



Jon Brown

Information Systems Security Officer
@ Montage Marketing Group

LinkedIn: @jonbrown2

MacAdmins Slack: @Jon Brown



What is **Expect**?

Expect is an extension to the Tcl scripting language written by Don Libes. The program automates interactions with programs that expose a text terminal interface. Expect, originally written in 1990 for the Unix platform, has since become available for Microsoft Windows and other systems. License Public Domain.



Don Libes

Computer Scientist @ NIST

LinkedIn: [@don-libes-243552](#)

Creator of Expect, STEP



Expect is a tool for automating interactive applications such as **telnet, ftp, passwd, fsck, rlogin, tip, etc.**

Commands (summarized)

- Close
 - closes the connection to the current process.
- Debug [-D]
 - controls a Tcl debugger allowing you to step through statements, set breakpoints, etc.
- Disconnect
 - disconnects a forked process from the terminal.
- Exit
 - disconnects a forked process from the terminal.
- Spawn
 - creates a new process running program args.

man expect

- Expect
 - waits until one of the patterns matches the output of a spawned process, a specified time period has passed, or an end-of-file is seen.
- Send [-flags] string
 - Sends string to the current process. For example, the command
 - send "hello world\r" sends the characters, h e l l o <blank> w o r l d <return> to the current process.
 - The -s flag forces output to be sent "slowly"
 - The -h flag forces output to be sent (somewhat) like a human actually typing

Commands (continued)

- **-c**
 - Specifies the command to execute before the script.
- **-d**
 - Provides a brief diagnostic output..
- **-f**
 - Specifies a file to read from.
- **-i**
 - Prompts commands interactively.
- **-b**
 - Reads file line by line (buffer).
- **-V**
 - Print version.

man expect

- Expect is located at:
 - `#!/usr/bin/expect`
- Use “whereis expect” to find the full path to Expect on your system.
- Basic Usage:
 - You need to have an interactive script or prompt to use Expect with.
 - Expect allows you to write prompts around expected user prompts.
 - User prompts in Bash scripts for example are noted as “read \$REPLY” variables.

```
#!/bin/bash
```

test.sh

```
echo "Enter your name"
```



```
read $REPLY
```



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



expect.exp

```
set timeout -1
```

```
spawn ./test.sh
```



```
expect "Enter your name\r"
```



```
send -- "I am Jon\r"
```



```
expect eof
```


Demo

Variables

```
#!/usr/bin/expect
```

```
set NAME "JON"
```

```
set AGE "45"
```



```
set PHONE [lindex $argv 0]
```



```
spawn ./expect_vars.sh
```

```
expect "Hello, who is this?\r"
```

```
send -- "$NAME\r"
```

```
expect "How old are you?\r"
```

```
send -- "$AGE\r"
```

```
expect "Whats your phone number?\r"
```

```
send -- "$PHONE\r"
```

```
expect eof
```

```
./expect_var.exp "555-55-5555"
```

Demo

Salesforce **Dataloader**

Challenges (summarized)

- First you have to install Java
 - Install Java Runtime Environment (JRE) version 17
- Data Loader itself is a .command file that is not signed and seemingly proudly so by Salesforce.
 - They actually say *(Just hit ignore when prompted to accept the unidentified developer warning).
- The .command file itself has to be packaged and deployed or hosted somewhere to then be downloaded potentially?

- Deploy Java (JRE)
- Deploy Data Loader
- Remove Quarantine Flag from Data Loader
- Install Data Loader
- Method to Uninstall Data Loader

Data Loader itself requires human interaction.

Here's what it looks like when you run it.

`#!/bin/bash`



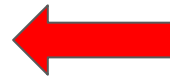
`install.sh`

`mkdir /var/tmp/dataloader_v60.0.1/`



`curl`

`https://a.sfdcstatic.com/developer-website/media/dataloader/dataloader_v60.0.1.zip > /var/tmp/dataloader_v60.0.1.zip`



`/usr/bin/unzip -d /var/tmp/dataloader_v60.0.1/
/var/tmp/dataloader_v60.0.1.zip`



`cd /var/tmp/dataloader_v60.0.1/`



`xattr -r -d com.apple.quarantine install.command`



`sleep 5`



`/usr/bin/expect<<EOF` 

install.sh

`spawn /var/tmp/dataloader_v60.0.1/install.command` 

`expect "Do you want to install Data Loader in the current
folder (/private/var/tmp/dataloader_v60.0.1)? \[Yes/No\]"`


`send -- "No"
send -- "\r"`

`expect "Provide the installation folder \[default:
dataloader\]"`

`send -- "\r"` 

`expect "Do you want to create a Desktop shortcut? \[Yes/No\]"`



```
send -- "Yes"
send -- "\r"
```



install.sh

```
expect "Do you want to create a shortcut in Applications
folder? \[Yes/No\]"
```


```
send -- "Yes"
send -- "\r"
```



```
expect eof
EOF
```



```
rm -rf /private/var/tmp/dataloader_v60.0.1
rm -rf /private/var/tmp/dataloader_v60.0.1.zip
```



```
#!/bin/bash
```

uninstall.sh

```
rm -rf /Applications/DataLoader\ 60.0.1
```

```
rm -rf ~/dataloader
```

```
rm -rf ~/Desktop/DataLoader\ 60.0.1
```

Demo

Autopkg Script

Aysiu / Mac-Scripts-and-Profiles

- We use a script created by Aysiu called /AutoPkgReviewAndRun.py
- The premise of the script is that it runs Autopkg recipes and it reviews and runs them and asks you to accept and verify and trust the script.
- All the recipes that we run we manage so we trust them, so we use expect to automate the answering of the questions.
- Here's how it would work.

- We're programing in Expect directly
- We're spawning the python script
- Telling expect what to search for
- Calling an EOF



<https://www.alansiu.net/>

```
#!/Library/AutoPkg/Python3/Python.framework/Versions/Current/bin/python3
```

```
import argparse
import os
import subprocess
import sys
```

```
# Where is the recipe list (one recipe per line) located?
# Recipe list should be one recipe per line, separated by a
carriage return ("\n")
```



```
recipe_locations = [
os.path.expanduser('~/.Library/AutoPkg/recipe_list.txt'),
    os.path.expanduser('~/.Library/Application
Support/AutoPkg/recipe_list.txt') ]
```



```
# Acceptable affirmative responses
```

```
affirmative_responses = ["y", "yes", "sure", "definitely"]
```



```
if desired_result not in verify_result: 
    print(err)
    confirmation = input("Do you trust these changes? (y/n)
")
    if confirmation.lower().strip() in
affirmative_responses:
        print("Updating trust info for {}".format(recipe))
        cmd = [ "/usr/local/bin/autopkg", 
"update-trust-info", recipe ]
        p = subprocess.Popen(cmd, stdout=subprocess.PIPE,
stderr=subprocess.PIPE,
            encoding='utf8')
        out, err = p.communicate()
        if err:
            print("Unable to update trust info:
{}".format(err))
```

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



expect.sh

```
set timeout -1
```

```
spawn /usr/local/bin/python3 AutoPkgReviewAndRun.py
```



```
expect {
```

```
    "Do you trust these changes?" {
```

```
        send -- "y\r"
```

```
        exp_continue
```



```
    }
```

```
    "Search GitHub AutoPkg repos for a*" {
```

```
        send -- "n\r"
```

```
        exp_continue
```



```
    }
```

```
eof
```



```
}
```


Demo

I deeply regret ever writing this

Do not put passwords in scripts

Firmware Script

If you lock your computer or
damage your computer you do
so at your own risk

Challenges, rotating a firmware password

- Assuming we know the password changing the password is pretty straight forward.
- Changing the users firmware password requires you to know the users firmware password.
- You must then enter the command for that firmware password change.
- You must then enter the new firmware password twice.
- If everyone has the same password it can be scripted.
- If we have multiple passwords in the mix than it can be scripted but its far more complicated.
- So long as we know all the different passwords that are in the mix.
- In this scenario while working at a company we had a situation where we had some computers with one password and another set with a different set of passwords.

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



Script #1

```
spawn sudo firmwarepasswd -setpasswd
```



```
expect {
```

```
    "Enter password:" {
```

```
        send "<PWGOESHERE>\r"
```

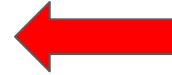


```
        exp_continue
```

```
    }
```

```
    "Enter new password:" {
```

```
        send "<PWGOESHERE>\r"
```



```
        exp_continue
```

```
    }
```

```
    "Re-enter new password:" {
```

```
        send "<PWGOESHERE>\r"
```



```
        exp_continue
```

```
    }
```

```
}
```

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



Script #2

```
set verifyPassword [exec sudo firmwarepasswd -check]
```



```
if { $verifyPassword eq "Password Enabled: Yes" } {
```

```
    spawn firmwarepasswd -delete
```

```
    expect "Enter password:"
```



```
    send "<PASS1>\r";
```

```
    expect {
```

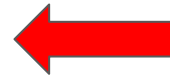


```
        "Password incorrect" {
```

```
            puts "Trying 2nd password"
```

```
            expect eof
```

```
            spawn firmwarepasswd -delete
```



```
            expect "Enter password:"
```


```
            send "<PASS2>\r";
```

```
            expect "Password removed"
```

```
            puts "Firmware Password Deleted using 2nd Password"
```

```
        }
```

```
"Password removed" {  
    puts "Firmware Password Deleted using 1st Password"  
}  
}  
} else {  
    puts "<result>Not Found</result>"  
}
```



**If you lock your computer or
damage your computer you do
so at your own risk**

Outcome?
15% Bricked

Auto Expect

What is Autoexpect?

- Autoexpect is an application to create an application! Once you start autoexpect, you then perform the actions that you want to script with Expect. Once you complete, it is well worthwhile to exercise the script to see that it does all that you really want it to do and that all error cases are handled.
- A script generated by autoexpect can be less readable and more difficult to modify than a script written by a human.
- When designing Expect scripts, it can be useful to look at autoexpect-generated scripts for inspiration and hints.

- autoexpect, one of the example programs written in and distributed with Expect, monitors an interactive shell session and generates an analogous Expect script.

Challenges with macOS

1. It's not pre-installed. Notice it says that Autoexpect is installed as part of Expect, but not with macOS, which means we need to install the homebrew version of expect to get autoexpect.

Install Homebrew

```
/bin/bash -c "$(curl -fsSL  
https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

Install Expect

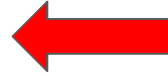
```
brew install expect
```

Validate

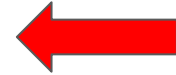
```
man autoexpect
```

So let's go back to Data Loader

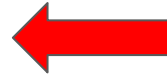
`/var/tmp/dataloader_v60.0.1/install.command`



`autoexpect /var/tmp/dataloader_v60.0.1/install.command`



`/var/tmp/dataloader_v60.0.1/install.exp`



When we open that file what do we see?

```
expect "Do you want to install Data Loader in the current  
folder (/private/var/tmp/dataloader_v60.0.1)? \[Yes/No\]"
```

```
send -- "No"  
send -- "\r"
```

```
expect "Provide the installation folder \[default:  
dataloader\]"
```

```
send -- "\r"
```

```
expect "Do you want to create a Desktop shortcut? \[Yes/No\]"
```

```
send -- "Yes"  
send -- "\r"
```

Final Demo

Questions?

Download



Review

