Применение Expect в администрировании системы Linux: Часть 2. Практическое применение Expect в системном администрировании

Алексей Снастин
независимый разработчик ПО
начальник отдела

11.03.2010

В первой статье рассматривались общие характеристики Expect, принципы и механизмы его функционирования. В данной статье описывается практическое применение Expect и некоторые аспекты его использования.

Несколько замечаний по использованию Expect

Прежде чем начать исследование административных скриптов, необходимо обратить внимание на некоторые вопросы, возникающие при использовании Expect в реальной работе.

Одна из самых существенных проблем: сама по себе программа Expect не обладает способностью распознавать промпты (строки приглашения) различных командных оболочек, а необходимость в этом возникает достаточно часто. Решить эту задачу помогает составление регулярного выражения, описывающего конечную часть строки промпта (именно конечную, потому что настройки всей строки приглашения отличаются большим разнообразием), и сохранение этого выражения в переменной. Можно использовать следующий фрагмент кода:

```
set prompt "(%|#|\\$) $";
catch {set prompt $env(EXPECT_PROMPT)}
...
expect -re $prompt
```

В первой строке переменной prompt присваивается регулярное выражение. Следует отметить, что для записи регулярных выражений применяется их диалект в языке Tcl. В скобках сгруппированы альтернативные варианты промпт-символов: % – для оболочек Cshell, # – для режима суперпользователя root, $ – для прочих оболочек. Символ "доллар" приходится экранировать, так как в регулярных выражениях он обозначает конец строки. Кроме того, необходимо защитить и сам символ "обратный слэш" от интерпретации его командной оболочкой. Далее следует символ "пробел", за ним признак конца строки ($).

Во второй строке выполняется встроенная команда `Tcl catch`, которая позволяет выполнить заданный код и вернуть управление в вызывающий скрипт даже при возникновении ошибки. В какой-то мере это аналог перехвата исключений в таких языках, как C++, C#, Java. Внутри блока `catch` выполняется присваивание переменной `prompt` значения из массива переменных среды `env` с индексом `EXPECT_PROMPT`. Иными словами, даже если переменной среды `EXPECT_PROMPT` не существует или она не содержит значения, выполнение скрипта не будет прервано, а переменная prompt сохранит значение, присвоенное ей в первой строке.

После этого в любой части скрипта, где ожидается приглашение командной оболочки, можно использовать инструкцию `expect -re $prompt`, в которой ключ `-re` предупреждает о том, что значение переменной prompt следует интерпретировать как регулярное выражение, а не как обычную строку.

Ещё одна особенность – все переменные, задействованные в командах Expect, по умолчанию являются локальными, если для них не указан явно модификатор `global`. Следствием этого является проблема, возникающая в случае выполнения команды spawn внутри процедуры. После выхода из такой процедуры идентификатор порождённого процесса `spawn_id` становится "невидимым" (область его видимости ограничена телом процедуры), следовательно, невозможно получить доступ и к самому процессу. В подобных ситуациях перед вызовом spawn необходимо вставить команду `global spawn_id`.

Примеры административных скриптов

Достаточно часто перед администратором встаёт задача создания большой группы новых пользователей. Вводить данные вручную, чередуя команды useradd и passwd, – весьма утомительное занятие. Конечно, у каждого опытного администратора есть своё "секретное оружие" – мне доводилось видеть хитроумные скрипты, которые в той или иной мере автоматизируют процесс создания учётных записей. Но знание Expect позволяет администратору решить ту же задачу с меньшими трудозатратами. Ниже приведён пример, в котором имена и пароли вновь создаваемых учётных записей пользователей считываются из заранее подготовленного текстового файла. Разумеется, такой файл с конфиденциальной информацией должен храниться в защищённом месте, но в данном примере для простоты предположим, что файл находится в текущем рабочем каталоге и имеет следующий вид.

Содержимое текстового файла newusers:

```
anna gp63r2d4
boris iyb1z9sd
victor fm571jq0
...
sergei e2h5mw7z
tatyana pv92x3au
```

А сам скрипт выглядит так:

```
#!/usr/bin/expect
set ufile [open "./newusers" r]
foreach uline [split [read $ufile] "\n"] {
  set username [lindex $uline 0]
  set password [lindex $uline 1]
```

```
  if { [string length $username] > 0 } {
    spawn "/bin/bash"
    send "useradd $username\r"
    expect -re "# $"
    sleep 1
    log_user 0
    send "passwd $username\r"
    expect "Новый пароль UNIX : "
    send "$password"
    expect "Повторите ввод нового пароля UNIX : "
    send "$password"
    expect " успешно обновлены."
    log_user 1
    puts stdout "Пользователь $username создан.\r"
  }
}
close $ufile
```

Начинается скрипт с процедуры открытия файла, содержащего имена и пароли пользователей, в режиме "только для чтения". Дескриптор открываемого файла присваивается переменной `ufile`.

Во второй строке инициализируется цикл перебора элементов списка foreach с переменной цикла `uline`. Поскольку файл считывается в виде непрерывного потока символов, мы воспользуемся командой языка Tcl для разделения этого потока символов на элементы списка с помощью символа-разделителя "`\n`" (переход на новую строку). Таким образом, каждая строка файла newusers становится элементом списка и поочерёдно, в порядке считывания, передаётся в переменную `uline`, что является обязательным требованием конструкции `foreach`.

С точки зрения языка Tcl (следовательно, и с точки зрения Expect) содержимое переменной `uline` также является списком, т.е. в паре "`имя_пользователя пароль`" имя является элементом списка `uline` с индексом `0`, а пароль – элементом с индексом `1`. Этим мы воспользуемся, чтобы создать ещё две переменные – `username` и `password`, извлекая значения для них по соответствующему индексу (Tcl-команда `lindex`).

Проверка длины имени пользователя необходима потому, что если в конце файла newusers случайно окажется пустая строка, то последним в перебираемом списке будет пустой элемент, который следует исключить из обработки.

Для каждого элемента, успешно прошедшего проверку, порождается процесс /bin/bash (если у вас не установлена эта командная оболочка, то внесите необходимые изменения). В этот процесс посылается команда добавления нового пользователя с указанием текущего имени, содержащегося в переменной username. Мы должны дождаться появления символа приглашения командной оболочки. Команда "`sleep 1`" приостанавливает выполнение на 1 секунду. Это не обязательно, но на современных сверхбыстрых компьютерах с многоядерными процессорами позволяет избежать некоторых "неожиданностей".

Команда "`log_user 0`" отключает вывод протокола диалога send-expect на экран. Далее следует диалог ввода и подтверждения пароля для только что созданного пользователя,

после чего вывод диалоговых сообщений снова включается (`log_user 1`) и выводится сообщение об успешном создании учётной записи пользователя.

Когда все записи из файла newusers считаны и обработаны, происходит выход из цикла `foreach`. Остаётся лишь закрыть файл и завершить выполнение скрипта.

Данный скрипт можно поместить в `crontab` и запланировать его выполнение во время минимальной загрузки системы (например, в ночное время), когда массовое создание новых пользователей никому не будет мешать.

Чтобы не утруждать себя сочинением десятков уникальных паролей, можно воспользоваться скриптом `mkpasswd`, входящим в стандартный комплект установки пакета Expect. Этот скрипт позволяет регулировать длину генерируемого пароля, количество цифровых символов, букв верхнего и нижнего регистра, специальных символов. Но в этом случае сгенерированные и назначенные пароли нужно каким-либо образом сохранять, чтобы потом сообщить их пользователям. Кстати, `mkpasswd` умеет также присваивать созданный пароль существующему пользователю.

Вместе с программой Expect устанавливаются и другие скрипты. Например, passmass выполняет смену пароля на нескольких сетевых компьютерах одновременно. Скрипт `dislocate` позволяет отключать процессы от терминала и вновь подключать их, а `unbuffer` запрещает буферизацию вывода. Чрезвычайно интересен скрипт `kibitz`, обеспечивающий взаимодействие двух (и более) пользователей с одной командной оболочкой. Для каждого из этих и прочих "штатных" скриптов имеется man-страница, а кроме того, вы можете изучать исходные коды самих скриптов, что является наилучшим способом освоения Expect по мнению самого Дона Либиса (исправляю неточность в написании его фамилии в первой статье цикла).

Заключение

В данной статье было описано практическое применение Expect и приведены примеры скриптов для решения задач системного администрирования.

В следующей статье цикла мы рассмотрим работу Expect в сетевой среде.

Об авторе

Алексей Снастин

Алексей Снастин - независимый разработчик ПО, консультант и переводчик с английского языка технической и учебной литературы по ИТ. Принимал участие в разработке сетевых офисных приложений типа клиент/сервер на языке C в среде Linux.