

Rogério Ferreira

Automatizando Tarefas ao Extremo Com Shell e Expect

open**SUSE**[™]

Linux para mentes abertas

<http://rogerioferreira.objectis.net>
rogeriotux@gmail.com



Liberdade de escolha



www.opensuse.org

Rogério Ferreira

Inaugurou a Seção de Segurança da Revista Linux Magazine e a Seção Programando.com da Revista Locaweb. Foi autor de Projeto de Software Livre em Governo Estadual e teve participação em Projeto de Zope e Plone no Governo Federal. Palestrante em importantes eventos de Software Livre, como PyCon Brasil, CONISLI, FLISOL e FISL. Possui as Certificações, LPIC-1, LPIC-2, LPIC-3 e LPI-302 (Mixed Environment). É Idealizador e Instrutor do Treinamento Hands-On, Virtualização Profissional com Xen, da Linux Solutions. Em parceria com A Casa do Linux criou o HowToDay. É embaixador do openSUSE.



Automatizar Tarefas

A necessidade de automatizar processos e tarefas é algo natural do curso da evolução humana. Máquinas reduziram um número expressivo de pessoas em linhas de produção no decorrer dos anos, e sistemas de informática substituíram processos exaustivos, executados por inúmeros indivíduos.

Administradores de Sistemas e Desenvolvedores de Software, tem grande necessidade de automatizar tarefas ao extremo, para que tenham mais controle do seu ambiente e a intervenção humana seja cada vez mais reduzida, para que falhas provenientes de mentes exaustas e desatentas sejam minimizadas ou quem sabe eliminadas (eu sei que isso não é possível...).

Linguagens de Script

Hoje em dia temos um arsenal espetacular de linguagens de script para uso geral, que possibilitam nos aproximarmos desta meta (reduzir problemas). Linguagens como: Shell, Python, Ruby, TCL, Lua, e o velho e “discriminado” Perl, e muitas outras. Cada uma destas linguagens tem suas peculiaridades e no que se propõem a fazer, são muito poderosas. Não existe solução única para problemas, assim como doenças, que muitas vezes são tratadas com diversos medicamentos diferentes, podemos solucionar nossos problemas informáticos com uma destas linguagens ou com a mistura de algumas delas.

Expect

Com esta proposta de automatizar tarefas em sistemas UNIX (eu diria UNIX-Like), Don Libes, um cientista de computação do NIST (National Institute of Standards and Technology), criou “Expect” em 1990, como uma extensão para a linguagem de scripting TCL, para interagir com aplicações como telnet, ftp, passwd, ssh, e outros. Expect possui a mesma convenção de sintaxe que TCL, acrescido de comandos específicos (**spawn**, **expect** e **send**, que são os mais usados) para interação com aplicações.

Expect é um programa para controle de aplicações interativas. Estas aplicações interagem por meio de um prompt e esperam que o usuário entre com algumas teclas como resposta. Imagine os passos na ação de mudar a senha de um usuário num sistema Linux...

Expect (Cont.)

```
$ passwd
```

```
Changing password for user ferreira.
```

```
Changing password for ferreira
```

```
(current) UNIX password: *****
```

```
New UNIX password: *****
```

```
Retype new UNIX password: *****
```

```
passwd: all authentication tokens updated successfully.
```

Note a interação que foi feita acima:

1-> O usuário disparou o comando `passwd` no terminal e teclou `enter`;

2-> O prompt esperou que o usuário digitasse a sua senha atual e teclasse `enter`;

3-> O usuário digitou a senha atual e teclou `enter`;

4-> O prompt esperou que o usuário digitasse a sua nova senha e teclasse `enter`;

5-> O usuário digitou a sua nova senha e teclou `enter`;

6-> O prompt esperou que o usuário repetisse a sua nova senha e teclasse `enter`;

7-> O usuário digitou novamente a sua nova senha e teclou `enter`.

Expect (Cont.)

Um script em Expect para executar a intervenção anterior, teria a seguinte aparência:

```
#!/usr/bin/expect -f

set timeout -l

spawn /usr/bin/passwd
expect "(current) UNIX password: "
send "SUASENHAATUAL\r"
expect "New UNIX password: "
send "SUANOVASENHA\r"
expect "Retype new UNIX password: "
send "SUANOVASENHA\r"

expect eof
```

Ao automatizar programas interativos, você será capaz de resolver problemas que você nunca teria sequer considerado antes. Expect pode lhe poupar horas de trabalho.

Instalando o Expect

Para instalar o Expect no openSUSE é muito simples:

```
# # zypper install expect
```

E no CentOS não é diferente:

```
# yum install expect
```

No Debian se assemelha também:

```
# aptitude install expect
```

Após a instalação você pode acessar o interpretador do Expect, para conhecer um pouco sobre ele:

```
$ expect
```

```
expect|.l> puts "Alo, Brasil!"
```

```
Alo, Brasil!
```

Conhecendo um pouco de TCL

O comando `puts` é usado pra imprimir valores. Podemos definir uma variável e imprimir o valor desta variável com `puts`:

```
expectl.2> set ip "192.168.0.100"
```

```
192.168.0.100
```

```
expectl.3> puts "O IP do servidor de backup eh $ip"
```

```
O IP do servidor de backup eh 192.168.0.100
```

Com o comando `set` é possível definir variáveis, listas e vetores. O caractere `$` substitui o nome da variável por seu conteúdo.

Para definir uma lista é muito simples:

```
expectl.4> set uf {"SP" "RJ" "RS" "PR" "MG" "AM"}
```

```
"SP" "RJ" "RS" "PR" "MG" "AM"
```

Conhecendo um pouco de TCL (Cont.)

Para acessar o posição 4 (sendo que o valor do primeiro elemento da lista é 0) da lista usamos o comando `lindex` seguido do nome da lista:

```
expectl.5> lindex $uf 3  
PR
```

Para acessarmos da posição 1 à 4 da lista usamos o comando `lrange`:

```
expectl.6> lrange $uf 1 4  
RJ RS PR MG
```

NOTA: Para sair no interpretador use o comando `exit`.

Conhecendo um pouco de TCL (Cont.)

Podemos iterar pela lista com foreach:

```
expectl.7> foreach estado $uf {  
+> puts $estado  
+> }  
SP  
RJ  
RS  
PR  
MG  
AM
```

Se quisermos imprimir somente o estado que satisfaça nossa exigência, podemos usar o if:

```
expectl.8> foreach estado $uf {  
+> if {$estado == "AM"} {  
+> puts $estado  
+> }  
+> }  
AM
```

Automatizando o rsync

```
#!/usr/bin/expect -f

# Define o timeout. O -l significa que vai ficar aguardando uma resposta por tempo indeterminado.
set timeout -l

# Define as variaveis
set user [lindex $argv 0]
set host [lindex $argv 1]
set rdir [lindex $argv 2]
set ldir [lindex $argv 3]
set pass [lindex $argv 4]

# Executa o rsync
spawn rsync -avz ${user}@${host}:${rdir} ${ldir}

# Caso ainda nao tenha sido baixada a chave do host remoto eh passado o "yes" e em seguida quando
# solicitada a senha, a mesma eh passada para o comando. Caso a chave jah senha sido baixada,
# somente a senha eh passada para o comando.
expect {
    "*re you sure you want to continue connecting*"
    {
        send "yes\n"
        expect {
            "*assword*"
            {
                send "${pass}\n"
            }
        }
    }
    "*assword*"
    {
        send "${pass}\n"
    }
}

# Finaliza a instrucao
expect eof
```

Automatizando o rsync (Cont.)

O comando `spawn` inicia a interação com o programa em questão (`rsync`). `argv` é uma lista que contém os argumentos passados na linha de comando.

Agora podemos testar o nosso script:

```
# chmod +x sinc.exp  
# ./sinc.exp "usuario" "host.exemplo.com" "/remote-dir/" "/local-dir/" "senha"
```

Caso você se sinta desconfortável em passar a senha como parâmetro de linha comando, você pode colocá-lo dentro do próprio script, ou até mesmo armazená-lo dentro de um arquivo com permissões restritas. Ou caso você goste de codificar, pode escrever um script que criptografe o arquivo de senhas e no momento de passá-lo como parâmetro para o script, possa descriptografá-lo.

Monitorando o SSH

```
#!/usr/bin/expect -f

# Define as variaveis
set user [lindex $argv 0]
set host [lindex $argv 1]
set pass [lindex $argv 2]
set time [lindex $argv 3]

# Define o timeout passado por parametro
set timeout ${time}

# Executa o comando date remoto
# Soh para testar o ssh
spawn ssh ${user}@${host} date

expect {
    "*re you sure you want to continue connecting*"
    {
        send "yes\n"
        expect {
            "*assword*"
            {
                send "${pass}\n"
            }
        }
    }
    "*assword*"
    {
        send "${pass}\n"
    }
    timeout
    {
        exit 1
    }
    "Connection refused"
    {
        exit 2
    }
}
# Finaliza a instrucao
expect eof
```

Monitorando o SSH (Cont.)

Agora podemos testar o nosso script:

```
$ chmod +x monssh.exp
```

```
$ ./monssh.exp "usuario" "host.exemplo.com" "senha" "20"
```

Se no intervalo de segundos definido como parametro o host responder, a variável \$? vai retornar 0:

```
$ echo $?
```

```
0
```

Caso contrário, vai retonar 1, conforme definimos no script (timeout {exit 1}):

```
$ echo $?
```

```
1
```

Juntando Shell com Expect

Programa que copia arquivos de texto de um host remoto para o host local, com a data do dia anterior:

```
#!/bin/sh
```

```
HOST="$1"
```

```
USER="$2"
```

```
PASS="$3"
```

```
DIR_SRC="$4"
```

```
DIR_DST="$5"
```

```
DATE=$(date --date='yesterday' +%F)
```

```
EXT="$6"
```

Juntando Shell com Expect (Cont.)

```
conn_ssh_list() {
  expect -c "
  set timeout -l
  spawn ssh $USER@$HOST ls -ltr --full-time $DIR_SRC/*. $EXT
  expect {
    \"*re you sure you want to continue connecting*"
  {
    send \ "yes\n\ "
    expect {
      \"*assword*"
    {
      send \ "$PASS\n\ "
    }
  }
}
    \"*assword*"
  {
    send \ "$PASS\n\ "
  }
}
  expect eof"
}
```

Juntando Shell com Expect (Cont.)

```
conn_ssh_copy() {
  FILES=$(conn_ssh_list | grep $DATE | awk '{print $9}')

  for FILE in $FILES
  do
    expect -c "
      set timeout -l
      spawn scp $USER@$HOST:$FILE $DIR_DST/
      expect {
        \"*re you sure you want to continue connecting*"
        {
          send \n\n
          expect {
            \"*assword*"
            {
              send \n\n
            }
          }
          \"*assword*"
          {
            send \n\n
          }
        }
      }
    expect eof"
  done
}
```

conn_ssh_copy

Escrevendo um Plugin para o Nagios com Expect e FreeTDS

- Dependências
 - FreeTDS
 - Expect

/caminho-do-plugin/mon_sql_agent

```
#!/bin/sh

HOST="$1"
TIMESEC=50
USER="USUARIOSQL"
PASS="SENHAUSUARIOSQL"
TODAY=$(date +%Y%m%d)
SQL="select count(*) from msdb.dbo.sysjobhistory where run_status = 0 and run_date = $TODAY"

exec_sql() {
expect -c "set timeout $TIMESEC
spawn tsq -S $HOST -U $USER
expect {
    \"Password: \"
        {
            send \"$PASS\r\"
            expect \">\"
            send \"$SQL\r\"
            send \"go\r\"
            expect \">\"
            send \"exit\r\"
        }
    timeout
    {
        puts \"\n\"
    }
}
expect eof"
}
```

/caminho-do-plugin/mon_sql_agent (Cont.)

```
RESULT=$(exec_sql | tail -n +9 | egrep -v '(!>|affected)' | sed 's/.5//')
```

```
case "$RESULT"
```

```
in
```

```
0)
```

```
    echo "MSSQL Agent OK"
```

```
    exit 0
```

```
;;
```

```
[1-9]*)
```

```
    echo "Algum job falhou no MSSQL Agent"
```

```
    exit 1
```

```
;;
```

```
*)
```

```
    echo "Possivel problema de conectividade com o host"
```

```
    exit 2
```

```
;;
```

```
esac
```

Incluindo o Plugin no Nagios

```
# Plugin para Monitorar MSSQL Agent
define command{
    command_name check_mssql_agent_CLIENTE X
    command_line /caminho-do-script/mon_sql_agent host-que-serah-monitorado
}
# MSSQL Agent
define service{
    service_description Monitora MSSQL Agent do CLIENTE X
    host_name host-que-serah-monitorado
    check_command check_mssql_agent_CLIENTE_X
    servicegroups grp-sqlagent
    use srv-generico
}
```

Mais informações

- Artigo em Revista sobre Expect, pp. 52-55:
 - <http://www.locaweb.com.br/downloads/Locaweb20.pdf>
- Treinamentos e Consultoria em Software Livre (Hands-On Automatizando Tarefas com Shell e Expect):
 - www.howtoday.com.br
 - www.acasadolinux.com.br
- Treinamento e Consultoria Oficial em Novell SUSE Linux Enterprise:
 - www.komputer.com.br