Впервые опубликовано на developerWorks 01.09.2005

Экзамен LPI 201: Настройка работ и автоматическое выполнение заданий

Администрирование Linux, средний уровень (LPIC-2) тема 213

Автоматизация периодически запускаемых заданий

Конфигурирование cron

Демон `cron` используется для периодического запуска комманд. Вы можете использовать `cron` для широкого круга задач по поддержке и администрированию. Если это событие или задание возникает с определенной регулярностью, его следует обслуживать при помощи `cron`. `Cron` пробуждается каждую минуту и проверяет, не нужно ли чего-нибудь сделать, но он не может запускать эти задания чаще чем раз в минуту. (Если вам требуется такая функциональность, возможно вам необходим демон, а не "cron job.") Cron журналирует свои действия через механизм syslog.

Cron ищет свои конфигурационные файлы, в которых устанавливаются переменные окружения и команды, которые следует исполнять, в разных местах. Первый из них -- это /etc/crontab, содержащий системные задания. Каталог /etc/cron.d/ может содержать различные конфигурационные файлы, дополняющие /etc/crontab. Отдельные пакеты могут добавлять файлы (с именами, соответсвующими имени паета) в /etc/cron.d/, но системному администратору следует использовать /etc/crontab.

ПОльзовательские конфигурации для cron хранятся в /var/spool/cron/crontabs/$USER. Однако, они должны быть созданы при помощи программы `crontab`. При помощи `crontab` пользователи могут задавать свои собственные периодически запускаемые задания.

Ежедневный, еженеделный и ежемесячный запуск заданий

Задания, которые должны выполняться ежедневно, еженедельно или ежемесячно -- а это наиболее распространенная ситуация -- описываются согласно специальным соглашениям. Каталоги /etc/cron.daily/, /etc/cron.weekly/ и /etc/cron.monthly/ созданы для хранения наборов соответсвующих скриптов. Добавление или удаление скриптов в этих каталогах -- это простейший путь для управления системными заданиями. Например, система, которую обслуживаю я, осуществляет ежедневную ротацию файлов журналов при помощи скрипта:

Листинг 1. Пример скриптового файла, запускаемого ежедневно

```
 $ cat /etc/cron.daily/logrotate
 #!/bin/sh
```

```
 test -x /usr/sbin/logrotate || exit 0
 /usr/sbin/logrotate /etc/logrotate.conf
```

Cron и anacron

Вы можете использовать `anacron` для выполнения команд с частотой в определенное количество дней. В отличие от `cron`, `anacron` проверяет каждое задание, которое должно было быть исполненено в течение последних n дней (где n -- это период, определенный для данного задания, в отличиие от проверки текущего времени для определения момента запуска). Если оно не запускалось, `anacron` производит запуск с задержкой, указанной в минутах в параметре `delay`. Таким образом, на машинах, которые не включены постоянно, переодически запускаемые задачи выполняются единожды, когда машина работает (конечно, точное время запуска может варьироваться, но задание не будет забыто).

`Anacron` читает список заданий из конфигурационного файла /etc/anacrontab. Каждая запись включает в себя период в днях, задержку в минутах, уникальный идентификатор задания и команду оболочки. Например, на одной из Linux систем, которую я поддерживаю, `anacron` используется для ежедневного, еженедельного и ежемесячного запуска заданий, даже если машина была выключена в тот момент, когда оно должно было быть запущено:

Листинг 2. Пример конфигурационного файла anacron

```
      $ cat /etc/anacrontab
      # /etc/anacrontab: configuration file for anacron
      SHELL=/bin/sh
      PATH=/sbin:/bin:/usr/sbin:/usr/bin
      # These replace cron's entries
      1         5  cron.daily    nice run-parts --report /etc/cron.daily
      7        10  cron.weekly   nice run-parts --report /etc/cron.weekly
      @monthly 15  cron.monthly  nice run-parts --report
/etc/cron.monthly
```

Содержимое crontab

Формат /etc/crontab (или содержимое файлов /etc/cron.d/) несколько отличается от пользовательских файлов `crontab`. По сути дела, они содержат одно дополнительное поле, указывающее пользователя, под которым должна запускаться данная команда. Для пользовательских `crontab` файлов это не нужно, так как они уже содержат имя пользователя в своем названии (/var/spool/cron/crontabs/$USER).

Каждая строчка /etc/crontab или устанавливает переменную окружения, или описывает задание. Комментарии и пустые строки игнорируются. Для заданий `cron` первые пять полей задают время запуска (где каждое поле может задаваться списком или диапазоном). Поля обозначают минуты, часы, дни месяца, месяцы, дни недели (разделяются пробелами или табуляциями). Asterisk (*) в любой позиции обозначает любой. Например, для запуска задания в полночь по вторникам и четвергам с августа по октябрь, следует сделать так:

```
# line in /etc/crontab
0 0 * 7-9 2,5 root /usr/local/bin/the-task -opt1 -opt2
```

Использование специальных scheduling переменных

Наиболее распространенными scheduling pattern'ам назначены сокращенные имена, которые вы можете использовать в первых пяти полях:

@reboot
: Запускать один раз, при старте.

@yearly
: Запускать один раз в год, "0 0 1 1 *".

@annually
: Тоже самое, что и @yearly.

@monthly
: Запускать один раз в месяц, "0 0 1 * *".

@weekly
: Запускать один раз в неделю, "0 0 * * 0".

@daily
: Запускать один раз в день, "0 0 * * *".

@midnight
: Тоже самое, что и @daily.

@hourly
: Запускать раз в час, "0 * * * *".

Например, конфигурационный файл может содержать:

```
@hourly root /usr/local/bin/hourly-task
0,29 * * * * root /usr/local/bin/twice-hourly-task
```

Использование crontab

Для установки пользовательских заданий используйте команду crontab (в отличие от файла /etc/crontab). Конкретнее, `crontab -e` запускает текстовый редактор для правки файла. Вы можете вывести ваш текущий список заданий при помощи `crontab -l` и удалить задание при помощи `crontab -r`. Или же вы можете задать `crontab -u user` для управления заданиями указанного пользователя `user`, но по умолчанию это будете вы (ограничения на права доступа играют свою роль).

Если в системе пристутсвует файл /etc/cron.allow, то он должен содержать имена всех пользователей, которым разрешено управление такими заданиями. С другой стороны, если файла /etc/cron.allow нет, то пользователь не должен быть помещен в файл /etc/cron.deny, если ему должно быть разрешено управление заданиями. Если ни одного из этих файлов нет, кто угодно может использовать `crontab`.

Автоматизация одноразовых заданий

Использование команды at

Если вам надо запустить задание в какое-то время, вы можете использовать

команду `at`, которая берет команды со стандартного ввода STDIN или из файла (через опцию `-f`) и принимает описание времени запуска в различных, достаточно гибких, форматах.

Семейство команд, связанных с `at` включает в себя: `atq` -- выводит список отложенных заданий; `atrm` -- удаляет задание из очереди; и `batch` -- работает подобно `at`, за исключением того, что она откладывает выполнение задания до тех пор, пока загрузка системы не будет низкой.

Права

Подобно /etc/cron.allow и /etc/cron.deny, команда `at` имеет файлы /etc/at.allow и /etc/at.deny для управления правами. Файл /etc/at.allow, если он присутствует, должен содержать всех пользователей, которым разрешено управлять запуском заданий. С другой стороны, если файла /etc/at.allow нет, пользователь должен отсутствовать в /etc/at.deny, если запуск заданий ему разрешен. Если ни одного из этих файлов не существует, все могут использовать `at`.

Указание времени

Обратитесь к справочному руководству man для получения полной информации о вашей версии `at`. Вы можете указать конкретное время в часах и минутах виде HH:MM для события, которое должно произойти, когда это время настанет. (Если это время уже прошло, это означает, что событие наступит завтра). Если вы используете 12-часовую систему измерения времени, вы можете также добавлять a.m. или p.m. Вы можете указывать дату в виде MMDDYY, MM/DD/YY, DD.MM.YY или `month-name-day`. Вы можете также прибавлять время к текущему следующим образом: `now + N units`, где N это число, а units это minutes, hours, days или weeks. Слова today и tomorrow имеют очевидное значение ("сегодня" и "завтра"), так же как midnight (полночь) и noon (полдень), teatime это 4 p.m. Несколько примеров:

```
% at -f ./foo.sh 10am Jul 31 % echo 'bar -opt' | at 1:30 tomorrow
```

Точное определение временных спецификации см. в /usr/share/doc/at/timespec.

Заметки о скриптах

Внешние ресурсы

О awk, Perl, bash и Python существует большое количество отличных книг. Соавтор это руководства (естественно) рекомендует собственное издание, Text Processing in Python , в качестве хорошей стартовой точки по написанию скриптов на Python.

Большинство скриптов, которые пишутся с ориентацией на системное администрирование, ориентируются на манипуляуию текстовыми данными, такими как извлечение величин из системных журналов и конфигурационных файлов и генерация отчетов и сводок. Сюда также относятся процедуры очистки системы и рассылка извещений о результатах исполнения заданий.

В Linux большинство обычных скриптов для системных администраторов пишутся на bash. Сам по себе bash имеет относительно мало встроенных возможностей, но зато может с легкостью использовать внешние программы (включая такие стандартные утилиты, как ls, find, rm и cd) и программы для обработки текстов (подобные тем, которые можно найти в GNU text utilities).

Заметки о bash

Одной из наиболее полезных установок, которые можно включать в bash скрипты, применяемые для обработки заданий, является опция `set -x`, выводящая исполняемые команды на стандартный вывод ошибок STDERR. Это очень полезно при отладке скриптов, когда они не выполняют того, что от них ожидается. Другая полезная для тестирования опция -- `set -n`, которая помогает отследить синтаксические проблемы в скрипте без его реального исполненеия. Разумеется, у вас не возникнет потребности использовать -n при запуске программы через `cron` или `at`, но для запуска и проверки их работоспособности это может быть полезно.

Листинг 3. Простейшее задание для cron, запускающее bash скрипт

```
#!/bin/bash
exec 2>/tmp/my_stderr
set -x
# functional commands here
```

В этом случае вывод на STDERR перенаправляется в файл и выводит запускаемые команды на STDERR. Последующее изучение этого файла может оказаться полезным.

Системное руководство man для bash конечно хорошее, но уж слишком большое. Наиболее интересными являются опции, доступные через встроенную команду set.

Обычной задачей програмирования для системного администрирования является процесс сбора файлов, как правило файлов, найденных по тем или иным критериям командой `find`. Однако, при наличии в именах файлов пробелов или символов перевода строки могут возникать проблемы. Большое количество процессов, использущих перебор в цикле и обработке имен файлов, могут исполняться некорректно при наличии пробельных символов в именах. Например, следующие две команды различаются:

```
% rm foo bar baz bam
% rm 'foo bar' 'baz bam'
```

Первая команда удаляет четыре файла (они для этого должны существовать); вторая же удаляет только два файла, каждый из которых включает пробелы в имени файла. Имена файлов с пробелами достаточно обычное явление для мультимедийного контента.

К счастью, GNU версия команды `find` имеет опцию `-print0`, ограничивающую каждый результат NULL'ем; а команда `xargs` имеет соответствующую опцию `-0` для обработки аргументов, разделенных NULL'ем. Совмещая и то и другое, вы можете удалить затерявшиеся файлы, содержащие в именах пробельные символы:

Листинг 4. Очистка имен файлов от пробелов

```
#!/bin/bash
# Cleanup some old files
set -x
find /home/dqm \( -name '*.core' -o -name '#*' \) -print0 \
 | xargs -0 rm -f
```

Perl taint mode

Perl имеет удобную опцию -T для переключения в taint mode. В этом режиме Perl предпринимает некоторые предосторожности, связанные с повышением защищенности, главной из которых является накладывание ограничений на команды, связанными с внешним вводом. Если вы используете запуск через `sudo`, taint mode может быть включен по умолчанию, но более надежным является запуск ваших административных скриптов через:

```
#!/usr/local/bin/perl -T
```

Как только вы сделаете это, все аргументы командной строки, переменные окружения, информация о locale (см. perllocale), результаты работы системных вызовов (`readdir(), readlink()`, переменные `shmread()`, сообщения порождаемые `msgrcv()`, поля `password, gcos` и `shell`, возвращаемые `getpwxxx()`) и ввод из всех файлов, помеченных как "tainted". Такие данные не могут использоваться прямо или косвенно любыми командами, ни запускаемыми через вложенный вызов командного интерпретатора, ни в любой команде, модифицирующей файлы, каталоги или процессы, за некоторыми исключениями.

Существует возможность провести отмену taint-режима (untaint) для отдельных внешних переменных через исчерпывающую проверку по определенным образцам:

Листинг 5. Untainting внешних переменных окружения

```
if ($data =~ /^([-\@\w.]+)$/) {
   $data = $1;                     # $data now untainted
} else {
   die "Bad data in $data";      # log this somewhere
```

Пакеты Perl CPAN

Одна из полезнейших вещей, доступных в Perl, -- это наличие стандартного механизма для установки дополнительных пакетов; он называется Comprehensive Perl Archive Network (CPAN). RubyGems обладает аналогичной функциональностью. Python, к несчастью, пока не обладает механизмом автоматической установки, но зато имеет достаточно широкий набор в комплекте поставки. Более простые языки, подобные `bash` и `awk`, не имеют возможностей для установки расширений, подобных описанным выше.

Страницы руководства `man` по команде `cpan` -- это хорошая отправная точка,

особенно если перед вами стоит задача, которая, как вы думаете, во многом уже решена другими. Кандидатами таких модулей могут являться те, что вы можете найти на CPAN.
`cpan` может работать в интерактивном режиме и режиме командной строки. Единожды сконфигурированный (запустите интерактивный режим и в ходе первого такого запуска у вас будут запрошены опции конфигурации), `cpan` отслеживает взаимозависти и загружает их автоматически. Например, положим перед вами стоит задача, которая требует обработки конфигурационных файлов в формате YAML (yaml Ain't Markup Language). Установка поддержки для YAML так же проста, как:

```
% cpan -i YAML # maybe with 'sudo' first
```

После установки ваши скрипты могут включать `use YAML`; в начале файла. Это позволяет вам использовать все возможности, предоставленные разработчиком пакета.