pexpect: the power of TCL's expect in python

Admin

Author:	*Mike Pennington - mike[~at~]pennington{dot}net*
Version:	*200910231928*
Venue:	*pyTexas 2009*
URL:	*http://www.pennington.net/tutorial*
License:	*Creative Commons Attribution-Noncommercial-Share Alike 3.0 US License*
Copyright:	*© 23 Oct 2009, David Michael Pennington - All rights reserved*
Plug:	*I'm available to consult, but I'm giving you the tools to do it yourself*

Assumptions

- You are reasonably familiar with python

Assumptions

- You are reasonably familiar with python
- You have a good understanding of regular expressions

Assumptions

- You are reasonably familiar with python
- You have a good understanding of regular expressions
- You have a need to interactively automate some text-console

Classic TCL/expect script...

```
#!/usr/bin/expect

set aRouter "HotCoffee.lab.net"
set login_prompt "Username: "
set userid "mpenning"

spawn /usr/bin/telnet $aRouter
expect {
  -re "$login_prompt\s*" {
     send "$userid"
  }
  timeout { error "Connect failed... received
$expect_out(buffer) instead of $login_prompt\r" }
}
```

Why do I want pexpect?

- Long answer:

Why do I want pexpect?

- Long answer:
  - Buffering low-level interactive character streams is not easy

Why do I want pexpect?

- Long answer:
  - Buffering low-level interactive character streams is not easy
  - pexpect provides a simple framework to handle...

Why do I want pexpect?

- Long answer:
  - Buffering low-level interactive character streams is not easy
  - pexpect provides a simple framework to handle...
    - Expired passwords or authentication service failure

Why do I want pexpect?

- Long answer:
  - Buffering low-level interactive character streams is not easy
  - pexpect provides a simple framework to handle...
    - Expired passwords or authentication service failure
    - Bugs / Crashes

Why do I want pexpect?

- Long answer:
  - Buffering low-level interactive character streams is not easy
  - pexpect provides a simple framework to handle...
    - Expired passwords or authentication service failure
    - Bugs / Crashes
    - Other interactive if-then decisions

Why do I want pexpect?

- Long answer:
  - Buffering low-level interactive character streams is not easy
  - pexpect provides a simple framework to handle...
    - Expired passwords or authentication service failure
    - Bugs / Crashes
    - Other interactive if-then decisions
    - See the next slide

Why do I want pexpect?

- Long answer:
  - Buffering low-level interactive character streams is not easy
  - pexpect provides a simple framework to handle...
    - Expired passwords or authentication service failure
    - Bugs / Crashes
    - Other interactive if-then decisions
    - See the next slide
  - Use Expect?

Why do I want pexpect?

- Long answer:
  - Buffering low-level interactive character streams is not easy
  - pexpect provides a simple framework to handle...
    - Expired passwords or authentication service failure
    - Bugs / Crashes
    - Other interactive if-then decisions
    - See the next slide
  - Use Expect?
    - I think TCL is clunky... I avoid it when I can

Why do I want pexpect?

- Long answer:
    - Buffering low-level interactive character streams is not easy
    - pexpect provides a simple framework to handle...
        - Expired passwords or authentication service failure
        - Bugs / Crashes
        - Other interactive if-then decisions
        - See the next slide
    - Use Expect?
        - I think TCL is clunky... I avoid it when I can
        - Sometimes expect's regexps have portability issues

Why do I want pexpect?

- Long answer:
  - Buffering low-level interactive character streams is not easy
  - pexpect provides a simple framework to handle...
    - Expired passwords or authentication service failure
    - Bugs / Crashes
    - Other interactive if-then decisions
    - See the next slide
  - Use Expect?
    - I think TCL is clunky... I avoid it when I can
    - Sometimes expect's regexps have portability issues
    - We're already using python. One language is better than two

Why do I want pexpect?

- Long answer:
  - Buffering low-level interactive character streams is not easy
  - pexpect provides a simple framework to handle...
    - Expired passwords or authentication service failure
    - Bugs / Crashes
    - Other interactive if-then decisions
    - See the next slide
  - Use Expect?
    - I think TCL is clunky... I avoid it when I can
    - Sometimes expect's regexps have portability issues
    - We're already using python. One language is better than two

Why do I want pexpect?

- Long answer:
  - Buffering low-level interactive character streams is not easy
  - pexpect provides a simple framework to handle...
    - Expired passwords or authentication service failure
    - Bugs / Crashes
    - Other interactive if-then decisions
    - See the next slide
  - Use Expect?
    - I think TCL is clunky... I avoid it when I can
    - Sometimes expect's regexps have portability issues
    - We're already using python. One language is better than two
  -
    ```
    # Either one works...
    if (python.love > TCL.love) == True:
        return "pexpect"
    else:
        return "expect"
    ```

I love p?[Ee]xpect because...

- Screen scraping: What else can you do if there is no SNMP MIB or XML / JSON API?

I love p?[Ee]xpect because...

- Screen scraping: What else can you do if there is no SNMP MIB or XML / JSON API?
- Secure backups: pull-model backup jobs (configure / secure one host instead of many)

I love p?[Ee]xpect because...

- Screen scraping: What else can you do if there is no SNMP MIB or XML / JSON API?
- Secure backups: pull-model backup jobs (configure / secure one host instead of many)
- Dev test software / hardware... put foo through 1000 reconfigurations and SIGHUPS. Does it leak memory or crash?

I love p?[Ee]xpect because...

- Screen scraping: What else can you do if there is no SNMP MIB or XML / JSON API?
- Secure backups: pull-model backup jobs (configure / secure one host instead of many)
- Dev test software / hardware... put foo through 1000 reconfigurations and SIGHUPS. Does it leak memory or crash?
- automated deployment / provisioning:

I love p?[Ee]xpect because...

- Screen scraping: What else can you do if there is no SNMP MIB or XML / JSON API?
- Secure backups: pull-model backup jobs (configure / secure one host instead of many)
- Dev test software / hardware... put foo through 1000 reconfigurations and SIGHUPS. Does it leak memory or crash?
- automated deployment / provisioning:
    - You have 200 Cisco routers / switches... reconfigure them quickly

I love p?[Ee]xpect because...

- Screen scraping: What else can you do if there is no SNMP MIB or XML / JSON API?
- Secure backups: pull-model backup jobs (configure / secure one host instead of many)
- Dev test software / hardware... put foo through 1000 reconfigurations and SIGHUPS. Does it leak memory or crash?
- automated deployment / provisioning:
  - You have 200 Cisco routers / switches... reconfigure them quickly
  - Some SW installers ask many questions... automate!

I love p?[Ee]xpect because...

- Screen scraping: What else can you do if there is no SNMP MIB or XML / JSON API?
- Secure backups: pull-model backup jobs (configure / secure one host instead of many)
- Dev test software / hardware... put foo through 1000 reconfigurations and SIGHUPS. Does it leak memory or crash?
- automated deployment / provisioning:
  - You have 200 Cisco routers / switches... reconfigure them quickly
  - Some SW installers ask many questions... automate!
- Services verification:

I love p?[Ee]xpect because...

- Screen scraping: What else can you do if there is no SNMP MIB or XML / JSON API?
- Secure backups: pull-model backup jobs (configure / secure one host instead of many)
- Dev test software / hardware... put foo through 1000 reconfigurations and SIGHUPS. Does it leak memory or crash?
- automated deployment / provisioning:
  - You have 200 Cisco routers / switches... reconfigure them quickly
  - Some SW installers ask many questions... automate!
- Services verification:
  - Dialup modem lines... use them or loose them

I love p?[Ee]xpect because...

- Screen scraping: What else can you do if there is no SNMP MIB or XML / JSON API?
- Secure backups: pull-model backup jobs (configure / secure one host instead of many)
- Dev test software / hardware... put foo through 1000 reconfigurations and SIGHUPS. Does it leak memory or crash?
- automated deployment / provisioning:
  - You have 200 Cisco routers / switches... reconfigure them quickly
  - Some SW installers ask many questions... automate!
- Services verification:
  - Dialup modem lines... use them or loose them
  - NOC console-server access checks

I love p?[Ee]xpect because...

- Screen scraping: What else can you do if there is no SNMP MIB or XML / JSON API?
- Secure backups: pull-model backup jobs (configure / secure one host instead of many)
- Dev test software / hardware... put foo through 1000 reconfigurations and SIGHUPS. Does it leak memory or crash?
- automated deployment / provisioning:
  - You have 200 Cisco routers / switches... reconfigure them quickly
  - Some SW installers ask many questions... automate!
- Services verification:
  - Dialup modem lines... use them or loose them
  - NOC console-server access checks
  - Network latency measurement *from the router's perspective*

I love p?[Ee]xpect because...

- Screen scraping: What else can you do if there is no SNMP MIB or XML / JSON API?
- Secure backups: pull-model backup jobs (configure / secure one host instead of many)
- Dev test software / hardware... put foo through 1000 reconfigurations and SIGHUPS. Does it leak memory or crash?
- automated deployment / provisioning:
  - You have 200 Cisco routers / switches... reconfigure them quickly
  - Some SW installers ask many questions... automate!
- Services verification:
  - Dialup modem lines... use them or loose them
  - NOC console-server access checks
  - Network latency measurement *from the router's perspective*
  - SMTP / IMAP / whatever SLA monitoring

What is a pty?

- pexpect requires pty support

What is a pty?

- pexpect requires pty support
- pty = Pseudo Terminal

What is a pty?

- pexpect requires pty support
- pty = Pseudo Terminal
- Fancy name for the way ssh and telnet interface with the server

What is a pty?

- pexpect requires pty support
- pty = Pseudo Terminal
- Fancy name for the way ssh and telnet interface with the server
- Consequently, pexpect currently:

What is a pty?

- pexpect requires pty support
- pty = Pseudo Terminal
- Fancy name for the way ssh and telnet interface with the server
- Consequently, pexpect currently:
    - Supports: *BSD, OSX, Linux, Cygwin python

What is a pty?

- pexpect requires pty support
- pty = Pseudo Terminal
- Fancy name for the way ssh and telnet interface with the server
- Consequently, pexpect currently:
  - Supports: *BSD, OSX, Linux, Cygwin python
  - Supports with a caveat: Solaris Python (according to the FAQ)

What is a pty?

- pexpect requires pty support
- pty = Pseudo Terminal
- Fancy name for the way ssh and telnet interface with the server
- Consequently, pexpect currently:
    - Supports: *BSD, OSX, Linux, Cygwin python
    - Supports with a caveat: Solaris Python (according to the FAQ)
    - No support: Native win32 Python

Installation

- Download from pypi or use `easy_install`

Note

pexpect 2.4 does not have a full documentation tree on pypi. Use http://pexpect.sourceforge.net/pexpect.html for 2.3 API docs

Installation

- Download from pypi or use `easy_install`
- Do not install from sourceforge...

Note

pexpect 2.4 does not have a full documentation tree on pypi. Use http://pexpect.sourceforge.net/pexpect.html for 2.3 API docs

Installation

- Download from pypi or use `easy_install`
- Do not install from sourceforge...
    - Sourceforge shows up way before pypi in google searches for 'pexpect'

Note

pexpect 2.4 does not have a full documentation tree on pypi. Use http://pexpect.sourceforge.net/pexpect.html for 2.3 API docs

Installation

- Download from pypi or use `easy_install`
- Do not install from sourceforge...
  - Sourceforge shows up way before pypi in google searches for 'pexpect'
  - As of this writing, SF.net is one version behind the pypi tarball

Note

pexpect 2.4 does not have a full documentation tree on pypi. Use http://pexpect.sourceforge.net/pexpect.html for 2.3 API docs

Contrived Example

- Assume we're in a directory with a file named `foosay.txt` which has the following contents:

Contrived Example

- Assume we're in a directory with a file named `foosay.txt` which has the following contents:

Contrived Example

- Assume we're in a directory with a file named `foosay.txt` which has the following contents:

-
  ```
      Lorem ipsum dolor sit amet,
      consectetur adipiscing elit.
  ```

Contrived Example

- Assume we're in a directory with a file named `foosay.txt` which has the following contents:
-
```
    Lorem ipsum dolor sit amet,
    consectetur adipiscing elit.
```
- See next slide for example...

Contrived Example (cont'd)

```
>>> import pexpect
>>> child = pexpect.spawn('cat foosay.txt')
>>> child.expect(pexpect.EOF)
0
>>> print child.before
Lorem ipsum dolor sit amet,
consectetur adipiscing elit.

>>> child.before
'Lorem ipsum dolor sit amet,\r\nconsectetur
adipiscing elit.\r\n'
>>>
```

Basic pexpect overview

- Take special note of the `\r\n` sequence used in the output above...

Basic pexpect overview

- Take special note of the `\r\n` sequence used in the output above...
  - that was on a debian Linux box,

Basic pexpect overview

- Take special note of the `\r\n` sequence used in the output above...
  - that was on a debian Linux box,
  - You say "But `\r\n` is a *DOS* EOL terminator!"

Basic pexpect overview

- Take special note of the `\r\n` sequence used in the output above...
  - that was on a debian Linux box,
  - You say "But `\r\n` is a *DOS* EOL terminator!"
  - Welcome to ptys. They use a CRLF combo (`\r\n`), even in *nix

Basic pexpect overview

- Take special note of the `\r\n` sequence used in the output above...
    - that was on a debian Linux box,
    - You say "But `\r\n` is a *DOS* EOL terminator!"
    - Welcome to ptys. They use a CRLF combo (`\r\n`), even in *nix
- `pexpect.spawn()` starts an interactive session in a pty using the args supplied

Basic pexpect overview

- Take special note of the `\r\n` sequence used in the output above...
  - that was on a debian Linux box,
  - You say "But `\r\n` is a *DOS* EOL terminator!"
  - Welcome to ptys. They use a CRLF combo (`\r\n`), even in *nix
- `pexpect.spawn()` starts an interactive session in a pty using the args supplied
- Push input to the pty

Basic pexpect overview

- Take special note of the `\r\n` sequence used in the output above...
  - that was on a debian Linux box,
  - You say "But `\r\n` is a *DOS* EOL terminator!"
  - Welcome to ptys. They use a CRLF combo (`\r\n`), even in *nix
- `pexpect.spawn()` starts an interactive session in a pty using the args supplied
- Push input to the pty
  - `pexpect.send('foo')`: Send 'foo' to the pty

Basic pexpect overview

- Take special note of the `\r\n` sequence used in the output above...
  - that was on a debian Linux box,
  - You say “But `\r\n` is a *DOS* EOL terminator!”
  - Welcome to ptys. They use a CRLF combo (`\r\n`), even in *nix
- `pexpect.spawn()` starts an interactive session in a pty using the args supplied
- Push input to the pty
  - `pexpect.send('foo')`: Send 'foo' to the pty
  - `pexpect.sendline('foo')`: Send 'foo\n' to the pty

Basic pexpect overview

- Take special note of the `\r\n` sequence used in the output above...
  - that was on a debian Linux box,
  - You say "But `\r\n` is a *DOS* EOL terminator!"
  - Welcome to ptys. They use a CRLF combo (`\r\n`), even in *nix
- `pexpect.spawn()` starts an interactive session in a pty using the args supplied
- Push input to the pty
  - `pexpect.send('foo')`: Send 'foo' to the pty
  - `pexpect.sendline('foo')`: Send 'foo\n' to the pty
  - `pexpect.send()` wasn't required in the last example

Basic pexpect overview

- Take special note of the `\r\n` sequence used in the output above...
  - that was on a debian Linux box,
  - You say “But `\r\n` is a *DOS* EOL terminator!”
  - Welcome to ptys. They use a CRLF combo (`\r\n`), even in *nix
- `pexpect.spawn()` starts an interactive session in a pty using the args supplied
- Push input to the pty
  - `pexpect.send('foo')`: Send 'foo' to the pty
  - `pexpect.sendline('foo')`: Send 'foo\n' to the pty
  - `pexpect.send()` wasn't required in the last example
- Use `pexpect.expect()` to read a response from the pty

Basic pexpect overview

- Take special note of the `\r\n` sequence used in the output above...
  - that was on a debian Linux box,
  - You say "But `\r\n` is a *DOS* EOL terminator!"
  - Welcome to ptys. They use a CRLF combo (`\r\n`), even in *nix
- `pexpect.spawn()` starts an interactive session in a pty using the args supplied
- Push input to the pty
  - `pexpect.send('foo')`: Send 'foo' to the pty
  - `pexpect.sendline('foo')`: Send 'foo\n' to the pty
  - `pexpect.send()` wasn't required in the last example
- Use `pexpect.expect()` to read a response from the pty
- Use `pexpect.close()` to gracefully close the session

Basic pexpect overview (cont'd)

You really wanted to know about `child.before`, right?

```
[snip]
>>> child.expect(pexpect.EOF)
0
>>> child.before
'Lorem ipsum dolor sit amet,\r\nconsectetur
adipiscing elit.\r\n'
>>>
```

- `child.before` returns all strings between the last match and the current match.

Basic pexpect overview (cont'd)

You really wanted to know about `child.before`, right?

```
[snip]
>>> child.expect(pexpect.EOF)
0
>>> child.before
'Lorem ipsum dolor sit amet,\r\nconsectetur
adipiscing elit.\r\n'
>>>
```

- `child.before` returns all strings between the last match and the current match.
  - `child` is an arbitrary name for pexpect's spawn object

Basic pexpect overview (cont'd)

You really wanted to know about `child.before`, right?

```
[snip]
>>> child.expect(pexpect.EOF)
0
>>> child.before
'Lorem ipsum dolor sit amet,\r\nconsectetur
adipiscing elit.\r\n'
>>>
```

- `child.before` returns all strings between the last match and the current match.
    - `child` is an arbitrary name for pexpect's spawn object
    - The matched string is *excluded* from `child.before`

Basic pexpect overview (cont'd)

You really wanted to know about `child.before`, right?

```
[snip]
>>> child.expect(pexpect.EOF)
0
>>> child.before
'Lorem ipsum dolor sit amet,\r\nconsectetur
adipiscing elit.\r\n'
>>>
```

- `child.before` returns all strings between the last match and the current match.
  - `child` is an arbitrary name for pexpect's spawn object
  - The matched string is *excluded* from `child.before`
- `child.after` gets the matched string and everything after it

pexpect.run()

Simplifying the non-interactive shell exec in the last example...

```
>>> foo = pexpect.run('cat foosay.txt')
>>> print foo
Lorem ipsum dolor sit amet,
consectetur adipiscing elit.

>>> foo
'Lorem ipsum dolor sit amet,\r\nconsectetur
adipiscing elit.\r\n'
>>>
```

pexpect.run() (optional shell exit code)

- pexpect.run() also supports a shell exit code

```
>>> (foo, status) = \
...      pexpect.run('cat foosay.txt', \
...      withexitstatus = True)
>>> foo
'Lorem ipsum dolor sit amet,\r\nconsectetur
adipiscing elit.\r\n'
>>> status
0
>>>
```

pexpect.run() (optional shell exit code)

- `pexpect.run()` also supports a shell exit code
    - 0 is clean exit

```
>>> (foo, status) = \
...       pexpect.run('cat foosay.txt', \
...       withexitstatus = True)
>>> foo
'Lorem ipsum dolor sit amet,\r\nconsectetur
adipiscing elit.\r\n'
>>> status
0
>>>
```

pexpect.run() (optional shell exit code)

- `pexpect.run()` also supports a shell exit code
  - 0 is clean exit
  - > 0 is a problem

```
>>> (foo, status) = \
...      pexpect.run('cat foosay.txt', \
...      withexitstatus = True)
>>> foo
'Lorem ipsum dolor sit amet,\r\nconsectetur
adipiscing elit.\r\n'
>>> status
0
>>>
```

pexpect.run() (shell exit code - cont'd)

- Note status after we ask for a bogus file...

```
>>> (foo, status) = pexpect.run('cat imNotHere.txt', \
...      withexitstatus=True)
>>> status
 1
>>>
```

Contrived interactive example

```
>>> import pexpect
>>> child = \
...     pexpect.spawn('ssh geeky.getaway.local')
>>> child.expect('assword:')
0
>>> child.sendline('b1gS3cr3t~')
10
>>> child.expect('$')
0
>>> child.before
"Welcome back...\r\nPlease don't break
anything\r\n[me@geeky ~]"
>>> child.after
'$'
>>>
```

pexpect errors

- If the datastream times out, pexpect will raise `pexpect.TIMEOUT`

pexpect errors

- If the datastream times out, pexpect will raise `pexpect.TIMEOUT`
- If the datastream ends, pexpect will raise `pexpect.EOF`

pexpect errors

- If the datastream times out, pexpect will raise `pexpect.TIMEOUT`
- If the datastream ends, pexpect will raise `pexpect.EOF`
- Use them in `pexpect.expect()`

pexpect errors

- If the datastream times out, pexpect will raise `pexpect.TIMEOUT`
- If the datastream ends, pexpect will raise `pexpect.EOF`
- Use them in `pexpect.expect()`
- Caveats

pexpect errors

- If the datastream times out, pexpect will raise `pexpect.TIMEOUT`
- If the datastream ends, pexpect will raise `pexpect.EOF`
- Use them in `pexpect.expect()`
- Caveats
  - There are multiple reasons for seeing `pexpect.TIMEOUT` or `pexpect.EOF`

pexpect errors

- If the datastream times out, pexpect will raise `pexpect.TIMEOUT`
- If the datastream ends, pexpect will raise `pexpect.EOF`
- Use them in `pexpect.expect()`
- Caveats
  - There are multiple reasons for seeing `pexpect.TIMEOUT` or `pexpect.EOF`
    - Unreliable network connectivity

pexpect errors

- If the datastream times out, pexpect will raise `pexpect.TIMEOUT`
- If the datastream ends, pexpect will raise `pexpect.EOF`
- Use them in `pexpect.expect()`
- Caveats
    - There are multiple reasons for seeing `pexpect.TIMEOUT` or `pexpect.EOF`
        - Unreliable network connectivity
        - Overutilized host (i.e. disk swapping?)

pexpect errors

- If the datastream times out, pexpect will raise `pexpect.TIMEOUT`
- If the datastream ends, pexpect will raise `pexpect.EOF`
- Use them in `pexpect.expect()`
- Caveats
  - There are multiple reasons for seeing `pexpect.TIMEOUT` or `pexpect.EOF`
    - Unreliable network connectivity
    - Overutilized host (i.e. disk swapping?)
    - Bugs in OS

pexpect errors

- If the datastream times out, pexpect will raise `pexpect.TIMEOUT`
- If the datastream ends, pexpect will raise `pexpect.EOF`
- Use them in `pexpect.expect()`
- Caveats
  - There are multiple reasons for seeing `pexpect.TIMEOUT` or `pexpect.EOF`
    - Unreliable network connectivity
    - Overutilized host (i.e. disk swapping?)
    - Bugs in OS
  - I can't remember a fatal problem in the pexpect library

pexpect errors

- If the datastream times out, pexpect will raise `pexpect.TIMEOUT`
- If the datastream ends, pexpect will raise `pexpect.EOF`
- Use them in `pexpect.expect()`
- Caveats
  - There are multiple reasons for seeing `pexpect.TIMEOUT` or `pexpect.EOF`
    - Unreliable network connectivity
    - Overutilized host (i.e. disk swapping?)
    - Bugs in OS
  - I can't remember a fatal problem in the pexpect library
  - Script failures are often useful host/network monitoring subsystems in themselves

pexpect errors

- If the datastream times out, pexpect will raise `pexpect.TIMEOUT`
- If the datastream ends, pexpect will raise `pexpect.EOF`
- Use them in `pexpect.expect()`
- Caveats
    - There are multiple reasons for seeing `pexpect.TIMEOUT` or `pexpect.EOF`
        - Unreliable network connectivity
        - Overutilized host (i.e. disk swapping?)
        - Bugs in OS
    - I can't remember a fatal problem in the pexpect library
    - Script failures are often useful host/network monitoring subsystems in themselves
    - Script failures alone cannot replace a real Enterprise Monitoring System (like nagios or HPOV)

pexpect decision tree

- We said the real world isn't very nice to linear assumptions

pexpect decision tree

- We said the real world isn't very nice to linear assumptions
- This implies a decision tree for pexpect sessions

pexpect decision tree

- We said the real world isn't very nice to linear assumptions
- This implies a decision tree for pexpect sessions
    - If foo1, then bar1

pexpect decision tree

- We said the real world isn't very nice to linear assumptions
- This implies a decision tree for pexpect sessions
  - If foo1, then bar1
  - If foo2, then bar2...

Simple pexpect decision tree

General usage:

```
pexpect.expect([ foo1, foo2, ...])
```

Example:

```
ii=child.expect(['sername:',
    pexpect.EOF, pexpect.TIMEOUT])
if ii==0:
    ## Do something here
    pexpect.send(username)
elif ii==1:
    ## Log an error; page sysadmin
elif ii==2:
    ## Log an error; send logs in an email
```

Regular expression notes

Note

- pexpect reads streams of characters one character at a time

Regular expression notes

Note

- pexpect reads streams of characters one character at a time
    - This is a non-trivial difference

Regular expression notes

Note

- pexpect reads streams of characters one character at a time
    - This is a non-trivial difference
    - Regular Expressions that require look-ahead don't work as you expect

Regular expression notes

Note

- pexpect reads streams of characters one character at a time
    - This is a non-trivial difference
    - Regular Expressions that require look-ahead don't work as you expect

- Matching the end of a line

Regular expression notes

Note

- pexpect reads streams of characters one character at a time
    - This is a non-trivial difference
    - Regular Expressions that require look-ahead don't work as you expect

- Matching the end of a line
    - $ requires 'normal' python REs to look-ahead; don't use it

Regular expression notes

Note

- pexpect reads streams of characters one character at a time
  - This is a non-trivial difference
  - Regular Expressions that require look-ahead don't work as you expect

- Matching the end of a line
  - $ requires 'normal' python REs to look-ahead; don't use it
  - '`\r\n`' will accomplish the same things that $ was intended to

Regular expression notes (cont'd)

- When using wildcards, 'anchor' your regexps as much as possible

Regular expression notes (cont'd)

- When using wildcards, 'anchor' your regexps as much as possible
    - Anchoring just means bounding the regexp with a non-wildcard

Regular expression notes (cont'd)

- When using wildcards, 'anchor' your regexps as much as possible
  - Anchoring just means bounding the regexp with a non-wildcard
  - An anchored regexp to a generic bash root prompt looks something like this: `\r\n\S+.+?#`

Regular expression notes (cont'd)

- When using wildcards, 'anchor' your regexps as much as possible
    - Anchoring just means bounding the regexp with a non-wildcard
    - An anchored regexp to a generic bash root prompt looks something like this: `\r\n\S+.+?#`
        - '`\r\n`' anchors the left-hand side

Regular expression notes (cont'd)

- When using wildcards, 'anchor' your regexps as much as possible
  - Anchoring just means bounding the regexp with a non-wildcard
  - An anchored regexp to a generic bash root prompt looks something like this: `\r\n\S+.+?#`
    - '`\r\n`' anchors the left-hand side
    - '#' anchors the right-hand side

Regular expression notes (cont'd)

- When using wildcards, 'anchor' your regexps as much as possible
  - Anchoring just means bounding the regexp with a non-wildcard
  - An anchored regexp to a generic bash root prompt looks something like this: `\r\n\S+.+?#`
    - '`\r\n`' anchors the left-hand side
    - '`#`' anchors the right-hand side
- Due to single character processing, pexpect regexps are *non greedy*

Regular expression notes (cont'd)

- When using wildcards, 'anchor' your regexps as much as possible
  - Anchoring just means bounding the regexp with a non-wildcard
  - An anchored regexp to a generic bash root prompt looks something like this: `\r\n\S+.+?#`
    - '`\r\n`' anchors the left-hand side
    - '`#`' anchors the right-hand side
- Due to single character processing, pexpect regexps are *non greedy*
  - That means I really didn't need to add `?` after `.+` above

Regular expression notes (cont'd)

- When using wildcards, 'anchor' your regexps as much as possible
  - Anchoring just means bounding the regexp with a non-wildcard
  - An anchored regexp to a generic bash root prompt looks something like this: `\r\n\S+.+?#`
    - '`\r\n`' anchors the left-hand side
    - '`#`' anchors the right-hand side
- Due to single character processing, pexpect regexps are *non greedy*
  - That means I really didn't need to add `?` after `.+` above
  - I like the consistency with other regexps, and it doesn't hurt pexpect

Regular expression match groups

- Match groups are the key to extracting data...

Regular expression match groups

- Match groups are the key to extracting data...
- Example:

Regular expression match groups

- Match groups are the key to extracting data...
- Example:
  - `child.expect('[Ss]ome*([Rr]eg\w+?)*[Hh]ere')`

Regular expression match groups

- Match groups are the key to extracting data...
- Example:
  - `child.expect('[Ss]ome*([Rr]eg\w+?)*[Hh]ere')`
  - `child.match.group(foo)`

Regular expression match groups

- Match groups are the key to extracting data...
- Example:
  - `child.expect('[Ss]ome*([Rr]eg\w+?)*[Hh]ere')`
  - `child.match.group(foo)`
    - `foo` is an integer index

Regular expression match groups

- Match groups are the key to extracting data...
- Example:
  - `child.expect('[Ss]ome*([Rr]eg\w+?)*[Hh]ere')`
  - `child.match.group(foo)`
    - `foo` is an integer index
    - *`foo = 0` is the entire regexp match*

Regular expression match groups

- Match groups are the key to extracting data...
- Example:
  - `child.expect('[Ss]ome*([Rr]eg\w+?)*[Hh]ere')`
  - `child.match.group(foo)`
    - `foo` is an integer index
    - `foo = 0` *is the entire regexp match*
    - `foo = 1` *matches the first parenthesis*

Regular expression match groups

- Match groups are the key to extracting data...
- Example:
  - `child.expect('[Ss]ome*([Rr]eg\w+?)*[Hh]ere')`
  - `child.match.group(foo)`
    - `foo` is an integer index
    - `foo = 0` *is the entire regexp match*
    - `foo = 1` *matches the first parenthesis*
    - `foo = 2` matches the second parenthesis (if you had one)

Regular expression match groups

- Match groups are the key to extracting data...
- Example:
  - `child.expect('[Ss]ome*([Rr]eg\w+?)*[Hh]ere')`
  - `child.match.group(foo)`
    - `foo` is an integer index
    - `foo = 0` *is the entire regexp match*
    - `foo = 1` *matches the first parenthesis*
    - `foo = 2` matches the second parenthesis (if you had one)
- Illustrate with an example... Let's suppose you wanted to build a generic host connector that...

Regular expression match groups

- Match groups are the key to extracting data...
- Example:
  - `child.expect('[Ss]ome*([Rr]eg\w+?)*[Hh]ere')`
  - `child.match.group(foo)`
    - `foo` is an integer index
    - `foo = 0` *is the entire regexp match*
    - `foo = 1` *matches the first parenthesis*
    - `foo = 2` matches the second parenthesis (if you had one)
- Illustrate with an example... Let's suppose you wanted to build a generic host connector that...
  - Takes a list of hosts, usernames, and passwords

Regular expression match groups

- Match groups are the key to extracting data...
- Example:
  - `child.expect('[Ss]ome*([Rr]eg\w+?)*[Hh]ere')`
  - `child.match.group(foo)`
    - `foo` is an integer index
    - `foo = 0` *is the entire regexp match*
    - `foo = 1` *matches the first parenthesis*
    - `foo = 2` matches the second parenthesis (if you had one)
- Illustrate with an example... Let's suppose you wanted to build a generic host connector that...
  - Takes a list of hosts, usernames, and passwords
  - Validates that each host has a prompt that meets IT standards

Regular expression match groups

- Match groups are the key to extracting data...
- Example:
  - `child.expect('[Ss]ome*([Rr]eg\w+?)*[Hh]ere')`
  - `child.match.group(foo)`
    - `foo` is an integer index
    - `foo = 0` *is the entire regexp match*
    - `foo = 1` *matches the first parenthesis*
    - `foo = 2` matches the second parenthesis (if you had one)
- Illustrate with an example... Let's suppose you wanted to build a generic host connector that...
  - Takes a list of hosts, usernames, and passwords
  - Validates that each host has a prompt that meets IT standards
  - Assume prompts are limited to a single line

Regular expression match groups

- Match groups are the key to extracting data...
- Example:
  - `child.expect('[Ss]ome*([Rr]eg\w+?)*[Hh]ere')`
  - `child.match.group(foo)`
    - `foo` is an integer index
    - `foo = 0` *is the entire regexp match*
    - `foo = 1` *matches the first parenthesis*
    - `foo = 2` matches the second parenthesis (if you had one)
- Illustrate with an example... Let's suppose you wanted to build a generic host connector that...
  - Takes a list of hosts, usernames, and passwords
  - Validates that each host has a prompt that meets IT standards
  - Assume prompts are limited to a single line
- The requirement explicitly requires you assume very little about the prompt

Regular expression match groups

- Match groups are the key to extracting data...
- Example:
    - `child.expect('[Ss]ome*([Rr]eg\w+?)*[Hh]ere')`
    - `child.match.group(foo)`
        - `foo` is an integer index
        - `foo = 0` *is the entire regexp match*
        - `foo = 1` *matches the first parenthesis*
        - `foo = 2` matches the second parenthesis (if you had one)
- Illustrate with an example... Let's suppose you wanted to build a generic host connector that...
    - Takes a list of hosts, usernames, and passwords
    - Validates that each host has a prompt that meets IT standards
    - Assume prompts are limited to a single line
- The requirement explicitly requires you assume very little about the prompt
- Is this possible?

Regular expression match groups (cont'd)

Obvously the answer is yes. It also helps a bit to know how the host terminates lines. Some hosts (like the Cisco ASA in this example) add an extra CR at the end of the line

```
>>> child = pexpect.spawn('ssh %s@%s' % (usr, addr))
>>> child.sendline(passwd)
>>> child.expect('')
>>> child.send('\n\n')
>>> child.expect('\r\n\r(.+?)\r\n\r')
>>> child.match.group(1)
'mpenning-fw> '
>>> child.match.group(0)
'\r\n\rmpenning-fw> \r\n\r'
>>>
```

pxssh

- So far I've been using `pexpect.spawn()` for ssh sessions

pxssh

- So far I've been using `pexpect.spawn()` for ssh sessions
- Perhaps you've wondered about those corner cases in the real world...

pxssh

- So far I've been using `pexpect.spawn()` for ssh sessions
- Perhaps you've wondered about those corner cases in the real world...
    - New server key

pxssh

- So far I've been using `pexpect.spawn()` for ssh sessions
- Perhaps you've wondered about those corner cases in the real world...
  - New server key
  - Public-key authentication without the password

pxssh

- So far I've been using `pexpect.spawn()` for ssh sessions
- Perhaps you've wondered about those corner cases in the real world...
  - New server key
  - Public-key authentication without the password
- `pxssh` is a subclass of `pexpect` that specializes in ssh interaction

pxssh

- So far I've been using `pexpect.spawn()` for ssh sessions
- Perhaps you've wondered about those corner cases in the real world...
    - New server key
    - Public-key authentication without the password
- `pxssh` is a subclass of `pexpect` that specializes in ssh interaction
    - The API is different

pxssh

- So far I've been using `pexpect.spawn()` for ssh sessions
- Perhaps you've wondered about those corner cases in the real world...
    - New server key
    - Public-key authentication without the password
- `pxssh` is a subclass of `pexpect` that specializes in ssh interaction
    - The API is different
    - It automagically recognizes prompts

pxssh

- So far I've been using `pexpect.spawn()` for ssh sessions
- Perhaps you've wondered about those corner cases in the real world...
    - New server key
    - Public-key authentication without the password
- `pxssh` is a subclass of `pexpect` that specializes in ssh interaction
    - The API is different
    - It automagically recognizes prompts
    - I build my own ssh handler with pexpect because I like the consistency, but you might like `pxssh`

pxssh

- So far I've been using `pexpect.spawn()` for ssh sessions
- Perhaps you've wondered about those corner cases in the real world...
  - New server key
  - Public-key authentication without the password
- `pxssh` is a subclass of `pexpect` that specializes in ssh interaction
  - The API is different
  - It automagically recognizes prompts
  - I build my own ssh handler with pexpect because I like the consistency, but you might like `pxssh`
- See the pxssh docs on SF.net

Logging

- When you're configuring things, you *need* to log what you're doing

Logging

- When you're configuring things, you *need* to log what you're doing
- Human error and bugs are always a risk... even without automated config changes

Logging

- When you're configuring things, you *need* to log what you're doing
- Human error and bugs are always a risk... even without automated config changes
    - Maybe you didn't find this case in the lab

Logging

- When you're configuring things, you *need* to log what you're doing
- Human error and bugs are always a risk... even without automated config changes
    - Maybe you didn't find this case in the lab
    - Yes, I did assume you tested the script in the lab :-)

Logging

- When you're configuring things, you *need* to log what you're doing
- Human error and bugs are always a risk... even without automated config changes
    - Maybe you didn't find this case in the lab
    - Yes, I did assume you tested the script in the lab :-)
    - If something goes wrong in production, you need logs to send to your boss and the vendor

Logging to a file

```
>>> child = pexpect.spawn('ssh buggy.host.local')
>>> wh = open('task01.buggy.log', 'w')
>>> child.logfile = wh
>>> # -> insert child.expect() and child.send()
>>> #    interaction here
>>> child.logfile = sys.stdout
>>> wh.close()
>>> child.close()
```

Hiding passwords

- There are a couple of different deployment models for this kind of automation

Hiding passwords

- There are a couple of different deployment models for this kind of automation
- How do we protect passwords well?

Hiding passwords

- There are a couple of different deployment models for this kind of automation
- How do we protect passwords well?
- If spawned by cron or anacron

Hiding passwords

- There are a couple of different deployment models for this kind of automation
- How do we protect passwords well?
- If spawned by cron or anacron
    - You could use `ssh-agent` for the user calling the script

Hiding passwords

- There are a couple of different deployment models for this kind of automation
- How do we protect passwords well?
- If spawned by cron or anacron
  - You could use `ssh-agent` for the user calling the script
    - `ssh-agent` requires certificate authentication and and admin to type passwords at least once per server reboot.

Hiding passwords

- There are a couple of different deployment models for this kind of automation
- How do we protect passwords well?
- If spawned by cron or anacron
  - You could use `ssh-agent` for the user calling the script
    - `ssh-agent` requires certificate authentication and and admin to type passwords at least once per server reboot.
    - `ssh-agent` is also a bit involved to setup securely. Google for it and you'll get good tutorials like http://mah.everybody.org/docs/ssh

Hiding passwords

- There are a couple of different deployment models for this kind of automation
- How do we protect passwords well?
- If spawned by cron or anacron
  - You could use `ssh-agent` for the user calling the script
    - `ssh-agent` requires certificate authentication and and admin to type passwords at least once per server reboot.
    - `ssh-agent` is also a bit involved to setup securely. Google for it and you'll get good tutorials like http://mah.everybody.org/docs/ssh
    - Note: some hosts don't support SSH public key authentication

Hiding passwords

- There are a couple of different deployment models for this kind of automation
- How do we protect passwords well?
- If spawned by cron or anacron
  - You could use `ssh-agent` for the user calling the script
    - `ssh-agent` requires certificate authentication and and admin to type passwords at least once per server reboot.
    - `ssh-agent` is also a bit involved to setup securely. Google for it and you'll get good tutorials like http://mah.everybody.org/docs/ssh
    - Note: some hosts don't support SSH public key authentication
  - If `ssh-agent` doesn't work for you, passwords need to be stored in the script or some other local file

Hiding passwords

- There are a couple of different deployment models for this kind of automation
- How do we protect passwords well?
- If spawned by cron or anacron
  - You could use `ssh-agent` for the user calling the script
    - `ssh-agent` requires certificate authentication and and admin to type passwords at least once per server reboot.
    - `ssh-agent` is also a bit involved to setup securely. Google for it and you'll get good tutorials like http://mah.everybody.org/docs/ssh
    - Note: some hosts don't support SSH public key authentication
  - If `ssh-agent` doesn't work for you, passwords need to be stored in the script or some other local file
    - If your hosts support it, use two-factor SSH public keys with *non-empty* passwords.

Hiding passwords

- There are a couple of different deployment models for this kind of automation
- How do we protect passwords well?
- If spawned by cron or anacron
    - You could use `ssh-agent` for the user calling the script
        - `ssh-agent` requires certificate authentication and and admin to type passwords at least once per server reboot.
        - `ssh-agent` is also a bit involved to setup securely. Google for it and you'll get good tutorials like http://mah.everybody.org/docs/ssh
        - Note: some hosts don't support SSH public key authentication
    - If `ssh-agent` doesn't work for you, passwords need to be stored in the script or some other local file
        - If your hosts support it, use two-factor SSH public keys with *non-empty* passwords.
        - At least it prevents someone from seeing the password and using it themselves on another device / user acct.

Hiding passwords (cont'd)

- Spawned manually

Hiding passwords (cont'd)

- Spawned manually
    - You can hide credentials pretty easily

Hiding passwords (cont'd)

- Spawned manually
  - You can hide credentials pretty easily
  - Use stdlib's *getpass* module...

Hiding passwords (cont'd)

- Spawned manually
  - You can hide credentials pretty easily
  - Use stdlib's *getpass* module...

Hiding passwords (cont'd)

- Spawned manually
  - You can hide credentials pretty easily
  - Use stdlib's *getpass* module...

-
  ```
  passwd = getpass.getpass("Enter password: ")
  ```

Questions or Comments?