<https://br-linux.org/artigos/dicas_expect.htm>

# Linux in Brazil (Expect )

### A FERRAMENTA DE PROGRAMAÇÃO EXPECT

**por João Alexandre Lôbo Marques (alexandre@prce.mpf.gov.br)**

*Nota do editor: embora o autor tenha baseado sua pesquisa na distribuição de Linux da Conectiva, o expect é uma ferramenta disponível por padrão na maior parte das distribuições.*

Se você trabalha com scripts e tem a necessidade de desenvolver rotinas em linguagens de scritp que estabeleçam comunicação entre servidores, EXPECT é uma ferramenta que poderá lhe ajudar e muito. Em primeiro lugar porque é uma ferramenta que torna simples procedimentos difíceis de serem implementados remotamente, tais como automatizar telnet, rlogin ou fsck. Em segundo lugar porque é simples e já vem junto com o Conectiva LINUX.

Dando uma definição mais formal, EXPECT é uma ferramenta para automatizar aplicações que exijam alguma interatividade entre máquinas clientes e servidoras, podendo ser citadas diversas dessas aplicações tais como telnet, ftp, passwd, fsck, rlogin, tip, etc. A ferramenta EXPECT realmente transforma esta tarefa em algo simples e trivial, além de também poder ser utilizada para testar estes mesmos tipos de aplicação. E outro aspecto muito interessante é que com a adição da ferramenta Tk, podem ser desenvolvidas aplicações para rodar em interfaces gráficas (GUI) X11.

O site oficial EXPECT ( [The Expect Home Page](http://expect.nist.gov/)) é um excelente ponto de partida para conhecer melhor como funciona a ferramenta. Apesar de não ser uma tradução pura e simples, baseamos este trabalho neste site para trazer para nossa língua portuguesa informações sobre mais esta ferramenta.

Podem ser encontradas diversas fontes de documentação para ajudar um usuário interessado em trabalhar com a EXPECT. A seguir está uma lista de tópicos interessantes:

Você poderá pegar o EXPECT além dos exemplos no próprio site oficial expect.nist.gov. O download poderá ser feito nos formatos expect.tar.Z ou expect.tar.gz. O arquivo gz é mais simples de se trabalhar e seu download é mais rápido. No entanto, se você preferir, há também a versão Z da distribuição. Dependendo da escolha, na linha de comando, digite:

uncompress expect.tar.Z

tar -xvf expect.tar

ou

tar -xzvf expect.tar.gz

Isto criará um diretório contendo a distribuição do EXPECT. Entre neste diretório e siga as instalação do arquivo README.

ATENÇÃO: O EXPECT requer que já esteja instalada no sistema a ferramenta Tcl. Se você ainda não possui o Tcl instalado, você poderá baixá-lo a partir dos links tcl.tar.Z or gz no site oficial. Para instalar, descompacte e siga as instruções do arquivo README, da mesma forma que com o EXPECT.

Outros itens interessantes do site oficial são:

**\* Exploring Expect é um livro da O'Reilly que serve como excelente fonte para o uso do Expect. Possui centenas de exemplos e também inclui um tutorial sobre o Tcl**

\* Expect FAQ (Frequently Asked Questions List) - Este FAQ é bastante interessante e traz informações gerais e dados técnicos sobre o uso do EXPECT.

\* Lista de exemplos que vem com o próprio Expect - Esta é uma simples lista. Existem exemplos mais atualizados que podem ser obtidos fazendo-se o download da distribuição completa do EXPECT. No entanto, esta lista é bastante útil para o usuário sentir o poder de fogo do EXPECT e ver na prática como ele realmente simplifica tarefas.

\* Arquivo EXPECT - Aqui você encontra scripts gerais de contribuição voluntária

\* The Expect README - Este é o próprio arquivo de README que vem com a instalação do EXPECT. O README tem diversas informações que podem ser bastante úteis. A princípio, há uma descrição da distribuição do software, tudo sobre bug reports, suporte, e outrs informações.

\* História do desenvolvimento do EXPECT e diferenças entre versões. A versão atual do EXPECT (segundo o site expect.nist.gov até a publicação deste arquivo é a 5.31.5, criada em 06/03/2000 - Mon Mar 06 10:35:04 EST 2000.

\* Links para informações relacionadas ao EXPECT

\* Além disso, não deixe de olhar a "man page" do EXPECT (man expect)

### CARACTERÍSTICAS DO EXPECT

**A melhor forma de aprendermos uma linguagem é realmente trabalharmos com ela na prática. Por isso, não será dada uma lista de todos os comandos com suas funções e parâmetros. Isto pode ser encontrado com um pouco de pesquisa nas páginas de manual do EXPECT ou nos sites indicados.**

Será feita uma breve apresentação dos principais comandos e características e em seguida partiremos logo para alguns exemplos no intuito de dar uma visão geral desta ferramenta de programação.

Iniciando uma apresentação da linguagem temos que a primeira linha de um script deve ser sempre a mostrada logo a seguir:

#!/caminho/para/expect -f

No caso do meu sistema Conectiva LINUX 4.0 está em " #!/usr/bin/expect -f ". Você poderá localizar onde está o interpretador do seu EXPECT através do comando:

$find / -name expect

Esta primeira chamada indica onde está o interpretador do EXPECT e o ativa para verificar e executar o que está contido no arquivo do script em questão.

I - A atribuição de valores a variáveis segue o padrão de outras linguagens de script para shell, utilizando o comando "set":

set valor

II - Da mesma forma que nas linguagens Perl ou PHP, a referência a uma variável deve sempre ser precedida do caractere "$".

III - O comando para executar um programa no shell atual é o "spawn"

IV - O comando para enviar uma mensagem para a tela do usuário é o "send\_user"

V- Para enviar um pacote com uma string ou comando utiliza-se o "send"

VI- Existem rotinas de teste de "timeout" de finalização e de erro inerentes ao sistema. Isto quer dizer que pode-se fazer referência por exemplo a um erro padrão ocorrido apenas referenciando-se no código com a adição da cláusula:

erro

{

código de tratamento

}

### EXEMPLO -

EXECUTANDO UM COMANDO EM SEGUIDA AO SU

Este é um programa que visa automatizar um comando após o usuário entrar em uma seção "su". Pode ser utilizado por exemplo para listar o diretório automaticamente.

O script se chamará "su2" e o comando desejado deve ser passado como parâmetro na linha de comando. Assim, para o exemplo de listagem de diretório, o comando a ser dado deverá ser:

$su2 ls

A listagem do script com diversos comentários está a seguir:

#!/usr/bin/expect -f   
#Lembre que a primeir linha sempre deverá referenciar o interpretador expect   
set timeout -1   
log\_user 0 ;# Faz com que o programa não esteja em modo de login do usuário   
spawn su ;# Executa o comando su   
system stty -echo ;# Prepara para receber a senha do usuário. Desabilita o echo do teclado   
send\_user "Password: " ; #Mostra ao usuário a string   
expect\_user -re "(.\*)\n" ;#Recebe a digitação da senha   
set password $expect\_out(1,string) ;# Coloca na variável $password o valor digitado pelo usuário   
system stty echo ;# Retorna o echo de caractere   
expect "\*Password:" ;# Apaga o prompt de requisição de senha   
log\_user 1 ;# Faz com que o programa entre em modo de login do usuário   
send "$password\r" ;# Envia a variável $password   
expect \*Sorry\* exit \*#\* ;# Trata algum erro que possa acontecer e sai do programa   
eval send [lrange $argv 1 end] \\r ;# Envia o argumento passado para ser executado   
interact ;# Permite que o usuário interaja com o novo shell e envie novos comandos

### EXEMPLO

MONITORANDO UMA REDE E AVISANDO AO ADMINISTRADOR

#!/usr/bin/expect -f   
# Este script executa o comando ping para uma lista prévia de hosts   
# Se algum estiver down, envia um PAGER para o administrador do sistema   
# Atenção: Este exemplo poderá ter que ser ajustado ao sistema de pager utilizado.   
# Deverá funcionar facilmente. No entanto, caso haja problemas   
# podem ser encontradas informações adicionais com scripts específicos para pagers   
# no próprio site de Archive do Expect (expect.nist.gov)   
#############################   
# Executar ping para cada host   
# #############################   
foreach host $argv   
{   
spawn ping $host ;# Executa o ping para o host em questão   
  
set timeout 2 ;# Determina um timeout de 2 segundos   
expect "alive" ;# Testa se recebeu resposta   
{   
continue ;# Se sim, continua o loop "foreach"   
}   
timeout ;# Rotina de tratamento do timeout   
{   
lappend deadhosts $host ;# Grava o host que não respondeu   
}   
close wait   
}   
##########################   
# Envia Mensagem via Pager   
# ##########################   
if [info exists deadhosts] ;# Testa se algo foi gravado em "deadhosts" na rotina de timeout acima.   
{   
spawn tip modem ;# Executa o programa tip   
expect "connected"   
send "ATD1234567\r" ;# Envia a string de discagem   
expect "user:"   
send "user123\r" ;# Envia o nome do usuário   
expect "msg:"   
send "not responding to ping: $deadhosts\r" ;# Envia a mensagem informando o(s) host(s) fora do ar   
expect eof   
}

### CONCLUSÃO

A idéia deste artigo foi apresentar o EXPECT de uma maneira simples e direta para quem nunca tinha ouvido falar nele (como eu mesmo há algum tempo atrás) e estava com esta potente ferramenta embaixo do nariz, assim como também para dar uma visão prática para aqueles que já haviam ouvido falar mas não tinham começado a criar aplicações para seu sistema.   
Conforme já foi dito, o EXPECT realmente é uma ferramenta que pode ser bastante útil caso você seja um administrador de sistemas que se trabalhe com diversos servidores, ou mesmo que necessite automatizar tarefas (e qual administrador não precisa?).   
Especificamente no meu caso, implementei diversas rotinas, tais como login remoto, telnet e outras aplicações que perdi muito tempo buscando soluções sem sucesso. Boa sorte para os felizardos que partirem para trabalhar neste time.

**por João Alexandre Lôbo Marques (alexandre@prce.mpf.gov.br)**

<https://ivanix.wordpress.com/2013/07/09/comando-expect/>

## [Comando expect](https://ivanix.wordpress.com/2013/07/09/comando-expect/)

[09/07/2013](https://ivanix.wordpress.com/2013/07/09/comando-expect/)[Nix](https://ivanix.wordpress.com/author/ivanix/)

Recentemente, tive que fazer a instalação de um software em diversos (ok, muitos) servidores. Imediatamente, pensei em automatizar a tarefa quando me deparei com um problema: a instalação era feita em modo interativo, isto é, esperava receber algumas respostas. Pesquisando no google e conversando com algumas pessoas, fui apresentada ao expect [**1**], que me ajudou a resolver o problema (viva a comunidade opensource! \o/).

O expect é uma ferramenta para automatizar um processo que recebe comando interativos, isto é, que exibe um prompt e espera que o usuário digite alguma resposta. Alguns exemplos de aplicações interativas são: passwd, ftp, fsck, ssh, entre outras.

Três comandos compõe a estrutura do expect: spawn, expect e send e o uso dele é simples. O comando spawn inicia o processo, expect aguarda por uma sequência de strings de um processo e send envia uma string para um processo (comandos, senhas e outros).

**Instalação**

expect é uma extensão Tcl [**2**] e pode ser instalado em praticamente todas distribuições Linux, seja através de um gerenciador de pacotes como por exemplo apt-get no Debian, yum no Red Hat ou ainda através do código fonte [**3**]. No meu caso, estou utilizando o Linux Mint (derivado do Ubuntu, que por sua vez deriva do Debian) e para instalar executei:

**$ sudo apt-get install expect**

**Como o expect trabalha**

Scripts expect são escritos como a maioria de outros scripts. Assim como no Bash ou Perl, onde a primeira linha com o shebang[**4**] indica o programa ou shell que será utilizado para executar os comandos do script, expect também utiliza essa formatação. Para localizar o binário do expect, basta utilizar o comando which:

**$ which expect**   
**/usr/bin/expect**

É necessário saber o path do binário pois essa informação será colocada na primeira linha do script.

A maioria dos scripts começam com o processo que se quer interagir, como por exemplo “spawn ssh ivani@localhost” ou “spawn ftp localhost”. Em uma forma simples, o processo é iniciado com o comando spawn que gera uma saída onde o expect irá procurar padrões para em seguida, com base nos resultados, enviar comandos através do comando send.

**O problema**

Pacote que durante a instalação faz algumas perguntas tais como aceite de licença e path da instalação.

**Solução encontrada**

**1)** Criação da lista de hosts onde será feita a instalação.

**2)** Criação do script install\_app.exp com os comandos a serem executados e as respostas esperadas. Lembrando que o Linux não reclama com esse negócio de extensão, apenas utilizei .exp para ficar organizado e eu saber que se trata de um script expect.

**3)** Criação de um shell script que executa o script install\_app.exp em cada host que consta na lista criada no item 1.

No lab, a simulação da instalação do pacote foi feita com um instalador java antigo (jre-6u7-linux-i586.bin) que tem o mesmo comportamento da aplicação que precisei instalar.

**1) Lista de hosts**

Arquivo texto com os hosts listados linha a linha. Exemplo:

**$ cat hosts.txt**   
**192.168.1.100**

**2) Script install\_app.exp**

Corpo do script expect que faz a instalação remota da aplicação;  linhas foram numeradas para facilitar a explicação.

**1 #!/usr/bin/expect -f**  
**2 set HOST [lindex $argv 0]**  
**3 exp\_internal 1**  
**4 spawn ssh ivani@$HOST**  
**5 expect “%”**  
**6 send “/home/ivani/jre-6u7-linux-i586.bin \r”**  
**7 send “%q”**  
**8 expect “% EOF”**  
**9 send “yes \r”**  
**10 expect “% Done.”**  
**11 send “exit \r”**  
**12 expect eof**

**Linha 1** – #!/usr/bin/expect

Indica o path do interpretador de comandos que será utilizado.

**Linha 2** – set HOST [lindex $argv 0]

Atribui variáveis. Scripts criados com o expect devem ter o formato Tcl; assim, não é possível utilizar variáveis como $1, por exemplo, para pegar parâmetros passados na linha de comando. É necessário utilizar a variável argv junto com o comando [lindex $argv 0], que faz com o primeiro parâmetro indicado na linha de comando (no caso, o nome do host) seja colocado na variável $HOST.

**Linha 3** – exp\_internal 1

Ativa a função de debug e irá mostrar todas as ações que o expect está realizando. Para desligar basta retirar a linha do script ou simplesmente trocar 1 por 0.

**Linha 4** – spawn ssh ivani@$HOST

Inicia o processo da aplicação interativa, no caso, o ssh.

**Linha 5** – expect “%”

Aguarda uma instrução (nesse caso, o prompt).

**Linha 6** – send “/home/ivani/jre-6u7-linux-i586.bin \r”

Envia um comando para ser executado remotamente, e oparâmetro \r é o <enter> esperado.

**Linha 7** – send “%q”

No pacote que está sendo instalado, o comando more é acionado para exibir uma licença e é necessário pressionar espaço para ler ou ir direto para o fim do arquivo, onde o usuário será questionado se quer ou não aceitar. Aqui, foi enviado “q” para sair da licença.

**Linhas – 8 e 9:**

expect “% EOF”  
send “yes \r”

Quando se pressiona “q” para sair da licença, é enviado um “EOF” indicando saída da leitura da licença e aparece para o usuário se quer ou não aceitar a licença; nesse ponto, o script envia a resposta yes.

**Linhas 10 e 11:**

expect “% Done.”  
send “exit \r”

Ao encontrar o termo “Done”, é enviado o comando exit para desconectar da sessão ssh.

**Linha 12** – expect eof

Garante que o script aguarde a execução dos comandos antes de retornar o controle para o script .sh e finalizar a conexão SSH.

**3) Script install\_java.sh**

Shell script que chama o script expect para instalar o pacote nos servidores.

Esse shelll é bem simples, foi criado um laço que irá ler o arquivo hosts.txt e para cada linha nesse arquivo, será executado o script java.exp.

A seguir o script:

**1 #!/bin/bash -x**  
**2**   
**3 for host in $(cat hosts.txt)**  
**4 do**   
**5 /home/ivani/Scripts\_Learning/java.exp $host**  
**6 done**

Abaixo, trechos da execução do script install\_java.sh  (partes foram suprimidas para o post não ficar tãão longo…).

ivani@nix ~/Scripts\_Learning $ **pwd**  
**/home/ivani/Scripts\_Learning**  
ivani@nix ~/Scripts\_Learning $ **./install\_java.sh**   
++ cat hosts.txt  
+ for host in ‘$(cat hosts.txt)’  
+ /home/ivani/Scripts\_Learning/java.exp 192.168.1.100  
**spawn ssh ivani@192.168.1.100**  
Linux lab 2.6.32-5-686 #1 SMP Fri Feb 15 15:48:27 UTC 2013 i686

The programs included with the Debian GNU/Linux system are free software;  
the exact distribution terms for each program are described in the  
individual files in /usr/share/doc/\*/copyright.

[…]  
Last login: Tue Jul 9 20:58:21 2013 from 192.168.1.101  
ivani@lab:~$ **/home/ivani/jre-6u7-linux-i586.bin**

Sun Microsystems, Inc. Binary Code License Agreement

for the JAVA SE RUNTIME ENVIRONMENT (JRE) VERSION 6

SUN MICROSYSTEMS, INC. (“SUN”) IS WILLING TO LICENSE THE  
SOFTWARE IDENTIFIED BELOW TO YOU ONLY UPON THE CONDITION  
THAT YOU ACCEPT ALL OF THE TERMS CONTAINED IN THIS BINARY  
CODE LICENSE AGREEMENT AND SUPPLEMENTAL LICENSE TERMS  
(COLLECTIVELY “AGREEMENT”). PLEASE READ THE AGREEMENT  
CAREFULLY. BY DOWNLOADING OR INSTALLING THIS SOFTWARE, YOU  
ACCEPT THE TERMS OF THE AGREEMENT. INDICATE ACCEPTANCE BY  
SELECTING THE “ACCEPT” BUTTON AT THE BOTTOM OF THE  
AGREEMENT. IF YOU ARE NOT WILLING TO BE BOUND BY ALL THE  
TERMS, SELECT THE “DECLINE” BUTTON AT THE BOTTOM OF THE  
AGREEMENT AND THE DOWNLOAD OR INSTALL PROCESS WILL NOT  
CONTINUE.

[…]

Do you agree to the above license terms? [yes or no]  
**yes**   
Unpacking…  
Checksumming…  
Extracting…  
UnZipSFX 5.50 of 17 February 2002, by Info-ZIP (Zip-Bugs@lists.wku.edu).  
creating: jre1.6.0\_07/  
creating: jre1.6.0\_07/bin/  
inflating: jre1.6.0\_07/bin/java   
[…]

**Done.**  
ivani@lab:~$ **exit**   
logout  
**Connection to 192.168.1.100 closed.**  
ivani@nix ~/Scripts\_Learning $

Até o próximo post :)!  
**Referências**

[1] <http://en.wikipedia.org/wiki/Expect>

[2] <http://en.wikipedia.org/wiki/Tcl>

[3] <http://www.nist.gov/el/msid/expect.cfm>

[4] <http://en.wikipedia.org/wiki/Shebang_(Unix)>

[Getting Started With Expect](http://oreilly.com/catalog/expect/chapter/ch03.html) – O’Reilly

<https://en.wikipedia.org/wiki/Expect>

# Expect

From Wikipedia, the free encyclopedia

|  |  |
| --- | --- |
| [https://upload.wikimedia.org/wikipedia/en/thumb/9/99/Question_book-new.svg/50px-Question_book-new.svg.png](https://en.wikipedia.org/wiki/File:Question_book-new.svg) | This article **needs additional citations for** [**verification**](https://en.wikipedia.org/wiki/Wikipedia:Verifiability). Please help [improve this article](https://en.wikipedia.org/w/index.php?title=Expect&action=edit) by [adding citations to reliable sources](https://en.wikipedia.org/wiki/Help:Introduction_to_referencing_with_Wiki_Markup/1). Unsourced material may be challenged and removed. *(May 2017)* *(*[*Learn how and when to remove this template message*](https://en.wikipedia.org/wiki/Help:Maintenance_template_removal)*)* |

|  |  |
| --- | --- |
| Expect | |
| [**Original author(s)**](https://en.wikipedia.org/wiki/Software_developer) | [Don Libes](https://en.wikipedia.org/wiki/Don_Libes) |
| [**Developer(s)**](https://en.wikipedia.org/wiki/Software_developer) | Nils Carlson |
|  | |
| [**Last release**](https://en.wikipedia.org/wiki/Software_release_life_cycle) | 5.45.1 / 15 August 2012; 5 years ago |
| **Development status** | Discontinued |
| **Written in** | [Tcl](https://en.wikipedia.org/wiki/Tcl) |
| [**Operating system**](https://en.wikipedia.org/wiki/Operating_system) | [POSIX](https://en.wikipedia.org/wiki/POSIX), [Windows](https://en.wikipedia.org/wiki/Microsoft_Windows) |
| [**License**](https://en.wikipedia.org/wiki/Software_license) | [Public domain](https://en.wikipedia.org/wiki/Public_domain)[[1]](https://en.wikipedia.org/wiki/Expect#cite_note-1) |
| **Website** | [expect.sourceforge.net](http://expect.sourceforge.net/) |

This article is about the Unix tool. For a definition of the word "expect", see the Wiktionary entry [expect](https://en.wiktionary.org/wiki/expect).

**Expect**, an extension to the [Tcl](https://en.wikipedia.org/wiki/Tcl) scripting language written by [Don Libes](https://en.wikipedia.org/wiki/Don_Libes), is a program to automate interactions with programs that expose a [text terminal](https://en.wikipedia.org/wiki/Text_terminal) interface. Expect was originally written in 1990 for [Unix](https://en.wikipedia.org/wiki/Unix) systems, but since became available for [Microsoft Windows](https://en.wikipedia.org/wiki/Microsoft_Windows) and other systems.

* ***[icon](https://en.wikipedia.org/wiki/File:Green_bug_and_broom.svg)***[***Software Testing portal***](https://en.wikipedia.org/wiki/Portal:Software_Testing)

## Contents

* [1 Basics](https://en.wikipedia.org/wiki/Expect#Basics)
* [2 Usage](https://en.wikipedia.org/wiki/Expect#Usage)
* [3 Alternatives](https://en.wikipedia.org/wiki/Expect#Alternatives)
  + [3.1 C#](https://en.wikipedia.org/wiki/Expect#C.23)
  + [3.2 Java](https://en.wikipedia.org/wiki/Expect#Java)
  + [3.3 Scala](https://en.wikipedia.org/wiki/Expect#Scala)
  + [3.4 Groovy](https://en.wikipedia.org/wiki/Expect#Groovy)
  + [3.5 Perl](https://en.wikipedia.org/wiki/Expect#Perl)
  + [3.6 Python](https://en.wikipedia.org/wiki/Expect#Python)
  + [3.7 Ruby](https://en.wikipedia.org/wiki/Expect#Ruby)
  + [3.8 Shell](https://en.wikipedia.org/wiki/Expect#Shell)
  + [3.9 Go](https://en.wikipedia.org/wiki/Expect#Go)
* [4 References](https://en.wikipedia.org/wiki/Expect#References)
* [5 Further reading](https://en.wikipedia.org/wiki/Expect#Further_reading)
* [6 External links](https://en.wikipedia.org/wiki/Expect#External_links)

## Basics

Expect is used to automate control of interactive [applications](https://en.wikipedia.org/wiki/Application_software) such as [Telnet](https://en.wikipedia.org/wiki/Telnet), [FTP](https://en.wikipedia.org/wiki/FTP), [passwd](https://en.wikipedia.org/wiki/Passwd_(command)), [fsck](https://en.wikipedia.org/wiki/Fsck), [rlogin](https://en.wikipedia.org/wiki/Rlogin), [tip](https://en.wikipedia.org/wiki/Tip_(unix_utility)), [SSH](https://en.wikipedia.org/wiki/Secure_Shell), and others. Expect uses [pseudo terminals](https://en.wikipedia.org/wiki/Pseudo_terminal) (Unix) or emulates a console (Windows), starts the target program, and then communicates with it, just as a human would, via the terminal or console interface. [Tk](https://en.wikipedia.org/wiki/Tk_(framework)), another Tcl extension, can be used to provide a [GUI](https://en.wikipedia.org/wiki/GUI).

Expect has [regular expression](https://en.wikipedia.org/wiki/Regular_expression) pattern matching and general program capabilities, allowing simple scripts to intelligently control programs such as Telnet, FTP, and SSH, all of which lack a [programming language](https://en.wikipedia.org/wiki/Programming_language), [macros](https://en.wikipedia.org/wiki/Macro_(computer_science)), or any other program mechanism.

## Usage

Expect serves as a "glue" to link existing utilities together. The general idea is to try to figure out how to make Expect utilize the system's existing tools rather than figure out how to solve a problem inside of Expect.

A key usage of Expect involves commercial software products. Many of these products provide some type of command-line interface, but these usually lack the power needed to write [scripts](https://en.wikipedia.org/wiki/Scripting_programming_language). They were built to service the users administering the product, but the company often does not spend the resources to fully implement a robust scripting language. An Expect script can spawn a shell, look up environmental variables, perform some Unix commands to retrieve more information, and then enter into the product's command-line interface armed with the necessary information to achieve the user's goal. After looking up information inside the product's command-line interface, the script can make an intelligent decision about what action to take, if any.

Every time an Expect operation is completed, the results are stored in a local variable called $expect\_out. This allows the script to harvest information to feed back to the user, and it also allows conditional behavior of what to send next based on the circumstances.

A common use of Expect is to set up a testing suite, whether it be for programs, utilities or embedded systems. [DejaGnu](https://en.wikipedia.org/wiki/DejaGnu) is a testing suite written using Expect for use in testing. It has been used extensively for testing [GCC](https://en.wikipedia.org/wiki/GNU_Compiler_Collection) and is very well suited to testing remote targets such as embedded development.

One can automate the generation of an expect script using a tool called 'autoexpect'. This tool observes your actions and generates an expect script using heuristics. Though generated code may be large and somewhat cryptic, one can always tweak the generated script to get the exact code.

# Assume $remote\_server, $my\_user\_id, $my\_password, and $my\_command were read in earlier

# in the script.

# Open a telnet session to a remote server, and wait for a username prompt.

spawn telnet $remote\_server

expect "username:"

# Send the username, and then wait for a password prompt.

send "$my\_user\_id\r"

expect "password:"

# Send the password, and then wait for a shell prompt.

send "$my\_password\r"

expect "%"

# Send the prebuilt command, and then wait for another shell prompt.

send "$my\_command\r"

expect "%"

# Capture the results of the command into a variable. This can be displayed, or written to disk.

set results $expect\_out(buffer)

# Exit the telnet session, and wait for a special end-of-file character.

send "exit\r"

expect eof

Another example is a script that automates FTP:

# Set timeout parameter to a proper value.

# For example, the file size is indeed big and the network speed is really one problem,

# you'd better set this parameter a value.

set timeout -1

# Open an ftp session to a remote server, and wait for a username prompt.

spawn ftp $remote\_server

expect "username:"

# Send the username, and then wait for a password prompt.

send "$my\_user\_id\r"

expect "password:"

# Send the password, and then wait for an ftp prompt.

send "$my\_password\r"

expect "ftp>"

# Switch to binary mode, and then wait for an ftp prompt.

send "bin\r"

expect "ftp>"

# Turn off prompting.

send "prompt\r"

expect "ftp>"

# Get all the files

send "mget \*\r"

expect "ftp>"

# Exit the ftp session, and wait for a special end-of-file character.

send "bye\r"

expect eof

Below is an example that automates SFTP (with password):

#!/usr/bin/env expect -f

# procedure to attempt connecting; result 0 if OK, 1 otherwise

proc connect {passw} {

expect {

"Password:" {

send "$passw\r"

expect {

"sftp\*" {

return 0

}

}

}

}

# timed out

return 1

}

#read the input parameters

set user [lindex $argv 0]

set passw [lindex $argv 1]

set host [lindex $argv 2]

set location [lindex $argv 3]

set file1 [lindex $argv 4]

set file2 [lindex $argv 5]

#puts "Argument data:\n";

#puts "user: $user";

#puts "passw: $passw";

#puts "host: $host";

#puts "location: $location";

#puts "file1: $file1";

#puts "file2: $file2";

#check if all were provided

if { $user == "" || $passw == "" || $host == "" || $location == "" || $file1 == "" || $file2 == "" } {

puts "Usage: <user> <passw> <host> <location> <file1 to send> <file2 to send>\n"

exit 1

}

#sftp to specified host and send the files

spawn sftp $user@$host

set rez [connect $passw]

if { $rez == 0 } {

send "cd $location\r"

set timeout -1

send "put $file2\r"

send "put $file1\r"

send "ls -l\r"

send "quit\r"

expect eof

exit 0

}

puts "\nError connecting to server: $host, user: $user and password: $passw!\n"

exit 1

it should be noted that using passwords as command-line arguments, like in this example, is a huge security hole, as any other user on the machine can read this password by running "[ps](https://en.wikipedia.org/wiki/Ps_(Unix)" \o "Ps (Unix))". You can, however, add code that will prompt you for your password rather than giving your password as an argument. This should be more secure. See the example below.

stty -echo

send\_user -- "Enter Password: "

expect\_user -re "(.\*)\n"

send\_user "\n"

stty echo

set PASS $expect\_out(1,string)

Another example of automated ssh login in user machine:

#timeout is a predefined variable in expect which by default is set to 10 sec

#spawn\_id is another default variable in expect.

#It is good practice to close spawn\_id handle created by spawn command

set timeout 60

spawn ssh $user@machine

while {1} {

expect {

eof {break}

"The authenticity of host" {send "yes\r"}

"password:" {send "$password\r"}

"\*\]" {send "exit\r"}

}

}

wait

close $spawn\_id

## Alternatives

Various projects implement Expect-like functionality in other languages, such as C#, Java, Scala, Groovy, Perl, [Python](https://en.wikipedia.org/wiki/Python_(programming_language)), [Ruby](https://en.wikipedia.org/wiki/Ruby_(programming_language)), Shell and Go. These are generally not exact clones of the original Expect, but the concepts tend to be very similar.

### C#

* [Expect.NET](http://blog.iwanek.eu/expect-net/) — Expect functionality for C# (.NET)
* [DotNetExpect](https://github.com/CBonnell/dotnetexpect) — An Expect-inspired console automation library for .NET

### Java

* [expect4j](https://github.com/cverges/expect4j) — an attempt at a Java clone of the original Expect
* [ExpectJ](http://expectj.sourceforge.net/) — a Java implementation of the Unix expect utility
* [Expect-for-Java](https://github.com/ronniedong/Expect-for-Java) — pure Java implementation of the Expect tool
* [expect4java](https://github.com/iTransformers/expect4java)  - a Java implementation of the Expect tool, but supports nested clousures. There is also wrapper for Groovy language DSL.

### Scala

* [scala-expect](https://github.com/Lasering/scala-expect) — a Scala implementation of a very small subset of the Expect tool.

### Groovy

* [expect4groovy](https://github.com/iTransformers/expect4groovy)  - a Groovy DSL implementation of Expect tool.

### Perl

* [Expect.pm](http://sourceforge.net/projects/expectperl) — [Perl](https://en.wikipedia.org/wiki/Perl) module (newest version at [metacpan.org](https://metacpan.org/pod/Expect))

### Python

* [Pexpect](https://github.com/pexpect/pexpect) — [Python](https://en.wikipedia.org/wiki/Python_(programming_language)) module for controlling interactive programs in a pseudo-terminal
* [winpexpect](https://pypi.python.org/pypi/winpexpect) — port of pexpect to the Windows platform

### Ruby

* [RExpect](http://rubyforge.org/projects/rexpect) — a drop in replacement for the expect.rb module in the standard library.
* [Expect4r](https://github.com/jesnault/expect4r) — Interact with Cisco IOS, IOS-XR, and Juniper JUNOS CLI

### Shell

* [Empty](http://empty.sourceforge.net) — expect-like utility to run interactive commands in the UNIX shell-scripts

### Go

* [GoExpect](https://github.com/google/goexpect) - expect-like package for the Go language

## References

* 1. [*"Expect FAQ: Our company policy requires a license to use Expect. Where can we get a license?"*](http://expect.sourceforge.net/FAQ.html#q6).

## Further reading

* Libes, Don (1995). Exploring Expect: A Tcl-Based Tool for Automating Interactive Programs. O'Reilly & Associates, Inc. [*ISBN*](https://en.wikipedia.org/wiki/International_Standard_Book_Number) [*1-565-92090-2*](https://en.wikipedia.org/wiki/Special:BookSources/1-565-92090-2).
* "[Advanced Programming in Expect: A Bulletproof Interface](http://www.cotse.com/dlf/man/expect/bulletproof1.htm)"

## External links

* [Official website](http://expect.sourceforge.net/)
* [Expect](https://sourceforge.net/projects/expect/) on [SourceForge.net](https://en.wikipedia.org/wiki/SourceForge)
* [Expect page](http://wiki.tcl.tk/201) — on The Tcler's Wiki
* [IBM Blog](https://www.ibm.com/developerworks/community/blogs/brian/entry/when_to_use_expect_scripting_and_when_to_avoid_it10?lang=en)

<http://wiki.tcl.tk/17378>

["expect\_out(buffer) has the content of the previous send"](http://wiki.tcl.tk/_/ref?N=17378)

Updated 2015-03-05 18:21:32 by [Jorge](http://wiki.tcl.tk/22837) [▲](http://wiki.tcl.tk/_/diff?N=17378#diff0)

The titular quote is something newcomers to [Expect](http://wiki.tcl.tk/201) often say--as one of [Expect's frequently-made mistakes](http://wiki.tcl.tk/18013), it indicates a misconception or two.

This explanation deserves its own [Wiki](http://wiki.tcl.tk/11) page: Expect is organized in terms of a dialogue. It does not have a direct notion of "the content of the previous send"; instead, we all have the expectation that, after a command is sent, what appears before the next prompt is the result of that command. That's a convention, though. From Expect's perspective, it's all just characters going back and forth, with no privileged concept of "command", "prompt", or so on.

Let's do a simple example: what do you expect from

set prompt {$ }

spawn ssh $user@$host

expect "Password: "

send $password\r

expect $prompt

send ls\r

puts "The output is '$expect\_out(buffer)'."

? You'll probably see something like the **$host** logon message.

To achieve what people think they want, it's necessary to write instead

spawn ssh $user@$host

expect "Password: "

send $password\r

expect $prompt

send ls\r

expect $prompt

puts "The output is '$expect\_out(buffer)'."

Do you see the difference?

It's sometimes useful to

expect \* ; # Receive \*anything\* returned immediately.

*My experience has been this is hardly ever useful - it means "match on anything - including nothing" and usually returns nothing. But it does it very quickly. RJ*

and/or

expect timeout ; # Receive everything returned up to $timeout seconds.

when one "just wants the answer".

Also, note that **$expect\_out(buffer)** doesn't really hold what people want; it typically needs to be filtered down at least to eliminate the prompt.

Also be aware of the "[match\_max](http://wiki.tcl.tk/9407) <max buffer size>" expect buffer size. if you do not get all the response you expect then you should probably increase the size.

"[How to access the result of a remote command in Expect](http://wiki.tcl.tk/2958)" makes somewhat different computations to achieve similar results.

<http://tcl.tk/man/tcl8.5/TclCmd/Tcl.htm#M27>

## [Tcl8.5.19/Tk8.5.19 Documentation](http://tcl.tk/man/tcl8.5/contents.htm) > [TclCmd](http://tcl.tk/man/tcl8.5/TclCmd/contents.htm) > Tcl

### [Tcl/Tk Applications](http://tcl.tk/man/tcl8.5/UserCmd/contents.htm) | [Tcl Commands](http://tcl.tk/man/tcl8.5/TclCmd/contents.htm) | [Tk Commands](http://tcl.tk/man/tcl8.5/TkCmd/contents.htm) | [Tcl Library](http://tcl.tk/man/tcl8.5/TclLib/contents.htm) | [Tk Library](http://tcl.tk/man/tcl8.5/TkLib/contents.htm)

[NAME](http://tcl.tk/man/tcl8.5/TclCmd/Tcl.htm" \l "M2)

Tcl - Tool Command Language

[SYNOPSIS](http://tcl.tk/man/tcl8.5/TclCmd/Tcl.htm" \l "M3)

[DESCRIPTION](http://tcl.tk/man/tcl8.5/TclCmd/Tcl.htm" \l "M4)

[[1]](http://tcl.tk/man/tcl8.5/TclCmd/Tcl.htm" \l "M5) **[Commands.](http://tcl.tk/man/tcl8.5/TclCmd/Tcl.htm" \l "M5)**

[[2]](http://tcl.tk/man/tcl8.5/TclCmd/Tcl.htm" \l "M6) **[Evaluation.](http://tcl.tk/man/tcl8.5/TclCmd/Tcl.htm" \l "M6)**

[[3]](http://tcl.tk/man/tcl8.5/TclCmd/Tcl.htm" \l "M7) **[Words.](http://tcl.tk/man/tcl8.5/TclCmd/Tcl.htm" \l "M7)**

[[4]](http://tcl.tk/man/tcl8.5/TclCmd/Tcl.htm" \l "M8) **[Double quotes.](http://tcl.tk/man/tcl8.5/TclCmd/Tcl.htm" \l "M8)**

[[5]](http://tcl.tk/man/tcl8.5/TclCmd/Tcl.htm" \l "M9) **[Argument expansion.](http://tcl.tk/man/tcl8.5/TclCmd/Tcl.htm" \l "M9)**

[[6]](http://tcl.tk/man/tcl8.5/TclCmd/Tcl.htm" \l "M10) **[Braces.](http://tcl.tk/man/tcl8.5/TclCmd/Tcl.htm" \l "M10)**

[[7]](http://tcl.tk/man/tcl8.5/TclCmd/Tcl.htm" \l "M11) **[Command substitution.](http://tcl.tk/man/tcl8.5/TclCmd/Tcl.htm" \l "M11)**

[[8]](http://tcl.tk/man/tcl8.5/TclCmd/Tcl.htm" \l "M12) **[Variable substitution.](http://tcl.tk/man/tcl8.5/TclCmd/Tcl.htm" \l "M12)**

**[$](http://tcl.tk/man/tcl8.5/TclCmd/Tcl.htm" \l "M13)***[name](http://tcl.tk/man/tcl8.5/TclCmd/Tcl.htm" \l "M13)*

**[$](http://tcl.tk/man/tcl8.5/TclCmd/Tcl.htm" \l "M14)***[name](http://tcl.tk/man/tcl8.5/TclCmd/Tcl.htm" \l "M14)***[(](http://tcl.tk/man/tcl8.5/TclCmd/Tcl.htm" \l "M14)***[index](http://tcl.tk/man/tcl8.5/TclCmd/Tcl.htm" \l "M14)***[)](http://tcl.tk/man/tcl8.5/TclCmd/Tcl.htm" \l "M14)**

**[${](http://tcl.tk/man/tcl8.5/TclCmd/Tcl.htm" \l "M15)***[name](http://tcl.tk/man/tcl8.5/TclCmd/Tcl.htm" \l "M15)***[}](http://tcl.tk/man/tcl8.5/TclCmd/Tcl.htm" \l "M15)**

[[9]](http://tcl.tk/man/tcl8.5/TclCmd/Tcl.htm" \l "M16) **[Backslash substitution.](http://tcl.tk/man/tcl8.5/TclCmd/Tcl.htm" \l "M16)**

[\](http://tcl.tk/man/tcl8.5/TclCmd/Tcl.htm" \l "M17)**[a](http://tcl.tk/man/tcl8.5/TclCmd/Tcl.htm" \l "M17)**

[\](http://tcl.tk/man/tcl8.5/TclCmd/Tcl.htm" \l "M18)**[b](http://tcl.tk/man/tcl8.5/TclCmd/Tcl.htm" \l "M18)**

[\](http://tcl.tk/man/tcl8.5/TclCmd/Tcl.htm" \l "M19)**[f](http://tcl.tk/man/tcl8.5/TclCmd/Tcl.htm" \l "M19)**

[\](http://tcl.tk/man/tcl8.5/TclCmd/Tcl.htm" \l "M20)**[n](http://tcl.tk/man/tcl8.5/TclCmd/Tcl.htm" \l "M20)**

[\](http://tcl.tk/man/tcl8.5/TclCmd/Tcl.htm" \l "M21)**[r](http://tcl.tk/man/tcl8.5/TclCmd/Tcl.htm" \l "M21)**

[\](http://tcl.tk/man/tcl8.5/TclCmd/Tcl.htm" \l "M22)**[t](http://tcl.tk/man/tcl8.5/TclCmd/Tcl.htm" \l "M22)**

[\](http://tcl.tk/man/tcl8.5/TclCmd/Tcl.htm" \l "M23)**[v](http://tcl.tk/man/tcl8.5/TclCmd/Tcl.htm" \l "M23)**

[\](http://tcl.tk/man/tcl8.5/TclCmd/Tcl.htm" \l "M24)**[<newline>](http://tcl.tk/man/tcl8.5/TclCmd/Tcl.htm" \l "M24)***[whiteSpace](http://tcl.tk/man/tcl8.5/TclCmd/Tcl.htm" \l "M24)*

[\\](http://tcl.tk/man/tcl8.5/TclCmd/Tcl.htm" \l "M25)

[\](http://tcl.tk/man/tcl8.5/TclCmd/Tcl.htm" \l "M26)*[ooo](http://tcl.tk/man/tcl8.5/TclCmd/Tcl.htm" \l "M26)*

[\](http://tcl.tk/man/tcl8.5/TclCmd/Tcl.htm" \l "M27)**[x](http://tcl.tk/man/tcl8.5/TclCmd/Tcl.htm" \l "M27)***[hh](http://tcl.tk/man/tcl8.5/TclCmd/Tcl.htm" \l "M27)*

[\](http://tcl.tk/man/tcl8.5/TclCmd/Tcl.htm" \l "M28)**[u](http://tcl.tk/man/tcl8.5/TclCmd/Tcl.htm" \l "M28)***[hhhh](http://tcl.tk/man/tcl8.5/TclCmd/Tcl.htm" \l "M28)*

[[10]](http://tcl.tk/man/tcl8.5/TclCmd/Tcl.htm" \l "M29) **[Comments.](http://tcl.tk/man/tcl8.5/TclCmd/Tcl.htm" \l "M29)**

[[11]](http://tcl.tk/man/tcl8.5/TclCmd/Tcl.htm" \l "M30) **[Order of substitution.](http://tcl.tk/man/tcl8.5/TclCmd/Tcl.htm" \l "M30)**

[[12]](http://tcl.tk/man/tcl8.5/TclCmd/Tcl.htm" \l "M31) **[Substitution and word boundaries.](http://tcl.tk/man/tcl8.5/TclCmd/Tcl.htm" \l "M31)**

### NAME

Tcl - Tool Command Language

### SYNOPSIS

Summary of Tcl language syntax.

### DESCRIPTION

The following rules define the syntax and semantics of the Tcl language:

[1] **Commands.**

A Tcl script is a string containing one or more commands. Semi-colons and newlines are command separators unless quoted as described below. Close brackets are command terminators during command substitution (see below) unless quoted.

[2] **Evaluation.**

A command is evaluated in two steps. First, the Tcl interpreter breaks the command into *words* and performs substitutions as described below. These substitutions are performed in the same way for all commands. The first word is used to locate a command procedure to carry out the command, then all of the words of the command are passed to the command procedure. The command procedure is free to interpret each of its words in any way it likes, such as an integer, variable name, list, or Tcl script. Different commands interpret their words differently.

[3] **Words.**

Words of a command are separated by white space (except for newlines, which are command separators).

[4] **Double quotes.**

If the first character of a word is double-quote (“"”) then the word is terminated by the next double-quote character. If semi-colons, close brackets, or white space characters (including newlines) appear between the quotes then they are treated as ordinary characters and included in the word. Command substitution, variable substitution, and backslash substitution are performed on the characters between the quotes as described below. The double-quotes are not retained as part of the word.

[5] **Argument expansion.**

If a word starts with the string “{\*}” followed by a non-whitespace character, then the leading “{\*}” is removed and the rest of the word is parsed and substituted as any other word. After substitution, the word is parsed as a list (without command or variable substitutions; backslash substitutions are performed as is normal for a list and individual internal words may be surrounded by either braces or double-quote characters), and its words are added to the command being substituted. For instance, “cmd a {\*}{b [c]} d {\*}{$e f "g h"}” is equivalent to “cmd a b {[c]} d {$e} f "g h"”.

[6] **Braces.**

If the first character of a word is an open brace (“{”) and rule [5] does not apply, then the word is terminated by the matching close brace (“}”). Braces nest within the word: for each additional open brace there must be an additional close brace (however, if an open brace or close brace within the word is quoted with a backslash then it is not counted in locating the matching close brace). No substitutions are performed on the characters between the braces except for backslash-newline substitutions described below, nor do semi-colons, newlines, close brackets, or white space receive any special interpretation. The word will consist of exactly the characters between the outer braces, not including the braces themselves.

[7] **Command substitution.**

If a word contains an open bracket (“[”) then Tcl performs *command substitution*. To do this it invokes the Tcl interpreter recursively to process the characters following the open bracket as a Tcl script. The script may contain any number of commands and must be terminated by a close bracket (“]”). The result of the script (i.e. the result of its last command) is substituted into the word in place of the brackets and all of the characters between them. There may be any number of command substitutions in a single word. Command substitution is not performed on words enclosed in braces.

[8] **Variable substitution.**

If a word contains a dollar-sign (“$”) followed by one of the forms described below, then Tcl performs *variable substitution*: the dollar-sign and the following characters are replaced in the word by the value of a variable. Variable substitution may take any of the following forms:

**$***name*

*Name* is the name of a scalar variable; the name is a sequence of one or more characters that are a letter, digit, underscore, or namespace separators (two or more colons).

**$***name***(***index***)**

*Name* gives the name of an array variable and *index* gives the name of an element within that array. *Name* must contain only letters, digits, underscores, and namespace separators, and may be an empty string. Command substitutions, variable substitutions, and backslash substitutions are performed on the characters of *index*.

**${***name***}**

*Name* is the name of a scalar variable. It may contain any characters whatsoever except for close braces.

There may be any number of variable substitutions in a single word. Variable substitution is not performed on words enclosed in braces.

[9] **Backslash substitution.**

If a backslash (“\”) appears within a word then *backslash substitution* occurs. In all cases but those described below the backslash is dropped and the following character is treated as an ordinary character and included in the word. This allows characters such as double quotes, close brackets, and dollar signs to be included in words without triggering special processing. The following table lists the backslash sequences that are handled specially, along with the value that replaces each sequence.

\**a**

Audible alert (bell) (0x7).

\**b**

Backspace (0x8).

\**f**

Form feed (0xc).

\**n**

Newline (0xa).

\**r**

Carriage-return (0xd).

\**t**

Tab (0x9).

\**v**

Vertical tab (0xb).

\**<newline>***whiteSpace*

A single space character replaces the backslash, newline, and all spaces and tabs after the newline. This backslash sequence is unique in that it is replaced in a separate pre-pass before the command is actually parsed. This means that it will be replaced even when it occurs between braces, and the resulting space will be treated as a word separator if it is not in braces or quotes.

\\

Backslash (“\”).

\*ooo*

The digits *ooo* (one, two, or three of them) give an eight-bit octal value for the Unicode character that will be inserted. The upper bits of the Unicode character will be 0.

\**x***hh*

The hexadecimal digits *hh* give an eight-bit hexadecimal value for the Unicode character that will be inserted. Any number of hexadecimal digits may be present; however, all but the last two are ignored (the result is always a one-byte quantity). The upper bits of the Unicode character will be 0.

\**u***hhhh*

The hexadecimal digits *hhhh* (one, two, three, or four of them) give a sixteen-bit hexadecimal value for the Unicode character that will be inserted.

Backslash substitution is not performed on words enclosed in braces, except for backslash-newline as described above.

[10] **Comments.**

If a hash character (“#”) appears at a point where Tcl is expecting the first character of the first word of a command, then the hash character and the characters that follow it, up through the next newline, are treated as a comment and ignored. The comment character only has significance when it appears at the beginning of a command.

[11] **Order of substitution.**

Each character is processed exactly once by the Tcl interpreter as part of creating the words of a command. For example, if variable substitution occurs then no further substitutions are performed on the value of the variable; the value is inserted into the word verbatim. If command substitution occurs then the nested command is processed entirely by the recursive call to the Tcl interpreter; no substitutions are performed before making the recursive call and no additional substitutions are performed on the result of the nested script.

Substitutions take place from left to right, and each substitution is evaluated completely before attempting to evaluate the next. Thus, a sequence like

set y [set x 0][incr x][incr x]

will always set the variable *y* to the value, *012*.

[12] **Substitution and word boundaries.**

Substitutions do not affect the word boundaries of a command, except for argument expansion as specified in rule [5]. For example, during variable substitution the entire value of the variable becomes part of a single word, even if the variable's value contains spaces.

Expect

======

EXPECT Script for Cisco / Juniper Config Backup

Firstly I am asking this as a beginner in scripting, currently I have an expect script which automatically logs in to predefined Cisco devices and runs certain commands, I would like to update my script so that this same script can also backup Juniper devices.

Ideally what I would like the script to do is (in pseudo code)

login / send credentials

expect prompt

send command "show version"

if output contains "JUNOS" then

send command 1

send command 2

send command 3

otherwise if output contains "Cisco" then

send command 1

send command 2

send command 3

Im sorry if this has been asked before, but I have tried searching and searching and couldn't find an answer if anyone can assist with this I would really appreciate it. I have also included my current expect script for your reference (this script gets called by a BASH Script)

set timeout 5

set hostname [lindex $argv 0]

set username "user"

set password "pass"

spawn ssh $username@$hostname

expect "Password:"

send "$password\n"

expect "#" {

send "terminal length 0\n"

send "show running-config\n"

expect "end\r"

send "\n"

send "exit\n"

}

---- UPDATE ---

Thanks for your input Dinesh - I have updated my script to include what you provided (as below)

set timeout 5

set hostname [lindex $argv 0]

set username "user"

set password "pass"

set prompt "(>|#|\\\$) $"

spawn ssh $username@$hostname

expect "\*assword:"

send $password\r

send \r

expect -re $prompt {

send "show version\r"

expect -re $prompt

expect \*;

set output $expect\_out(buffer);

#Basic string check logic with Tcl

if { [string first "JUNOS" $output ]!=-1 } {

send "show configuration | display set | no-more"

expect -re $prompt

send "exit\r"

} else {

send "terminal length 0\r"

expect -re $prompt

send "show run\r"

expect "end"

send \r

expect -re $prompt

send "exit\r"

}

}

However when i run this script the issue I have is that the output of the "show version" doesn't seem to be matching my "string check" and the script therefore ignores the "if" statement and proceeds with the "else" statement.

The output of the "show version" command is below - what will I need to modify so that the "JUNOS" string gets matched?

user@host> show version

Hostname: host

Model: srx240h

JUNOS Software Release [11.4R7.5]

--- EDIT 2: Output from the script

05c4rw@testpc:~/script$ ./ssh.sh

spawn ssh user@juniperhost

## LOGIN BANNER - Removed for brevity

user@juniperhost's password:

--- JUNOS 11.4R7.5 built 2013-03-01 11:40:03 UTC

user@juniperhost> show version

Hostname: juniperhost

Model: srx240h

JUNOS Software Release [11.4R7.5]

user@juniperhost> show configuration | display set | no-more

set version 11.4R7.5

## \*\*\* OUTPUT REMOVED FOR BREVITY / PRIVACY \*\*\*

## \*\*\* END OF OUTPUT from previous command

user@juniper> spawn ssh user@ciscohost

password:

## LOGIN BANNER - removed for brevitiy

ciscohost#05c4rw@testpc:~/script$

configuration scripting expect cisco

shareimprove this question

edited Jan 13 '15 at 5:15

asked Jan 12 '15 at 6:14

05c4rw

112

Place the code expect \* before sending show version and print the value of output. Update the same here. – Dinesh Jan 13 '15 at 4:08

Hi Dinesh, thanks for that I have updated the configuration and now it is successfully running the "if" statement however the "else" statement doesn't appear to be working - as a test I have two devices in the list which it automatically logs in to the Juniper first and the Cisco after that - after successfully completing the Juniper part (if) it logs in to the Cisco Device and sits in the prompt before exiting (not issuing any commands) – 05c4rw Jan 13 '15 at 4:33

In that case, print the value of output (just to know for our convenience) and check what it contains before going to the if-else loop. – Dinesh Jan 13 '15 at 4:36

I've placed the output in the main question due to formatting issues, thanks. – 05c4rw Jan 13 '15 at 5:15

What I meant is to print the variable output's value. The code can not skip the if-else loop. Code flow should either with if or else – Dinesh Jan 13 '15 at 8:03

add a comment

2 Answers

active

oldest

votes

up vote

1

down vote

set timeout 5

set hostname [lindex $argv 0]

set username "user"

set password "pass"

spawn ssh $username@$hostname

expect "Password:"

send "$password\r"

expect "#" {

send "terminal length 0\r"

expect "#"

# This is to clean up the previous expect\_out(buffer) content

# So that, we can get the exact output what we need.

expect \*;

send "show running-config\r"

expect "end"

#Now, the content of 'expect\_out(buffer)' has the whole 'show run' output

set output $expect\_out(buffer);

#Basic string check logic with Tcl

if { [string first "JUNOS" $output ]!=-1 } {

# Apply your logic here

# send "command1"

# expect "#"

} else {

# Same as above

# I assume, there are 2 possibilities. So, keeping 'else' part alone.

# Have 'else if', if you have more than 2.

}

}

Notice that each line sent by the script is terminated with \r. This denotes a return character and is exactly what you would press if you entered these lines at the shell, so that is exactly what Expect has to send. It is a common mistake to terminate send commands to a process followed by \n. In this context, \n denotes a linefeed character. You do not interactively end lines with a linefeed. So Expect must not either. So, always use \r.

You can have a look at here if you are interested to know more about the why you need expect \*. (which is a separate story)

I can see that there are some commands used only with send in your example. Basically, expect will work with two feasible commands such as send and expect. In this case, if send is used, then it is mandatory to have expect (in most of the cases) afterwards. (while the vice-versa is not required to be mandatory)

This is because without that we will be missing out what is happening in the spawned process as expect will assume that you simply need to send one string value and not expecting anything else from the session.

-=-=-=-=-=-=-=-=-=-=-=-=-=

Shell script with expect module to backup running config of router via console

I am a Linux enthusiast trying my hand at shell scripting lately.

#!/usr/bin/expect-f

cd /home/vuser/Net\_backup

now=$(date +"%m\_%d\_%y")

touch /home/vuser/Net\_backup/ASR$now.txt

sleep 5s

set Password "prizer"

set password "banger"

set Trial "32"

set bye "exit"

set quitconsole "q"

set exitcons "logout"

spawn telnet xx.xx.xx.xx

expect "password:"

send "$password\n"

expect "admin>"

send "$Trial\n"

sleep 5s

expect "ASR1001>"

send "enable\n"

expect "Password:"

send "$Password\n"

expect "ASR1001#"

send "terminal length 0\n"

expect "ASR1001#"

send "show run\n" | tee /home/vuser/Net\_backup/ASRnow.txt

expect "ASR1001#"

send "$bye\n"

sleep 5s

xdotool key ctrl+]

sleep 5s

send "$quitconsole\n"

expect "admin >"

send "$exitcons\n"

sleep 5s

exit

I intend to write a shell script which would perform the following functions .

(1) Create a file in a specific folder (2) Append the system date to the file name in the format ""%m\_%d\_%y"" (3) Login to a console server (4) access a specific port on a console server to login to the router (5) go to the super user prompt on the router (6) set terminal length to 0 to negate the need for carriage return input (7) run the command "show run" on the router and capture this output to the file created in step (2) (8) Exit from the router command prompt (9)Exit from the console server (this needs three steps first send keystrokes (ctrl+]) then

the command q or quit and finally logout to comeback to the linux prompt) (10) Exit the script .

I have written the script which is given above ..

My script however gets stuck on the console server and does not move forward, the script proceeds after I manually logout from the console server however the intended output is not achieved

Appreciate your help on this .

Thank you Avinash

shell expect

shareimprove this question

edited Feb 9 '14 at 6:05

asked Feb 8 '14 at 19:10

avan1729

814

add a comment

2 Answers

active

oldest

votes

Não encontrou uma resposta? Pergunte em Stack Overflow em Português.

✕

up vote

1

down vote

While developing expect scripts, you want to enable debugging output. Always add this near the top: exp\_internal 1

This is wrong: #!/usr/bin/expect-f -- you need a space before the option.

Expect is not shell. These lines need to be changed:

now=$(date +"%m\_%d\_%y")

touch /home/vuser/Net\_backup/ASR$now.txt

sleep 5s

send "show run\n" | tee /home/vuser/Net\_backup/ASRnow.txt

Use \r instead of \n to "hit enter" for your send commands.

If your expect patterns are correct, there's no need to sleep.

Here's a quick untested rewrite

#!/usr/bin/expect -f

cd /home/vuser/Net\_backup

set now [clock format [clock seconds] "%m\_%d\_%y"]

exec touch ASR$now.txt

set Password "prizer"

set password "banger"

set Trial "32"

set bye "exit"

set quitconsole "q"

set exitcons "logout"

spawn telnet 192.168.95.96

expect "password:"

send "$password\r"

expect "admin>"

send "$Trial\r"

expect "ASR1001>"

send "enable\r"

expect "Password:"

send "$Password\r"

expect "ASR1001#"

send "terminal length 0\r"

expect "ASR1001#"

send "show run\r"

expect -re "(.\*)ASR1001#"

set fh [open ASR$now.txt w]

puts $fh $expect\_out(1,string)

close $fh

send "$bye\r"

sleep 5

send "^]" ;# here, in your editor, enter a literal ctrl-]

sleep 5

send "$quitconsole\r"

expect "admin >"

send "$exitcons\r"

expect eof

shareimprove this answer

edited Feb 14 '14 at 20:29

avan1729

814

answered Feb 8 '14 at 21:35

glenn jackman

139k21114200

The script doesnt work , I will have to try something similar though . – avan1729 Feb 10 '14 at 18:57

Hi glenn made some modifications to the script it mostly works now ,however I am not able to capture the output of the file . Please see my inputs below ... – avan1729 Feb 14 '14 at 20:05

add a comment

up vote

0

down vote

#! /usr/bin/expect -f

set systemTime [clock seconds]

set now [clock format $systemTime -format "%m\_%d\_%y"]

exec touch ASR$now.txt

set Password "prizer"

set password "banger"

set bye "exit"

set quitconsole "q"

set exitcons "logout"

spawn ssh admin@192.168.95.96

expect "password:"

send "$password\r"

sleep 5

expect "admin>"

send "32\r"

sleep 5

send "\r"

expect "ASR1001>"

send "enable\r"

expect "Password:"

send "$Password\r"

expect "ASR1001#"

send "terminal length 0\r"

expect "ASR1001#"

send "show run \r"

set output [open "ASR$now.txt" "w"] <<<<<<<<<<<<<<< Fails here

send "$bye\r"

sleep 5

send "ctrl-]"

sleep 5

send "$quitconsole\r"

expect "admin>"

send "$exitcons\r"

expect eof

shareimprove this answer,

-=-=-=-=-=-=-=-=-=-=-=-=-=-

#!/bin/bash

##Enter Username and Password Details:

userName=<UserName>

password=<Password>

expect -c "

spawn ssh ${useName}@10.8.100.100 ##put your own IP here

expect "password: " ##or whatever password prompt you get

send "$password\r"

epxect -re "Last Login: " ##or whatever the end of your welcome message is

send "su -i"

expect "password for <UserName>"

send "$password\r"

"

=-=-=-=-=-=-=-=-=

Mixing bash and expect is not a good way to achieve the desired effect. I'd try to use only Expect:

#!/usr/bin/expect

eval spawn ssh -oStrictHostKeyChecking=no -oCheckHostIP=no usr@$myhost.example.com

#use correct prompt

set prompt ":|#|\\\$"

interact -o -nobuffer -re $prompt return

send "my\_password\r"

interact -o -nobuffer -re $prompt return

send "my\_command1\r"

interact -o -nobuffer -re $prompt return

send "my\_command2\r"

interact

Sample solution for bash could be:

#!/bin/bash

/usr/bin/expect -c 'expect "\n" { eval spawn ssh -oStrictHostKeyChecking=no -oCheckHostIP=no usr@$myhost.example.com; interact }'

This will wait for enter and than return (for a moment) interactive session.

--=-=-=-=-=-=-=-=

Simple BASH scripts using EXPECT for Cisco Devices

Publicado em 31 de agosto de 2015

31

5

2

Daniel Gilbertson

Daniel Gilbertson

Security Consultant at Presidio

Managing a large amount of devices on a network can be tricky, especially without management software to run backups or push mass configuration changes. In this post, we are going to be using SSH with expect scripts to run tftp backups of Cisco devices and also show how to push configuration changes.

The backup script has been configured for both Catalyst (IOS) and Nexus (NXOS) based devices. Whereas, the config push script should work on any Cisco device.

Requirements:

running tftp server

expect

To get expect on Debian based distros:

sudo apt-get install expect

The script has three parts:

A list of IP addresses or FQDN hostnames

A bash script to get login information, read the IP list, then send it to the expect script

And lastly, the expect script to open an SSH session, login, and run commands

First, you'll want to create a file named backup-IP-list, and put the IP addresses of the devices you want to run backups for.

Example:

10.0.0.1

10.0.0.2

10.0.0.3

10.0.0.4

10.0.0.5

Then, create a file named backup.sh, and put this in it. This is where it will prompt you for username and credentials. You will notice that as you type, the cursor will not move and nothing will show up; a little security from shoulder suffers or having plain-text passwords in the terminal history.

#!/bin/bash

# Collect username, ssh, & enable passwords

echo "Enter the username: "

read -s -e user

echo "Enter the SSH password: "

read -s -e password

echo "Enter the Enable password: "

read -s -e enable

# Open device list & send the collected information to script

for device in `cat backup-IP-list`; do

./expectbackup.sh $device $user $password $enable ;

done

Next, we create the expect script. Create a file named expectbackup.sh, and put this in it.

#!/usr/bin/expect -f

# Set variables

set hostname [lindex $argv 0]

set username [lindex $argv 1]

set password [lindex $argv 2]

set enablepassword [lindex $argv 3]

set date [exec date +%F]

set server "XXX.XXX.XXX.XXX"

# Log results

log\_file -a backup-$date.log

# Announce device & time

send\_user "\n"

send\_user ">>>>> Working on $hostname @ [exec date] <<<<<\n"

send\_user "\n"

# Don't check keys

spawn ssh -o StrictHostKeyChecking=no $username\@$hostname

# Connection issues & priv password

expect {

timeout { send\_user "\nTimeout Exceeded - Check Host\n"; exit 1 }

eof { send\_user "\nSSH Connection To $hostname Failed\n"; exit 1 }

"\*assword:" { send "$password\r" }

}

# Enable password

expect {

default { send\_user "\nLogin Failed - Check Password\n"; exit 1 }

"\*#" { send "\r" }

"\*>" {

send "enable\n"

expect "\*assword:"

send "$enablepassword\r"

}

}

# Backup config to tftp server

expect {

"\*#" {

send "copy running-config tftp:\r"

expect {

"Enter destination filename:\*" {

send "\r"

expect "(If no input, current vrf 'default' is considered): "

send "\r"

expect "\*server:"

send "$server\r"

expect "\*(please wait)..."

send "exit\r"

}

"Address or name of remote host []?\*" {

send "$server\r"

expect "\*filename\*"

send "\r"

expect "\*copied\*"

send "exit\r"

}

}

}

}

Make sure all three of these files are located in the same directory. Then all you have to do to run it is open a terminal, and type ./backup.sh

That's it!!!

Now if you wanted to make configuration changes, the scripts are pretty much the same, but instead of running backups, we get into conf t and send commands.

Lets start by creating a file named config-IP-list, and add IP addresses.

Then create a file named ciscoconfig.sh, and put this in it.

#!/bin/bash

# Collect username, ssh, & enable passwords

echo "Enter the username: "

read -s -e user

echo "Enter the SSH password: "

read -s -e password

echo "Enter the Enable password: "

read -s -e enable

# Open device list & send the collected information to script

for device in `cat config-IP-list`; do

./configpush.sh $device $user $password $enable ;

done

Lastly, create a file named configpush.sh, and add this to it.

#!/usr/bin/expect -f

# Set variables

set hostname [lindex $argv 0]

set username [lindex $argv 1]

set password [lindex $argv 2]

set enablepassword [lindex $argv 3]

set date [exec date +%F]

# Log results

log\_file -a config-$date.log

# Announce device & time

send\_user "\n"

send\_user ">>>>> Working on $hostname @ [exec date] <<<<<\n"

send\_user "\n"

# Don't check keys

spawn ssh -o StrictHostKeyChecking=no $username\@$hostname

# Connection issues & priv password

expect {

timeout { send\_user "\nTimeout Exceeded - Check Host\n"; exit 1 }

eof { send\_user "\nSSH Connection To $hostname Failed\n"; exit 1 }

"\*assword:" { send "$password\r" }

}

# Enable password

expect {

default { send\_user "\nLogin Failed - Check Password\n"; exit 1 }

"\*#" { send "\r" }

"\*>" {

send "enable\n"

expect "\*assword:"

send "$enablepassword\r"

}

}

# configuration changes

expect {

"\*#" {

send "conf t\r"

expect "\*config)#"

send "hostname DanIsAwesome\r"

expect "\*config)#"

send "end\r"

expect "\*#"

send "copy run start\r"

expect "\*(please wait)..."

send "exit\r"

}

}

You will notice in that script, we changed the hostname of the device to DanIsAwesome, but that was just an example of what you can do.

To do more, just add:

send "COMMAND\r"

expect "\*config)#"

for every command you want to send.

-=-=-=-=-=-=-=-=-=-

Expect Script: Parse Specific Info From Cisco CLI

Ask Question

up vote

0

down vote

favorite

I'm new to Expect scripting, and trying to parse Cisco router ACL output.

I'm specifically trying to parse IP addresses between the first 2 remark lines. Any help greatly appreciated.

Here's one of my random attempts. It parses IP addresses from the ACL, but doesn't stop at the 2nd remark line:

send "show run | section ip access-list extended OUTSIDE\r\n"

set ip {}

expect {

"remark ##### DENIED HOSTS #####\r\n" {

expect -re {(\d+\.\d+\.\d+\.\d+)} {

set ip "${ip}$expect\_out(0,string)"

puts $ip

set ip {}

exp\_continue

}

}

}

Not only does the script not stop at the 2nd remark line like I want it to, but it doesn't seem to have an exit point and waits for Expect to timeout.

Random sample ACL for reference:

ip access-list extended OUTSIDE

remark ##### DENIED HOSTS #####

deny ip host 2.2.2.2 any

deny ip host 9.9.9.9 any

deny ip host 8.7.6.5 any

deny ip host 5.6.7.8 any

deny ip host 5.5.5.5 any

deny ip host 4.4.4.4 any

deny ip host 3.3.3.3 any

remark ########################

permit tcp any any eq 22

deny ip any any

permit ip host 2.3.5.1 any

I basically want to parse addresses from 2.2.2.2 to 3.3.3.3, but then exit once the script reaches the 2nd remark line.

parsing expect cisco command-line-interface

shareimprove this question

edited Jul 22 '14 at 3:59

asked Jul 17 '14 at 23:42

Jason

32

Please format your post. Reading an essay is just not fun here... – slayedbylucifer Jul 18 '14 at 5:47

add a comment

3 Answers

active

oldest

votes

Não encontrou uma resposta? Pergunte em Stack Overflow em Português.

✕

up vote

2

down vote

accepted

With exp\_continue, it is always better to add the case where we want to terminate from expect. So that, if the exit case arrived, we can terminate from the expect. Else, obviously timeout will happen.

E.g.

expect{

“p” {send “\r\r\r”; exp\_continue}

“+” {incr foo; exp\_continue}

“i” {puts "test"; exp\_continue}

“quit” exit

}

As you can see, if the word 'quit' comes, then it will be exited.

Lets come to your question. You need to exit from expect when the word

'remark ########################' arrives. But, if we code like,

expect {

"remark ##### DENIED HOSTS #####\r\n" {

expect {

"remark ########################" {exit}

-re {(\d+\.\d+\.\d+\.\d+)} {

set ip "${ip}$expect\_out(0,string)"

puts "-->$ip"

set ip {}

exp\_continue

}

}

}

}

Program will terminate the expect while the ip parsing is in progress at the middle of anything.

Instead of doing this way, you can better get the whole output and try to parse it as a separate variable, to avoid overhead within expect.

send "show run | section ip access-list extended OUTSIDE\r\n"

expect {

-re {deny\s+ip\s+host\s+(.\*)remark} { puts "match found" }

timeout {puts "timeout happened"}

}

Now we have the match and it will be available in the variable expect\_out(1,string).

Then we can do regexp in the variable and parse the ip address.

puts [ regexp -inline -all {\d.\d.\d.\d} $expect\_out(1,string) ]

You can try one more way as below. Instead of doing regular expression at the expect, you can simply get the whole output and then do the regexp as below.

send "sh run | section ip access-list extended OUTSIDE\r"

#Assuming your router's hostname is `Router`

expect "Router#"

#output will hold the command output.

set output expect\_out(buffer)

puts $output

set ip\_list [regexp -inline -all {\d.\d.\d.\d} $output]

foreach ip $ip\_list {

puts $ip

}

shareimprove this answer

edited Jul 23 '14 at 10:03

answered Jul 22 '14 at 8:48

Dinesh

8,835154980

sorry...accidentally replied as an "edit" to your response somehow :-). Listing here as well. Don't see an option to delete the edit update I made. – Jason Jul 22 '14 at 22:38

Thanks, you are a mad genius :-). The only issues that I'm seeing are that it's printing the IP addresses twice for some reason, and it also printed 2.3.5.1, which was a "permit ip host" address listed after the 2nd remark line. Any ideas? Also, any idea how to add a newline after each IP address it's printing out? I tried adding "\n" in various places in the puts line, but no luck. Here's the output I got using your updates: match found 2.2.2.2 9.9.9.9 8.7.6.5 5.6.7.8 5.5.5.5 4.4.4.4 3.3.3.3 2.3.5.1 2.2.2.2 9.9.9.9 8.7.6.5 5.6.7.8 5.5.5.5 4.4.4.4 3.3.3.3 – Jason Jul 22 '14 at 22:40

cannot format in this comment area to save my life... – Jason Jul 22 '14 at 22:44

Can you try print the value of expect\_out(0,string) and expect\_out(1,string)? – Dinesh Jul 23 '14 at 6:04

You can do formatting in the comment section too. Click on the 'help' link available near the 'Add Comment' button. – Dinesh Jul 23 '14 at 6:05

show 3 more comments

up vote

0

down vote

Thanks to Dinesh!!!

Reposting the final working code (or a mix of Dinesh-provided code) here so it's not lost in all the text above (and so I can format more easily than in the comment section).

send "show run | section ip access-list extended OUTSIDE\n"

expect -re {remark(.\*)remark}

set output $expect\_out(buffer)

#puts $output

puts "\r"

set ip\_list [regexp -inline -all {\d.\d.\d.\d} $output]

foreach ip $ip\_list {

puts $ip

}

I commented out the 1st puts line, but left it there to use when testing to see what's actually being stored in $output.

Here's the ACL that was parsed from the router CLI:

ip access-list extended OUTSIDE

remark ##### DENIED HOSTS #####

deny ip host 2.2.2.2 any

deny ip host 9.9.9.9 any

deny ip host 8.7.6.5 any

deny ip host 5.6.7.8 any

deny ip host 5.5.5.5 any

deny ip host 4.4.4.4 any

deny ip host 3.3.3.3 any

remark ########################

permit tcp any any eq 22

deny ip host 82.28.82.28 any

permit ip host 2.3.5.1 any

I know, it's a completely convoluted ACL, but it worked great for testing.

Here's Expect script output after parsing the ACL:

2.2.2.2

9.9.9.9

8.7.6.5

5.6.7.8

5.5.5.5

4.4.4.4

3.3.3.3

Any / all IP addresses after the 2nd ACL remark line are ignored as desired!

Thanks Dinesh!

-=-=-=-=-=-=-=-=-=-=-=

Script to automate tasks

Hi

I`m looking to automate some functions in a web application.

Functions like enable/disable interfaces on a Cisco 3550 switch.

Anybody know if there are some good scrips, or examples anywhere to get me started?

Solved! Go to Solution.

LAN Switching and Routing

Everyone's tags (3)

automatescripttasks

1 person had this problem

1 ACCEPTED SOLUTION

siddhartham

siddhartham Silver

Silver

‎07-16-2012 07:11 AM

Script to automate tasks

Perpaal,

Below are the example scripts that shuts down an interface and logs the session into a directory.

File1- IP-list -- contains a list of IPs that you want to execute the script

File2- ssh.sh -- reads the IP-list and execute File3 which is enableint script

File3 - enableint.sh --- script to shuts down an interface

Make ssh.sh and enableint.sh files as executable with chmod +x and you just need to run ./ssh.sh

File1-IP-list

10.10.1.1

10.2.3.2

File2- ssh.sh

#!/bin/bash

while read ipadd

do

#echo $ipadd

export ipadd

./enableint.sh $ipadd

done < IP-list

File3- enableint.sh

#!/usr/bin/expect -f

set timeout 20

set IPaddress [lindex $argv 0]

set Username "username"

set Password "password"

set Directory /home/Desktop/logs

log\_file -a $Directory/session\_$IPaddress.log

send\_log "### /START-SSH-SESSION/ IP: $IPaddress @ [exec date] ###\r"

spawn ssh -o "StrictHostKeyChecking no" $Username@$IPaddress

expect "\*assword: "

send "$Password\r"

expect "#"

send "conf t\r"

expect "(config)#"

send "int g0/0\r"

expect "(config-if)#"

send "shut\r"

expect "(config-if)#"

send "exit"

expect "(config)#"

send "exit"

expect "#"

send "wr mem\r"

expect "#"

send "exit\r"

sleep 1

send\_log "\r### /END-SSH-SESSION/ IP: $IPaddress @ [exec date] ###\r"

exit

Siddhartha

Siddhartha

-=-=-=-=-=-=--

An expect script to automatiaclly log into cisco routers.

#!/usr/bin/expect

set timeout 5

set hostname [lindex $argv 0]

set username "username"

set password "password"

set enablepassword "password"

spawn telnet $hostname

expect "Username:" {

send "$username\n"

expect "Password:"

send "$password\n"

expect ">" {

send "en\n"

expect "Password:"

send "$enablepassword\n"

}

interact

}

-=-=-=-=-=-=-

#!/bin/bash

wget http -O FWnew http://rules.emergingthreats.net/fwrules/FWrev

if cmp -s "FWcurrent" "FWnew"

then

echo "same revision number"

rm FWnew

else

mv FWnew FWcurrent

wget -O fw.txt http://rules.emergingthreats.net/fwrules/emerging-PIX-CC.rules

wget http://mirror1.malwaredomains.com/files/domains.txt

cat fw.txt | cut -d" " -f6 > ips.txt

tail -n +44 ips.txt > cc.txt

rm ips.txt

rm fw.txt

cat domains.txt | grep fastflux | cut -d$'\t' -f3 > fastflux.txt

/usr/bin/expect -c '

set list [open cc.txt r]

set filedata [read $list]

close $list

set iplist [open ip.txt r]

set ipadd [read $iplist]

close $iplist

set fflist [open fastflux.txt r]

set dlist [read $fflist]

close $fflist

set username username

set password password

foreach add $ipadd {

spawn ssh $username@$add

expect -re ".\*(yes/no)" {

send "yes\r"

exp\_continue

}

expect -re ".\*password:"

send -- "$password\r"

expect -re ".\*>"

send "enable\r"

expect -re ".\*Password:"

send -- "$password\r"

expect -re ".\*#"

send "config t\r"

expect -re ".\* (config)#"

send "no dynamic-filter blacklist\r"

expect -re ".\* (config)#"

send "dynamic-filter enable\r"

send "dynamic-filter drop blacklist interface outside\r"

send "dynamic-filter blacklist\r"

expect -re ".\* (config-llist)#"

foreach line $filedata {

send "address $line 255.255.255.255\r"

expect -re ".\*#"

}

foreach line $dlist {

send "name $line\r"

expect -re ".\*#"

}

send "exit\r"

expect ".\* (config)#"

send "exit\r"

expect ".\*#"

send "write mem\r"

expect ".\*#"

send "exit\r"

send "exit\r"

}

'

rm cc.txt

rm domains.txt

rm fastflux.txt

fi

-=-=-=-=-=