Тема 2. Регулярные выражения и управление заданиями

Регулярные выражения

Выражение	Описание	Пример	Что означает пример
a	Обычные символы обозначают сами себя	*apple*	Строка «apple»
[...]	Любой символ, заключенный в [ ]	*[02468]*	Любая из цифр 0,2,4,6,8
[^...]	Любой символ, не заключенный в [ ]	*[^13579]*	Любой символ кроме нечетных цифр
[x-x]	Диапазон символов	*[A-Z]*	Любая большая латинская буква
.	Любой одиночный символ	*c.t*	cut, cat,c9t и т.д.
^	Начало строки	*^[0-9]*	Строки, начинающиеся с цифры
$	Конец строки	*/bin/sh$*	Строки кончающиеся bin/sh
*****	Ноль или больше предшествующих символов	*[a-z]**	Любая последовательность символов в нижнем регистре или ничего
?	Ноль или один предшествующий символ	*https?://*	http:// и https://
+	Один или более предшествующих символов	*T+*	T, TT, TTT, TTTT и т.д.
{n}	n повторов предыдущих символов	*[0-9]{3}*	Последовательность из 3 цифр, например 124, 111, 743
{n,}	n или больше предыдущих символов	*0{3,}*	000, 0000, 00000 и т.д.
{n,m}	От n до m предыдущих символов	*[A-Z]{2,3}*	Строки вроде AB, ABC, YY, ZZZ

Синтаксис grep:

grep 'patern' textfile

Команда grep может работать в двух режимах: простые регулярные выражения и расширенные регулярные выражения. Для использования расширенных регулярных выражений необходимо указывать ключ **-E**.

Пример (регулярное выражение, распознающее IP-адреса):
([0-9]{1,3}\.){3}[0-9]{1-3} - режим расширенных regexp'ов
\([0-9]\{1,3\}\.\)\{3\}[0-9]\{1-3\} - режим простых regexp'ов

Выполните пример по регулярным выражениям:

1. Создайте файл hello.txt с некоторой текстовой информацией, включающей и пустые строки.

2. Выполните в окне терминала: grep –v ‘^$’ ~/hello.txt
3. Команда выведет все непустые строки файла. Ключ v означат поиск несовпадения.

Управление заданиями

1. Откройте окно терминала и введите такую команду:

yes > /dev/null

Команда *yes* просто выводит на экран (или в /dev/null, как в нашем примере) ‘y’, пока вы не прервёте ее. Предоставленная сама себе, команда будет выполняться вечно, так что оставим её транжирить циклы процессора, а сами откроем новый терминал и запустим в нём top. Вы должны увидеть, что yes использует до 100% процессорного времени. Затем оставьте окно с *top* запущенным и вернитесь в терминал с *yes*.

Нажатие Ctrl+C посылает текущему процессу сигнал SIGINT и обычно приводит к его прерыванию. Можете также нажать Ctrl+\, чтобы послать процессу сигнал SIGQUIT, сигнал немедленно завершит процесс и создаст дамп памяти (core dump). Впрочем, куда важнее, что вы можете использовать Ctrl+Z, чтобы послать SIGSTOP.

Большинство сигналов в Linux отправляются программе и обрабатываются ею. Например, нажатие Ctrl+C посылает SIGINT процессу, а тот его обрабатывает. Процесс может проигнорировать сигнал (вэтом случае Ctrl+C ничего не делает), а может выполнить некоторые действия перед завершением работы. Но два сигнала – SIGSTOP и SIGKILL – процессу не отправляются. Вместо этого они передаются непосредственно ядру, потому что требуют внешнего воздействия на процесс.

2. Давайте попробуем применить это к команде *yes*. Выбрав данный терминал, нажмите Ctrl+Z. Вы должны увидеть такой вывод:

[1]+ Stopped yes > /dev/null

Это означает, что наша команда (yes > /dev/null) остановлена, и на неё можно сослаться как на задание №1. Терминал теперь в ваших руках, так что попробуйте запустить fg (foreground) для реанимации *yes*. Внимание: отправка SIGSTOP и запуск fg похожи на нажатие паузы на вашем CD-плейере, поскольку процесс на самом деле не прекращает существование, а просто временно приостанавливает работу.

3. Итак, *yes* снова работает. Нажмите Ctrl+Z, чтобы опять приостановить процесс, но теперь введите bg, чтобы *Bash* возобновил его работу и оставил его в фоновом режиме – как будто вы запустили yes > / dev/null &. Теперь *yes* работает в фоновом режиме, и Ctrl+Z не сможет остановить его. Вместо этого мы должны послать сигнал SIGSTOP непосредственно процессу, используя команду:

kill -SIGSTOP %1

%1 означает, что *Bash* должен послать сигнал заданию №1, т.е. вашей команде *yes*. Если вы опасаетесь, что не упомните все цифровые идентификаторы, самое время познакомиться с *jobs*. У этой команды нет интересных параметров, но она возвращает список всех созданных нами заданий.

Вы, вероятно, недоумеваете, почему задание №1 имеет знак «минус» после номера, задание №2 не имеет никакого знака, а №3 – знак «плюс». Так *Bash* отмечает, что рассматривается в качестве текущего (+) и предыдущего (-) задания, и вы можете

использовать это для краткости. Например, если перевели команду *yes* в фоновый режим, вы можете воскресить её, набрав %+ (или даже %%). Для ссылки на предыдущее задание используйте %-.

Автоматизация работы

Управление заданиями превосходно работает, когда вы подключены к терминалу, но что если вы хотите, чтобы программы останавливались и запускались, когда вас нет на месте? Рассмотрим две команды по автоматизации задач.

Команда *at* обычно работает интерактивно, то есть вы сначала вызываете программу, указав время, когда задание должно выполниться, затем вводите свои команды и нажимаете Ctrl+D, чтобы сохранить задание. Вот пример «диалога» с *at*:

```
user@user:~$ at midnight
at> du / > ~/diskusage
at <EOT>
job 1 at 2007-02-13 00:00
```

В первой строке запускаем *at* и указываю полночь (midnight) как время старта для задания. При этом *at* запустится, и появится приглашение at>, показывающее, что можно вводить содержимое задания. Задание, которое мы хотим выполнить в полночь – выяснить, сколько дискового пространства используется на моём компьютере, так что запускаем *du* и перенаправляем её вывод в файл в домашнем каталоге. <EOT> – это Ctrl+D, что приводит к сохранению задания, выводу его номера и сообщению, когда оно будет выполнено (сегодня в полночь).

Узнать, какие задания уже поставлены в очередь, вы можете, набрав atq, которая распечатает что-то наподобие

```
1 2007-02-13 00:00 a user
```

Первое, второе и четвёртое поля – это соответственно номер задания, время, когда оно будет выполнено, и кто его создал, но о третьем поле нужно сказать особо: это приоритет задания. В Linux можно выбрать много очередей заданий, и чем дальше буква от начала алфавита, тем ниже приоритет. Очередь «a» имеет наивысший приоритет, и будет исполнена с нормальным для пользователя значением nice (то есть так быстро, как только сможет).

Если вы дождётесь полуночи, ваше задание будет выполнено, как и планировалось. Но если вы передумаете, используйте команду *atrm*, чтобы удалить задание:

```
atrm 1
```

Есть несколько способов указать время в *at*, и midnight – лишь один из них. Из предопределённых есть tomorrow (завтра), noon (полдень) и teatime (4 часа вечера), но вы можете указывать и точное время, например, 16:00 (те же 4 вечера) или комбинировать эти значения (16:00 tomorrow). Простейший способ – указывать относительное время, используя now (сейчас), например, так:

```
at now + 5 minutes
at now + 3 hours
at now + 4 weeks
```

Если время выполнения для вас не имеет значения, забудьте про *at* и используйте *batch*. Различие заключается в том, что *batch* начнёт выполнять ваши задания сразу

же, как только загрузка системы опустится ниже 0,8 (т.е. машина будет не слишком занята). Синтаксис намного проще, поскольку не нужно указывать время: просто наберите batch, нажмите Enter, добавьте свои команды и нажмите Ctrl+D, чтобы сохранить задание.

Введя atq, вы увидите ваше задание в очереди «B», что означает запуск с более низким приоритетом, чем у других заданий и большинства программ в системе. По этой причине ваше задание стартует только тогда, когда система бездействует, но если оно запустится, а в следующую секунду машину потребует другая работа, ваше задание тихонько переберется в фоновый режим и отдаст ресурсы процессора. Оно не останавливается, но из-за более низкого приоритета получит намного меньше процессорного времени.

Никто не любит вводить одну и ту же команду снова и снова, так что если вы хотите, чтобы *at* или *batch* читали ваши задания из файла, просто используйте -f имяфайла перед указанием времени, например, так:

```
at -f myjob.job tomorrow
```

или для batch

```
batch -f myjob.job
```

Нет ничего прекраснее, чем заставить ваш компьютер делать кучу работы за вашей спиной, полностью автоматически.

Задание

1) Создать пять текстовых файлов
2) Поместить в них адреса электронной почты вида name@domain.domain и любую другую информацию
3) Создайте файл задания user.job в домашнем каталоге.
4) Используя утилиту grep и регулярные выражения напишите в файле user.job команду задания, выводящего отсортированный список уникальных адресов электронной почты в файл allemails.lst.
5) Поставьте на выполнение это задание (user.job) через одну минуту с помощью at.