Применение Expect в администрировании системы Linux: Часть 3.Практическое применение Expect в сетевом администрировании

Алексей Снастин
независимый разработчик ПО
начальник отдела

25.03.2010

В этом цикле статей описаны основные функциональные принципы и рассмотрены вопросы практического применения Expect, а также приведены примеры скриптов для решения задач системного администрирования.

1. Введение

Сразу отмечу, что приведенные в статье примеры скриптов представляют собой предельно упрощённые решения задач сетевого администрирования. Это сделано преднамеренно – для удобства восприятия читателями. После того как вы освоите синтаксис и семантику команд, усовершенствовать эти скрипты можно будет самостоятельно.

2. Выполнение команды настройки на удалённом компьютере

Достаточно часто в локальной сети возникает необходимость выполнения команды настройки среды на какой-либо рабочей станции. Рассмотрим такую ситуацию: на одном из компьютеров периодически требуется синхронизация даты и времени из-за неполадок аппаратного таймера.

Эту проблему поможет решить следующий скрипт. Предположим, что с датой и временем некорректно обращается компьютер по имени comp052.

© Copyright IBM Corporation 2010

```
01: #!/usr/bin/expect
02: spawn ssh <имя_пользователя>@comp052
03: expect "password:"
04: send "<пароль_пользователя>\r"
05: expect -re "\\$ $"
06: send "su\r"
07: expect "(Password:|Пароль:)"
08: send "<пароль_root>\r"
09: expect -re "# $"
10: exec date > /tmp/datesync.tmp
11: exec cat /tmp/datesync.tmp
12: set newtime [exec cat /tmp/datesync.tmp]
13: send "date -s \"$newtime\"\r"
14: expect -re "# $"
15: send "exit\r"
16: expect -re "\\$ $"
17: send "logout\r"
18: expect "closed."
19: exit 0
```

Со всеми командами вы уже знакомы по примерам в предыдущих статьях цикла. Отмечу лишь, что двухступенчатая процедура регистрации в качестве обычного пользователя и последующего перехода в ранг суперпользователя обусловлена требованиями безопасности при работе через ssh. Практически во всех системах соблюдается рекомендация: запретить непосредственный вход суперпользователя root по протоколу SSH.

Команда вывода содержимого файла /tmp/datesync.tmp (строка 11) на локальную консоль позволяет зафиксировать в системном журнале время выполнения данного скрипта.

Подразумевается, что вместо условных "шаблонов" <имя_пользователя>, <пароль_пользователя> <пароль_root> подставляются реально существующие на обслуживаемом компьютере (в данном примере – comp52) имя, пароль и пароль суперпользователя соответственно.

Описанный выше скрипт можно поместить в системную таблицу crontab и запланировать его выполнение с требуемой периодичностью.

3. Автоматизация выполнения команды rsync на удалённом компьютере

Ещё одна задача сетевого администрирования – резервное копирование определённых данных и параметров настройки на сетевых компьютерах. В небольшой сети для администратора не составит труда выполнить несколько команд rsync вручную. А если сеть состоит из нескольких десятков или даже сотен машин? Снова зовём на помощь Expect.

В следующем примере команда rsync использует протокол SSH – безопасность превыше всего.

Скрипт принимает два аргумента: имя хоста, с которого производится резервное копирование, и пароль суперпользователя root на этом хосте.

```
01: #!/usr/bin/expect -f
02: if {[llength $argv] != 2} {
```

```
03:   puts "Вызов: auto_rsync.exp <ИМЯ_ХОСТА> <ПАРОЛЬ_ROOT>"
04:   exit 1
05: }
06: set hostname [lindex $argv 0]
07: set password [lindex $argv 1]
08: set timeout -1
09: spawn date
10: expect -re "# $"
11: spawn rsync -av -e ssh $hostname:/etc /archive/sys
12: expect "password:" {send "$password\r"}
13: expect -re "# $"
14: spawn date
15: expect -re "# $"
16: spawn rsync -av -e ssh $hostname:/usr/etc /archive/sys
17: expect "password:" {send "$password\r"}
18: expect -re "# $"
19: spawn date
20: expect -re "# $"
21: spawn rsync -av -e ssh $hostname:/usr/work /archive/works
22: expect "password:" {send "$password\r"}
23: expect -re "# $"
24: spawn date
25: expect -re "# $"
26: exit 0
```

Здесь необходимы некоторые пояснения. В строке идентификации (01) использован флаг -f, непосредственно предваряющий файл, из которого считываются команды. Собственно говоря, он как раз и предназначен для #!-нотации с тем, чтобы в командной строке могли быть заданы другие аргументы. В нашем случае такими аргументами являются имя хоста и пароль суперпользователя.

В строках 02–05 выполняется проверка наличия названных выше аргументов, поскольку без них скрипт работать не будет.

С инициализацией переменных (строки 06 и 07) вы уже знакомы, а вот в строке 08 для переменной timeout задаётся значение -1, которое означает, что прерывания выполнения по тайм-ауту не будет, т.е. время ожидания не ограничено.

Фиксация даты и времени (строка 09 и далее по тексту) будет выполняться перед началом каждой операции и после её окончания.

В строках 11, 16 и 21 выполняется команда rsync, в которой флаг -a инициализирует режим архивирования, флаг -v позволяет выводить подробную информацию о ходе выполнения команды, а флаг -e предоставляет возможность задать командную оболочку: в нашем случае это ssh.

Скрипт можно выполнять из командной строки или запланировать его выполнение с помощью системных средств (cron или at), или поместить в скрипт-"обёртку" командной оболочки, в котором используется цикл для перебора всех необходимых хостов.

4. Организация совместной работы пользователей

Как уже было отмечено ранее, в состав установочного пакета Expect включены уже готовые скрипты, предназначенные для выполнения задач администрирования. Один из этих скриптов – kibitz – предоставляет весьма любопытные возможности.

Дон Либис с изрядной долей юмора дал такое имя (в переводе с английского kibitz – вмешиваться в чужие дела; давать непрошеные советы) скрипту, который позволяет двум или даже нескольким пользователям одновременно работать с одной программой: это может быть командная оболочка, текстовый редактор и т.д. Ограничение только одно – это должны быть программы с символьным (текстовым) интерфейсом. Запуск программ с графическим интерфейсом возможен, но при этом не гарантируется нормальный вид и поведение таких программ. Впрочем, это ограничение устраняется с помощью ещё одного скрипта xkibitz, который позволяет нормально работать с X-программами, а кроме того, обеспечивает динамическое подключение и отключение пользователей во время работы.

Для того чтобы начать работу, пользователь (назовём его user1) запускает скрипт, передавая ему в качестве аргумента имя другого пользователя, с которым он намеревается взаимодействовать (пусть это будет user2):

```
kibitz user2
```

При этом активизируется новый экземпляр командной оболочки (или какая-либо другая программа, которую можно задать в командной строке), а пользователю user2 предлагается также выполнить скрипт kibitz. В приглашение включается уникальный идентификатор для того, чтобы избежать конфликтов с другими сеансами kibitz. В действительности этот идентификатор представляет не что иное, как просто идентификатор исходного процесса kibitz. Второй пользователь получает приблизительно такое сообщение:

```
Can we talk? Run: kibitz -4077
EOF
```

После того как user2 выполнит указанную в сообщении команду (kibitz -4077), все нажатия клавиш обоих пользователей становятся вводом для общей командной оболочки. Соответственно, оба пользователя получают вывод оболочки.

Для того чтобы завершить сеанс kibitz, достаточно просто выйти из "совместной" командной оболочки, нажав клавиши Ctrl-D или введя команду exit. После выхода одного из пользователей происходит автоматический выход из оболочки и второго пользователя, в результате чего сеанс kibitz завершается.

В сетевой среде, когда вызываемый пользователь работает на другом хосте, команда инициализации сеанса kibitz выглядит следующим образом:

```
kibitz user2@host02
```

Канал связи создаётся с помощью утилиты rlogin. Если удалённый компьютер запрашивает пароль, то kibitz перехватывает этот запрос и предлагает вызывающему пользователю ввести требуемый пароль. Все прочие подробности установления соединения скрыты от пользователей.

После установления соединения и инициализации сеанса kibitz вторым пользователем всё выглядит так, как если бы оба пользователя работали на одном компьютере. Завершение сеанса производится точно так же, как описано выше.

Если необходимо организовать сеанс с участием более двух пользователей, то можно воспользоваться таким вариантом команды:

```
kibitz user2@host02 kibitz user3@host03 kibitz user4@host04
```

Такой "массовый сеанс" вполне пригоден, например, для обучения и для консультаций, когда опытный пользователь наглядно показывает начинающим, как выполняются различные операции, следит за их действиями, отвечает на вопросы и т.п. Разумеется, для организации подобного сеанса необходимо, чтобы на всех хостах-участниках был установлен пакет программ Expect.

Запуск консольного текстового редактора в kibitz-сеансе позволяет провести "интерактивное совещание" с ведением протокола или организовать совместное редактирование документов.

4.1. Использование kibitz как команды, встраиваемой в Expect-скрипты

Несмотря на то что kibitz является интерактивным скриптом (а с другой стороны – именно благодаря этому), его можно применять как команду, включаемую в сценарии автоматизации, написанные на Expect.

Предположим, что имеется Expect-скрипт, запускаемый сервисом cron или переведённый в фоновый режим выполнения. Обобщённая схема такого скрипта может выглядеть следующим образом:

```
01: spawn <некоторый_процесс>; set proc_id $spawn_id
    ...
    <Expect-команды>
    ...
    # предположим, что в ходе выполнения возможно возникновение проблемы
    # или вопроса, требующего вмешательства пользователя
02: spawn kibitz -noproc <имя_пользователя>
03: send "Возникла ситуация, требующая вмешательства пользователя\n"
04: interact -u $proc_id
05: send "Проблема решена, продолжаю выполнение\n"
```

Здесь при инициализации сеанса kibitz (строка 02) используется флаг "-noproc", который позволяет обойтись без создания нового процесса, а вместо этого напрямую подключить

создаваемый сеанс к текущему процессу. Таким образом, обеспечивается доступ заданного пользователя к выполняющемуся процессу, в котором возникла проблема.

Возможность выполнения пользователем необходимых действий предоставляется командой interact (строка 04). Эта Expect-команда устанавливает связь между двумя "сущностями". Первая "сущность" – это пользователь, которому передаётся управление процессом. Вторая "сущность" определяется с помощью флага "-u" с аргументом, которым обязательно должен быть идентификатор текущего процесса, требующего взаимодействия с пользователем.

После того как пользователь выполнит все необходимые действия и завершит интерактивный kibitz-сеанс, выдаётся соответствующее сообщение (строка 05), управление возвращается Expect-скрипту и его работа продолжится.

5. Заключение

Expect можно считать диалектом языка программирования Tcl, специализированным для выполнения задач администрирования. Основное предназначение Expect – объединение двух и более интерактивных программ и обеспечение их функционирования без участия пользователя.

С помощью всего лишь нескольких базовых команд можно создавать скрипты различной степени сложности, обеспечивающие взаимодействие многочисленных программ и автоматизацию их выполнения.

В этом цикле статей были описаны основные функциональные принципы и рассмотрены вопросы практического применения Expect, а также приведены примеры скриптов для решения задач системного администрирования.

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.

© Copyright IBM Corporation 2010
(www.ibm.com/legal/copytrade.shtml)
Торговые марки
(www.ibm.com/developerworks/ru/ibm/trademarks/)