Вспомогательные материалы к лекции № 3

Основы работы в GNU/Linux

Фёдоров Станислав, ст. преп. кафедры ИУС

Сюжетная линия

- Управление заданиями
- Автоматизация работы
- Сценарии командного процессора

Управление заданиями: посылка сигналов процессу

- Нажатие Ctrl+C посылает текущему процессу сигнал SIGINT и обычно приводит к его прерыванию
- Ctrl+\ посылает процессу сигнал SIGQUIT, сигнал немедленно завершит процесс и создаст дамп памяти (core dump)
- Использование Ctrl+Z посылает сигнал SIGSTOP

Управление зданями: обработка сигналов

- Большинство сигналов в Linux отправляются программе и обрабатываются ею
  - Например, нажатие Ctrl+C посылает SIGINT процессу, а тот его обрабатывает
- Процесс может проигнорировать сигнал
- 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

Автоматизация работы: команда at

- Команда *at* обычно работает интерактивно:
  - вы сначала вызываете программу, указав время, когда задание должно выполниться,
  - затем вводите свои команды и нажимаете Ctrl+D, чтобы сохранить задание

Автоматизация работы. Пример. Команда at

user@user:~$ at midnight

at> du / > ~/diskusage

at <EOT>

job 1 at 2007-02-13 00:00

- В первой строке запускаем *at* и указываем 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 компьютере
- <EOT> – это Ctrl+D, что приводит к сохранению задания, выводу его номера и сообщению, когда оно будет выполнено (сегодня в полночь)

Автоматизация работы. Очередь заданий

- 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)

Автоматизация работы. Удаление задания

- Если вы дождётесь полуночи, ваше задание будет выполнено, как и планировалось. Но если вы передумаете, используйте команду *atrm*, чтобы удалить задание:
  - atrm 1

Автоматизация работы. Указание времени

- 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)
- 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

Автоматизация работы. Команда batch

- Если время выполнения для вас не имеет значения, 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

Автоматизация работы. Приоритет заданий, созданных batch

- 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

Вопросик

Требуется запускать задания с помощью на выполнения при загрузке системы ниже 60 % (а не ниже 80 %). Как решать такую задачу?

Автоматизация работы. Помещение заданий в файлы

- 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

Интерактивный наглядный пример

- помещение задания, использующего регулярные выражения, в файл и его запланированное выполнение через некоторое время

Сценарии (Scipts)

Строка *shebang*

- Все сценарии должны начинаться со строки "#!/bin/bash" или с указания любого другого процессора, который вы предпочитаете
- Эта строка называется shebang
- Она уведомляет оболочку о том, какой командный процессор должен использоваться для этого сценария
- Указанный путь должен быть абсолютным (вы не можете просто написать "bash", к примеру), а shebang должен находиться на первой строке скрипта без любых символов перед ним

Пример Hello World

```
#!/bin/sh
echo "Hello World"
```

- сделать сценарий исполняемым, используя команду "chmod".
  - chmod 744 firstshellscript.sh
- или
  - chmod +x firstshellscript.sh

Выполнение сценария

- Это может быть сделано вводом имени сценария в командную строку с указанием его пути
- Если сценарий находится в текущей директории, это очень просто:

  bash$ ./firstshellscript.sh

  Hello World

Пошаговое выполнение сценария

- Если вы хотите увидеть выполнение пошагово, что очень полезно для отладки - тогда выполните сценарий с опцией '-x' (что означает 'раскрыть аргументы'):

  sh -x firstshellscript.sh

  + echo 'Hello World'

  Hello World

Комментарии в сценарии

- В сценариях командного процессора все строки, начинающиеся с #, являются комментариями.

  # Это строка комментария.

  # Это еще одна строка комментария.
- Вы также можете написать многострочные комментарии, используя двоеточие и одинарные кавычки:

  : 'Это комментарий.

  Это снова комментарий.

  Кто бы мог подумать, что это еще один комментарий.'
- Примечание: Это не будет работать, если в комментариях будут символы одинарной кавычки

Системные переменные

- Системные переменные определяются и хранятся в окружении *родительского процессора*. Они также называются переменными окружения. Имена этих переменных состоят из заглавных букв, и могут быть показаны командой 'set'. Примерами системных переменных являются PWD, HOME, USER. Значения этих системных переменных могут быть показаны по отдельности командой 'echo'. Например 'echo $HOME' выведет значение, хранящееся в системной переменной HOME
- Когда устанавливаете системную переменную, не забудьте использовать команду 'export', чтобы сделать ее доступной *дочерним процессорам* (любые процессоры, которые запущены из текущего, включая сценарии):

  bash$ SCRIPT_PATH=/home/blessen/shellscript

  bash$ export SCRIPT_PATH
- Современные командные процессоры также позволяют сделать это одной командой:

  bash$ export SCRIPT_PATH=/home/blessen/shellscript

Пользовательские переменные

- Вмена не могут начинаться с цифр, записываются в нижнем регистре и вместо пробела используется знак подчеркивания

  define_tempval=blessen
- Не должно быть пробелов перед или за знаком равенства

  bash$ echo $define_tempval

  blessen
- Следующий сценарий устанавливает переменную "username" и показывает ее значение.

  #!/bin/sh

  username=blessen

  echo "username - $username"

Аргументы командной строки

- Это переменные, которые содержат аргументы выполняемого сценария. Доступ к этим переменным может быть получен через имена $1, $2, ... $n, где $1 первый аргумент командной строки, $2 второй, и так далее. Аргументы располагаются после имени сценария и разделены пробелами. Переменная $0 - это имя сценария. Переменная $# хранит количество аргументов командной строки, это количество ограничено 9 аргументами в старых шеллах и практически неограниченно в современных.
- Рассмотрим сценарий, который возьмет два аргумента командной строки и покажет их. Мы назовем его 'commandline.sh':

  #!/bin/sh

  echo "Первый агрумент - $1"

  echo "Второй агрумент - $2"
- Если запустить 'commandline.sh' с аргументами командной строки "blessen" и "lijoe" результат будет таким:

  bash$ ./commandline.sh blessen lijoe
- Первый аргумент - blessen
- Второй аргумент - lijoe

Переменная кода возврата

- Эта переменная сообщает нам, была ли последняя команда успешно выполнена. Она обозначается $?. Нулевое значение означает, что команда была успешно выполнена
- Любые другие числа означают, что команда была выполнена с ошибкой (также некоторые программы, такие как 'mail', используют ненулевое значение возврата для отображения состояния, а не ошибки)
- Таким образом, это очень полезное свойство при написании сценариев

Область видимости переменной

- При написании сценариев процессоров видимость переменных используется для различных задач
- В процессорах есть два типа видимости: глобальная и локальная
- Локальные переменные определяются, используя ключевое слово "local" перед именем переменной, все остальные переменные, кроме связанных с аргументами функции, - глобальные, и поэтому доступны из любого места сценария

Область видимости переменной

- Сценарий, приведенный ниже демонстрирует различные видимости локальной и глобальной переменной:

```
#!/bin/sh
display()
{
   local local_var=100
   global_var=blessen
   echo "локальная переменная внутри функции равна $local_var"
   echo "глобальная переменная внутри функции равна $global_var"
}
echo "====внутри функции===="
display
echo "=====вне функции======"
echo "локальная переменная вне функции равна $local_var"
echo "глобальная переменная вне функции равна $global_var"
```

-

Область видимости переменной

- Запущенный сценарий выводит следующий результат:
  ====внутри функции====
  локальная переменная внутри функции равна 100
  глобальная переменная внутри функции равна blessen
  ======вне функции======
  локальная переменная вне функции равна
  глобальная переменная вне функции равна blessen
- Заметьте отсутствие значения для локальной переменной вне функции

Ввод-вывод в сценариях

- Для ввода с клавиатуры используется команда 'read'. Эта команда считает значения, набранные на клавиатуре, и присвоит каждое определенной переменной.

  read <имя_переменной>
- Для вывода используется команда 'echo'.

  echo <имя_переменной>

Арифметические операции

- sum=`expr 12 + 20`
- echo $[12 + 10]
- echo $(12 + 10)

Оператор условия "if"

```
#!/bin/sh
echo "Введите имя пользователя:"
read username
if [ "$username" = "blessen" ]
then
 echo 'Успешно!!! Вы зашли.'
else
 echo 'Извините, неправильное имя пользователя.'
fi
```

- Не забывайте всегда закрывать переменную в условии в двойные кавычки; если этого не сделать ваш сценарий вызовет ошибку синтаксиса, когда переменная будет пуста.

Сравнение переменных

- В сценариях процессора можно выполнять различные сравнения. Если значения сравниваемых переменных являются числами, вы должны использовать следующие опции:
- -eq Равно
  -ne Не равно
  -lt Меньше
  -le Меньше или равно
  -gt Больше
  -ge Больше или равно
- Если переменные являются строками, вы должны использовать эти опции:
  - = Равно
  - != Не равно
  - < Первая строка отсортирована перед второй
  - > Первая строка отсортирована после второ

Цикл "for"

- Самый часто используемый цикл - цикл "for". В сценариях существует два типа: первый аналогичен циклу "for" в языке программирования Си, а второй является циклом итерации (обработки списков).
- Синтаксис первого типа цикла "for" (этот тип доступен только в современных процессорах):

  for ((<начальное значение>; <условие>; <инкремент/декремент>))
  do
   <любые выражения или операторы>
  done
- Синтаксис второго, более распространенного типа цикла "for":

  for <переменная> in <список>
  do
   <любые выражения или операторы>
  done

Цикл "while"

- Цикл "while" - еще один полезный цикл, используемый во всех языках программирования. Цикл продолжает выполняться до тех пор, пока условие перестанет быть верным.

```
while [ <условие> ]
do
 <любые выражения или операторы>
done
```

- Следующий сценарий присваивает значение "1" переменной "num" и прибавляет единицу к "num" каждый раз, когда "num" меньше 5.

```
#!/bin/sh
num=1
while [$num –lt 5]
do
num=$[$num + 1];
echo $num;
done
```

Операторы "select" и "case"

- Подобно конструкции "switch/case" в языке программирования Си комбинация операторов "select" и "case" обеспечивает сценариям такую же функциональность. Оператор "select" не является частью конструкции "case", но оба приведены для иллюстрации того, как они могут быть использованы в сценариях.
- Синтаксис оператора select:

  select <переменная> in <список>

  do

  <любые выражения или операторы>

  done
- Синтаксис оператора case:

  case $<переменная> in

  <выбор1>) <любые выражения или операторы> ;;

  <выбор2>) <любые выражения или операторы> ;;

  *) echo "Извините, неправильный выбор" ;;

  esac

Функции

- Синтаксис функции:
  <имя_функции> ()
  { # начало функции
   <операторы>
  } # конец функции
- Функции вызываются, когда их имя встречается в коде программы, возможно с аргументами, располагающимися за именем функции. Например:

```
#!/bin/sh
sumcalc ()
{
 sum=$[$1 + $2]
}
echo "Введите первое число:"
read num1
echo "Введите второе число:"
read num2
sumcalc $num1 $num2
echo "Сумма чисел: $sum"
```

Отладка сценариев процессоров

- Теперь нам нужно отлаживать наши программы. Для этого мы используем опции '-x' и '-v' шелла. Опция '-v' выводит больше информации о ходе выполнения программы. Опция '-x' подробно распишет каждую простую команду, цикл "for", оператор "case", оператор "select" или арифметический оператор, показывая значение PS4, за каждой командой и ее аргументы или список слов. Попробуйте их - они могут быть очень полезны, когда вы не можете понять, где находится проблема в вашем сценарии

Вопросик

Какие недостатки несёт интерпретация сценариев?

Вопросик

Почему в сценариях не реализуется поддержка чисел с плавающей запятой?

Вопросик

Как ускорить работу, выполняемую сценарием?

Спасибо за внимание