку **.ssh** в домашнем каталоге пользователя, под которым хотим выполнять операции без ввода пароля:

```
# ssh raph@22.22.22.22 'mkdir /home/raph/.ssh'
```

Передадим содержимое сгенерированного SSH ключа, через подключение к получателю и запишем его в файл **authorized_keys** в папку **.ssh**, которую создали предыдущей командой:

```
# cat /root/.ssh/id_dsa.pub | ssh raph@22.22.22.22 'cat
>> /home/raph/.ssh/authorized_keys'
```

Все просто, главное не запутаться. Теперь можно использовать команды **ssh** и **scp** на сервере-источнике по отношению к получателю без ввода пароля пользователя. Например:

```
# scp /etc/rc.conf raph@22.22.22.22:/home/raph/rc.conf
```

Если все равно вы видите запрос пароля, проверьте конфигурационный файл демона **sshd** на сервере-получателе:

```
# vi /etc/ssh/sshd_config
   PubkeyAuthentication  yes
   AuthorizedKeysFile    .ssh/authorized_keys
```

Держим у себя описание доменной зоны (named)

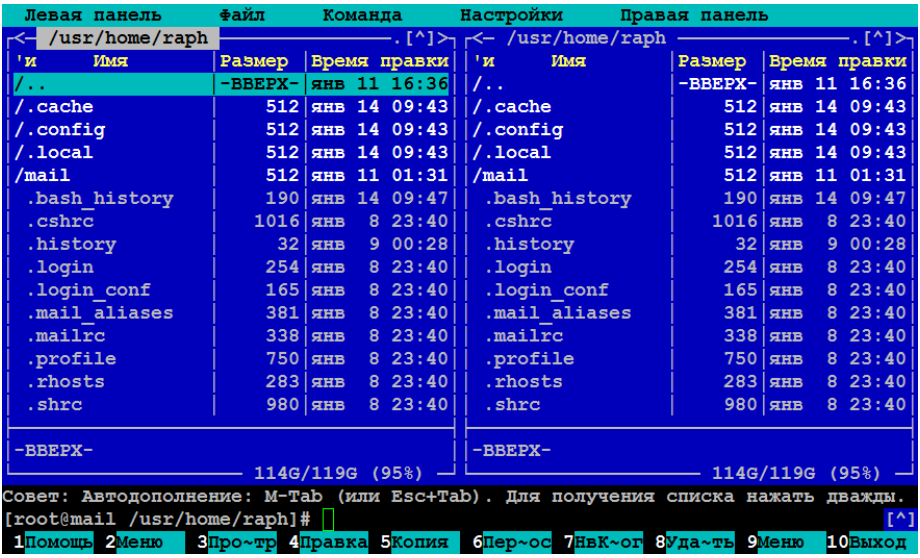
Если ваш Интернет-домен зарегистрирован и поддерживается регистратором, тогда все просто. Но бывает и так (все реже и реже, но все же), что вы вынуждены обслуживать свой домен самостоятельно. Почему бы не сделать это прямо у себя, т.е. выступить первичным DNS сервером своего же доменного имени. А вторичный взять на каком-либо бесплатном сервисе, например **secondary.net.ua**. В этом случае, нужно создать у себя сам файл-описание вашего Интернет-домена:

```
# vi /etc/namedb/named.conf
   zone "example2.com" {
       type master;
       file "/etc/namedb/example2.com";
   };

# vi /etc/namedb/example2.com
   $TTL     10800
   @    IN   SOA    ns.example2.com. root.example2.com. (
                    2013010101 ; Serial
                    10800      ; Refresh
                    3600       ; Retry
                    604800     ; Expire
                    86400 )    ; Minimum
   @    IN   NS     ns.example2.com.
   @    IN   NS     ns.secondary.net.ua.
   @    IN   MX     10 gw.example2.com.
   @    IN   A      66.66.66.66
   gw   IN   A      66.66.66.66
   www  IN   CNAME  ns

# /etc/rc.d/named restart
```

Мы рассмотрели вариант, когда у вас всего один сервер FreeBSD. Конечно будет гораздо лучше, если этот ваш домен будет «жить» на каком-либо другом вашем сервере.

Файловый менеджер Midnight Commander (mc-light)

Для более удобной и наглядной работы в файловой системе можно установить файловый менеджер:



Наверняка все узнали на скриншоте (рис. 21) старый добрый Norton, Volcov, Far, в общем, кто к чему привык в свое время. Для FreeBSD тоже есть свой файловый менеджер – Midnight Commander. Он много чего умеет, уж точно не меньше, чем его аналоги, правда, установка его займет около 20 минут. Но лучше поставить его облегченный вариант **mc-light** – это будет гораздо быстрее:

```
# cd /usr/ports/misc/mc-light/
# make install clean
```

Пользоваться им несложно, однако, автор не рекомендует привыкать к нему новичкам. Все-таки сначала нужно почувствовать уверенность в командной строке.

Глава 8
Программирование скриптов

О том, насколько полезным является умение программировать на языке командного интерпретатора, наверное, говорить не стоит. Потратив время на изучение работы интерпретатора, вы сможете писать очень серьезные и сложные программы. Особенно хорошо это будет получаться после изучения сотни специализированных команд FreeBSD, предназначенных для решения узких задач.

Вот шесть основных причин, по которым стоит изучать программирование на языке командного интерпретатора в FreeBSD:

1. Его легко выучить. Если вы часто работаете с командной строкой FreeBSD, вы, наверняка, уже знакомы со многими командами, используемыми в сценариях;

2. Он экономит время. Для решения некоторых задач, требующих часы или даже дни программирования на C, достаточно 5-10 минут программирования на языке интерпретатора;

3. Этот язык позволяет автоматизировать рутину. Скажем, если требуется внести одно и то же изменение в 100 файлов, достаточно воспользоваться циклом для автоматизации этого процесса;

4. Он позволяет изучить новые полезные методы работы с системой. При программировании вы столкнетесь с командами, о которых никогда раньше не слышали, и узнаете о новых способах решения известных задач. Вы лучше изучите командную строку FreeBSD и поймете, какой мощью она обладает;

5. Он развивает креативный потенциал. Вы всегда сможете создать нужную вам программу на языке командного интерпретатора. Одним из главных преимуществ FreeBSD является философия «Делай по-своему»;

6. Он обучает думать. Возможно, это идеальный язык для того, чтобы научиться программировать, поскольку он, работая со знакомыми командами, позволяет сосредоточиться на логике программы.

Описание языка

В этом разделе книги мы очень кратко пройдемся по основным операторам и оставим читателю возможность изучить искусство программирования в shell самостоятельно экспериментальным путем, показав, правда, несколько интересных практических примеров. Итак, перейдем к изучению одного из самых мощных средств FreeBSD.

По традиции обучение начинается с написания программы, которая выводит на экран «Hello, World!». Создадим в своем домашнем каталоге папку **scrpts**, внутри которой и будем упражняться:

```
# mkdir /home/raph/scrpts
# cd /home/raph/scrpts
# vi hello.sh
   #!/bin/sh
   # Программа "Hello, World!"
   echo "Hello, World!"
   exit 0

# chmod 0755 hello.sh
# ./hello.sh
Hello, World!
```

Скрипт должен начинаться с последовательности символов **#!**, которая сообщает, что за ней следует имя интерпретатора для выполнения сценария. Вторая строка в нашем примере – это комментарий. Третья строка – это оператор **echo**, который выводит саму фразу «Hello, World!» в стандартный вывод (на экран). Последняя строка завершает работу программы и возвращает код родительской программе. Код завершения, равный **0**, указывает, что программа нормально завершила работу, код, отличный от **0**, сообщает об ошибке.

Переменные.

В именах переменных учитывается регистр. Они могут содержать буквы, цифры и символ подчеркивания. Имя переменной не может начинаться с цифры. Кроме того, не следует вначале имени переменной использовать подчеркивание. Желательно применять описательные имена, чтобы код программы было легко читать. В простейшем случае значение переменной присваивается следующим образом:

```
myvar=5
```

Для доступа к информации в переменной перед ее именем следует указывать символ **$**:

```
echo ${myvar}
```

Фигурные скобки не являются обязательными, однако они улучшают читаемость кода, выделяя имена переменных. Значение одной переменной можно присвоить другой:

```
newvar=$myvar
```

Создание переменной среды похоже на создание переменной интерпретатора. Только ее необходимо экспортировать:

```
MYVAR=5 export
```

Теперь **MYVAR** является переменной среды, которая будет доступна другим программам, запущенным в этом сеансе интерпретатора.

Кроме присвоения значений переменным и их использования в сценарии, командный интерпретатор предоставляет возможность обрабатывать их ввод – команда **read**. Обычно этот ввод набирается пользователем (или читается из файла, если применяется перенаправление):

```
# vi hello2.sh
   #!/bin/sh
   echo -n "Please enter your name: "
   read name
   echo "Hello, $name!"
   exit 0
```

Команда **read** может иметь несколько аргументов. Запустим этот скрипт и по запросу введем какое-нибудь имя:

```
# ./hello2.sh
Please enter your name: Ivan
Hello, Ivan!
# _
```

Получить информацию от пользователя можно также при чтении аргументов командной строки. Командный интерпретатор автоматически сохраняет значения этих аргументов в специальных переменных **$1** - **$9**. Переменная **$0** содержит имя самой программы, переменная **$@** – все аргументы, а **$#** – количество аргументов:

```
# vi yourname.sh
   #!/bin/sh
   echo "The name of the program is: $0"
   echo "The total number of arguments: $#"
   echo "The complete argument string is: #@"
   echo "Your first name is: $1"
   echo "Your last name is: $2"
   exit 0
```

Вот пример запуска программы:

```
# ./yourname.sh Ivan Petrov
The name of the program is: ./yourname.sh
The total number of arguments: 2
The complete argument string is: Ivan Petrov
Your first name is: Ivan
Your last name is: Petrov
# _
```

Подстановка команд.

Подстановка команд позволяет запустить команду и присвоить ее вывод переменной. Выполняемую команду следует заключить в символы `. Не путайте их с одинарными кавычками. Символ ` – это символ обратной кавычки (на клавиатуре он, как правило, совмещен с символом тильды ~). Например, оператор:

```
TodayDate=`date`
```

запускает команду **date** и присваивает ее вывод переменной **TodayDate**. После этого к значению переменной можно обращаться обычным образом.

Арифметические операции в сценариях.

Простейшие арифметические операции с целыми числами можно выполнять при помощи **expr** и подстановки команд:

var3=`expr var1 + var2` – сложение;
var3=`expr var1 - var2` – вычитание;
var3=`expr var1 \* var2` – умножение;
var3=`expr var1 / var2` – целое от деления;
var3=`expr var1 % var2` – остаток от деления.

Попытка использования чисел с плавающей точкой приведет к ошибке. Кроме того, изменение порядка операций с помощью скобок не поддерживается. Символы, имеющие специальное значение для интерпретатора, следует экранировать (*****).

Команда **expr** позволяет работать и с логическими выражениями, значением которых является истина (**expr** возвращает **1**) или ложь (**expr** возвращает **0**):

expr var1 = var2 – равно;
expr var1 != var2 – не равно;
expr var1 \> var2 – больше;
expr var1 \< var2 – меньше;
expr var1 \>= var2 – больше или равно;
expr var1 \<= var2 – меньше или равно.

Не забываем экранировать специальные символы **<** и **>**. Команда **expr** не ограничивается сравнением лишь чисел. Эта команда позволяет сравнивать также и строки. И хотя **expr** подходит для простых операций в сценариях, возможности этой команды весьма ограничены. Для операций с числами с плавающей точкой, а также обработки сложных выражений, где порядок операций изменен, применяется команда **bc**. Она представляет собой специальный язык программирования, предназначенный для математических действий. Применяется на практике обычно так:

```
var3=`echo "$var1+$var2" | bc -l`
var3=`echo "(100-$var1)/(100+$var2)" | bc -l`
```

Данный пример демонстрирует простейший способ использования команды **bc**. На самом же деле, она является мощным средством, возможности которого гораздо шире.

Циклы.

Иногда требуется повторять действие или группу действий до тех пор, пока соблюдаются определенные условия. В таких случаях стоит воспользоваться циклами. Командный интерпретатор поддерживает три вида циклических конструкций: **while**, **until** и **for**.

Цикл **while** выполняет операторы, заключенные в нем, до тех пор, пока условие цикла является истинным:

```
i=1
while [ $i –le 10 ]
do
  echo $i
  i=`expr $i + 1`
done
```

Команда **while** содержит условие, заключенное в квадратные скобки. На самом деле, они представляют собой сокращенную запись команды под названием **test**. Последняя часто используется в сценариях командного интерпретатора. Команда **test** использует достаточно прозрачный синтаксис, близкий к Фортрану:

-eq – истина, если операнды равны;
-ne – истина, если операнды не равны;
-gt – истина, если первый операнд больше второго;
-ge – истина, если первый операнд больше или равен второму;
-lt – истина, если первый операнд меньше второго;
-le – истина, если первый операнд меньше или равен второму.

Цикл **until** по смыслу противоположен циклу **while**. Он выполняет последовательность операций до тех пор, пока условие не станет истинным:

```
i=1
until [ $i -gt 10 ]
do
  echo $i
  i=`expr $i + 1`
done
```

Циклы **while** и **until** позволяют работать с логическими операторами **AND** и **OR**. Логическое выражение **AND** возвращает значение истина лишь в том случае, когда оба операнда истинны, а выражение **OR** – когда лишь один из операндов имеет значение истина:

while [$var1 -gt 10] && [$var1 -lt 20] – истина, если **10 < var1 < 20**;

while [$var1 -lt 10] || [$var1 -gt 20] – истина, если **var1 < 10** или **var1 > 20**.

Цикл **for** отличается от **while** и **until**. Вместо проверки истинности условия цикл **for** выполняет операторы внутри тела цикла в зависимости от количества аргументов в списке. Цикл **for** содержит переменную, которая при каждой итерации получает следующий аргумент из списка. Цикл **for** продолжается до тех пор, пока список не будет исчерпан:

```
for num in `jot 10 10 20`
do
sq_root=`echo "scale=3; sqrt($num)" | bc -l
echo $sq_root
done
```

Эта программа выводит квадратные корни чисел от 10 до 20.

В программировании сценариев используются так же операторы **true** и **false**. Их единственным назначением является возвращение значения истина (**1**) или ложь (**0**), соответственно. Этими операторами можно воспользоваться для создания бесконечных циклов.

Если вдруг возникает необходимость программно прервать бесконечный цикл, то стоит воспользоваться одним из двух следующих операторов: **break** или **continue**. Оператор **break** прерывает цикл немедленно, независимо от того, выполнено ли условие окончания цикла. Оператор **continue** заставляет цикл перейти к началу и проверить условие.

Условные операторы.

Очень часто программу нужно построить так, чтобы определенные участки кода выполнялись только при определенных условиях. Существует две общих формы условных операторов: **if** и **case**. Кроме того, есть еще логические операторы **AND** и **OR.**

Операторы **if** проверяют числовые выражения. Если условие истинно, выполняются операторы внутри блока **if**. Если оно ложно, то либо выполняются операторы внутри блока **else**, либо не выполняется ничего:

```
#!/bin/sh
if [ $# -ge 1 ]
then
  echo "You supplied $# arguments."
else
  echo "Usage: $0 filel file2..."
fi
exit 0
```

Часть **then** оператора **if** является обязательной, а часть **else** – нет. Операторы в блоке **then** выполняются тогда, когда условие истинно. Но иногда возникает необходимость выполнить действия

лишь тогда, когда выражение ложно. В этом случае нужно воспользоваться двоеточием:

```
if [ $# -ge 1 ]
then
  :
else
  echo "Usage: $0 filel file2..."
fi
```

В некоторых случаях возникает необходимость проверить два или несколько разных условий и предпринять различные действия в зависимости от результатов каждого этапа. Для этих целей используется оператор **elif**. Когда используется оператор **elif**, программа вначале выполняет оператор **if**. Если его условие истинно, выполняется его код, а затем управление передается следующему оператору (т.е. оператору, расположенному после **fi**). Если условие ложно, проверяется условие в первом операторе **elif**. Если оно истинно, выполняются операторы из его блока, и программа переходит к концу блока **if.** Если оно ложно, проверяется следующий оператор **elif** и т.д. Фактически, условия проверяются до тех пор, пока одно из них не даст значение истина. Если такого условия нет, ничего не происходит либо запускаются операторы, заключенные в блоке **else** (если он присутствует).

Второй условный оператор – **case**, аргументом которого является переменная, содержит блоки, выполнение которых зависит от значения переменной. Его лучше использовать при проверке строковых значений, т.к. он сравнивает переменную с конкретными значениями, кроме того поддерживает символы-заместители и конвейеры:

```
#!/bin/sh
echo "Do you really want to shutdown? (yes, no)"
read ans
case "$ans" in
[Yy]|[Yy][Ee][Ss])
  echo "OK. Good bye."
  shutdown –h now
  ;;
[Nn]|[Nn][Oo])
  echo "OK. Go on."
  ;;
*)
  echo "Error. Please, type yes or no."
  ;;
esac
exit 0
```

Логические операторы **AND** и **OR** (**&&** и **||**) в некоторых случаях заменяют операторы **if**. Код завершения первой команды используется как условие запуска второй. Мы часто пользуемся ими в командной строке. Например:

```
# tar czvf backup.tar.gz ./scripts && rm -r ./scripts
```

Если первая операция прошла успешно, то выполняется вторая. Если нет, то вторая команда не выполняется. Другими словами: "Необходимо выполнить команды A и B. Но если команда A невыполнима, то не следует исполнять и B". Второй пример:

```
# tar czvf backup.tar.gz ./scripts || echo "Operation failed."
```

Если первая операция прошла успешно, то вторую выполнять не нужно. Если нет, то выполнить вторую. Другими словами: "Если A невыполнимо, исполнить B. Но если A завершилось успешно, B не выполнять".

Функции.

Теперь стоит отметить то, что иногда приходится повторять один и тот же набор команд в скрипте. Здесь нам на помощь приходят функции. Они представляют собой группы операторов, вызываемые одной командой. Их можно рассматривать как "минипрограммы внутри программ". О важности и полезности функций говорить не будем, т.к. надеемся, что это и так понятно. Прежде всего, если определенный набор операций требуется выполнить в нескольких местах программы, достаточно воспользоваться лишь одной командой. Во-вторых, если вы захотите изменить то, как исполняется операция, достаточно будет внести изменения лишь в одном фрагменте кода:

```
on_exit() {
   echo "Good bye."
   mail admin@example.com < ./report.txt
   rm ./report.txt
}
...
on_exit
...
```

Между вызовом функции и вызовом другой программы существует важное различие. Функция выполняется текущим интерпретатором, а отдельная программа запускается в другой копии командного интерпретатора. Это значит, что функциям доступны и переменные среды, и внутренние переменные вызывающей, ее программы. Отдельной программе, исполняемой другим интерпретатором, они недоступны.

Файловые дескрипторы.

Файловые дескрипторы представляют собой числовые идентификаторы, устанавливаемые ядром при запуске каждого нового процесса. По умолчанию командный интерпретатор открывает три файловых дескриптора:

F.D. 0 STDIN. Это стандартный входной поток. Обычно ввод поступает с клавиатуры, однако его можно перенаправить из файла или какого-либо другого источника;

F.D. 1 STDOUT. Это стандартный выходной поток. Обычно вывод поступает на экран, однако, как вы понимаете, его также можно перенаправить;

F.D. 2 STDERR. Это стандартный поток ошибок. Обычно выводится на экран, но и его можно перенаправить.

Для открытия файлового дескриптора используется команда **exec**:

```
#!/bin/sh
exec > ./testfile.txt
echo "Line 1 of the file"
echo "Line 2 of the file"
echo "Line 3 of the file"
exit 0
```

Оператор **exec** во второй строке примера перенаправляет поток **STDOUT** в файл **testfile.txt**. В результате, операторы **echo** выводят информацию в **testfile.txt**, а не на экран, хотя в каждом из них поток не перенаправлен явно. Если в файл нужно вывести более одной строки текста, эффективнее использовать файловый дескриптор, а не перенаправление вывода в каждом операторе. Этот же подход применяется при работе с дескриптором **STDIN** и командой **read**:

```
#!/bin/sh
exec < ./testfile.txt
while read string do
echo $string
done
exit 0
```

Если файл **testfile.txt**, созданный в предыдущем примере, существует, то программа выведет на экран по очереди все его строки. Пример иллюстрирует важную концепцию использования дескриптора с командой **read**. Поскольку между последовательными вызовами **read** файл не закрывается, **read** помнит последнюю прочитанную строку и передвигает указатель на следующую строку. Таким образом, **read** автоматически последовательно читает все строки файла.

Отладка сценариев.

И в завершение коснемся отладки сценариев командного интерпретатора, которая рано или поздно вам понадобится. Хотя интерпретатор не имеет полноценного отладчика, он обеспечивает простейшие возможности для мониторинга всех выполняемых действий. Трассировка включается посредством редактирования первой строки:

```
#!/bin/sh -xv
```

После этого, при запуске программы, вы будете видеть на экране последовательно все действия, которые она выполняет.

Пример 1. Простой и полезный скрипт backup.sh

Теперь немного попрактикуемся. Создадим еще у себя в домашнем каталоге папку **backup** – для сохранения резервных копий основных конфигурационных файлов скриптом, который мы прямо сейчас и напишем:

```
# mkdir /home/raph/backup
# vi /home/raph/scrpts/backup.sh
   #!/bin/sh
     # записываем в переменную значение текущей даты
   date=`date "+%Y%m%d"`
     # записываем в переменную путь к каталогу бэкапов
   path="/home/raph/backup"
     # создаем временный каталог
   mkdir ${path}/temp
     # копируем в temp все, что нужно сохранить
   echo "Сохранение каталога /etc"
   cp -r /etc ${path}/temp/
   echo "Сохранение каталога /usr/local/etc"
   cp -r /usr/local/etc ${path}/temp/usr_local_etc
   echo "Сохранение каталога /usr/local/www"
   cp -r /usr/local/www ${path}/temp/usr_local_www
   echo "Сохранение каталога /usr/src/sys/amd64/conf"
   cp -r /usr/src/sys/amd64/conf ${path}/temp/kernel
   echo "Сохранение каталога /home/raph/scrpts"
   cp -r /home/raph/scrpts ${path}/temp/
     # меняем права на все, что скопировали
   chown -R raph:wheel ${path}/*
     # архивируем все и удаляем temp
   echo "Архивирование резервной копии..."
   cd ${path}/temp/
   tar czf ${path}/backup_${date}.tar.gz ./*
   rm -r ${path}/temp
   echo "Выполнено. Имя файла: backup_${date}.tar.gz"
     # завершаем выполнение программы
   exit 0
```

Сначала мы определили две текстовые переменные – текущую дату и путь к месту хранения бэкапов. Затем создали временную папку, в которую скопировали все наши конфигурационные файлы вместе со своими скриптами. После этого мы заархивировали все в файл **backup_[текущая дата].tar.gz** и удалили временную папку. Добавим нашему скрипту прав на выполнение и посмотрим на результат выполнения программы:

```
# chmod ugo+x /home/raph/scrpts/backup.sh
# /home/raph/scrpts/backup.sh
Сохранение каталога /etc
Сохранение каталога /usr/local/etc
Сохранение каталога /usr/local/www
Сохранение каталога /usr/src/sys/amd64/conf
Сохранение каталога /home/raph/scrpts
Архивирование резервной копии...
Выполнено. Имя файла: backup_20130101.tar.gz
# _
```

Результат – архив с текущими копиями всех конфигурационных файлов, веб-сайтов, а так же ваших рабочих скриптов, можно забирать из своего домашнего каталога по FTP, а можно отправлять себе по почте 1 раз в неделю, если прописать выполнение скрипта в **/etc/crontab** и добавить команду **mail**.

Пример 2. Общение с сервером при помощи смс

Если вы окажетесь где-то, где нет никаких коммуникаций, кроме смс, и вам очень срочно нужно будет выполнить пару команд на вашем сервере, ну или хотя бы перезагрузить его, то вы должны подготовить это заранее. Мы будем использовать возможность отправлять смс на электронную почту и наоборот – электронную почту на смс. Итак:

```
# vi /home/raph/scrpts/sms.sh
   #!/bin/sh
     # записываем строку с кодом CMD (команда), если есть
   oper1=`grep CMD: /tmp/sms.txt`
     # записываем строку с кодом RBT (reboot), если есть
   oper2=`grep RBT: /tmp/sms.txt`
     # проверяем, не пустая ли переменная 1
   if [ $oper1 ]
   then
     # если нет, то записываем саму команду
     cmd=`echo $oper1 | cut -d: -f2`
     # выполняем ее и записываем ответ
     ans=`$cmd`
     # ответ отправляем обратно смской
     echo $ans | mail 380671234567@sms.kyivstar.net
   fi
     # проверяем, не пустая ли переменная 2
   if [ $oper2 ]
   then
     # если нет, то записываем время перезагрузки
     time=`echo $oper2 | cut -d: -f2`
     # перезагружаем сервер по времени
     shutdown -r $time
   fi
   rm /tmp/sms.txt
     # завершаем выполнение программы
   exit 0

# chmod ugo+x /home/raph/scrpts/sms.sh
```

Ну а для того, чтобы сохранить электронное письмо, как текстовый файл, и получить наш **sms.txt**, нужно создать алиас. Запись следующего вида продублирует письмо в наш почтовый ящик, сохранит его, как текстовый файл, и запустит скрипт обработки:

```
# vi /etc/mail/aliases
   sms: raph, /tmp/sms.txt, "|/home/raph/scrpts/sms.sh"

# newaliases
```

У каждого мобильного оператора свои настройки. Мы рассмотрели практический пример работы через украинского мобильного оператора «Киевстар». Сообщение смс будет приходить нам на почту **sms@example.com**. Примеры формата смс сообщения:

sms@example.com CMD:ipfw add 500 allow ip from any to any

sms@example.com RBT:now

Отправив подобную смс на номер **555** (такой номер определил «Киевстар»), мы получим его, как письмо по почте. Оно сохранится и запустится скрипт обработки, в котором сначала мы вытаскиваем из текстового файла строки с ключевыми словами **CMD** (команда) и **RBT** (перезагрузка). Далее, если есть строка, которая начинается с **CMD**, выполняется команда, а ее ответ отправляется на ваш мобильный, как ответная смс. Если есть строка **RBT**, то выполняется перезагрузка по времени, после чего удаляется временный файл с сообщением. Вы можете придумать свои варианты, насколько будет необходимость.

Пример 3. Управление соединениями VPN IPSEC

В предыдущей главе мы рассматривали возможность построения VPN при помощи **ipsec**. Хорошо, если эта сеть небольшая и настраивается один раз. А если большая и часто меняется? Напишем скрипт, который будет читать настройки из текстового файла и поднимать наши туннельные соединения по требованию. Для начала, создадим файл с настройками:

```
# vi /home/raph/scrpts/vpn.conf
   66.66.66.66:192.168.0:1
   77.77.77.77:10.77.77:10
   88.88.88.88:10.0.88:100
```

То есть, мы находимся в центральном офисе. Удаленный офис №1 имеет внешний IP-адрес **66.66.66.66**, внутреннюю локальную сеть **192.168.0.0/24** и шлюз **192.168.0.1**; удаленный офис №2 – внешний IP **77.77.77.77**, локальная сеть **10.77.77.0/24** и шлюз **10.77.77.10**; удаленный офис №3 – внешний IP **88.88.88.88**, локальная сеть **10.0.88.0/24** и шлюз **10.0.88.100**. Теперь напишем сам скрипт:

```
# vi /home/raph/scrpts/vpn.sh
   #!/bin/sh
     # присваиваем начальное значение счетчику
   i=0
     # создаем переменные с параметрами нашей сети
     # o=наш, e=внеш, i=внутр, n=сеть, a=адрес.
   oea="22.22.22.22"
   oin="10.0.0"
   oia="1"
     # сохраняем данные об уже построенных туннелях
   ifconfig | grep tunnel | cut -f5 -d' ' > /tmp/ifcfg.txt
     # начинаем читать конфигурационный файл
   exec < $1
     # начинаем перебирать строки конфигурационного файла
   while read str
   do
```

```
    # создаем переменные с параметрами удаленной сети
    # r=удален, e=внеш, i=внутр, n=сеть, a=адрес.
    rea=`echo $str | cut -f1 -d':'`
    rin=`echo $str | cut -f2 -d':'`
    ria=`echo $str | cut -f3 -d':'`
    # ищем эту сеть в уже построенных туннелях
    s1=`grep $rea /tmp/ifcfg.txt`
    # номер правила для фаерволла (3010, 3020..)
    nn=`expr $i \* 10 + 3000`
    # проверяем значение второго параметра
    if [ $2 -eq "up" ]
    then
    # если up, то проверяем не поднят ли уже этот туннель
      if [ -z $s1 ]
      then
    # если не поднят, то поднимаем gif, route, ipfw
        ifconfig gif$i create
        ifconfig gif$i tunnel $oea $rea
        ifconfig gif$i inet $oin.$oia $rin.$ria netmask
255.255.255.0
        route add $rin.0/24 $rin.$ria
        ipfw add $nn allow ip from any to any via gif$i
      fi
    # проверяем значение второго параметра еще раз
    elif [ $2 -eq "down" ]
    then
    # если down, то удаляем gif, route, ipfw
      ifconfig gif$i destroy
      route delete $rin.0/24 $rin.$ria
      ipfw delete $nn
    else
    # если не up и не down, просим уточнить действие
     echo 'Use "up" or "down" parameter...'
    fi
    i=`expr $i + 1`
  done
  rm /tmp/ifcfg.txt
    # завершаем выполнение программы
  exit 0
```

```
# chmod ugo+x /home/raph/scrpts/vpn.sh
```

Построить туннели к удаленным серверам можно выполнив наш скрипт с параметрами:

```
# /home/raph/scrpts/vpn.sh /home/raph/scrpts/vpn.conf up
```

При этом, в начале мы определяем счетчик **i** и параметры нашей сети, затем командой **ifconfig** проверяем какие туннели у нас уже построены и сохраняем эту информацию в текстовый файл. Далее начинаем считывать построчно из нашего **vpn.conf** параметры удаленных сетей. Если вторым параметром скрипта идет слово **up**, то мы проверяем, не построен ли уже этот туннель – если нет, то строим. Если же вторым параметром идет слово **down**, то мы уничтожаем текущий интерфейс и все его настройки.

Запуск этого скрипта можно поместить в каталог **/usr/local/etc/rc.d/** (например другим скриптом), тогда VPN будет строиться автоматически при загрузке сервера, а при появлении новой удаленной сети, нужно будет просто добавить ее параметры в **vpn.conf** и запустить скрипт вручную.

Пример 4. Контроль трафика пользователей (ipa)

В конце главы 4 мы настроили подсчет общего входящего и исходящего трафика при помощи пакета **ipa**. По такому же принципу мы можем считать трафик конкретных хостов локальной сети по определенным портам, т.е. смоделировать счетчики так, как мы посчитаем нужным, используя возможности **ipfw**.

В этом примере мы проанализируем трафик, проходящий через **squid**, и воспользуемся лимитами и потоками **ipa** для его ограничения. Для начала напишем скрипт, который будет добавлять в наш фаерволл счетчики всех возможных хостов локальной сети (например это будет область IP-адресов вашего DHCP сервера с 10.0.0.10 по 10.0.0.99):

```
# vi /home/raph/scrpts/ipa_ipfw.sh
   #!/bin/sh
     # присваиваем начальное значение счетчику адресов
   ip=10
     # описываем нашу локальную сеть
   nt="10.0.0"
     # выполняем цикл, пока счетчик меньше или равен 99
   while [ $ip -le 99 ]
   do
     # номер правила для фаерволла (1010, 1011..)
     nn=`expr $i + 1000`
     # добавляем правило-счетчик в фаерволл
     ipfw add $nn count tcp from me 3128 to $nt.$ip
     # увеличиваем значение счетчика на 1
     ip=`expr $ip + 1`
   done
   exit 0
```

То есть мы добавили в фаерволл 90 правил от 1010 до 1099, которые будут считать трафик от вашего **squid** в локальную сеть к каждому хосту персонально. Теперь напишем скрипты для блокирования и ограничения скорости доступа к прокси-серверу:

```
# vi ipa_access.sh
   #!/bin/sh
     # находим адрес по номеру правила-счетчика
   ip=`expr $1 - 1000`
   nt="10.0.0"
     # номер правила для фаерволла (2010, 2011..)
   nn=`expr $1 + 1000`
     # проверяем значение второго параметра
   if [ $2 = deny ]
   then
     # если deny, то закрываем доступ
     ipfw add $nn deny tcp from $nt.$ip to me 3128
     # проверяем значение второго параметра еще раз
   elif [ $2 = allow ]
   then
     # если up, то удаляем блокирующее правило
     ipfw delete $nn
   fi
   exit 0

# vi ipa_speed.sh
   #!/bin/sh
     # находим адрес по номеру правила-счетчика
   ip=`expr $1 - 1000`
   nt="10.0.0"
     # номер правила для фаерволла (2010, 2011..)
   nn=`expr $1 + 2000`
     # проверяем значение второго параметра
   if [ $2 = down ]
   then
     # если down, то понижаем скорость
     ipfw add $nn pipe $ip tcp from me to $nt.$ip out
     ipfw pipe $ip config bw 256Kbit/s
     # проверяем значение второго параметра еще раз
   elif [ $2 = up ]
   then
     # если up, то удаляем тормозящее правило
     ipfw delete $nn
   fi
   exit 0
```

Необходимые нам скрипты готовы. Теперь нужно немного усовершенствовать **ipa**, а именно научить ее работать с лимитами и порогами для локальных пользователей. Приведем файлы **ipa.conf** и **ipastat.conf** к следующему виду:

```
# vi /usr/local/etc/ipa.conf
     # служебные модули и параметры
   ac_mod "ipa_ipfw.so";
   db_mod "ipa_db_sdb.so";
     # используем не только абсолютные пути для команд
   only_abs_paths = no;
   global {
     # время обновления и фиксирования статистики
     update_time = 5m;
     append_time = 3h;
     ac_list = ipfw;
     db_list = sdb;
     ipfw:maxchunk = 1G;
     sdb:db_group = wheel;
   }
     # при начале работы запускаем ipa_ipfw.sh
   startup { exec root "/home/raph/scrpts/ipa_ipfw.sh"; }

     # шаблон для правил IN, OUT (общие счетчики)
   rulepat "[A-Z]" {
     # определяем лимит
     limit LIM {
     # значение лимита = 50 Гб
       limit = 50G;
     # перезапуск недостигнутого лимита = 1 месяц
       restart { restart = +M; }
     # при достижении лимита = проинформировать письмом
       reach { exec root "echo \"OUR %rule% TRAFFIC IS OVE
R 50G!!!\" | mail raph@example.com"; }
     # перезапуск достигнутого лимита = 1 месяц
       expire { expire = +M; }
     }
   }
```

```
    # шаблон для правил 1010, 1011..1099 (персональные)
  rulepat "[0-9]" {
    # определяем лимит
    limit LIM {
    # значение лимита = 1 Гб
      limit = 1G;
    # перезапуск недостигнутого лимита = 1 неделя
      restart { restart = +W; }
    # при достижении лимита = ipa_access.sh deny, письмо
      reach { exec root "/home/raph/scrpts/ipa_access.s
h %rule% deny"; exec root "echo \"LIMIT %rule% WAS ACTIV
ATE D!\" | mail admin"; }
    # достиг.лим. = 1 неделя, ipa_access.sh allow, письмо
      expire { expire = +W; exec root "/home/raph/scrpt
s/ipa_access.sh %rule% allow"; exec root "echo \"LIMIT
%rule% WAS DEACTIVATED!\" | mail admin"; }
    }

    # определяем порог
    threshold THR {
    # значение порога = 100 Мб +/- 1 Мб
      threshold = 100M;
      threshold_balance = 1:-:1;
      threshold_deviation = 1M;
    # временной снимок = 1 час
      threshold_time_width = 1h;
    # время скольжения = 10 минут
      threshold_time_slice = 10m;
    # при непревышении порога = ipa _speed.sh up, письмо
      below_threshold { exec root "/home/raph/scrpts/ip
a_speed.sh %rule% up"; exec root "echo \"SPEED %rule% WA
S UP!\" | mail admin"; }
    # при превышении порога = ipa _speed.sh down, письмо
      above_threshold { exec root "/home/raph/scrpts/ip
a_speed.sh %rule% down"; exec root "echo \"SPEED %rule%
WAS DOWN!\" | mail admin"; }
    }
  }
```

```
    # правила-счетчики общего трафика
  rule IN { ipfw:rules = 800; }
  rule OUT { ipfw:rules = 900; }
    # правила-счетчики персонального трафика
  rule 1010 { ipfw:rules = 1010; }
  rule 1011 { ipfw:rules = 1011; }
   ...
  rule 1099 { ipfw:rules = 1099; }

# vi /usr/local/etc/ipastat.conf
  st_mod "ipa_st_sdb.so";
  dynamic_rules = yes;
  dynamic_limits = yes;
  dynamic_thresholds = yes;
  global { st_list = sdb; }
```

Пожалуй, примерно так. В начале мы определили, что при запуске **ipa** нужно добавить счетчики трафика для локальной сети в фаерволл (скрипт **ipa_ipfw.sh**). Затем определили два шаблона правил – для счетчиков общего трафика и для персональных счетчиков пользователей. Для общих счетчиков мы установили лимит на 1 месяц в 50Гб трафика, причем при достижении этого лимита нужно просто проинформировать нас по почте. Для персональных счетчиков немного сложнее. Мы определили лимит на неделю в 1Гб, причем, при его достижении, пользователю будет закрыт доступ к прокси-серверу до начала следующей недели (скрипт **ipa_access.sh** с параметрами **deny** и **allow**). Так же мы определили скользящий порог – если кто-то за последний час времени скачает больше 100Мб, то ему понизится скорость доступа к прокси-серверу до тех пор, пока не перестанет качать (скрипт **ipa_speed.sh** с параметрами **down** и **up**). О каждом событии блокирования/открытия доступа или понижения/восстановления скорости будет так же отправлено уведомление.

Не забудьте сделать все наши новые скрипты исполняемыми и перезапустить **ipa**:

```
# chmod ugo+x /home/raph/scrpts/ipa*
# /usr/local/etc/rc.d/ipa restart
```

Кстати, чтобы не добавлять в конфигурационный файл все 90 персональных правил вручную, можно тоже написать скрипт (а лучше скопировать **ipa_ipfw.sh** и изменить там одну строчку):

```
# cd /home/raph/scrpts/
# cp ipa_ipfw.sh ipa_conf.sh
# vi ipa_conf.sh
   #!/bin/sh
   ip=10
   nt="10.0.0"
   while [ $ip -le 99 ]
   do
     nn=`expr $i + 1000`
     # добавляем строку в ipa.conf
     echo "rule $nn { ipfw:rules = $nn; }" >> /usr/local
/etc/ipa.conf
     ip=`expr $ip + 1`
   done
   exit 0

# chmod ugo+x ipa_conf.sh
# ./ipa_conf.sh
# /usr/local/etc/rc.d/ipa restart
```

Теперь вы имеете достаточно мощную внутреннюю билинговую систему для ограничения беспорядков, которые могут натворить пользователи с вашим Интернет-каналом. Просматривать можно так:

```
# ipastat -q -r 1010 -l LIM
# ipastat -q -r 1010 -t THR
```

А вообще, у системы **ipa** достаточно подробная документация на русском языке, за что большое спасибо ее разработчику.

Основные команды и конфигурационные файлы

Работа в файловой системе

ls	Вывести содержимое каталога
cd	Перейти в каталог
pwd	Вывести текущий каталог
cp	Копировать файлы и каталоги
mv	Переместить или переименовать файлы и каталоги
touch	Создать пустой файл
mkdir	Создать каталог
rm	Удалить файл или каталог
rmdir	Удалить каталог
ln	Создать ссылку
find	Найти файл или каталог
locate	Найти файл или каталог
mount	Монтировать файловую систему
umount	Демонтировать файловую систему
tar	Архивировать файлы и каталоги

Работа с пользователями и правами

adduser	Создать нового пользователя
rmuser	Удалить пользователя
passwd	Изменить пароль пользователя
vipw	Редактировать /etc/passwd и базу данных пользователей
sudo	Получить права администратора
visudo	Редактировать файл /etc/sudoers
chmod	Изменить права доступа к файлам и каталогам
chown	Изменить права владения файлами для пользователей
chgrp	Изменить права владения файлами для групп

Работа с текстовыми файлами

more	Вывести поэкранно содержимое файла
less	Вывести поэкранно содержимое файла
grep	Найти и вывести шаблон в файле
cat	Вывести содержимое файла
wc	Подсчитать число строк, слов и символов в файле
diff	Сравнить содержимое файлов и вывести различия
fmt	Преобразовать содержимое файла к формату эл. письма
cut	Вывести определенный столбец или поле из файла
head	Вывести первые строки файла
tail	Вывести последние строки файла
sort	Сортировать содержимое файла
vi	Открыть файл в текстовом редакторе vi

Работа в системе

man	Вывести справочную информацию о программе
date	Вывести или изменить текущую дату и время
cal	Вывести календарь
ps	Вывести список процессов, запущенных в системе
top	Вывести статистику использования ресурсов процессами
kill	Завершить работу процесса по идентификатору
killall	Завершить работу процессов по имени программы
shutdown	Остановить или перезагрузить систему
halt	Остановить систему
reboot	Перезагрузить систему
uptime	Вывести время работы системы с момента включения
pkg_info	Вывести список пакетов, установленных в системе
pkg_add	Установить пакет в систему
pkg_delete	Удалить пакет из системы

make	Получить файлы порта и собрать его
make install	Инсталлировать собранный порт
make deinstall	Деинсталлировать собранный порт
make clean	Удалить рабочие файлы сборки порта
make distclean	Удалить рабочие файлы порта и дистрибутивы

Работа в сети

ifconfig	Вывести или настроить параметры сетевых интерфейсов
route	Создать маршрут в таблице маршрутизации
ping	Отправить тестовые пакеты хосту по сети
traseroute	Вывести маршрут к удаленному хосту
netstat	Вывести текущую таблицу маршрутизации
nslookup	Обработать dns-запрос
dig	Обработать dns-запрос
ipfw	Вывести или задать правила системы ipfw
trafshow	Вывести активные соединения через интерфейс
tcpdump	Вывести информацию о трафике через ipfw
ssh	Подключиться к виртуальной консоли удаленного хоста
scp	Копировать файлы и каталоги по сети по ssh

Основные конфигурационные файлы системы

/etc/motd	Приветственное сообщение
/etc/rc.conf	Параметры автозагрузки
/etc/rc.local	Дополнительные команды автозагрузки
/etc/rc.firewall	Основные правила фаерволла
/etc/passwd	Пользователи системы
/etc/master.passwd	Пользователи системы с паролями
/etc/group	Группы пользователей системы
/etc/services	Конфигурация сервисов и портов
/etc/ppp/ppp.conf	Конфигурация ppp
/etc/adduser.conf	Конфигурация adduser
/etc/sudoers	Конфигурация sudo
/etc/resolv.conf	Конфигурация dns-серверов сети
/etc/hosts	Конфигурация хостов сети
/etc/inetd.conf	Конфигурация inetd
/etc/crontab	Конфигурация cron
/etc/ftpusers	Черный список пользователей ftp
/etc/ftpchroot	Корневой каталог для пользователей ftp
/etc/namedb/named.conf	Конфигурация named
/etc/mail/sendmail.cf	Конфигурация sendmail
/etc/mail/freebsd.mc	Конфигурация sendmail для правки
/usr/src/sys/i386/conf/	Ядро 32-разрядной системы
/usr/src/sys/amd64/conf/	Ядро 64-разрядной системы

Подведение итогов

Мы хорошо потрудились и в награду получили сервер, за который не будет стыдно. На самом деле мы с вами рассмотрели лишь некоторые из возможных путей реализации основных и типичных офисных задач. Потенциала нашего сервера хватит большинству компаний абсолютно разного калибра – от небольших, до многопользовательских с распределенной структурой. Если говорить более конкретно, то мы реализовали полноценный граничный шлюз, который обеспечивает:

- Прозрачный доступ пользователей локальной сети в Интернет, как через прокси-сервер, так и напрямую;
- Фильтрацию и перенаправление HTTP трафика, а так же сбор всей статистики работы сотрудников в Интернете;
- Авторизированный и анонимный доступ к своим и общим файловым ресурсам по FTP;
- Отправку электронной почты с аутентификацией удаленных пользователей с шифрованием SSL или без;
- Прием и хранение электронной почты и доступ к ней по протоколам POP3, IMAP4 с шифрованием SSL или без;
- Антивирусный контроль электронной почты и эффективную управляемую систему фильтрации спама;
- Веб-сервер с поддержкой PHP и MySQL, а так же с возможным виртуальным хостингом сайтов;
- Сбор статистики по количеству пройденного трафика и загрузке Интернет-канала в реальном времени;
- Защиту сервера и локальной сети от вторжений, фильтрацию и перенаправление всего проходящего трафика.

Таким образом у нас получилась довольно мощная реализация основных Интернет-задач современной компании на одном физическом и логическом сервере. Хорошо это или плохо? Наверное было бы лучше, если бы мы имели возможность разделить функции и один сервер у нас был бы, как шлюз, фаерволл и прокси, второй – почтовый, а третий – веб-сервер. Для этого не обязательно иметь три физических сервера – можно воспользоваться популярной сегодня технологией виртуализации.

Кроме того, мы надеемся, что вы приобрели хороший опыт в работе с FreeBSD и у вас пропал тот первичный страх, который был при первом входе в консоль. Если это так, то сейчас вы имеете хороший фундамент и импульс для движения вперед и совершенствования полученных знаний и навыков. Наша цель достигнута.

Теперь самое время изменить приветствие, которое вы видите при входе в систему:

```
# vi /etc/motd

             ======================================
                     GATEWAY.EXAMPLE.COM
             WellCome to Example FreeBSD Server!
             ======================================

     Admin: Korney A. Kornienko
    E-mail: raph@example.com
    Mobile: +380671234567
     Skype:
       ICQ:
```

Подумайте о своих последователях. Ведь вы устанавливаете сервер не на один год и, кто знает, может быть когда-нибудь кому-то понадобится ваша помощь. А пока просто приятно будет – в сервере все должно быть прекрасно :)

Источники информации

При написании данной книги были использованы следующие источники информации:

- Книга: Майкл Эбен, Брайан Таймэн **«FreeBSD. Искусство достижения равновесия»;**
- Книга: Ральф Гильдебрандт, Патрик Кеттер **«Postfix. Подробное руководство»;**
- Команда **`man`** и каталог **`/usr/local/share/doc`**;
- Официальный сайт FreeBSD: ***http://www.freebsd.org***;
- Официальные сайты устанавливаемых программ;
- Поисковая машина **Google** и множество найденных при ее помощи статей, блогов, форумов и факов…

Обратная связь

Все свои отзывы, мнения, вопросы, пожелания, предложения и прочие мысли по поводу этой книги вы можете отправить автору с официального промо-сайта: ***http://freebsdbook.com.ua*** при помощи специальной формы обратной связи, либо напрямую: ***author@freebsdbook.com.ua***. Любой полезный конструктив не будет оставлен без внимания и ответа.

Наукове видання

Корнієнко К.О.
FREEBSD 9. КОРПОРАТИВНИЙ ІНТЕРНЕТ-СЕРВЕР.
(російською мовою)

Підписано до друку 06.03.2013.
Друк офсетний. Папір офсетний.
Гарнітура Таймс. Формат 64x90/16.
Умов. друк. арк. 11.0. Наклад 500 прим.

ФОП Декет В.М., тел.: (044) 592-06-85, 592-06-86, delia_print@i.ua.
Свідоцтво суб'єкта видавничої справи ДК №1791 від 25.05.2004.