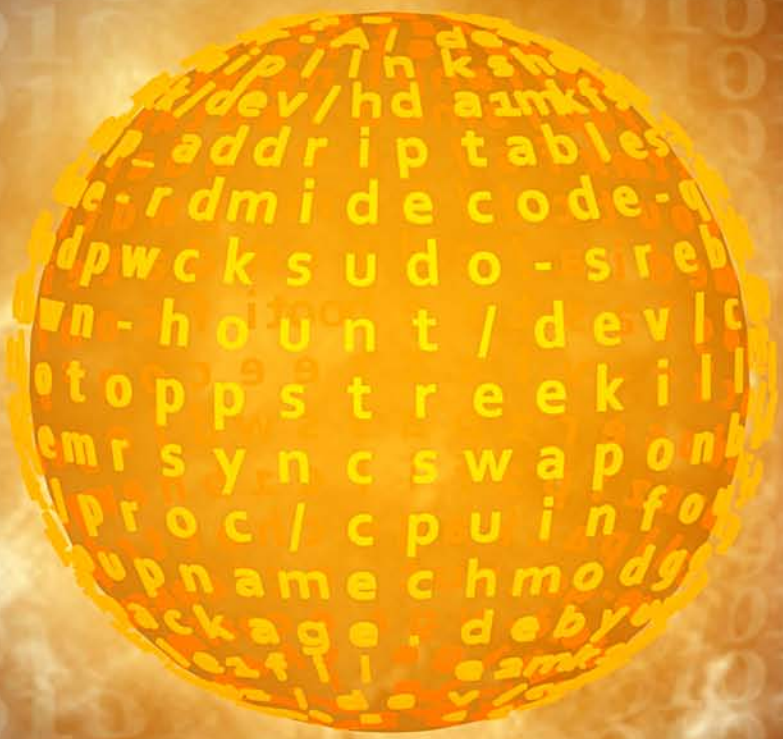
user And LINUX

№1 январь 2011

Больше чем user

Wi-fi в Ubuntu
Раздаем интернет

Программирование на Bash
Часть I: краткое введение



vnStat:
мониторинг сетевых интерфейсов

Communigate
Часть I: Обзор, назначение, возможности, применение

Обратный туннель SSH

UA Linux
open source ■ open future

Время обновлять сервера! Выпущена новая версия комплекта для разворачивания полноценного сервера на базе Ubuntu Server. Подробнее на странице проекта: http://ualinux.com/index.php/who-usp

UA Linux

open source ■ open future

Разработка комплекта дополнений Ubuntu Applications Pack

IT аутсорсинг

Поддержка и консультирование пользователей

Разработка, внедрение и сопровождение эффективных IT-решений на базе открытого ПО

Выпуск ежемесячного журнала User And Linux

Внедрение систем виртуализации

Тестирование оборудования а так же доработка под него программного обеспечения

От редакции

Мы рады приветствовать Вас, наши дорогие читатели, на страницах нового издания «U&L. Больше чем user»!

Это дополнение к журналу было разработано специально для профессионалов: системных администраторов, программистов, просто продвинутых пользователей и тех, кто стремиться ими стать.

«U&L. Больше чем user» - станет азбукой, отличной шпаргалкой и подсказкой даже для самых опытных пользователей СПО на базе GNU/Linux. Мы постараемся собирать самые интересные и полезные материал для Вас и надеемся, что наша работа будет нужна всем, кто стремиться узнать как можно больше о мире СПО.

Мы очень надеемся, что наше новое издание обретет своего читателя и будет им востребовано. Хочется пожелать удачи во всех новых начинаниях как нашей редакции так и Вам, нашим уважаемым читателям!

Ирина Сикач
Главный редактор журнала
«U&L. Больше чем user»

НАША КОМАНДА:

Главный редактор:
Ирина Сикач

Редакторы:
Алексей Невенчаный
Владимир Попов
Надежда Козаченко
Дмитрий Бутолин

Дизайн и верстка:
Виталий Панин
Игорь Шарай
Александр Никитин

Подбор материала:
Владимир Попов
Ирина Сикач
Алексей Невенчаный

Адрес журнала в интернете:
http://ualinux.com/index.php/journal

Обсуждение журнала в форуме:
http://ualinux.com/index.php/forum

По вопросам
преобретения журнала:
http://ualinux.com/index.php/get-journal

Адрес редакции:
Украина, 03040, г.Киев
а/я 56 (Невенчаный)
Email: journal@ualinux.com

Тип издания:
электронный/печатный
Регулярность: ежемесячный
Дата выпуска: 17.01.2011
Тираж: *более10 000 копий.

* указано сумарное количество загрузок прошлого выпуска журнала с первичных источников, а также загрузок с других известных ftp, http и torrent серверов

Дебет Плюс

СИСТЕМА УПРАВЛЕНИЯ ПРЕДПРИЯТИЕМ

- Полностью готова для применения на Украине
- Есть бесплатная версия
- Есть WEB-интерфейс
- Работает на Linux, Windows, MacOS
- Работает с PostgreSQL, MySQL, Oracle, MS SQL, DB2
- Бизнес логика открыта (можно разрабатывать и распространять собственные модули на коммерческой основе)

Бухгалтерия становится ближе!

Сайт: www.debet.kiev.ua
E-mail: debet@debet.kiev.ua
Тел.: 098-061-67-63, 099-242-72-01, 063-892-25-88

Содержание

Console

Hi-tech

Programming

Other

Ограничение скорости с помощью iptables

Несмотря на наличие специальных утилит для разграничения скорости передаваемых пакетов, ограничивать скорость можно и с помощью iptables.

Для этой цели используется модуль limit. Пакеты, проходящие через правило с этим критерием будут соответствовать такому условию до тех пор, пока лимит не исчерпается.

Например:

```
iptables -I FORWARD -i
eth0 -p tcp -s 192.0.2.12
-m limit --limit 3/hour
--limit-burst 5 -j ACCEPT
```

Такому правилу будут соответствовать первые 5 пакетов, пришедшие из интерфейса eth0 с адреса 192.0.2.12. Каждый проходящий пакет увеличивает счетчик на единицу, пока тот не достигнет значения параметра --limit-burst. Последующие пакеты такому правилу соответствовать не будут, а перейдут к следующему в таблице правил iptables (что с ними будет потом – наше правило не волнует).

В течение следующих 20 минут (это задает параметр —limit 3/hour, что значит три раза за час уменьшить значение счетчика на единицу) ни один пакет через это правило не пройдет (не прошедшие пакеты передаются следующим правилам). По истечении заданного интервала счетчик уменьшается на единицу. Теперь правило может принять еще один пакет. Если пакет придет – счетчик снова достигнет значения limit-burst. А если не придет в течение еще 20 минут, то в дальнейшем правило сможет принять сразу два пакета, и т.д.

www.oper777.co.cc

Ограничение количества сессий с одного ip адреса

Как ограничить количество устанавливаемых клиентами сетевых соединений, проходящих через шлюз в единицу времени (на шлюзе стоит Linux)? Я нашел всего два модуля для iptables, которые помогут мне помочь в решении данной задачи. Это модуль connlimit и модуль recent.

connlimit может работать только с tcp протоколом и ограничивает количество установленных tcp сессий для каждого клиента достаточно простым правилом. Приведенный ниже пример ограничит количество TCP сессий от каждого клиента из сети 172.16.0.0/12 в 200.

Например:

```
A FORWARD -s 172.16.0.0/12
-p tcp --syn -m connlimit
--connlimit-above 200 -j
DROP
```

Но, как показало практическое тестирование, connlimit очень сильно нагружает процессор, что делает его применение на роутере нецелесообразным, т.к. несколько клиентов в сети могут создать шторм сетевых пакетов и вывести роутер из рабочего режима, затруднив остальным пользователям комфортную работу.

recent работает по следующему принципу: первым правилом iptables вы назначаете пакетам, удовлетворяющим определенным критериям, уникальное имя, которое впоследствии будет использоваться этим модулем для наложения ограничений. Например присвоим всем сетевым пакетам с состоянием NEW (новое соединение), проходящие через сетевой интерфейс eth0 нашего роутера имя SYNF.

```
  -A FORWARD -i eth0 -m state
--state NEW -m recent --set
--name SYNF --rsource
```

Вторым правилом мы скажем модулю, что если от какого-то клиента поступает пакетов помеченных как SYNF больше 10 за 10 секунд, то такие пакеты должны отбрасываться.

```
-A FORWARD -i eth0 -m state
--state NEW -m recent
--update --seconds 10
--hitcount 10 --rttl --name
SYNF --rsource -j DROP
```

Тестирование показало, что использование модуля recent не влияет на нагрузку роутера. Причем с помощью комбинации модулей state и recent можно делать достаточно интересные ограничения в сетевом трафике, препятствуя распространению спама, флуда и пр.

www.centos.alt.ru

Обратный туннель SSH

Обратный туннель применяется в том случае, когда нужно попасть на машину, защищенную межсетевым экраном или находящуюся за NAT.

Принцип действия заключается в том, что соединение инициирует удаленная машина, а мы попадаем туда по уже готовому соединению. В такой туннель можно отправить любой трафик, не только ssh.

Итак, со стороны компьютера за NAT используется следующее заклинание:

```
  # ssh user@server -R
5544:localhost:22 -N
```

На человеческом языке это звучит как: «подключись на server под именем user таким образом, чтобы удаленный порт 5544 соединился с локальным 22 и не запускай интерактивную оболочку». Если мы хотим, чтобы пользователи с server могли попасть на эту машину, на ней должен быть запущен sshd. Если же мы пробрасываем порт для другой службы, то, естественно, это требование не обязательное.

На машине server достаточно выполнить команду

```
ssh localhost -p 5544
```

чтобы оказаться на удаленной машине.

www.oper777.co.cc

Экранирующий символ `~` в secure shell SSH

Клиент OpenSSH обладает одной «фишкой», о которой многие не знают: экранирующий символ '~'. Чтобы воспользоваться им, сперва нажмите Enter (экранирующие символы распознаются только тогда, когда они

расположены в начале строки), затем введите символ тильды — '~'. Если ничего в терминале не вывелось, значит всё хорошо, экранирующий символ распознан.

Теперь нажмите Ctlr+Z, после чего ваша текущая сессия в SSH будет приостановлена и вас «выбросит» в оболочку вашей локальной системы. Удобная штука, если вам необходимо не прерывая SSH-сессию, что-то сделать на локальной машине, при этом не открывая лишних терминалов. Кстати, этот трюк работает и в screen и даже в Mutt. Теперь, как только вам понадобится вернуться в приостановленную SSH-сессию, используйте команду fg.

Ctrl+Z — не единственная команда, которую можно использовать после ввода экранирующего символа '~'. Для того, чтобы получить полный список возможных команд, воспользуйтесь последовательностью '~?'. Правда, эта последовательность не работает в Mutt, в то время как в screen всё отлично.

Другой последовательностью, которая может пригодится вам в работе, является последовательность '~C'. При помощи неё вы можете попасть в командную строку самого SSH-клиента. Оказавшись в ней, вы можете получить список доступных команд при помощи символа '?'. При помощи командной строки SSH вы можете настраивать перенаправления потоков, не перезапуская клиент SSH. После того, как вы настроите какие-либо перенаправления, вы можете получить их список при помощи последовательности '~#'.

Используя последовательность '~.' вы можете немедленно прервать SSH-сессию. Очень полезная штука, если сетевое соединение с сервером внезапно пропадёт, а вам необходимо закрыть SSH-клиент, не дожидаясь истечения таймаута или не закрывая окно с терминалом.

Также, если вдруг вам понадобится, вы можете изменить экранирующий символ с '~' на какой-нибудь другой. Это делается при помощи опции '-e' OpenSSH-клиента во время его запуска. Например, команда

```
$ ssh -e % me@remote.com
```

замените экранирующий символ на знак процента.

www.ashep.org

Защита POP3 с помощью Stunnel

Даже если POP3/IMAP-сервер не поддерживает ни один из вариантов безопасных протоколов (SPOP и IMAPS), можно применить Stunnel, который дает возможность создавать TCP-туннели, по которым данные пересылаются в зашифрованном виде.

Stunnel предназначен для универсального туннелирования TCP-соединений. Если Stunnel еще не установлен, можно загрузить его с сайта: http://www.stunnel.org (конечно, понадобится SSL-библиотека, к примеру OpenSSL). Со стороны сервера можно употреблять Stunnel, чтобы предоставлять сервисы SPOP и IMPAS пользователям.

Генерация ключа

Сначала надо сгенерировать ключ для сервера. В каталоге /etc/stunnel для этих целей можно найти сценарий generate-stunnel-key.sh, вызывающий библиотеку openssl для генерирования сертификатов x509.

```
USE_DH=0

openssl req -new x509 -days
365 -nodes -config .stunnel.
cnf \ -out stunnel.pem
-keyout stunnel.pem
```

```
test $USE_DH -eq 0 opensll
gendh 512 » stunnel.pem

openssl x509 -subject
-dates -fingerprint -noout
-in stunnel.pem

chmod 600 stunnel.pem rm -f
stunnel. rr.d
```

Если был вызван сценарий generate-stunnel-key.sh не из каталога /etc/stunnel, нужно переместить .pem-файлы в каталог /etc/stunnel.

(От редакции:
«Для лучшего понимания процесса генерации сертификатов полезно разобраться в нем более детально. В одном из следующих номеров мы опубликуем развернутый материал о работе с Openssl.»)

Конфигурация

После этого надо создать конфигурационный файл для Stunnel. Можно создать конфигурационный файл для туннелирования POP3-сервера, но если нужно применять Stunnel для туннелирования другого TCP-сервиса, то надо будет модифицировать конфигурацию Stunnel.

Stunnel поддерживает chroot-окружения, поэтому надо создать группу и пользователя, от имени которых будет запускаться Stunnel, а также структуру каталогов /chroot/stunnel:

```
# mkdir -p /chroot/stunnel
/chroot/stunnel/etc /
chroot/var /chroot/stunnel/
var/run

# groupadd stunnel

# useradd -g stunnel -s/
sbin/nologin
```

Нужно создать файл /etc/stunnel/stunnel.conf и добавить в него следующие записи:

```
chroot = /chroot/stunnel

setuid = stunnel

setgid = stunnel

pid = /stunnel.pid

debug = mail.notice

client = no
```

Кроме этого можно указать месторасположение только что созданного сертификата:

```
cert = /etc/stunnel/
stunnel.pem
```

Путь к сертификату надо указывать абсолютный (а не относительно chroot-окружения), потому что сертификат загружается до вызова chroot(). Это позволяет хранить сертификаты за пределами «песочницы», что обеспечивает дополнительную безопасность.

Теперь надо подумать о клиентах. Конечно, самый безопасный вариант когда клиенты предоставляют свои сертификаты, но это и самый непрактичный результат: представьте себе лицо обычного пользователя, которому надо сказать: «– Создайте сертификат X509». Поэтому с практической точки зрения лучшим будет вариант, когда пользователи будут пропускаться как с сертификатом, так и без него:

```
verify = 1
```

Осталось только добавить опции POP3:

```
[pop3 server]

accept = 995

connect = localhost:pop3
```

Первая строка [pop3 server] применяется для повышения читабельности конфигурационного файла, 995 – это стандартный порт для SPOP (этот порт будет прослушивать spop), а третья строка устанавливает имя узла и порт, которому будет перенаправлено (forward) соединение.

Все, что теперь осталось – запустить Stunnel (и конечно же, обеспечить его автоматический запуск):

```
# stunnel /etc/stunnel/
stunnel.conf
```

www.suvan.ru

vnStat: мониторинг сетевых интерфейсов

Наблюдение за сетевыми интерфейсами в Linux обычно не представляет собой какой-либо сложности. При помощи таких утилит как sar, Iperf, и vnStat администратор может легко и удобно отслеживать активность сетевых интерфейсов, при это можно наблюдать как мгновенную активность, так и за последнее время. Сегодня мы рассмотрим известную многим утилиту vnStat. Утилита vnStat ведёт лог трафика за определённый период и хранит собранную статистику по каждому выбранному администратором интерфейсу, предоставляя доступ к ней в случае необходимости. vnStat, в отличие от снифферов, например Wireshark, не собирает информацию непосредственно с сетевого интерфейса, а анализирует данные, предоставляемые ядром через файловые системы proc и sys, что даёт возможность использовать эту утилиту даже непривилегированным пользователем.

Установка

vnStat присутствует в стандартных репозиториях Ubuntu, так что установка не должна вызвать каких-либо осложнений:

```
sudo apt-get install vnstat
```

После установки будет выдано предупреждение о том, что база данных пуста. Не пугайтесь, это естественно, если в этой системе vnStat ещё никогда не работала. После того, как вы настроите vnStat должным образом, база данных будет наполняться информацией о статистике сетевого трафика, а пока она пуста.

Настройка и запуск

После установки vnStat необходимо инициализировать базы данных интерфейсов, чтобы демон смог начать работу по сбору статистики. Делается это простой командой (замените eth0 на имя нужного интерфейса):

```
do vnstat -i eth0 -u

Error: Unable to read
database «/var/lib/vnstat/
eth0».

Info: - A new database has
been created.
```

Повторите эту процедуру для всех интерфейсов, активность которых необходимо отслеживать.

Сбором статистики занимается демон vnStat — vnstatd. Init-скрипт для запуска/останова демона располагается в файле /etc/init.d/vnstat. После установки пакета будут созданы симлинки из каталогов /etc/rc*.d для автоматического запуска демона во время старта системы. Если вы не перезагружали систему после установки vnStat, его придётся запустить вручную:

```
sudo /etc/init.d/vnstat
start
```

Предварительно вы можете просмотреть и изменить в случае необходимости файл конфигурации vnStat, расположенный в файле /etc/vnstat.conf. Хотя, как мне кажется, настройки по умолчанию вполне пригодны для использования в большинстве случаев.

По умолчанию демон vnStat обновляет базу данных каждые 5 минут, а «снимает» информацию с интерфейсов каждые 30 секунд. Эти и другие параметры регулируются в файле конфигурации и при необходимости вы можете изменить их, предварительно проконсультировавшись с man vnstat.conf. Если вы не хотите ждать, пока демон обновит данные в БД, вы можете принудительно обновить данные:

```
sudo vnstat -u
```

Эта команда обновит БД всех интерфейсов. Если вам необходимо обновить информацию по какому-то определённому интерфейсу, воспользуйтесь опцией «-i», передав ей имя нужного интерфейса, например:

```
sudo vnstat -i eth0 -u
```

Просмотр статистики

Сводная информация по всем интерфейсам:

```
vnstat
```

```
File  Edit  View  Search  Terminal  Help
ashep@aserver:~$ vnstat

                    rx      /      tx      /     total    /   estimated
eth0:
      Nov '10     2.39 MiB  /    4.52 MiB  /    6.91 MiB  /    6.00 MiB
        today     2.39 MiB  /    4.52 MiB  /    6.91 MiB  /      27 MiB

tun0:
      Nov '10       47 KiB  /     179 KiB  /     226 KiB  /       0 KiB
        today       47 KiB  /     179 KiB  /     226 KiB  /      --

tun1: Not enough data available yet.
ashep@aserver:~$
```

Сводная информация по конкретному интерфейсу:

```
vnstat -i eth0
```

```
File  Edit  View  Search  Terminal  Help
ashep@aserver:~$ vnstat -i eth0
Database updated: Sat Nov 27 05:13:23 2010

   eth0 since 11/27/10

          rx:  2.39 MiB      tx:  4.52 MiB      total:  6.91 MiB

   monthly
                     rx      |     tx      |    total    |   avg. rate
     ------------------------+-------------+-------------+---------------
       Nov '10     2.39 MiB |    4.52 MiB |    6.91 MiB |    0.03 kbit/s
     ------------------------+-------------+-------------+---------------
     estimated        2 MiB |       4 MiB |       6 MiB |

   daily
                     rx      |     tx      |    total    |   avg. rate
     ------------------------+-------------+-------------+---------------
         today     2.39 MiB |    4.52 MiB |    6.91 MiB |    3.01 kbit/s
     ------------------------+-------------+-------------+---------------
     estimated        9 MiB |      18 MiB |      27 MiB |
ashep@aserver:~$
```

Отдельный интерфейс, по часам: `vnstat -i eth0 -h`

```
File  Edit  View  Search  Terminal  Help
ashep@aserver:~$ vnstat -i eth0 -h
 eth0                                                          05:18
  ^                                                              t
  |                                                           t  t
  |                                                           t  t
  |                                                           t  t
  |                                                           t  t
  |                                                           t rt
  |                                                          rt rt
  |                                                          rt rt
  |                                                          rt rt
  |                                                          rt rt
 -+------------------------------------------------------------------>
  |   06 07 08 09 10 11 12 13 14 15 16 17 18 19 20 21 22 23 00 01 02 03 04 05

 h   rx (KiB)    tx (KiB)      h   rx (KiB)    tx (KiB)      h   rx (KiB)    tx (KiB)
06          0           0     14          0           0     22          0           0
07          0           0     15          0           0     23          0           0
08          0           0     16          0           0     00          0           0
09          0           0     17          0           0     01          0           0
10          0           0     18          0           0     02          0           0
11          0           0     19          0           0     03          0           0
12          0           0     20          0           0     04       1267        2553
13          0           0     21          0           0     05       1468        2688
ashep@aserver:~$
```

Отдельный интерфейс, по дням:

`vnstat -i eth0 -d`

Отдельный интерфейс, по неделям:

`vnstat -i eth0 -w`

Отдельный интерфейс, по месяцам:

`vnstat -i eth0 -m`

Live-режим

Также vnStat можно использовать для просмотра статистики в режиме реального времени:

`vnstat -i eth0 -l`

Установка и настройка VnStat под Ubuntu

В этот статье я познакомлю Вас с системой мониторинга трафика vnStat.

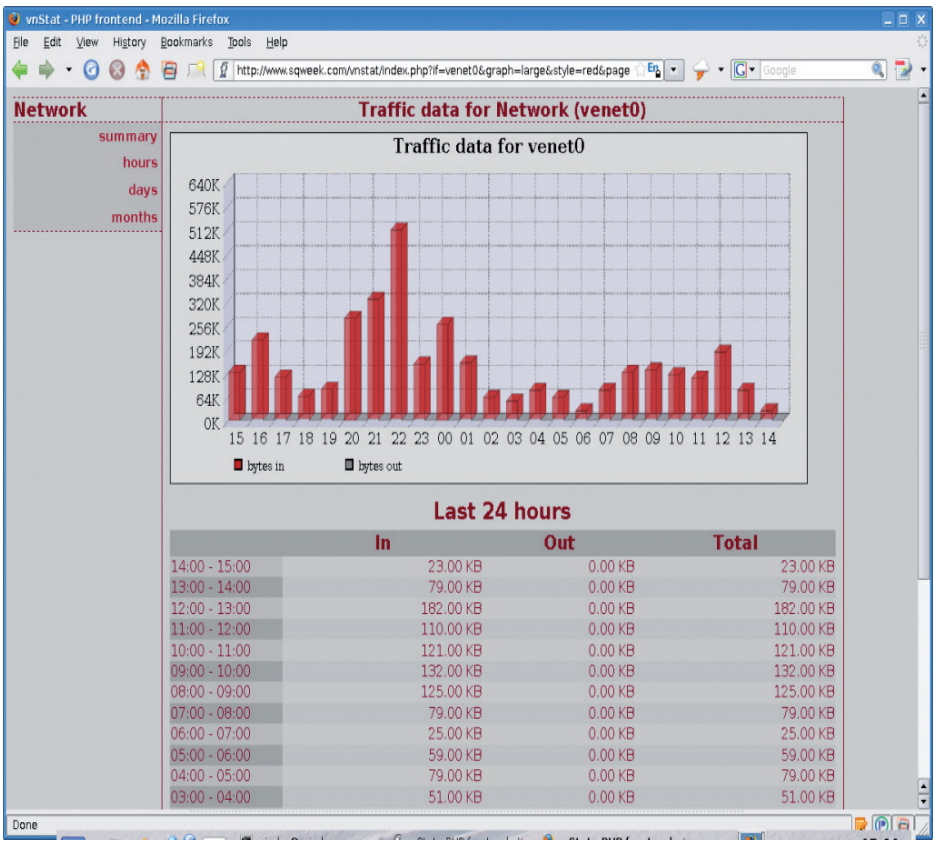
VnStat — простая консольная утилита, но можно к ней прикрутить Web-интерфейс и удобно смотреть сколько мегабайт было получено или отправлено на том или ином сетевом интерфейсе. Вот скриншот. Я аналогов этой утилиты не знаю, но уверен они точно есть.

Устаовка vnStat

Ставится одной командой:

```
sudo apt-get install vnstat
```

(От редакции. Данный обзор уместно дополнить авторским материалом из обзора утилиты vnstat. Мы приносим свои извинения за некоторый повтор материала, но, как кажется редакции, материалы дополняют друг друга.)



После установки vnStat готов к работе, без всяких дополнительных настроек зайдите в терминал и выполните команду man vnstat – вы увидите мануал по vnStat, а котором всё подробно написано. Но напишу пару подсказок, как всё работает.

Для того чтобы сказать vnstat'у, чтобы он отслеживал трафик через интерфейс eth1 надо выполнить команду:

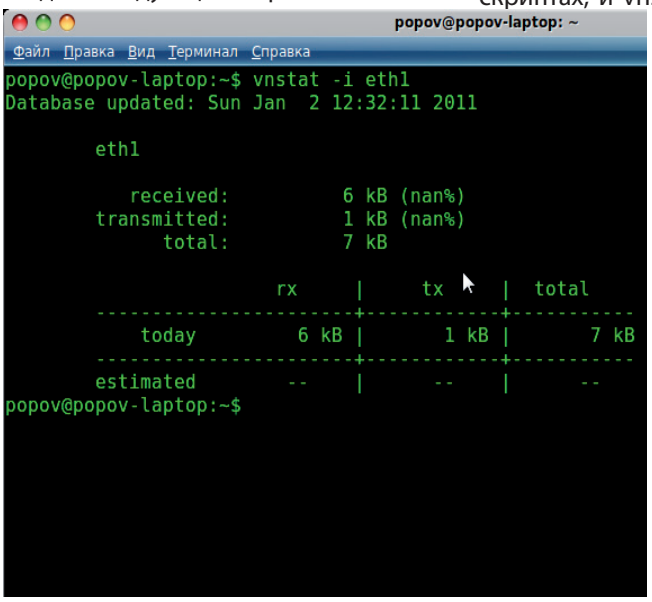
```
sudo vnstat -u -i eth1
```

Этой команды vnStat создаёт базу данных трафика для интерфейса eth1, лежать она будет в каталоге/var/lib/vnstat и называться просто eth1. Так же эта команда обновляет статистику по интерфейсу eth1 (-u значит update). Вообще статистика сама обновляется с какой-то периодичностью.

Для просмотра общей статистики трафика через интерфейс eth1 выполните команду:

```
vnstat -i eth1
```

результат этой команды будет выглядеть следующим образом.

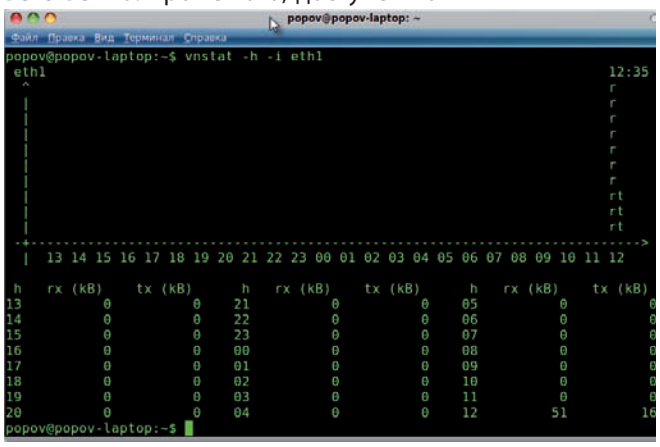


rx - исходящий трафик
tx - входящий трафик
estimated - ожидаемый

Команда

```
vnstat -h -i eth1
```

Выведет статистику по часам, выглядит это так



Также доступны параметры -d, -w, -m которые выводят статистику по дням, неделям и месяцам соответственно. Кроме того, доступен параметр -l, который выводит трафик в текущее время типа live. Можно составлять свои запросы к базе данных, – об этом можно почитать в мануале. Всё это конечно хорошо, но для удобного восприятия данных неплохо бы иметь графический интерфейс. Для vnstat таких интерфейсов существует несколько: gui web-интерфейс от производителей, написанный на CGI-скриптах, и vnstat PHP frontend – интерфейс, разработанный каким-то добрым человеком на PHP. Я пользуюсь интерфейсом на PHP, про него я и расскажу.

Скачать его можно отсюда www.sqweek.com/sqweek/index.php?p=1. Вы скачаете архив из php файлов, которые обеспечивают работу графического интерфейса. Создайте новый виртуальный хост в Apache или скопируйте эту папку в домашнюю папку уже существующего хоста и введите адрес в браузер, допустим http://site1.ru/nvStat. Вы попадёте на страницу статистики, но скорее всего вам будет показана нулевая статистика. Чтобы отобразить реальные данные, надо настроить конфигурацию интерфейса, что осуществляется через файл config.php, который вы скачали вместе с другими файлами vnstat PHP frontend.

В этом файле надо обратить внимание на эту часть

```
//
// There are two possible sources for vnstat data. If the $vnstat_bin
// variable is set then vnstat is called directly from the PHP script
// to get the interface data.
//
// The other option is to periodically dump the vnstat interface data to
// a file (e.g. by a cronjob). In that case the $vnstat_bin variable
// must be cleared and set $data_dir to the location where the dumps
// are stored. Dumps must be named 'vnstat_dump_$iface'.
//
// You can generate vnstat dumps with the command:
// vnstat --dumpdb -i $iface > /path/to/data_dir/vnstat_dump_$iface
//
$vnstat_bin = '';
$data_dir = './dumps';
```

Речь идет о том, что есть два способа получения данных web-интерфесом: первый – какой-то непонятный и второй – через дамп базы данных по интерфейсу (имеются ввиду те базы данных, которые вы создали при помощи команды sudo vnstat -u -i eth1, напомню – они лежат в /var/lib/vnstat). Для того, чтобы web-интерфейс получал данные о трафике через интерфейс eth1, надо сделать дамп базы eth1, я делаю это командой:

```
sudo vnstat –dumpdb -i eth1 > /var/lib/vnstat/dumps/vnstat_dump_eth1
```

Также нужно сразу указать в параметре $data_dir конфигурационного файла путь к директории с дампами базы данных в моём случае

$data_dir = '/var/lib/vnstat/dumps/';

Так как статистика всё время меняется, надо всё время обновлять этот дамп, с этим отлично справляется cron. Просто внесём эту команду в таблицу задач cron'а.

*/15 * * * * sudo vnstat -u -i eth1

*/15 * * * * sudo vnstat --dumpdb -i eth1 > /var/lib/vnstat/dumps/vnstat_dump_eth1

У меня ещё через cron обновляется сама база статистики каждые 15 минут. Всё теперь зайдите на сайт вашей статистики, там уже будет вся статистика.

www.tux-the-penguin.blogspot.com

Выводим информацию об устройствах

`lsdev` – информация об уже установленных устройствах. Для работы необходимо установить пакет procinfo: sudo apt-get install procinfo

`cat /proc/cpuinfo` – вывести полную информацию о модели процессора (частота, поддерживаемые инструкции и т. д.):

`cat /proc/meminfo` – вывести расширенную информацию о занимаемой оперативной памяти (MemTotal, MemFree, Buffers, Cached, SwapCached,HighTotal, HighFree, LowTotal и т. д.);

`grep SwapTotal /proc/meminfo` – показать размер раздела выделенного под swap;

`watch -n1 cat /proc/interrupts` – показать информацию о прерываниях;

`free -m` – информация о используемой и свободной ОЗУ и Swap-файле (-m указывает, что отображать нужно в Мб);

`lshal` – показать список всех устройств и их параметров;

`cat /proc/devices` – показать все устройства в системе (названия взяты из директории /proc/devices);

`lspci -tv` – показать обнаруженные PCI-устройства;

`lsusb -tv` – показать обнаруженные USB-устройства;

`sudo] dmidecode` – показать информацию о версии BIOS компьютера;

`gtf 1280 800 75` – выводит строку ModeLine для Вашего монитора на параметрах экрана 1280x800x75Hz;

www.mypingvin.org

Подключение PPPoE через Терминал

Наверное, каждый пользователь Ubuntu попадал в ситуацию, когда апплет сети не функционировал или отсутствовал вовсе. Конечно, всё это дело могло решиться перезагрузкой или перезапуском X.org, но мы, пользователи Linux, любим способы посложнее.

ATTENTION> Этот способ действует только для ADSL-модемов, подключаемых через порт Ethernet(Lan). Во время настройки модем должен быть ВКЛЮЧЁН!

1. В терминале вбиваем sudo pppoeconf и, соответственно, вводим запрашиваемый пароль.

2. Запустилась программа настройки соединения. Сейчас она вывела все доступные сетевые устройства. Нажимаем ДА.

3. Сейчас программа ищет ваш модем. Это будет занимать до 1 минуты.

4. На этом шаге программа спрашивает разрешение на использование опций defaultroute и nodetach. Эти опции ничем не вредны, а иногда и полезны, так что жмём ДА.

5. Тут вводим свой логин, выданный провайдером, например, pro_1234567.

6. А тут пароль, тоже выданный провайдером.

ATTENTION> Будьте внимательны при вводе. В случае ошибки, заново запускаем программу pppoeconf.

7. Здесь спрашивают про сохранение полученных DNS адресов. Лучше пользоваться этой опцией. Нажимаем ДА.

8. Тут нас спросят про ограничение MSS 1452 байтами. Лучше включить ограничение т.к. это позволяет избежать проблем с соединением.

9. На вопрос устанавливать ли соединение при запуске отвечаем как вам нравится (лично я ответил ДА, просто так мне удобнее).

10. Установить соединение сейчас? Конечно же, ДА!

ATTENTION> Разорвать соединение можно командой sudo poff dsl-provider, а подключиться sudo pon dsl-provider.

Теперь запускаем браузер и проверяем наличие соединения! Приятного интернет-сёрфинга!

Роман «MAGNUM» Баранов,
magnumcity@gmail.com,
Россия, Амурская обл., г. Свободный

Как раздать интернет через wi-fi в Ubuntu

1. Устанавливаем пакеты:

```
sudo apt-get install hostapd libpam-radius-auth
```

2. Идем в Network-Manager, выбираем вкладку «Беспроводные сети». Жмем добавить.

Имя соединения: На ваш выбор.

SSID: Любое какое хотите, чтобы не путаться, лучше то же, что и «Имя соединения».

Режим: Специальный.

mtu автоматически.

Остальные строчки на этой вкладке окна добавления нового соединения оставляем без изменений.

Жмем вкладку Ipv4.

Метод: Общий с другими компьютерами.

Идем на вкладку «Безопасность».

Выбираем тип шифрования, я выбрал WEP 40/128, другой тип шифрования у меня установить почему-то не получилось.

Вводите Ключ – 5 символов, которые нужно запомнить чтобы ввести их при подключении на другом устройстве. Жмете применить.

Всё, теперь ищите эту сеть на другом ноуте и подключайтесь.

www.ubuntu-repository.blogspot.com

Как заставить работать на Ubuntu принтеры Canon из серии LBP

Эти принтера хорошо распространены из-за своей неубиваемости и простой заправки картриджей. Но вот проблема - драйвера под линукс написаны довольно таки криво и многие, встретившись с проблеммой их установки, не знают что делать. Мне тоже пришлось столкнуться с такой проблеммой, так как на офисе и дома у меня стоят принтера Canon LBP 2900. Покопавшись в интернете и испробовав кучу различных инструкций и советов, я натолкнулся на готовый пакет, который ребята собрали для i386 и 64-битных систем. После установки пакета с этими принтерами больше нет проблемм.

Этот пакет подходит для таких принтеров: LBP-1120, LBP-1210, LBP2900, LBP3000, LBP3010, LBP3018, LBP3050, LBP3100, LBP3108, LBP3150, LBP3200, LBP3210, LBP3250, LBP3300, LBP3310, LBP3500, LBP5000, LBP5050, LBP5100, LBP5300, LBP7200C.

Инструкция очень проста.

1. Качаем пакет по этой сылке: http://codebin.cotescu.com/canon/lbp_driver/CanonCAPTdriver.tar.gz

2. Распаковываем в домашний каталог, советую каталог переименовать во что-то простое, например в canon.

3. Заходим в этот каталог:

```
user@home:~$ cd canon
```

4.И набираем волшебную команду, которая поставит драйвера (указывая свою модель принтера, в моем случае это был LBP2900):

```
user@home:~/canon$ sudo ./canonLBP_install.sh LBP2900
```

5. Перезагружаемся и в Администрирование->Печать выбираем по умолчанию LBP2900 или какой там у вас другой из этого списка.

Теперь можете наслаждаться печатью. Есть еще несколько нюансов, которые нужно учесть. Если во время печати закончилась бумага, кнопочка вам не поможет. Для этого я написал небольшой скриптик который перезапускает демон ccpd:

```
sudo /etc/init.d/ccpd restart
```

Но печать после этого придется запустить заново

Дмитрий Бутолин
www.ubuntu.dp.ua

D-Link DWA-525 на Ubuntu 10.04

Вот решил избавится от проводов и повтыкать дома в компы вайфайки, так получилось, что купил D-Link DWA-525. На Ubuntu сетевушка не ожила и пришлось погуглить что-бы найти как ее завести. Нагуглил кучу воплей и проклятий на D-Link, что сетевушка отстой и дрова корявые, что ни у кого толком не работает, хотя были и такие счастливчики, которые хвастались что все у них работает, но почему-то обьяснить как это случилось, никто не смог, пока не нашел простой способ новичка в линуксе, который без всяких выпадений на умняк, кучи умных слов и посыланий читать мануалы, объяснил что надо сделать...

Итак начнем.

1. Выколупываем сетевушку с компа, в смысле - выкручиваем,))).

2. Качаем драйвера с оффсайта сетевушки: http://ftp.dlink.ru/pub/Wireless/DWA-525/Drivers/Linux/ и распаковуем их в домашний каталог.

3. Получаем каталог с километровым именем - переименуем его например в 11

4. Запускаем Терминал - автоматом терминал обращается в ваш домашний каталог - home (так его назову, у вас он может называется именем вашего компа), в котором находится папочка с драйверами, она то как раз родимая нам и нужна,))). Ну что ж, зайдем в нее - пишем в терминале:

```
>user@home:~$ cd 11
```

5. Набираем:

```
>user@home:~/11$ make
```

нажимаем Enter и видим что что-то начало в терминалке происходить, побежали буквочки, что-то оно там шаманит, колдует... сидим и ждем когда доколдует,))))).

6. Оно там что-то сотворило и остановилось, что-то сделало,))). Мы в ответ набираем ему еще одни волшебные слова:

```
>user@home:~/11$ sudo make install
```

нажимаем Enter, нас попросило ввести пароль - мы его вводим, нажимаем Enter, видим что опять там что-то пишет, колдует. Когда все это дело прошло - вырубаем комп, втыкаем карточку в него, включаем комп и .....

Чудо!!! Сетевушка ищет вайфайную сетку,))).

Дмитрий Бутолин
www.ubuntu.dp.ua

Принтеры hewlett packard в Ubuntu

Не работает убунту из коробки с принтерами HP. Чтобы корректно работали нужно следующее. В терминале:

```
sudo apt-get install hplip-gui
```

это поставит графический интерфейс со всем необходимым.

Потом нужно запустить:

```
hp-setup
```

Если сначала запустить hp-setup, то возможны ошибки типа:

error: hp-setup requires GUI support (try running with –qt3). Also, try using interactive (-i) mode

или

warning: Qt/PyQt 4 initialization failed или warning: Qt/PyQt 4 initialization failed

www.usefree.com.ua

Установка Canon PIXMA iP1800 в Ubuntu

В процессе перевода одной из фирм на Linux, мне довелось подключать и настраивать струйный принтер PIXMA iP1800 к одному из офисных компьютеров, под управлением Ubuntu Linux. Этот процесс не был тривиальным из-за отсутствия драйверов в Ubuntu.

При подключении PIXMA iP1800 к Ubuntu, в левом верхнем окне уведомлений появилось сообщение о том, что iP1800 был опознан Ubuntu, но не может работать из-за отсутствия драйверов.

Вот действия, которые позволили запустить в работу вышеуказанный принтер:

Скачиваем драйвер для работы PIXMA iP1800 в Ubuntu:

```
wget http://linuxway.ru/
wp-content/uploads/2010/01/
cnijfilter-common_2.70-2_
i386.deb
```

```
wget http://linuxway.
ru/wp-content/
uploads/2010/01/cnijfilter-
ip1800series_2.70-2_i386.
deb
```

Затем устанавливаем скачанные пакеты:

```
sudo dpkg –install cnijfilter-
common_2.70-2_i386.deb
sudo dpkg –install cnijfilter-
ip1800series_2.70-2_i386.
deb
```

После успешной установки пакетов, переходим в Система – Администрирование – Печать, выбираем в окне настройки печати принтер iP1800 и, нажав правой кнопкой мыши, выбираем в меню “Изменить”. При выборе месторасположения драйвера, выберите файл:

/usr/share/cups/model/custom/canonip1800.ppd

Затем в меню настроек нажмите “Применить”.

Это все, после указанных выше действий, PIXMA iP1800 будет готов к работе в Ubuntu.

www.linuxway.ru

Установка принтера Canon iP1500 в Ubuntu 10.10

Мой старый принтер Canon iP1500 под 10-ой как всегда не работал, пришлось поковыряться, но в итоге он таки заработал.

Что бы заработал Canon iP1500 в Ubuntu 10.10: качаем и ставим пересобранные пакеты для Ubuntu кроме pstocanonbj (он хочет cupsys и конфликтует с имеющейся версией cups, как побороть этот выбрык читайте ниже)

```
sudo dpkg -i *.deb
```

http://www.filippoamaduzzi.com/blog/wp-content/uploads/2009/06/bjfilter-2_5_1-1_i386.deb

http://www.filippoamaduzzi.com/blog/wp-content/uploads/2009/06/libcnbj-2_5_0-1_i386.deb

http://www.filippoamaduzzi.com/blog/wp-content/uploads/2009/06/pstocanonbj_3_3-1_i386.deb

последний pstocanonbj потребует по зависимости libcupsys2 - он есть на launchpad

http://launchpadlibrarian.net/25606454/libcupsys2_1.3.9-17ubuntu1_all.deb

качаем, ставим.

Или качаем все пакеты одним архивом

http://ifile.it/c4yzr62/CanoniP1500-Linux.tar.gz

Распаковываем пакет pstocanonbj

```
dpkg --extract
pstocanonbj_3.3-1_i386.deb
/home/mk/tmp/
```

копируем файл pstocanonbj из той папки, в которую распаковано дерево deb пакета, в /usr/lib/cups меняем владельца и группу на рута

```
sudo chown root:root
pstocanonbj
```

копируем файл из той папки, в которую распаковано дерево deb пакета, в canonpixmaip1500.ppd в/usr/share/ppd меняем владельца и группу на рута

```
sudo chown root:root
canonpixmaip1500.ppd
```

Перезапускаем сервис печати (или перезагружаемся):

```
sudo /etc/init.d/cups
restart
```

После установки в меню создаем новый принтер через веб-интерфейс Cups http://localhost:631/

Документы печатает без проблем с разрешением 600 dpi, в 10.10 появилось разрешение 1200, но принтер его не поддерживает и так же появилась опция качества печати «High».

www.surrender-zen-way.blogspot.com

ПРОГРАММИРОВАНИЕ НА BASH

Часть I. Краткое введение

Данное введение в программирование на bash прельстило меня своей краткостью и содержательностью. В то же время я изменил некоторые примеры, потому что они делали слегка не то, что ожидается новичками. Начинающим текст будет полезен как отправная точка для начала написания скриптов. Опытным – как справочник. Удачного чтения!

Выражаю также благодарность Владимиру Черному (начальнику отдела образовательных проектов ALT Linux) за внесенные в текст правки.

Тема программирования на bash из разряда тех, которые могут быть рассмотрены и в пару, и в сотни страниц. Гарольд Родригес (Harold Rodriguez) объясняет эту тему в приведенном ниже руководстве из двух частей. Его прекрасный и яркий стиль позволил ему охватить все существенные черты программирования на bash буквально на нескольких страницах.

Если вы никогда не программировали на bash ранее – сейчас самое время начать. Даже если у вас мало знаний о том, что такое bash, вы вполне можете посмотреть на множество интересных скриптов, разбираемых Гарольдом.

ВВЕДЕНИЕ

Подобно остальным оболочкам, доступным в Linux, Bourne Again shell (bash) является не только, собственно, командной оболочкой, но и языком для написания сценариев (скриптов) (слова «сценарий» и «скрипт» обычно являются синонимами – прим. перев.). Скрипты позволяют в полной мере использовать возможности оболочки и автоматизировать множество задач, которые иначе потребуют для своего выполнения ввода множества команд. Многие программы, работающие внутри вашего компьютера с Linux – это скрипты. Если вы захотите узнать как они работают или изменить их, важно понимать их синтаксис и семантику. Кроме того, понимая язык bash, вы сможете писать свои собственные программы, чтобы выполнять разные задачи теми способами, которые выберете сами.

ТАК ВСЕ-ТАКИ ПРОГРАММИРОВАНИЕ (PROGRAMMING) ИЛИ НАПИСАНИЕ СКРИПТОВ (SCRIPTING)?

Новичков в программировании, как правило, озадачивает разница между, собственно, программированием и языками скриптов. Программы, написанные на каких-то языках программирования, обычно гораздо более мощные по возможностям и работают намного быстрее, чем программы, написанные на языках сценариев. Примеры языков программирования – C, C ++ и Java. Создание программы на каком-либо языке программирования обычно начинается с написания исходного кода (текстовый файл, содержащий инструкции о том, как будет работать окончательная программа), затем его необходимо скомпилировать (собрать) в исполняемый файл. Этот исполняемый файл не так легко переносить между различными операционными системами. Например, если вы напишете программу на C для Linux, вы не сможете запустить ее в Windows. Чтобы сделать это, вам придется перекомпилировать исходный код под Windows. Написание скрипта также начинается с написания исходного кода, который не компилируется в исполняемый файл. Вместо этого интерпретатор оболочки последовательно читает инструкции в файле исходного кода и выполняет их. К сожалению, поскольку интерпретатор должен читать каждую инструкцию, скорость исполнения скрипта обычно медленнее (намного медленнее – прим. перев.), чем у скомпилированной программы. Основным преимуществом скриптов является то, что вы можете легко переносить исходный файл в любую операционную систему и просто запускать их (естественно при наличии интерпретатора для этой операционной системы – прим. перев.).

Bash – это язык сценариев. Он отлично подходит для написания небольших программ, но если вы планируете

создавать какие-то мощные приложения, предпочтительнее выбрать для этого какой-нибудь язык программирования. Другие примеры скриптовых языков Perl, Lisp, и Tcl.

ЧТО НУЖНО ЗНАТЬ ДЛЯ НАПИСАНИЯ СВОИХ СКРИПТОВ?

Для этого необходимо знание основных команд Linux. Например, вы должны знать, как копировать, перемещать и создавать новые файлы. Обязательно умение использовать какой-нибудь текстовый редактор. Существуют три основных текстовых редактора в Linux: vi, emacs и pico (автор еще забыл nano, который лучше всего подходит начинающим, если не учитывать еще и mcedit. – Прим. перев.). Если вы не знакомы с vi или emacs, используйте pico или другой простой в использовании текстовый редактор.

ВНИМАНИЕ!!! ВНИМАНИЕ!!! ВНИМАНИЕ!!!

Не следует учиться программировать на bash из-под пользователя root! В противном случае – может случиться все что угодно! Я не буду нести никакой ответственности, если вы случайно допустите ошибку и испортите вашу систему. Вы предупреждены! Используйте учетную запись обычного пользователя без каких-либо привилегий. Вы можете даже создать нового пользователя только для обучения написанию сценариев. Таким образом, худшее, что произойдет в данном случае — это исчезновение данных в каталоге этого пользователя.

ВАША ПЕРВАЯ ПРОГРАММА НА BASH

Нашей первой программой будет классическая «Hello World». Конечно, если уже вы программировали раньше, то, должно быть, устали от таких примеров. Однако это – традиция и кто я такой, чтобы менять ее? Программа «Hello World» просто выводит слова «Hello World» на экран. Запустите текстовый редактор и наберите в нем следующее:

```
#!/bin/bash
echo «Hello World»
```

Первая строка сообщает Linux использовать интерпретатор bash для запуска этого скрипта. В этом случае bash находится в каталоге /bin. Если у вас bash находится где-то еще, сделайте соответствующие изменения в данной строке. Явное указание интерпретатора очень важно, удостоверьтесь еще раз, что указали его, поскольку данная строка говорит Linux какой именно интерпретатор нужно использовать для выполнения инструкций в скрипте. Следующее, что нужно сделать, это сохранить скрипт. Назовем его hello.sh. После этого вам нужно сделать скрипт исполняемым:

```
chmod u+x hello.sh
```

Если вы не понимаете, что делает эта команда, прочтите справочную страницу команды chmod:

```
man chmod
```

Как только это будет сделано, вы сможете запустить программу, просто набрав ее название:

```
./hello.sh
Hello World
```

Получилось! Это ваша первая программа! И хотя она скучная и не делает ничего полезного, она показывает как именно все работает. Просто запомните эту простую последовательность действий: напишите код, сохраните файл, сделайте его исполняемым с помощью chmod и запустите.

КОМАНДЫ, КОМАНДЫ И КОМАНДЫ

Что именно делает ваша первая программа? Она печатает на экран слова «Hello World». Каким образом она это делает? Она использует команды. В нашей программе мы написали только одну команду – echo «Hello World». Что именно тут команда? echo. Эта программа принимает один аргумент и печатает его на экран.

Аргументом является все, что следует после ввода названия программы. В нашем случае «Hello World» это и есть аргумент, переданный команде echo. При вводе команды ls /home/, аргументом команды ls является /home. Ну и что это все означает? А означает это то, что если у вас есть программа, которая принимает какой-то аргумент и выводит что-то на экран, вы можете использовать ее вместо echo. Предположим, что у нас есть программа под названием foo. Эта программа будет принимать один аргумент (строку из слов) и печатать их на экран. Тогда мы можем переписать нашу программу вот так:

```
#!/bin/bash
foo «Hello World»
```

Сохраните ее, сделайте исполняемой и перезапустите ее (примечание для новичков - этот пример работать не будет. – Прим. перев.):

```
./hello
Hello World
```

Точно такой же результат. Использовался ли тут какой-то уникальный код? Нет. Написали ли мы какую-то программу? Нет, если только вы не являетесь автором программы echo. Все, что вы сделали – просто встроили программу echo в наш скрипт и снабдили ее аргументом. Примером альтернативы использования команды echo

в реальном программировании является команда printf, которая имеет больше возможностей, если вы знакомы с программированием на C. Ну и на самом деле, точно такой же результат можно было бы получить и без написания скрипта:

```
echo «Hello World”
Hello World
```

Написание скриптов на bash предлагает широкий спектр возможностей и этому легко научиться. Как вы только что могли видеть, здесь используются разные команды Linux, чтобы писать собственные скрипты. Ваша программа-оболочка представляет собой несколько других программ, собранных вместе для выполнения какой-либо задачи.

ДРУГИЕ ПОЛЕЗНЫЕ ПРОГРАММЫ

Сейчас мы напишем программу, которая переместит все файлы в каталог, удалит его вместе с содержимым, а затем создаст это каталог заново. Это может быть сделано с помощью следующих команд (В примере, приведенном в оригинале автор показывает, что не зря рекомендовал делать приведенные упражнения под специально созданным пользователем. Результатом выполнения данной последовательности команд будет чистый каталог, в котором вы работаете. Скорее всего это будет ваша домашняя директория. Поэтому, если вы НЕ хотите удаления всех файлов в ней – НЕ выполняйте команды из оригинала статьи. А лучше последуйте совету автора и создайте отдельного пользователя специально для тренировки написания скриптов. Этот пример я немного расширил и теперь он не такой опасный. – Прим. перев.):

```
touch file1
mkdir trash
mv file1 trash
rm -rf trash
mkdir trash
```

Вместо того, чтобы вводить это все в интерактивном режиме, напишем скрипт, выполняющий эти команды:

```
#!/bin/bash
touch file1
mkdir trash
mv file1 trash
rm -rf trash
mkdir trash
echo “Файл удален!”
```

Сохраните его под именем clean.sh и теперь все, что нужно сделать – запустить его. Он переместит все файлы в каталог, удалит его, создаст заново каталог и даже напечатает сообщение об удалении файлов. Запомните, если вы обнаружите, что регулярно делаете нечто требующее набора одной и той же последовательности команд – это вполне можно автоматизировать написанием скрипта.

КОММЕНТАРИИ

Комментарии помогают сделать ваш код более читабельным. Они не влияют на то, что выводит программа. Они написаны специально для того, чтобы вы их прочли. Все комментарии в Bash начинаются с хэш-символа #, за исключением первой строки (#!/bin/bash), имеющей специальное назначение. Первая строка – не комментарий. Возьмем для примера следующий код:

```
#!/bin/bash
# Эта программа считает от 1 до 10:
for i in 1 2 3 4 5 6 7 8 9 10; do
echo $i
done
```

Даже если вы пока не понимаете скрипты на Bash, вы сразу же поймете, что делает приведенный выше пример, благодаря комментарию. Комментирование кода – хорошая практика. Со временем вы поймете, что, если вам нужно будет поддерживать ваши скрипты, то при наличии комментированного кода – делать это станет легче.

ПЕРЕМЕННЫЕ

Переменные это просто «контейнеры», которые содержат некоторые значения. Создавать их нужно по многим причинам. Вам нужно будет как-то сохранять вводимые пользователем данные, аргументы или числовые величины. Например:

```
#!/bin/bash
x=12
echo ”Значение переменной x - $x”
```

Здесь мы присвоили переменной x значение 12. Строка echo “Значение переменной x - $x” напечатает текущее значение x. При определении переменной не допускается наличие каких-то пробелов между именем переменной и оператором присваивания: «=». Синтаксис следующий:

имя_переменной=ее_значение

Обращение к переменным выполняется с помощью префикса «$» перед именем переменной. Именно таким образом мы получаем доступ к значению переменной x с помощью команды echo $x.

Есть два типа переменных – локальные и переменные окружения (среды). Переменные окружения устанавлива-

ются системой и имеют специальной назначение. Обычно их значение может быть выведено с помощью команды echo. Например, если ввести:

```
echo $SHELL
/bin/bash
```

Вы получили имя оболочки, запущенной в данный момент. Переменные среды задаются в файле /etc/profile и в ~/.bash_profile. Команда echo может применяться для проверки текущего значения переменной.

Примечание: задание переменных среды подробно описано в этой статье – «Как задавать переменные среды» – http://learnbyexamples.org/linux/linux-tip-how-to-set-shell-environment-variables-bash-shell.html. В статье также описаны некоторые особенности оболочки Bash.

Если у вас все еще возникают проблемы с пониманием того, зачем нужно использовать переменные, приведем пример:

```
#!/bin/bash
echo «Значение x – 12».
echo «У меня есть 12 карандашей».
echo «Он сказал мне, что значение x равно 12».
echo «Мне 12 лет.»
echo «Как получилось, что значение x равно 12?»
```

Хорошо, теперь предположим, что вы решите поменять значение x на 8 вместо 12. Что для этого нужно сделать? Вы должны изменить все строки кода, в которых говорится, что x равно 12. Но погодите... Есть другие строки кода, где упоминается это число, поэтому простую автозамену использовать не получится. Теперь приведем аналогичный пример, только с использованием переменных:

```
#!/bin/bash
x=12 # задаем переменной x значение 12
echo «Значение x = $x»
echo «У меня есть 12 карандашей»
echo «Он сказал мне, что значение x равно $x»
echo «Мне 12 лет»
echo «Как получилось, что значение x равно $x?»
```

Здесь мы видим, что $x выводит текущее значение переменной x равное 12. Поэтому теперь, если вы хотите задать новое значение x равное 8, то все что вам нужно сделать, это изменить одну строчку с x=12 на x=8, и в выводе все строки с упоминанием x также изменяться. Поэтому вам не нужно руками модифицировать остальные строки. Как вы увидите позже, переменные имеют и другие способы применения.

УПРАВЛЯЮЩИЕ ОПЕРАТОРЫ

Управляющие операторы делают вашу программу компактнее и позволяют ей принимать решения. И, что еще более важно, они позволяют нам выполнять проверку на наличие ошибок. До сих пор все, что мы сделали, это писали скрипты, которые просто исполняют набор инструкций в файле. Например:

```
#!/bin/bash
cp /etc/foo .
echo ”Готово”
```

Это небольшой скрипт, назовем его bar.sh, копирует файл с именем /etc/foo в текущий каталог и выводит «Готово» на экране. Эта программа будет работать при одном условии - файл /etc/foo должен существовать. В противном случае вот что произойдет:

```
./bar.sh
cp: /etc/foo: No such file or directory
Готово
```

Таким образом, как вы видите, существует проблема. Не у каждого, кто будет запускать вашу программу, будет файл /etc/foo. Наверное, было бы лучше, если бы ваша программа сначала проверяла наличие данного файла, а затем при положительном ответе – выполняла бы копирование, в противном случае – просто бы завершала работу. В псевдо-коде это выглядит так:

```
если /etc/foo существует, то
скопировать /etc/foo в текущую директорию
напечатать «Готово» на экране
в противном случае,
напечатать на экране «Этот файл не существует»
выход
```

Можно ли это сделать в Bash? Конечно! Набор управляющих операторов Bash включает в себя: if, while, until, for и case. Каждый из этих операторов является парным, то есть начинается он одним тегом и заканчивается другим. Например, если условный оператор if начинается с if и заканчивается fi. Управляющие операторы cp /etc/foo . – это не отдельные программы в системе, они встроены в bash.

```
if … else … elif … fi
```

Это один из наиболее распространенных операторов. Он позволяет программе принимать решения следующим образом – «если условие верно – делаем одно, если нет

– делаем что-то другое». Чтобы эффективно его использовать, сначала нужно научиться пользоваться командой test. Эта программа выполняет проверку условия (например, существует ли файл, есть ли необходимые права доступа). Вот переписанный вариант bar.sh :

```
#!/bin/bash
if test -f /etc/foo
then
```

Файл существует, копируем его и печатаем сообщение на экране

```
cp /etc/foo .
echo «Готово».
```

else # Файл не существует, поэтому мы печатаем сообщение

```
#и завершаем работу
echo «Этот файл не существует.»
exit
fi
```

Обратите внимание на переводы строки после then и else. Они не являются обязательными, но делают чтение кода гораздо более простым в том смысле, что делают логику программы более наглядной. Теперь запустите программу. Если у вас есть файл /etc/foo – он будет скопирован, в противном случае будет напечатано сообщение об ошибке. Команда test проверяет существование файла. Ключ -f проверяет, является ли аргумент обычным файлом. Ниже приведен список опций test (Не стоит пытаться запомнить их все, т.к. это все равно нереально. Его всегда можно посмотреть в руководстве команды test – man test. Прим. перев.):

КЛЮЧИ КОМАНДЫ TEST:

`-d` проверяет наличие файла и то, что он является каталогом
`-e` проверяет существование файла
`-f` проверяет наличие файла и то, что это обычный файл
`-g` проверяет наличие у файла SGID-бита
`-r` проверяет наличие файла и то, что он доступен на чтение
`-s` проверяет наличие файла и то, что его размер больше нуля
`-u` проверяет наличие у файла SUID-бита
`-w` проверяет наличие файла и то, что он доступен на запись
`-x` проверяет наличие файла и наличие у него прав на запуск

Оператор else используется, когда вы хотите, чтобы ваша программа еще что-то делала, если первое условие не выполняется. Существует также оператор elif, который может использоваться вместо еще одного if. elif означает «else if». Он используется, когда первое условие не выполняется, и вы хотите проверить еще одно условие.

Если вам не нравится приведенная форма записи if и test, есть сокращенный вариант.

Например, код:

```
if test -f /etc/foo
then
```

Можно записать вот так:

```
if [ -f /etc/foo ]; then
```

Квадратные скобки – это еще один вариант записи test. Если у вас есть опыт в программировании на C, этот синтаксис для вас может быть более удобным. Обратите внимание на наличие пробелов до и после каждой из скобок (Наличие пробелов объясняется просто: открывающая квадратная скобка – это команда оболочки. В этом можно легко убедиться набрав в консоли команду which [. А раз это отдельная команда, то ее нужно отделить пробелами от остальных опций. – Прим. перев.). Точка с запятой: «;» говорит оболочке о завершении одного оператора и начале следующего. Все, что находится после этого символа будет работать так, как будто находится в отдельной строке. Это делает код более удобным для чтения и, естественно, такая запись необязательна. Если вы предпочитаете другой вариант записи – then можно сразу поместить в другой строке.

Если вы используете переменные – их нужно помещать в кавычки. Например:

```
if [ «$name» -eq 5 ]; then
```

оператор -eq будет объяснен далее в этой статье.

while … do … done

Оператор while используется для организации циклов. Он работает так «пока (while) условие истинно, делать что-то». Рассмотрим это на примере:

```
#!/bin/bash
while true; do
echo «Нажмите CTRL-C для выхода»
done
```

true – это тоже программа. Единственное, что она тут делает – это запускает тело цикла снова и снова. Использование true считается медленным, потому что ваш скрипт должен запускать ее раз за разом. Можно использовать альтернативный вариант:

```
#!/bin/bash
while :; do
```

```
echo «Нажмите CTRL-C для выхода»
done
```

Это позволяет добиться точно такого же эффекта, но быстрее, потому что «:» – это встроенная функция bash. Единственное отличие состоит в принесении в жертву читабельности кода. Используйте из приведенных вариантов тот, который вам нравится больше. Ниже приведен гораздо более полезный вариант использования переменных:

```
#!/bin/bash
x=0;
while [ «$x» -le 10 ]; do
echo «Текущее значение x: $ x»
# Увеличиваем значение x:
x=$(expr $x + 1)
sleep 1
done
```

Здесь мы используем для проверки состояния переменной x запись с квадратными скобками. Опция -le означает «меньше или равно (less or equal)». Говоря обычным языком приведенный код говорит: «пока (while) x меньше или равен 10, выводить на экран текущее значение x, после чего добавлять к текущему значению x единицу». Оператор sleep 1 приостанавливает выполнение программы на одну секунду.

Ниже приведен список возможных операций сравнения целых чисел (полный список приведен в man test. – Прим. перев.):

x `-eq` y – x = y (equal)

x `-ne` y – x не равен y (not equal)

x `-gt` y – x больше либо равен y (greater than)

x `-lt` y – x меньше либо равен y (lesser than)

ОПЕРАТОРЫ СРАВНЕНИЯ СТРОК:

x = y – строка x идентична y
x != y – строка x не совпадает y
`-n` x – выражение истинно, если строка x ненулевой длины
`-z` x – выражение истинно, если строка x имеет нулевую длину

Скрипт, приведенный выше, нетрудно понять, за исключением, может быть, только этой строки:

```
x=$(expr $x + 1)
```

Комментарий приведенный выше он говорит нам, что он увеличивает x на 1. Но что означает запись $ (...)? Это переменная? Нет. На самом деле это способ сказать оболочке, что вы хотите запустить команду expr $x + 1, и присвоить результат ее выполнения переменной x. Любая команда, заключенная в $ (...) будет выполняться:

```
#!/bin/bash
me=$(whoami)
echo «Привет! Меня зовут $me»
```

Попробуйте сделать приведенный пример, и вы поймете, что я имею в виду. Приведенный выше код можно было бы сократить без каких-либо потерь вот так:

```
#!/bin/bash
echo «Привет! Меня зовут $(whoami)»
```

Вы сами можете выбрать, какая из записей вам ближе и понятнее. Существует и другой способ для выполнения команд или передачи результата их выполнения переменной. Как это сделать - будет объяснено позже. Пока используйте запись вида $(...).

until … do … done

Оператор until применяется аналогично приведенному выше while. Разница лишь в том, что условие работает наоборот. Цикл while выполняется до тех пор, пока условие истинно. Цикл until – до тех пор, пока условие не станет истинным. Например:

```
#!/bin/bash
x=0
until [ «$x» -ge 10 ]; do
echo «Текущее значение x равно $ x»
x=$(expr $x + 1)
sleep 1
done
```

Эта часть кода выглядит знакомой. Попробуйте ее набрать и посмотреть, что он делает. Приведенный цикл будет работать, пока x не станет больше или равен 10. Когда величина x достигнет значения 10, цикл остановится. Таким образом, последнее значение напечатанное значение x будет 9.

for … in … do … done

Цикл for используется, когда вам надо перебрать несколько значений переменной. Например, вы можете написать небольшую программу, которая печатает 10 точек:

```
#!/bin/bash
echo -n «Проверка системы на наличие ошибок»
for dots in 1 2 3 4 5 6 7 8 9 10; do
echo -n «.»
done
echo «Ошибок не обнаружено»
```

Опция -n команды echo предотвращает автоматический перевод строки. Попробуйте один раз вариант с -n и вариант без этой опции, чтобы понять, что я имею в виду.

Переменная dots последовательно принимает значения от 1 до 10 и одновременно скрипт печатает на экране точку.

Приведенный дальше пример показывает, что я имею в виду под выражением «переменная последовательно принимает несколько значений»:

```
#!/bin/bash
for x in paper pencil pen; do
echo «значение переменной x равно $x»
sleep 1
done
```

При запуске программы, вы видите, что x сначала имеет значение «pencil», а затем она принимает значение «pen». Когда у переменной заканчивается список возможных значений, цикл завершается.

Ниже приведен гораздо более полезный пример. Этот скрипт добавляет расширение .html для всех файлов в текущей директории (Этот скрипт действительно так и поступает и вам возможно это не нужно. Поэтому все-таки создайте отдельного пользователя, если вы еще до сих пор этого не сделали, и экспериментируйте под ним. – Прим. перев.):

```
#!/bin/bash
for file in *; do
echo «Добавляем расширение .html для файла $file …»
mv $file $file.html
sleep 1
done
```

Символ * имеет специальное значение, которое в данном случае означает «все в текущем каталоге», т.е. все файлы в каталоге. Переменная file последовательно принимает значения, соответствующие именам файлов в текущем каталоге. Затем используется программа mv для переименования файла в файл с расширением .html.

case … in … esac

Оператор case очень похож на if. Он отлично подходит для тех случаев, когда нужно проверить несколько условий и вы не хотите для этого использовать несколько вложенных операторов if. Поясним на примере:

```
#!/bin/bash
x=5 # инициализируем x значением 5
# проверяем значение x:
case $x in
0) echo «значение x равно 0»
;;
5) echo «значение x равно 5»
;;
9) echo «значение x равно 9»
;;
*) echo «значение неизвестно»
;;
esac
```

Оператор case проверяет переменную x на равенство трем значениям. В приведенном примере, он сначала проверит, равен ли x нулю 0, затем равен ли он 5, затем равен ли он 9. И, если все проверки завершились неудачно, скрипт выведет сообщение, что значение x определить не получилось. Помните, что «*» означает «все», а в этом случае, «любое другое значение, помимо указанных явно». Если x имеет любое другое значение, отличное от 0, 5 или 9, то это значение попадает во категорию «*». При использовании case каждое условие должно заканчиваться двумя точками с запятой.

Зачем нужно использовать case, когда вы можно использовать if? Ниже приведен пример эквивалентного скрипта, написанного с использованием if. Решение о том, что быстрее написать и легче прочесть, предлагается принять самостоятельно ;) :

```
#!/bin/bash
x=5 # инициализируем x значением 5
if [ «$x» -eq 0 ]; then
echo «Значение x равно 0»
elif [ «$x» -eq 5 ]; then
echo «значение x равно 5»
elif [ «$x» -eq 9 ]; then
echo «значение x равно 9»
else
echo «Значение x определить не удалось»
fi
```

ИСПОЛЬЗОВАНИЕ КАВЫЧЕК

Кавычки играют важную роль в написании скриптов оболочки. Существует три типа кавычек. Это: двойные кавычки «, одинарные ' (апостроф) и обратные ` (находятся слева от клавиши 1. – Прим. перев.). Имеет ли каждый из приведенных видов какое-то особое значение? Да.

Примечание: Статья Wildcards, Quotes, Back Quotes, Apostrophes in shell commands (* ? [] "' ') прекрасно описывает использование специальных символов. Пожалуйста, ознакомьтесь с ней в случае, если вы не знакомы с использованием этих специальных символов в скриптах оболочки. Ниже приведено краткое объяснение использования некоторых из них.

Двойные кавычки используются главным образом для объединения нескольких слов в строку и сохранения в ней пробелов. Например, «Эта строка содержит пробелы». Строка, заключенная в двойные кавычки рассматривается как единое целое. Например:

```
mkdir hello world
ls -F
```

```
hello/ world/
```

Здесь мы создали две директории. Команда mkdir принимает два слова hello и world, как два отдельных аргумента, и поэтому создает два каталога. Теперь посмотрим, что произойдет, если написать код таким образом:

```
$ mkdir “hello world”
$ ls -F
hello/ hello world/ world/
```

Команда создала каталог с именем из двух слов. Кавычки объединили два слова в один аргумент. (Главным образом, дело в том, что bash воспринимает пробел как разделитель всего, что только можно – опций, аргументов, отдельных команд. Внутри двойных кавычек пробел теряет свое специальное значение. – Прим. перев.).

Одинарные кавычки в основном используются для работы с переменными. Если переменная находится в двойных кавычках, то к ней можно обратиться через $имя_ переменной. Если переменная находится в одинарных кавычках – это невозможно. Чтобы пояснить это, приведем пример:

```
#!/bin/bash
x=5 # задаем x равным 5
# используем двойные кавычки
echo «Используем двойные кавычки, значение x равно $x»
# используем одинарные кавычки
echo 'Используем одинарные кавычки, значение x равно $x'
```

Почувствовали разницу? Вы можете использовать двойные кавычки, если вы не планируете использовать переменные для строки, которая в них находится. И да, если вам интересно, прямые кавычки также можно использовать для сохранения пробелов в строке тем же способом, что и двойные кавычки

```
mkdir 'hello world'
ls -F
hello world/
```

Обратные кавычки сильно отличаются от двойных и одинарных. Они не могут использоваться для сохранения пробелов. Если вы помните, выше мы использовали такую строку:

```
x=$(expr $x + 1)
```

Как вы уже знаете, результатом работы этой команды будет то, что выражение $x + 1 присваивается переменной x. Того же результата можно достичь и с использованием обратных кавычек:

```
x='expr $x + 1'
```

Какой тип кавычек лучше использовать? Тот, что вам больше нравится. Изучая скрипты вы найдете, что обратные кавычки используются чаще, чем запись $(...) . Тем не менее, я считаю, $ (...) легче читать, особенно если у вас код наподобие этого:

```
#!/bin/bash
echo “I am 'whoami'”
```

(На мой взгляд, лучше использовать именно запись типа $(...), потому что запись в обратных кавычках и одинарных можно легко перепутать при наборе кода и при его чтении. – Прим. перев.)

Это только начало. Вы узнаете еще много интересного в заключительной части этой статьи (надеюсь со временем, как только текст появится, осилить и его перевод. – Прим. перев.). А пока вы ждете – удачного вам написания скриптов...

Перевод Чернышов Антон,
Linux-преподаватель УЦ R-Style
www.tux-the-penguin.blogspot.com

Включаем NumLock при старте

Больше всего по утрам меня когда-то раздражал выключенный по-умолчанию NumLock в Ubuntu/Xubuntu/Kubuntu и иже с ними. Если у вас в пароле к учетной записи есть цифры — вы меня поймете. При нажатии на клавиши NumPad'а фокус вдруг начинал скакать по элементам управления, словно строптивый конь.

Как оказалось, решение такой серьезной проблемы для всех, у кого есть цифровая клавиатура, оказалось очень простым. Вот тут — www.help.ubuntu.com/community/NumLock — есть целая страница помощи с множеством способов для различных систем.

Я же предлагаю вам довериться мне и сделать все универсально и через консоль :)

Для начала установим маленькую (10 Кб) утилиту numlockx, управляющую NumLock'ом:

```
sudo apt-get install numlockx
```

А теперь добавим в конфигурационный файл «иксов» загрузку этой утилиты и включение NumLock'а:

```
sudo su -
echo /usr/bin/numlockx on >> /etc/X11/xinit/xinitrc
```

Вот и всё :)

www.pingvinus.ru

And User Linux

10 способов работы с FIND

GNU find является одной из наиболее часто используемых программ. На первый взгляд опции find и их синтаксис выглядят слегка непонятными. Однако, немного попрактиковавшись с find, вы сможете быстро и без труда находить любой файл в вашей системе. Чтобы облегчить вашу работу с find, рассмотрим десять способов её использования.

Имейте ввиду, что не все версии find одинаковы, и та, которую вы используете в Linux, будет отличаться от версий для Mac, BSD или Solaris. В основном синтаксис одинаков во всех версиях, но местами встречаются небольшие различия.

ПРОСТОЙ ПОИСК

Давайте начнём с простого. Если вам известно имя файла, но вы не знаете точно в каком каталоге он расположен, синтаксис find будет предельно прост. Просто сообщите find имя искомого файла:

```
find -name имя_файла
```

Если файл существует, то find выведет вам список путей, в которых встречается имя указанного вами файла. Выглядеть это будет примерно так:

```
jzb@kodos:~$ find -name filename
./projects/filename
jzb@kodos:~$
```

ПОИСК ПО РАЗМЕРУ

Иногда при поиске файла бывает нужно использовать его дополнительные атрибуты в качестве критерия поиска, а не только имя. Например, размер файла. Например, когда в вашей системе заканчивается свободное место в каком-то дисковом разделе и вам необходимо узнать, какие файлы занимают драгоценное дисковое пространство. При помощи find вы можете отыскать такие файлы и уже потом решить, за счёт чего можно высвободить необходимое дисковое пространство.

Для такого случая у find имеется опция -size, принимающая в качестве параметра размер, являющийся критерием поиска. Размер можно указывать начиная с байтов (b), заканчивая гигабайтами (G). Например, чтобы выполнить поиск файлов размером 100 килобайт, можно использовать команду:

```
find -size 100k
```

Однако такой вариант может не подойти в нашем случае. Более подходящим будет поискать файлы размеров больше (или меньше) заданного. Чтобы выполнить такой поиск, просто добавьте «+» или «-» к размеру, и find будет искать файлы соответственно большего или меньшего размера, чем указанный. Например, следующая команда найдёт все файлы размеров более 100 килобайт:

```
find -size +100k
```

а эта – менее, чем 100 килобайт:

```
find -size -100k
```

Также вы можете попросить find найти все пустые файлы:

```
find -empty -type f
```

Обратите внимание на указанную опцию -type с параметром «f», которая указывает find искать только обычные файлы. Если не указать это, то find выведет также и пустые каталоги.

ПОИСК ПО ВЛАДЕЛЬЦУ

Другой, часто используемый, вариант поиска — поиск по принадлежности файла какому-то пользователю или даже по его отсутствию. Например, вы переместили какие-то файлы в другую систему или же удалили какого-то пользователя, вероятно сделав файлы «сиротами». Отыскать такие файлы-сироты можно простой командой:

```
find -nouser
```

Для поиска файлов, принадлежащих какому-то конкретному пользователю, существуют опции -user и -uid.

Первая опция принимает как имя пользователя, так и его идентификатор, а вторая — только идентификатор. Например, если мне нужно будет найти все файлы, владельцем которых я являюсь, я воспользуюсь одной из команд:

```
find -user jzb
find -user 1000
find -uid 1000
```

Также, вам может понадобиться найти файлы принадлежащие пользователю А или пользователю Б. Для этого необходимо объединить два условия поиска при помощи оператора-о:

```
find -user root -o -user www-data
```

Такая команда будет искать файлы, владельцем которых является пользователь root или же пользователь www-data. Если же, например, вы хотите найти файлы, владельцем которых пользователь не является, используйте оператор «-not»:

```
find -not -user www-data
```

Естественно, операторы работают и с другими опциями. К примеру, следующая команда найдёт файлы, владельцем которых является www-data и которые размером не более ста килобайт:

```
find -user www-data -not -size +100k
```

ПОИСК ПО ГРУППЕ-ВЛАДЕЛЬЦУ

Ещё один способ использования find — поиск файлов, принадлежащих какой-то группе пользователей. Для этого используется опция «-group», параметром которой должно быть имя группы или её идентификатор. Например:

```
find -group admin
```

В повседневной жизни вы, вероятней всего, будете комбинировать эту опцию с другими. Например, если вам нужно отыскать файлы, принадлежащие определённому пользователю и группе.

ПОИСК ПО ПРАВАМ ДОСТУПА

Иногда может возникнуть необходимость найти файлы, доступные для записи кому угодно или файлы, имеющие какие-либо другие «плохие» разрешения. Подобный поиск find может осуществлять при помощи различных опций. Простейшие из них — это операторы -readable, -writable и -executable, которые работают применительно к пользователю, запустившему find. Имейте ввиду, что в слишком древних версиях find эти опции отсутствуют.

Другой способ искать файлы с определённым режимом доступа — использование опции -perm, позволяющей точно определять права доступа искомых файлов. Например, если вы хотите найти файлы, биты выполнения которых установлены для владельца и группы, используйте команду:

```
find -type f -perm -110
```

Здесь параметр «110» сообщает find набор битов доступа, а «-» заставляет игнорировать все остальные. Таким образом, если файл имеет ко всему прочему установленные биты чтения и записи, он также будет соответствовать критерию поиска, поскольку для find важно лишь то, что установлены указанные биты выполнения.

Если вам необходимо точное совпадение с указанным режимом доступа, то уберите опцию«-».

А что, если вам необходимо найти файлы, исполняемые владельцем или группой? В это случае вместо «-» используйте «/»:

```
find -type f -perm /110
```

Поиск по правам доступа используется часто, хотя и кажется несколько сложным, поэтому вам может потребоваться какое-то время, чтобы привыкнуть к его синтаксису. Особенно это касается новичков, которые ещё толком не разобрались с механизмом взведения битов доступа файлов. В этом случае чтение man-страницы find особенно рекомендуется.

ИСПОЛЬЗОВАНИЕ РЕГУЛЯРНЫХ ВЫРАЖЕНИЙ

Иногда вам может потребоваться использование регулярных выражений, чтобы определить критерии поиска. И find поддерживает их даже в большей степени, чем вы, возможно, ожидали. find не только поддерживает использование регулярных выражений, но и позволяет использовать различные их типы. Тип регулярного выражения можно определить при помощи опции -regextype, которая принимает параметры posix-awk, posix-egrep и тому подобные. В man-странице вы найдёте полный перечень поддерживаемых типов регулярных выражений вашей версией find.

Небольшой пример. Скажем, вам нужно найти файлы, имеющие расширения «.php» и «.js». Такое можно осуществить следующей командой:

```
find -regextype posix-egrep -regex
'.*(php|js)$'
```

Выглядит страшновато, не так ли? Эта команда говорит find использовать синтаксис регулярных выражений egrep (-regextype posix-egrep), а затем сообщает само регулярное выражение. Выражение обрамлено одинарными кавычками, чтобы оболочка не пыталась по-своему интерпретировать спецсимволы, использующиеся в выражении. В самом выражении «.*» означает любой символ, повторяющийся ноль или более раз. Часть выражения«(php|js)» сообщает о необходимости искать «php» или «js» (символ вертикальной черты используется для определения оператора «или»). И, наконец, знак доллара в конце выражения сообщает о том, что предыдущая часть выражения должна искаться в конце строки.

Так же, как и с правами доступа, регулярные выражения можно комбинировать.

Первая опция принимает как имя пользователя, так и его идентификатор, а вторая — только идентификатор. Например, если мне нужно будет найти все файлы, владельцем которых я являюсь, я воспользуюсь одной из команд:

```
find -user jzb
find -user 1000
find -uid 1000
```

Также, вам может понадобиться найти файлы принадлежащие пользователю А или пользователю Б. Для этого необходимо объединить два условия поиска при помощи оператора-о:

```
find -user root -o -user www-data
```

Такая команда будет искать файлы, владельцем которых является пользователь root или же пользователь www-data. Если же, например, вы хотите найти файлы, владельцем которых пользователь не является, используйте оператор «-not»:

```
find -not -user www-data
```

Естественно, операторы работают и с другими опциями. К примеру, следующая команда найдёт файлы, владельцем которых является www-data и которые размером не более ста килобайт:

```
find -user www-data -not -size +100k
```

ПОИСК ПО ГРУППЕ-ВЛАДЕЛЬЦУ

Ещё один способ использования find — поиск файлов, принадлежащих какой-то группе пользователей. Для этого используется опция «-group», параметром которой должно быть имя группы или её идентификатор. Например:

```
find -group admin
```

В повседневной жизни вы, вероятней всего, будете комбинировать эту опцию с другими. Например, если вам нужно отыскать файлы, принадлежащие определённому пользователю и группе.

ПОИСК ПО ПРАВАМ ДОСТУПА

Иногда может возникнуть необходимость найти файлы, доступные для записи кому угодно или файлы, имеющие какие-либо другие «плохие» разрешения. Подобный поиск find может осуществлять при помощи различных опций. Простейшие из них — это операторы -readable, -writable и -executable, которые работают применительно к пользователю, запустившему find. Имейте ввиду, что в слишком древних версиях find эти опции отсутствуют.

Другой способ искать файлы с определённым режимом доступа — использование опции -perm, позволяющей точно определять права доступа искомых файлов. Например, если вы хотите найти файлы, биты выполнения которых установлены для владельца и группы, используйте команду:

```
find -type f -perm -110
```

Здесь параметр «110» сообщает find набор битов доступа, а «-» заставляет игнорировать все остальные. Таким образом, если файл имеет ко всему прочему установленные биты чтения и записи, он также будет соответствовать критерию поиска, поскольку для find важно лишь то, что установлены указанные биты выполнения.

Если вам необходимо точное совпадение с указанным режимом доступа, то уберите опцию«-».

А что, если вам необходимо найти файлы, исполняемые владельцем или группой? В это случае вместо «-» используйте «/»:

```
find -type f -perm /110
```

Поиск по правам доступа используется часто, хотя и кажется несколько сложным, поэтому вам может потребоваться какое-то время, чтобы привыкнуть к его синтаксису. Особенно это касается новичков, которые ещё толком не разобрались с механизмом взведения битов доступа файлов. В этом случае чтение man-страницы find особенно рекомендуется.

ИСПОЛЬЗОВАНИЕ РЕГУЛЯРНЫХ ВЫРАЖЕНИЙ

Иногда вам может потребоваться использование регулярных выражений, чтобы определить критерии поиска. И find поддерживает их даже в большей степени, чем вы, возможно, ожидали. find не только поддерживает использование регулярных выражений, но и позволяет использовать различные их типы. Тип регулярного выражения можно определить при помощи опции -regextype, которая принимает параметры posix-awk, posix-egrep и тому подобные. В man-странице вы найдёте полный перечень поддерживаемых типов регулярных выражений вашей версией find.

Небольшой пример. Скажем, вам нужно найти файлы, имеющие расширения «.php» и «.js». Такое можно осуществить следующей командой:

```
find -regextype posix-egrep -regex
'.*(php|js)$'
```

Выглядит страшновато, не так ли? Эта команда говорит find использовать синтаксис регулярных выражений egrep (-regextype posix-egrep), а затем сообщает само регулярное выражение. Выражение обрамлено одинарными кавычками, чтобы оболочка не пыталась по-своему интерпретировать спецсимволы, использующиеся в выражении. В самом выражении «.*» означает любой символ, повторяющийся ноль или более раз. Часть выражения«(php|js)» сообщает о необходимости искать «php» или «js» (символ вертикальной черты используется для определения оператора «или»). И, наконец, знак доллара в конце выражения сообщает о том, что предыдущая часть выражения должна искаться в конце строки.

Так же, как и с правами доступа, регулярные выражения можно комбинировать.

РАБОТА СО ВРЕМЕНЕМ

Что, если вам понадобится найти файлы, основываясь на их возрасте? Иногда бывает, что знаешь, в каком промежутке времени файл был создан, а всё остальное — позабылось. Или же, вам может понадобиться отыскать какие-то старые файлы, которые пора удалить. В общем, причин может быть мнного. find в полном объёме умеет работать со временем, позволяя искать по времени последнего доступа к файлу (-atime), времени последнего изменения файла (-mtime), или по времени его создания (-ctime). Например, давайте все найдём файлы, которые были изменены за последние два дня:

```
find -mtime +2
```

Параметры опций, работающих со временем, можно интерпретировать как «N раз по 24 часа» и в действительности означают промежуток времени. Если вы передадите find опцию «+1», то она поймёт это как «не менее, чем 24 часа назад, но не более, чем 48». Эти опции вы также можете комбинировать, если нужно отыскать файлы, временные критерии поиска находятся в каком-то промежутке. Так, команда

```
find -mtime +2 -mtime -5
```

означает «два или более дня назад, но не более пяти дней назад».

РАБОТА С МИНУТАМИ

Иногда бывает нужно найти файлы, изменённые за последние 24 часа, и в этом случае рассмотренные опции *time по понятным причинам не подойдут. Однако, на этот случай у find припасены специальные опции -amin, -cmin, -mmin, которые работают подобно выше рассмотренным, с той разницей, что в качестве параметров они принимают минуты, а не сутки. Так что, если вам нужно найти какие-то файлы, изменённые, например, в течение рабочего дня — это те самые опции, которые вам помогут.

ОГРАНИЧЕНИЕ ПОИСКА

Иногда find выдаёт намного больше результатов поиска, чем вам нужно. При помощи опции -maxdepth вы можете ограничить find таким образом, чтобы она не «зарывалась» слишком глубоко. Например, если вы хотите найти все файлы с расширением «js» в каталоге wordpress, можно воспользоваться командой:

```
find wordpress -name '*js'
```

Но что, если вас интересуют файлы лишь из каталога верхнего уровня? Нет проблем: ограничьте поиск при помощи опции -maxdepth:

```
find wordpress -maxdepth 1 -name '*js'
```

Такая команда заставит искать find только в каталоге wordpress, не заходя в подкаталоги, которые в нём содержатся. Если вы хотите поискать в этих подкаталогах, но не соваться глубже — увеличьте параметр опции -maxdepth на единицу и т. д.

ДЕЙСТВИЯ НАД НАЙДЕННЫМИ ФАЙЛАМИ

Итак, вы нашли то, что искали. Что вы будете делать с найденным? Используя xargs или опцию find -exec, можно выполнять необходимые действия с найденными файлами.

Давайте представим, что вы хотите сменить владельца каких-то файлов с root на www-data. Для начала нужно все эти файлы найти, а затем уж менять их владельца. Смена владельца вручную по списку, полученному от find, звучит как-то скучно. Всё же, лучшим решением будет использовать опцию -exec:

```
find -user root -exec chown www-data {} \;
```

Такая команда заставляет find передавать пути всех найденных файлов утилите chown, которая и будет изменять владельца файлов. Легко и просто!

НАПОСЛЕДОК

Если вы при помощи find -exec собираетесь что-то удалять, то обязательно дважды проверьте, что именно находит find, прежде, чем передавать ей команду на удаление найденного.

www.ashep.org

маленькие хитрости

ФАЙЛ .HIDDEN

Не многие начинающие пользователи Ubuntu знают, что в наутилусе можно скрыть файлы и папки не переименовывая их. Достаточно лишь создать файл .hidden и прописать в нем то, что вы хотите убрать с глаз, например: Шаблоны и Firefox_wallpaper.png

www.vanoc.ru

КАК НАЙТИ НУЖНОЕ СОДЕРЖАНИЕ В МОРЕ ФАЙЛОВ?

```
find /путь/ -exec grep'регулярное выраже-
ние' {} \
```

КАК ВЫВЕСТИ НА ЭКРАН СПИСОК ЖЕЛЕЗА?

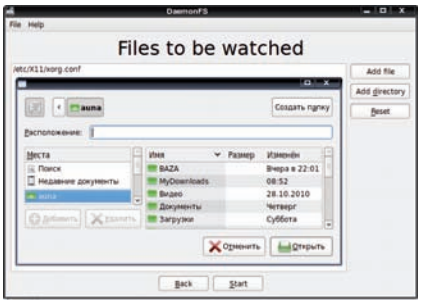
```
sudo dmidecode | more
```

http://linux.cpms.ru

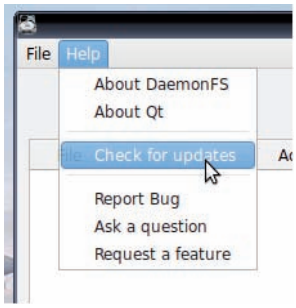
DaemonFS

DaemonFS - простой и маленький C++/Qt инструмент контролирующий в режиме реального времени изменение как отдельных файлов и каталогов так и всего жесткого диска (дисков).

DaemonFS позволяет отслеживать изменение как в множестве отдельных файлов, включая скрытые файлы, так и отдельных каталогов, включая подкаталоги (показывает дату последнего изменения).

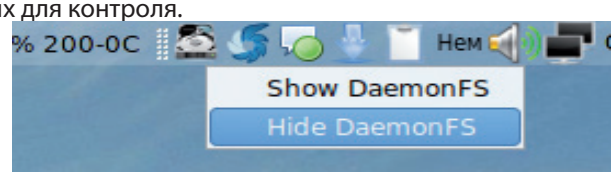


После запуска и определения файлов, каталогов или дисков целиком, они будут проиндексированы и могут быть использованы для контроля использования жесткого диска, анализа программного обеспечения и многого другого.



DaemonFS, будучи запущенным с правами администратора позволяет вернуть проиндексированные файлы и каталоги в исходное состояние (на момент индексации).

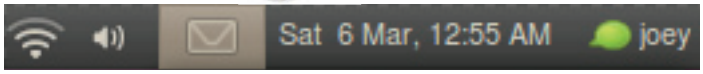
DaemonFS интегрируется в системный трей и показывает всплывающие уведомления о наличии каких либо изменений в тех или иных файлах или каталогах определённых для контроля.



Лицензия: GNU General Public License v3.0 (GPLv3)

www.zenway.ru

ДОБАВЛЯЕМ SKYPE, EMPATHY, THUNDERBIRD, ETC В MESSAGING MENU (UBUNTU 10.04)



После удаления и новой установки empathy – пропал значок "Чат" из messaging menu (значок письма) в Ubuntu 10.04. Кроме того, хотелось чтобы и другие программы отображались в messaging menu. Попробуем добавить их туда на примере skype.

Итак, чтобы добавить skype в Ubuntu messaging menu необходимо:

- Установить skype

```
sudo aptitude install skype
```

- Открыть на правку файл /usr/share/applications/skype.desktop

```
sudo gedit /usr/share/applications/skype.desktop
```

3. Добавить строку

```
StartupNotify=true
```

4. Сохранить и закрыть файл

5. Создать новый файл и открыть его на правку /usr/share/indicators/messages/applications/skype

```
sudo gedit /usr/share/indicators/messages/applications/skype
```

6. Добавить строку

```
/usr/share/applications/skype.desktop
```

7. Сохранить и закрыть файл
8. Перелогиниться

Получилось. Теперь skype отображается в messaging menu. Таким же путём вернул empathy.

www.howtolinux.ru

СРАВНЕНИЕ ФАЙЛОВ В LINUX (КОМАНДА DIFF)

Для сравнения двух или нескольких файлов в Linux есть команда diff. Она может сравнивать как отдельные файлы, так и каталоги. Рассмотрим синтаксис, опции команды diff и несколько примеров её использования.

СИНТАКСИС КОМАНДЫ DIFF

Команда diff имеет следующий синтаксис:

diff [опции] файлы-или-директории

Мы указываем опции и подаем на вход два или более файлов или директорий, которые нам нужно сравнить.

ОПЦИИ КОМАНДЫ DIFF

Рассмотрим основные опции команды diff. Я рассмотрю только те опции, которые сам использую наиболее часто.

`-E`	игнорировать изменения, связанные с добавлением символа табуляции в тексте
`-b`	игнорировать изменения, связанные с добавлением пробелов
`-w`	игнорировать изменения, связанные с добавлением пробелов и табуляции
`-B`	игнорировать новые пустые строки
`-p` (или `—show-c-function`)	показать название функции языка C, в которой найдены изменения
`-y` (или `—side-by-side`)	отобразить результаты в две колонки
`-r`	просматривать каталоги рекурсивно
`-X FILE`	исключить из поиска файлы, имена которых совпадают с шаблонами в файле FILE
`-d` (или `—minimal`)	попытаться найти как можно меньше изменений (то есть исключить ложные срабатывания)

Примеры использования команды diff

СРАВНЕНИЕ ДВУХ ТЕКСТОВЫХ ФАЙЛОВ

Для простого сравнения двух текстовых файлов с именами myfile1 и myfile2 выполним в терминале команду:

```
diff myfile1 myfile2
```

Вывод команды diff удобно перенаправить в файл с расширением diff. Большинство текстовых редакторов в Linux, например Gedit, распознают этот файл и подсвечивают его синтаксис. Чтобы направить результат сравнения в файл changes.diff нужно использовать символ перенаправления потока (>):

```
diff myfile1 myfile2 > changes.diff
```

Сравнение директорий, содержащих текстовые файлы

Рассмотрим пример сравнения двух директорий (mydir1 и mydir2), которые содержат текстовые файлы. Основное отличие здесь от примера выше состоит в том, что мы добавим опцию -r, означающую рекурсивный обход файлов в директориях.

```
diff -r mydir1 mydir2 > changes.diff
```

Теперь предположим, что в директориях, в которых мы сравниваем файлы, находится много «мусора», который мы не должны сравнивать. Создадим файл excludeFiles и запишем в него шаблоны и названия файлов, которые мы не должны сравнивать. Например, содержимое excludeFiles может иметь вид:

```
*.o
ChangeLog*
*.bak
*.exe
```

Теперь укажем команде diff, чтобы она использовала наш файл excludeFiles при сравнении каталогов:

```
diff -r -X excludeFiles mydir1 mydir2 > changes.diff
```

Таким образом, мы не сравниваем файлы, имена которых попадают под шаблоны в файле excludeFiles, например, vasya.exe или ChangeLog12.

Добавим еще несколько опций, которые описаны выше, чтобы улучшить результат сравнения:

```
diff -rwBd -X excludeFiles mydir1 mydir2 > changes.diff
```

Мы сравниваем файлы в директориях mydir1 и mydir2, игнорируя изменения, связанные с добавлением пустых строк, пробелов, табуляции, а также используем шаблоны имен файлов в excludeFiles, чтобы исключить из сравнения ненужные файлы.

ЗАКЛЮЧЕНИЕ

Дополнительную информацию по использованию команды diff в вашей системе Linux вы можете получить, выполнив команду:

```
man diff
```

Также существуют программы, которые позволяют сравнивать файлы, используя графический интерфейс. Например, программа Meld, которая в наглядном виде показывает где и что изменилось в файлах.

www.pingvinus.ru

ВРЕМЕННЫЕ ПАРАМЕТРЫ ФАЙЛА - ATIME, MTIME, CTIME

Сегодня я решил уточнить значения параметров mtime, atime и ctime, которые присутствуют у каждого файла вLinux. На первый взгляд все вроде понятно:
`mtime - modification time` - время последней модификации (изменения) файла, `atime - access time` - время последнего доступа к файлу, `ctime - change time` - время последнего изменения атрибутов файла (данных, которые хранятся в inode-области)

Но когда начинаешь спрашивать себя, когда меняются эти параметры, в частности, какие команды меняют их, а какие нет, и проверять на практике ответы на свои же вопросы - все не так очевидно. Особенно для каталогов, которые тоже являются разновидностью файлов в Linux. Вот решил поделится некоторыми экспериментами в этом направлении. Эксперименты проводил конечно же в своей Ubuntu 9.10 с файловой системой ext4. У ext4, кстати появилось два дополнительных временных параметра - время создания файла linux и время удаления файла linux, но о них в самом конце.

До начала своих экспериментов я предполагал следующее.

Параметр mtime - изменяется после того, как изменяется содержимое файла. Например, открыли файл командой nano, дописали что-то, сохранили, закрыли и время mtime поменялось. А как в случае с каталогами? Предположительно, время модификации для каталога изменяется когда в каталоге создаются/удаляются файлы и подкаталоги.

Параметр atime - изменяется тогда, когда мы получаем доступ к файлу, например, той же командой nano. Значит, atime должен измениться. Команды cat, less, tail выводят содержимое файла, значит мы получаем доступ к нему, но не меняем его поэтому mtime меняться не должен. А как быть с каталогами? Когда меняется atime для каталога? Тут я даже не знал, что себе ответить.

Параметр ctime - самый простой для моего понимания. Изменяется тогда, когда изменяются права доступа к файлу (командой chmod), изменяется владелец файла (команда chown), создаются жесткие ссылки на файл (команда ln). В этом плане различий с каталогом нет, с той лишь разницей, что на каталоги нельзя создавать жесткие ссылки.

Вот примерно так я понимал значение временных параметров файлов. Практические эксперименты несколько расширили и изменили мои познания. Во время экспериментов проверку временных параметров проверял командой stat, которая показывает сразу все три временных атрибута. Если использовать команду ls, то по умолчанию она выводит время mtime - ls -l. ls -lu (или ls –time=atime|access|use) - выводит время atime - время последнего доступа к файлу. ls -lc (или ls –time=ctime|status) - выводит время ctime - время последнего изменения атрибутов.

Начал с параметра atime. Создаю в домашнем каталоге, каталог timetest и проверяю его временные метки командой stat (вывод результата сокращаю для экономии места):

```
mkdir timetest
stat timetest
Access: 2010-07-29 16:08:53.330974403 +0300
Modify: 2010-07-29 16:08:53.330974403 +0300
Change: 2010-07-29 16:08:53.330974403 +0300
```

Сейчас все временные метку равны друг другу, так и должно быть. Далее захожу в каталог timetest и создаю пустой файл test, после чего проверяю временные метки каталога:

```
touch ./timetest/test
stat timetest
Access: 2010-07-29 16:08:53.330974403 +0300
Modify: 2010-07-29 16:34:24.442971906 +0300
Change: 2010-07-29 16:34:24.442971906 +0300
```

и файла:

```
stat timetest/test
Access: 2010-07-29 16:34:24.442971906 +0300
Modify: 2010-07-29 16:34:24.442971906 +0300
Change: 2010-07-29 16:34:24.442971906 +0300
```

После создания файла в каталоге, ожидаемо изменился параметр каталога atime и неожиданно параметр ctime. Что ж возьму себе на заметку: при создании файла в каталоге, у каталога изменяются временные параметры atime и ctime.

При создании подкаталога ситуация аналогичная. Попробую теперь удалить файл test и посмотреть как это повлияет на параметры каталога:

```
stat timetest/
Access: 2010-07-29 16:35:14.554977285 +0300
Modify: 2010-07-29 16:54:09.082973370 +0300
Change: 2010-07-29 16:54:09.082973370 +0300
rm timetest/test
stat timetest/
Access: 2010-07-29 16:56:07.634977694 +0300
Modify: 2010-07-29 16:56:13.386972802 +0300
Change: 2010-07-29 16:56:13.386972802 +0300
```

Интересный результат. Время доступа к каталогу изменилось, но на 6 секунд раньше, чем время mtime и ctime. Значит команда удаления файла не могла изменить это время. А что тогда? Проведя еще пару экспериментов догадался — это двойной tab который показывает содержимое каталога. То есть я набрал в консоли rm ti, затем нажал tab, чтобы дополнить до timetest, и затем нажал двойной tab,

чтобы отобразилось содержимое каталога, а это получается и есть операция доступа к каталогу, которая и меняет время atime. В процессе выяснения этого обстоятельства определил еще один интересный момент. Оказывается, время доступа atime меняется только в том случае, если оно меньше или равно времени mtime или ctime. Если время доступа меньше, то сколько бы я не нажимал на двойной таб - время доступа не менялось. Проверить этот момент очень просто. Командой touch -m меняю время модификации каталога на текущее и проверяю изменения:

```
touch -m timetest/
stat timetest/
Access: 2010-07-29 17:15:34.762968549 +0300
Modify: 2010-07-29 17:30:51.857519567 +0300
Change: 2010-07-29 17:30:51.854977549 +0300
```

Отмечаю, что снова изменился параметр ctime. Поэтому записываю себе еще одно предположение, что изменение времени модификации файла автоматически изменяет и время изменения атрибутов файла.

Теперь если снова выполнить двойной tab, то время доступа изменится:

```
stat timetest/
dir1/ file3 file4
stat timetest/
Access: 2010-07-29 17:31:17.410974316 +0300
Modify: 2010-07-29 17:30:51.857519567 +0300
Change: 2010-07-29 17:30:51.854977549 +0300
```

Также проверил, что команды ls, du меняют время доступа, а команды cd и pwd нет. Что впрочем логично.

С временем доступа atime для файлов ситуация аналогичная. При выводе содержимого файла командами cat, less, tail изменяется время доступа, но опять таки только в том случае, если оно меньше или равно времени mtime или ctime.

Далее решил проверить как себя ведет изменение времени модификации - mtime. Здесь для меня все обошлось без сюрпризов. Для каталога время модификации меняется всякий раз, когда в каталоге создаются/удаляются/переименовываются подкаталоги и файлы.

```
stat timetest/
Access: 2010-07-29 17:41:02.706976846 +0300
Modify: 2010-07-29 17:40:13.765539027 +0300
Change: 2010-07-29 17:40:13.762977529 +0300
rm timetest/file3
stat timetest/
Access: 2010-07-29 17:41:02.706976846 +0300
Modify: 2010-07-29 17:52:04.758977254 +0300
Change: 2010-07-29 17:52:04.758977254 +0300
mkdir timetest/dir2
stat timetest/
Access: 2010-07-29 17:41:02.706976846 +0300
Modify: 2010-07-29 17:52:33.238971947 +0300
Change: 2010-07-29 17:52:33.238971947 +0300
igor@adm-ubuntu:~$ rm -r timetest/dir2
igor@adm-ubuntu:~$ stat timetest/
Access: 2010-07-29 17:41:02.706976846 +0300
Modify: 2010-07-29 17:53:05.874971500 +0300
Change: 2010-07-29 17:53:05.874971500 +0300
touch timetest/file5
stat timetest/
Access: 2010-07-29 17:55:26.978977669 +0300
Modify: 2010-07-29 17:58:53.078971720 +0300
Change: 2010-07-29 17:58:53.078971720 +0300
mv timetest/file5 timetest/file6
stat timetest/
Access: 2010-07-29 17:55:26.978977669 +0300
Modify: 2010-07-29 17:59:40.314988184 +0300
Change: 2010-07-29 17:59:40.314988184 +0300
```

Также из примеров видно, что действительно при изменении времени mtime автоматически изменяется и время ctime. В тоже время временной параметр atime каталога timetest не меняется от действия команд rm, mkdir, mv.

Для файла параметр mtime меняется при изменении содержимого файла. Открыв файл в редакторе nano, изменив содержимое и сохранив изменения увидел следующую картину:

до изменения файла:

```
stat timetest/file7
Access: 2010-07-29 17:58:53.078971720 +0300
Modify: 2010-07-29 18:00:39.058974330 +0300
Change: 2010-07-29 18:27:06.246971667 +0300
```

после изменения и сохранения изменений:

```
stat timetest/file7
Access: 2010-07-29 18:29:22.666968437 +0300
Modify: 2010-07-29 18:29:28.798977680 +0300
Change: 2010-07-29 18:29:28.798977680 +0300
```

Обратите внимание, что в данной ситуации изменяются все временные параметры: atime - в момент открытия файла, mtime и ctime в момент сохранения изменений в файле. Кстати команда переименования файла mv не должна изменять временной параметр файла mtime. Дело в том, что в этом случае модифицируется только имя файла, которое хранится в inode-области, но не в самом файле. По логике не должна команда mv и менять время доступа. А вот параметр cname как раз и должен измениться. Но чего гадать - проверяю на практике:

```
stat timetest/file7
File: «timetest/file7»
Size: 8 Blocks: 8 IO Block: 4096 обычный
файл
Device: 804h/2052d Inode: 659938 Links: 1
Access: (0644/-rw-r--r--) Uid: ( 1000/
igor) Gid: ( 1000/ igor)
Access: 2010-07-29 18:34:20.438968510 +0300
Modify: 2010-07-29 18:34:24.438971636 +0300
Change: 2010-07-29 18:34:24.438971636 +0300
mv timetest/file7 timetest/file8
stat timetest/file8
File: «timetest/file8»
Size: 8 Blocks: 8 IO Block: 4096 обычный
файл
Device: 804h/2052d Inode: 659938 Links: 1
Access: (0644/-rw-r--r--) Uid: ( 1000/
igor) Gid: ( 1000/ igor)
Access: 2010-07-29 18:34:20.438968510 +0300
Modify: 2010-07-29 18:34:24.438971636 +0300
Change: 2010-07-29 18:40:28.914973652 +0300
```

Действительно изменился только параметр ctime. Здесь я привел вывод команды stat полностью, чтобы обратить внимание на то, что при переименовании файла,

действительно меняется имя файла, а inode - идентификатор файла остается неизменным - Inode: 659938.

Так я плавно подошел к параметру ctime. Для каталогов и для файлов этот параметр меняется после выполнения команд chmod, и chown.

```
stat timetest/dir1/
Access: 2010-07-29 18:49:00.602972037 +0300
Modify: 2010-07-29 18:49:00.602972037 +0300
Change: 2010-07-29 18:49:00.602972037 +0300

igor@adm-ubuntu:~$ chown igor:igor timetest/
dir1

igor@adm-ubuntu:~$ stat timetest/dir1/
Access: 2010-07-29 18:49:00.602972037 +0300
Modify: 2010-07-29 18:49:00.602972037 +0300
Change: 2010-07-29 18:52:07.154977433 +0300
igor@adm-ubuntu:~$ chmod o-r timetest/dir1
igor@adm-ubuntu:~$ stat timetest/dir1/
Access: 2010-07-29 18:49:00.602972037 +0300
Modify: 2010-07-29 18:49:00.602972037 +0300
Change: 2010-07-29 18:52:26.950977903 +0300
```

Для файла ctime меняется и после создания жесткой ссылки на файл:

```
stat timetest/file4
Access: 2010-07-29 18:54:14.007298839
+0300
Modify: 2010-07-29 17:11:32.442977462
+0300
Change: 2010-07-29 18:55:46.318977548
+0300
igor@adm-ubuntu:~$ ln timetest/file4
timetest/lnfile6
igor@adm-ubuntu:~$ stat timetest/file4
Access: 2010-07-29 18:54:14.007298839
+0300
Modify: 2010-07-29 17:11:32.442977462
+0300
Change: 2010-07-29 18:56:17.998971658
+0300
```

На этом буду заканчивать свой практический эксперимент, но в конце о двух новых временных атрибутах которые появились в файловой системе ext4. Это долгожданный многими crtime - create time - время создания файла, с которым очень часто путали параметр ctime. И параметр dtime - delete time - время удаления файла.

Как же посмотреть время создания файла - crtime? Так как функционал новый, то пока не нашел команды которая выдает эту информацию, но нашел как выходят из положения другие. Для этого используют команду debugfs. У меня в Ubuntu она шла по умолчанию. Чтобы воспользоваться программой нужны права администратора и имя раздела на котором находится файл.

Смотрю куда у меня примонтирован домашний каталог, в котором я работал:

```
mount | grep home
/dev/sda4 on /home type ext4 (rw)
```

Далее используем команду debugfs в таком виде:

```
sudo debugfs -R 'stat /igor/timetest/' /
dev/sda4
```

После нажатия Enter вижу следующую информацию:

```
Inode: 659937 Type: directory Mode: 0755
Flags: 0x80000
Generation: 3428284077 Version:
0x00000000:00000018
User: 1000 Group: 1000 Size: 4096
File ACL: 0 Directory ACL: 0
Links: 3 Blockcount: 8
Fragment: Address: 0 Number: 0 Size: 0
ctime: 0x4c51a4a1:ee2c6428 -- Thu Jul 29
18:56:17 2010
atime: 0x4c51a398:029e0340 -- Thu Jul 29
18:51:52 2010
mtime: 0x4c51a4a1:ee2c6428 -- Thu Jul 29
18:56:17 2010
crtime: 0x4c517d65:4ee9130c -- Thu Jul 29
16:08:53 2010
Size of extra inode fields: 28
EXTENTS:
(0): 2650522
```

Чтобы вернуться в консоль, нажимаю q. Видим параметр crtime равный Thu Jul 29 16:08:53 2010, что совпадает с временем создания (см. самый первый вывод команды stat).

А где же параметр dtime? А он появляется только, когда файл удален. А как тогда его посмотреть? А посмотреть можно по inode. На примере файла file8:

Смотрим сначала inode файла:

```
ls -li timetest/file8
659938 -rw-r--r-- 1 igor igor 8 2010-07-29
18:34 timetest/file8
```

inode равен 659938. Теперь я удаляю файл:

```
rm timetest/file8
```

и выполняю команду debugfs, где указываю не имя файла, а inode:

```
sudo debugfs -R 'stat <659938>' /dev/sda4
Inode: 659938 Type: regular Mode: 0644
Flags: 0x80000
Generation: 3428285866 Version:
0x00000000:00000001
User: 1000 Group: 1000 Size: 0
File ACL: 0 Directory ACL: 0
Links: 0 Blockcount: 0
Fragment: Address: 0 Number: 0 Size: 0
ctime: 0x4c51a95e:8afe201c -- Thu Jul 29
19:16:30 2010
atime: 0x4c519f7c:68a882f8 -- Thu Jul 29
18:34:20 2010
mtime: 0x4c51a95e:8afe201c -- Thu Jul 29
19:16:30 2010
crtime: 0x4c51972d:12d40d20 -- Thu Jul 29
17:58:53 2010
dtime: 0x4c51a95e -- Thu Jul 29 19:16:30
2010
Size of extra inode fields: 28
EXTENTS:
```

Вот и параметр dtime нашелся.

На этом все. Надеюсь, статья будет кому-то полезной.

www.igorka.com.ua

COMMUNIGATE

Часть I. Общий обзор, назначение, возможности, применение

Говоря о такой системе как Communigate, на мой взгляд, нужно начать с описания того, что это такое и для чего она предназначается. Communigate – это система, основанная на открытых стандартах и представляющая собой интегрированную платформу, в которой реализованы функции хранения и отправки электронных почтовых сообщений, ведения календаря, коммуникаций реального времени – голосовые, видео-, мгновенные сообщения, совместная работа в сетях IPv4 и Ipv6.

ОБЩЕЕ ОПИСАНИЕ ПОДСИСТЕМ COMMUNIGATE

Если систематизировать все подсистемы этой интегрированной среды, то функционально можно выделить следующие подразделы:

- управление идентификацией;
- управление хранением;
- передача почты;
- сигналы реального времени;
- среда для приложений реального времени;
- службы доступа к данным;
- передовые средства безопасности;
- многоуровневая система администрирования;
- возможность использования нескольких серверов.

Поскольку Communigate базируется на открытых стандартах, в ней поддерживаются все основные современные возможности для систем такого уровня, то есть Communigate по праву может называться современной промышленной коммуникационной средой операторского класса.

К числу её достоинств относят также возможность удаленного администрирования через любой Web-браузер. При этом доступны следующие свойства и особенности:

- настройка сервера и настройка всех его коммуникационных модулей и правил маршрутизации;
- создание и удаление пользователя, обновление любой информации о пользователе;
- мониторинг активности модулей;
- мониторинг системных журнальных файлов;
- работа как с очередями сообщений сервера, так и с индивидуальными сообщениями в очереди.

Помимо всего этого, есть специализированный лист рассылки (CGatePro-on@communigate.com) для обсуждения вопросов, связанных с Communigate и их архив для желающих получить дополнительную информацию. Рассмотрим более подробно подсистемы Communigate.

Все основные подсистемы Communigate выполнены в виде объектов, которые поддерживают строгую иерархию. На самом верхнем уровне находится множество доменов, затем по нисходящей идут пользователи, группы, списки рассылки, псевдонимы и переадресаторы.

У каждого пользователя имеется в распоряжении одна или несколько папок, в каждой из которых могут храниться сообщения. Помимо этих основных объектов, пользователь может иметь хранилище файлов и данные настроек, домены могут иметь сертификаты, файлы, определяющие вид Web-интерфейса пользователя, специальные приложения реального времени и многое другое. Все это тоже является объектами, поддерживаемыми общей структурой Communigate.

ДОМЕНЫ

Домены – это объекты Communigate, которые содержат другие объекты: пользователей, списки рассылки, группы. Каждый домен имеет имя (my.com, www.my_company.com и подобные этим).

Хотя каждый домен Communigate имеет своё имя, нет необходимости создавать отдельные домены Communigate для каждого имени домена, которое придется обслуживать. Домены Communigate могут иметь псевдонимы доменов, которые позволяют задавать несколько имён одному домену Communigate. Например, домен Communigate mycompany.com может иметь псевдоним домена mail.mycompany.com. В этом случае все ссылки на имя домена mail.mycompany.com будут обрабатываться так же, как и ссылки на домен mycompany.com.

Существует специальный домен Communigate, называемый главным доменом. Все остальные домены Communigate равнозначны. Главный домен создаётся сразу же после установки сервера и его имя указывается в установках «Общее». Если сервер обслуживает только один домен, то не обязательно создавать другие домены. При этом имя главного домена используется в качестве имени сервера. Каждый домен Communigate имеет свои собственные установки и наборы объектов домена.

ПОЛЬЗОВАТЕЛИ

Пользователь является основной единицей для обслуживания: каждый, кто обслуживается на сервере Communigate, должен быть пользователем сервера, зарегистрированным в одном из его доменов. Каждый пользователь имеет пароль и только сам пользователь (опционально администратор этого сервера/домена) имеет неограниченный доступ к своим данным.

При установке сервера в главном домене автоматически создаётся пользователь с учетной записью postmaster. Этому пользователю предоставляется полное право доступа на все возможные данные и настройки. Адресом электронной почты и сообщений для пользователя будет accountname@domainname, где accountname – это имя пользователя Communigate, a domainname – это имя домена Communigate, в котором существует рассматриваемый пользователь. Сообщения, направляемые на адрес пользователя, доставляются пользователю путем местной доставки или через компонент сигнальных сообщений.

Для того, чтобы присвоить несколько имён одному пользователю, администратор может создать псевдонимы для этого пользователя.

Каждый пользователь Communigate имеет свои собственные установки и наборы папок, а также может иметь собственное хранилище файлов.

ГРУППЫ

В доменах Communigate могут создаваться группы. Группы – это списки, содержащие имена пользователей и/или другие группы, отправка сообщений на имя группы приведёт к тому, что сообщение будет отправлено всем членам группы (по образу списка рассылки).

ПЕРЕАДРЕСАТОРЫ

В доменах Communigate могут быть переадресаторы. Каждый переадресатор имеет имя и адрес электронной почты для перенаправления. Если почта отправляется на адрес name@domain.com, где name является объектом типа переадресатор в домене Communigate domain.dom, то почта перенаправляется на электронный адрес, указанный в переадресаторе.

Объект «переадресатор» отличается от объекта «группа» по следующим критериям:

- переадресатор содержит только один адрес, а группа может содержать несколько адресов;
- переадресатор работает на уровне маршрутизатора, подставляя собственный адрес взамен указанного адреса, а объект «группа» обрабатывает сообщения, отправленные на адрес группы и генерирует новые копии, отправляемые всем членам группы.

СПИСКИ РАССЫЛКИ

В доменах Communigate могут быть созданы списки рассылки. Каждый список рассылки имеет имя и всегда принадлежит определенному пользователю из того же домена – владельцу списка рассылки.

Список рассылки содержит список подписчиков и работает с несколькими папками пользователя, владельца этого списка. Эти папки применяются для хранения и архивирования сообщений, создания дайджестов, хранения информации о запросах на подписку и хранения отчётов об ошибках. Группы и списки рассылки суть не одно и то же и отличаются друг от друга следующими параметрами:

- группы предназначены для небольшого числа членов, а списки рассылки предназначены для работы с большим числом подписчиков одного списка;
- группы в основном используются локальными пользователями в случае, если имя пользователя переименовывается или пользователь удаляется, он также переименовывается в группах или удаляется из всех групп этого домена;
- списки рассылки, кроме базовой функциональности рассылки почты, имеют множество дополнительных возможностей, – это автоматическая обработка запросов на подписку, обработка ошибок, архивирование и создание дайджестов, просмотр, контроль за политикой отправки сообщений в список.

ПАПКИ

Папка является базовой единицей хранения: сообщения, отправляемые пользователям, хранятся в соответствующих папках. Сообщения из папок могут быть прочитаны, могут быть отмечены различными флагами, скопированы в другие папки и удалены. Каждый пользователь может иметь одну или несколько папок. Папка INBOX является специальной папкой: она существует у каждого пользователя.

Inbox используется для хранения поступающих сообщений и создаётся автоматически при создании пользователя. Пользователь не может удалить папку INBOX, но может переименовать её. В этом случае немедленно создается новая пустая папка INBOX. Communigate позволяет администраторам создавать пользователей, которым разрешена работа только с одной папкой. Как правило, это и есть папка INBOX.

Сервер Communigate предоставляет доступ к папкам пользователя через модули POP, IMAP, Web-интерфейс пользователя, а также через другие модули. Папки Communigate могут иметь различные форматы. Администраторы и пользователи могут выбрать формат папки на этапе её создания.

ПСЕВДОНИМЫ ПОЛЬЗОВАТЕЛЯ

Псевдоним пользователя является альтернативным именем для

пользователя Communigate. Каждый пользователь может иметь один или несколько псевдонимов. Например, пользователь j.smith в домене domain2.com может иметь псевдонимы smith и jsmith. Почта, отправляемая на адрес smith@domain2.com, будет храниться у пользователя j.smith, и попытка входа на сервер как jsmith@domain2.com приведёт к входу от имени пользователя j.smith.

Можно также использовать переадресатор для задания альтернативных имён пользователя. Если создается переадресатор js в домене domain2.com, который указывает на адрес j.smith, то он фактически будет работать как еще один псевдоним пользователя j.smith. При этом, если переименовывается пользователь j.smith в james.smith, то все псевдонимы пользователя также изменятся вместе с ним – smith и jsmith останутся псевдонимами для james.smith.

Если удаляется пользователь, все его псевдонимы также будут удалены. Переименование или удаление пользователей не затрагивает их переадресаторов. Переадресаторы должны использоваться для создания «объектов», которые перенаправляют почту на другие домены или на другие почтовые серверы.

ОСНОВНЫЕ ВОПРОСЫ УСТАНОВКИ И ИСПОЛЬЗОВАНИЯ COMMUNIGATE

Установка Communigate и весь ее процесс зависит от платформы, которую вы собираетесь использовать. Разработчики предлагают широкий спектр дистрибутивов относительно практически всех известных на сегодняшний день системных платформ. Это Sun Solaris, Linux, MS Windows, MacOS X, FreeBSD, NetBSD, OpenBSD, BSDI BSD/OS, AIX, HP/UX, QNX, BeOS и многие другие. Для всего многообразия представленных систем, тем не менее, придется решать общие вопросы в процессе установки и их настройки.

ИМЯ ГЛАВНОГО ДОМЕНА

Для его задания нужно ввести правильное имя в поле «Имя главного домена» и нажать на кнопку модификации. Имя должно быть полным (однозначным) именем домена компании или организации.

Если необходимо обслуживать несколько доменов на сервере Communigate, то можно создать для этих целей дополнительные домены: каждый домен при этом независим от других и имеет свои установки, своих собственных пользователей и свою систему объектов.

ЯЗЫК И КОДИРОВКА

В Communigate языком по умолчанию является английский. Если большинство пользователей предпочитает использовать другой язык, в меню «язык» нужно изменить выбор языка по умолчанию. Можно также изменить используемую по умолчанию кодировку. Хотя большинство современных браузеров используют кодировку Unicode (UTF-8), при чтении сообщения, в котором не указана используемая кодировка, или при отправке сообщения в системы, поддерживающие старые стандарты передачи сообщений, используется кодировка по умолчанию.

Эти настройки могут быть установлены «по умолчанию» для всех пользователей используемого сервера.

ЧАСОВОЙ ПОЯС

Время на сервере необходимо контролировать. Для этого нужно убедиться, что установлен правильный часовой пояс. Затем нужно проверить время, установленное на самом сервере. Если какой-нибудь из этих параметров неверен, необходимо исправить эти настройки непосредственно в операционной системе. Возможно, при этом потребуется перезапустить операционную систему и/или сам сервер Communigate для того, чтобы новые установки времени вступили в силу.

Если выбрать «встроенный» пояс (фиктивный), то сервер будет использовать фиктивный часовой пояс, который имеет ту же разницу с Гринвичем, что и операционная система сервера. Этот часовой пояс не поддерживает переход на зимнее/летнее время и не может использоваться для отправки информации о повторяющихся событиях или встречах с пользователями других серверов.

Необходимо избегать использования «встроенного» часового пояса и применять его только в том случае, если часовой пояс, который нужно использовать, отсутствует в списке, что очень маловероятно.

СЕТЬ

Настройки сети находятся в области управления сервером, имеющей административные привилегии. Для их изменения нужно открыть сетевой раздел и внести требуемые изменения. При этом, если сервер подсоединён к корпоративной или домашней сети, нужно указать соответствующие сетевые адреса.

Если Communigate используется в корпоративной среде, большинство пользователей будут соединяться с сервером из внутренней локальной сети (LAN).

Для того, чтобы настроить адреса LAN, нужно использовать Web-интерфейс администратора. Таблица адресов LAN первоначально содержит адреса, полученные Communigate из настроек операционной системы сервера. При необходимости нужно исправить этот список, включив в него все LAN (локальные сети), которые должен обслуживать Communigate.

В разделе «Список сетевых адресов» описывается формат списка. Иногда требуется, чтобы все почтовые клиенты и клиенты реального времени (VoIP или мгновенные сообщения), устанавливающие соединения с LAN адресов, имели возможность ретран-

слировать (релеить) электронную почту и сигналы в любое место в Интернете. В этом случае нужно включить адреса LAN в список «Сетевые адреса клиентов».

Список адресов LAN используется для поддержки коммуникаций реального времени. К их числу относятся голосовые, видео- и другие подобные сообщения. При этом Communigate знает, какие адреса являются локальными NAT адресами, то есть какие адреса не могут быть доступны непосредственно из Интернета.

Для осуществления коммуникаций между компьютерами в LAN нужно использовать настройку «Адрес сервера в LAN» для выбора собственного IP-адреса сервера, используемого также операционной системой сервера. Communigate может обслуживать коммуникации реального времени и с помощью механизма дальнего прохождения NAT поддерживает SIP-протокол для удалённых клиентов, находящихся за NAT-устройствами. Для того что бы обнаружить клиентов за NAT, серверу нужно знать, какие адреса используются в удалённых сетях, находящихся за пределами своего маршрутизатора.

Для того что бы настроить адреса за NAT, используется также Web-интерфейс администратора. Для этого нужно открыть в области установки страницу «Сеть», затем открыть страницу NAT и внести требуемые изменения.

ПОЛЬЗОВАТЕЛИ

Для доступа к этому разделу необходимо открыть область «Пользователи» в Web-интерфейсе администратора и создать там требуемых пользователей.

Сам пользователь является основной единицей обслуживания: каждый, кто обслуживается на сервере Communigate, должен быть пользователем сервера.

Каждый пользователь защищен паролем и только сам пользователь (а также администраторы сервера и домена) могут получить полный доступ к своим данным. В главном домене автоматически создаётся пользователь с именем «postmaster». Ему предоставляется полное право доступа к информации.

ВЫВОДЫ

В статье описаны вопросы общей организации и назначения интеграционного программного комплекса Communigate. Рассмотрены основные компоненты, образующие логический состав сервера и составляющие его структуру.

Описаны общие вопросы, возникающие при выборе платформы для развертывания Communigate и установки на нее этой системы. В следующих статьях цикла мы подробнее рассмотрим нюансы настройки и практического использования Communigate.

Александр Деревянко, ведущий консультант-эксперт, консультант
www.ibm.com

SLACKWARE 13.0 НА ДОМАШНЕМ ПК

Slackware Linux – один из старейших дистрибутивов Linux. Его иногда называют «самым UNIX`овым». Поклонникам этого дистрибутива приписывают такие высказывания: «Если вы знаете Red Hat, то всё, что вы знаете, - это Red Hat, если вы знаете Slackware - вы знаете GNU/Linux.» ©Википедия .

Slackware считается слишком сложным для простого пользователя. Это далеко не так. Во всяком случае он ничуть не сложнее других дистрибутивов Linux. Возможно, в нем чуть меньше графических настроечных утилит с менюшками и кнопочками. Возможно, что некоторые настройки надо делать в файлах конфигурации, но ведь это приходится делать и в любом другом дистрибутиве. То, что раньше считалось исключительно серверной платформой, уже давно стало десктопом, и очень неплохим.

УСТАНОВКА

За последние лет 5 процесс установки ничуть не изменился, по крайней мере внешне. Что действительно не отнять, так это некий консерватизм и традиционность. Не знаю кому как, а я при переходе с версии 12.1 на 13.0 никаких затруднений не испытывал. Тяжелее было привыкнуть к KDE4, но в общем много времени это не заняло.

Начнем с того, что при установке никаких графических приложений, в том числе для изменения разделов диска нет. Неплохо бы иметь также хотя бы базовое знание английского – понятно, что установка именно на этом языке. Входим под root-ом, затем cfdisk (псевдографика тоже бывает вполне удобна), setup. Далее банально и несложно, хотя следует отметить некоторые моменты:

Выберите сразу раскладку «qwerty/ru-ms.map» и шрифт «koi8u-8x16.psfu.gz». Изменить, если потребуется, можно и позже. Установите пакеты KDEI, чтобы иметь в KDE поддержку других языков, включая русский. Не устанавливайте LILO автоматически. Выберите «Use expert lilo.conf setup menu».

После завершения установки и последующей перезагрузки, добавьте пользователя с помощью adduser. Slackware, в отличие, например, от Ubuntu, суперпользователя не прячет. Тем более, относиться надо с осторожностью и входить под root только в крайнем случае.

НАСТРОЙКА X

Не представляет ни сложности ни интереса, тем более, что на замену старому xorgconfig появился новый, более удобный xorgsetup.

РУСИФИКАЦИЯ

Сначала консоль. В файле «/etc/profile.d/lang.sh» заменить локаль «export LANG=ru_RU.KOI8-R». В конец файла «/etc/rc.d/rc.font» буквально, как есть, добавить следующее:

```
for n in 1 2 3 4 5 6 do
{
  echo -ne «\033(K» > /dev/tty$n;
  }
```

В конце не забыть нажать Enter, чтобы поставить конец строки. Перезагрузить систему. Готово – консоль, Midnight Commander, а также некоторые man-страницы русские, переключение раскладок – правый Ctrl.

Русификацию KDE проще и нагляднее показать в картинках.

Добавить язык - русский.

Включить переключение раскладок и индикатор.

Определить горячие клавиши переключения раскладок. После чего перезапустить X.

МОНТИРОВАНИЕ NTFS-РАЗДЕЛОВ, ДИСКОВОДОВ И ФЛЕШЕК ПОЛЬЗОВАТЕЛЕМ

Сразу нормально это не работает, а если работает, то ненормально. Причем, насколько заметно по блогам и форумам, не только в Slackware. Потратив несколько дней на изучение документации и разбирательства с файлами конфигурации HAL и ntfs-3g, я пришел к выводу, что эти штуки созданы исключительно для того, чтобы окончательно испортить мне жизнь. Идея в общем хорошая и состоит в том, чтобы обычный пользователь мог монтировать устройства для чтения и записи в один клик. Реализация оказалась, как это обычно бывает, не на высоте. ntfs-config собранный из сырцов (в дистрибутиве Slackware его нет) помог мало. Он устанавливает автоматическое монтирование, с некорректной поддержкой языка, даже без возможности размонтировать пользователем. Тогда зачем, спрашивается он такой нужен? Остаются документация и конфигурация вручную. Итак:

Удалите через менеджер пакетов старый ntfs-3g и скачайте свежий.

Соберите и установите. (я, вместо того чтобы сразу обновить этот драйвер, потерял напрасно кучу времени)

Так выглядит мой fstab (точнее его часть, которая относится к делу):

```
/dev/hda2  /          ext3    defaults              1 1
/dev/hda4  /mnt/hda4  ntfs-3g noauto,rw,user,locale=ru_RU.KOI8-R 0 0
/dev/hda1  /mnt/slax  ext3    noauto,user,rw        0 0
/dev/cdrom /mnt/cdrom auto    noauto,user,ro        0 0
/dev/sda1  /mnt/flash vfat    noauto,user,rw,codepage=866,iocharset=koi8-r 0 0
```

За исключением раздела NTFS все обычно и привычно.

Добавить себя в группы:

```
usermod -G users,disk,pludgdev,messagebus,haldaemon username
```

Поменять права доступа к точке монтирования NTFS раздела и, возможно, к устройству. Для флешек и дисковода это может не потребоваться.

```
chmod 777 /mnt/hda4
chmod 660 /dev/hda4
```

Это в моем конкретном случае. Ваш NTFS раздел может быть другим. Работает, монтируется по требованию, пользователем, как и должно быть.

Вот так, вкратце можно довести Slackware от установки до полного рабочего состояния. От себя добавлю, что я не профессионал, а обычный пользователь, просто перелопатил документацию и сеть в поисках правильных настроек.

p.s. Дополнительно по рекомендации доки по ntfs-3g,

```
chmod 4755 /bin/ntfs-3g
```

поставить suid бит, если его нет.

Андрей aka keedhost Кондратьев
www.keedhost.blogspot.com

По вопросам размещения вашей информации обращайтесь в редакцию журнала «User And LINUX»

Адрес для писем: journal@ualinux.com

UA Linux

open source ■ open future

Разработка комплекта дополнений Ubuntu Applications Pack

IT аутсорсинг

Поддержка и консультирование пользователей

Разработка, внедрение и сопровождение эффективных IT-решений на базе открытого ПО

Выпуск ежемесячного журнала User And Linux

Внедрение систем виртуализации

Тестирование оборудования а так же доработка под него программного обеспечения