NAME

chat − Automated conversational script with a modem

SYNOPSIS

chat [*options*] *script*

DESCRIPTION

The *chat* program defines a conversational exchange between the computer and the modem. Its primary purpose is to establish the connection between the Point-to-Point Protocol Daemon (*pppd*) and the remote's *pppd* process.

OPTIONS

−f *<chat file>*

Read the chat script from the chat *file*. The use of this option is mutually exclusive with the chat script parameters. The user must have read access to the file. Multiple lines are permitted in the file. Space or horizontal tab characters should be used to separate the strings.

−t *<timeout>*

Set the timeout for the expected string to be received. If the string is not received within the time limit then the reply string is not sent. An alternate reply may be sent or the script will fail if there is no alternate reply string. A failed script will cause the *chat* program to terminate with a non-zero error code. You can also use the TIMEOUT string in order to specify the timeout.

−r *<report file>*

Set the file for output of the report strings. If you use the keyword *REPORT*, the resulting strings are written to this file. If this option is not used and you still use *REPORT* keywords, the *stderr* file is used for the report strings.

−e Start with the echo option turned on. Echoing may also be turned on or off at specific points in the chat script by using the *ECHO* keyword. When echoing is enabled, all output from the modem is echoed to *stderr*.

−E Enables environment variable substitution within chat scripts using the standard *$xxx* syntax.

−v Request that the *chat* script be executed in a verbose mode. The *chat* program will then log the execution state of the chat script as well as all text received from the modem and the output strings sent to the modem. The default is to log through the SYSLOG; the logging method may be altered with the −S and −s flags.

−V Request that the *chat* script be executed in a stderr verbose mode. The *chat* program will then log all text received from the modem and the output strings sent to the modem to the stderr device. This device is usually the local console at the station running the chat or pppd program.

−s Use stderr. All log messages from '−v' and all error messages will be sent to stderr.

−S Do not use the SYSLOG. By default, error messages are sent to the SYSLOG. The use of −S will prevent both log messages from '−v' and error messages from being sent to the SYSLOG.

−T *<phone number>*

Pass in an arbitrary string, usually a phone number, that will be substituted for the \T substitution metacharacter in a send string.

−U *<phone number 2>*

Pass in a second string, usually a phone number, that will be substituted for the \U substitution metacharacter in a send string. This is useful when dialing an ISDN terminal adapter that requires two numbers.

script If the script is not specified in a file with the *−f* option then the script is included as parameters to the *chat* program.

CHAT SCRIPT

The *chat* script defines the communications.

A script consists of one or more "expect−send" pairs of strings, separated by spaces, with an optional

"subexpect−subsend" string pair, separated by a dash as in the following example:

ogin:−BREAK−ogin: ppp ssword: hello2u2

This line indicates that the *chat* program should expect the string "ogin:". If it fails to receive a login prompt within the time interval allotted, it is to send a break sequence to the remote and then expect the string "ogin:". If the first "ogin:" is received then the break sequence is not generated.

Once it received the login prompt the *chat* program will send the string ppp and then expect the prompt "ssword:". When it receives the prompt for the password, it will send the password hello2u2.

A carriage return is normally sent following the reply string. It is not expected in the "expect" string unless it is specifically requested by using the \r character sequence.

The expect sequence should contain only what is needed to identify the string. Since it is normally stored on a disk file, it should not contain variable information. It is generally not acceptable to look for time strings, network identification strings, or other variable pieces of data as an expect string.

To help correct for characters which may be corrupted during the initial sequence, look for the string "ogin:" rather than "login:". It is possible that the leading "l" character may be received in error and you may never find the string even though it was sent by the system. For this reason, scripts look for "ogin:" rather than "login:" and "ssword:" rather than "password:".

A very simple script might look like this:

ogin: ppp ssword: hello2u2

In other words, expect ....ogin:, send ppp, expect ...ssword:, send hello2u2.

In actual practice, simple scripts are rare. At the vary least, you should include sub-expect sequences should the original string not be received. For example, consider the following script:

ogin:−−ogin: ppp ssword: hello2u2

This would be a better script than the simple one used earlier. This would look for the same login: prompt, however, if one was not received, a single return sequence is sent and then it will look for login: again. Should line noise obscure the first login prompt then sending the empty line will usually generate a login prompt again.

COMMENTS

Comments can be embedded in the chat script. A comment is a line which starts with the **#** (hash) character in column 1. Such comment lines are just ignored by the chat program. If a '#' character is to be expected as the first character of the expect sequence, you should quote the expect string. If you want to wait for a prompt that starts with a # (hash) character, you would have to write something like this:

```
# Now wait for the prompt and send logout string
'# ' logout
```

SENDING DATA FROM A FILE

If the string to send starts with an at sign (@), the rest of the string is taken to be the name of a file to read to get the string to send. If the last character of the data read is a newline, it is removed. The file can be a named pipe (or fifo) instead of a regular file. This provides a way for **chat** to communicate with another program, for example, a program to prompt the user and receive a password typed in.

ABORT STRINGS

Many modems will report the status of the call as a string. These strings may be **CONNECTED** or **NO CARRIER** or **BUSY**. It is often desirable to terminate the script should the modem fail to connect to the remote. The difficulty is that a script would not know exactly which modem string it may receive. On one attempt, it may receive **BUSY** while the next time it may receive **NO CARRIER**.

These "abort" strings may be specified in the script using the *ABORT* sequence. It is written in the script as in the following example:

ABORT BUSY ABORT 'NO CARRIER' '' ATZ OK ATDT5551212 CONNECT

This sequence will expect nothing; and then send the string ATZ. The expected response to this is the string

OK. When it receives *OK*, the string ATDT5551212 to dial the telephone. The expected string is *CONNECT*. If the string *CONNECT* is received the remainder of the script is executed. However, should the modem find a busy telephone, it will send the string *BUSY*. This will cause the string to match the abort character sequence. The script will then fail because it found a match to the abort string. If it received the string *NO CARRIER*, it will abort for the same reason. Either string may be received. Either string will terminate the *chat* script.

CLR_ABORT STRINGS

This sequence allows for clearing previously set **ABORT** strings. **ABORT** strings are kept in an array of a pre-determined size (at compilation time); **CLR_ABORT** will reclaim the space for cleared entries so that new strings can use that space.

SAY STRINGS

The **SAY** directive allows the script to send strings to the user at the terminal via standard error. If **chat** is being run by pppd, and pppd is running as a daemon (detached from its controlling terminal), standard error will normally be redirected to the file /etc/ppp/connect−errors.

SAY strings must be enclosed in single or double quotes. If carriage return and line feed are needed in the string to be output, you must explicitly add them to your string.

The SAY strings could be used to give progress messages in sections of the script where you want to have 'ECHO OFF' but still let the user know what is happening. An example is:

```
ABORT BUSY
ECHO OFF
SAY "Dialling your ISP...\n"
'' ATDT5551212
TIMEOUT 120
SAY "Waiting up to 2 minutes for connection ... "
CONNECT ''
SAY "Connected, now logging in ...\n"
ogin: account
ssword: pass
$ \c
SAY "Logged in OK ...\n" etc ...
```

This sequence will only present the SAY strings to the user and all the details of the script will remain hidden. For example, if the above script works, the user will see:

```
Dialling your ISP...
Waiting up to 2 minutes for connection ... Connected, now logging in ...
Logged in OK ...
```

REPORT STRINGS

A **report** string is similar to the ABORT string. The difference is that the strings, and all characters to the next control character such as a carriage return, are written to the report file.

The report strings may be used to isolate the transmission rate of the modem's connect string and return the value to the chat user. The analysis of the report string logic occurs in conjunction with the other string processing such as looking for the expect string. The use of the same string for a report and abort sequence is probably not very useful, however, it is possible.

The report strings to no change the completion code of the program.

These "report" strings may be specified in the script using the *REPORT* sequence. It is written in the script as in the following example:

```
REPORT CONNECT ABORT BUSY '' ATDT5551212 CONNECT '' ogin: account
```

This sequence will expect nothing; and then send the string ATDT5551212 to dial the telephone. The expected string is *CONNECT*. If the string *CONNECT* is received the remainder of the script is executed. In addition the program will write to the expect−file the string "CONNECT" plus any characters which follow

it such as the connection rate.

CLR_REPORT STRINGS

This sequence allows for clearing previously set **REPORT** strings. **REPORT** strings are kept in an array of a pre-determined size (at compilation time); **CLR_REPORT** will reclaim the space for cleared entries so that new strings can use that space.

ECHO

The echo options controls whether the output from the modem is echoed to *stderr*. This option may be set with the *−e* option, but it can also be controlled by the *ECHO* keyword. The "expect−send" pair *ECHO ON* enables echoing, and *ECHO OFF* disables it. With this keyword you can select which parts of the conversation should be visible. For instance, with the following script:

```
ABORT   'BUSY'
ABORT   'NO CARRIER'
''      ATZ
OK\r\n  ATD1234567
\r\n    \c
ECHO    ON
CONNECT \c
ogin:   account
```

all output resulting from modem configuration and dialing is not visible, but starting with the *CONNECT* (or *BUSY*) message, everything will be echoed.

HANGUP

The HANGUP options control whether a modem hangup should be considered as an error or not. This option is useful in scripts for dialling systems which will hang up and call your system back. The HANGUP options can be **ON** or **OFF**.

When HANGUP is set OFF and the modem hangs up (e.g., after the first stage of logging in to a callback system), **chat** will continue running the script (e.g., waiting for the incoming call and second stage login prompt). As soon as the incoming call is connected, you should use the **HANGUP ON** directive to reinstall normal hang up signal behavior. Here is an (simple) example script:

```
ABORT   'BUSY'
''      ATZ
OK\r\n  ATD1234567
\r\n    \c
CONNECT \c
'Callback login:' call_back_ID
HANGUP OFF
ABORT "Bad Login"
'Callback Password:' Call_back_password
TIMEOUT 120
CONNECT \c
HANGUP ON
ABORT "NO CARRIER"
ogin:−−BREAK−−ogin: real_account
etc ...
```

TIMEOUT

The initial timeout value is 45 seconds. This may be changed using the **−t** parameter. You can also specify "TIMEOUT 0".

To change the timeout value for the next expect string, the following example may be used:

```
ATZ OK ATDT5551212 CONNECT TIMEOUT 10 ogin:−−ogin: TIMEOUT 5 assword: hello2u2
```

This will change the timeout to 10 seconds when it expects the login: prompt. The timeout is then changed to 5 seconds when it looks for the password prompt.

The timeout, once changed, remains in effect until it is changed again.

SENDING EOT

The special reply string of *EOT* indicates that the chat program should send an EOT character to the remote. This is normally the End-of-file character sequence. A return character is not sent following the EOT. The EOT sequence may be embedded into the send string using the sequence *^D*.

GENERATING BREAK

The special reply string of *BREAK* will cause a break condition to be sent. The break is a special signal on the transmitter. The normal processing on the receiver is to change the transmission rate. It may be used to cycle through the available transmission rates on the remote until you are able to receive a valid login prompt. The break sequence may be embedded into the send string using the *\K* sequence.

ESCAPE SEQUENCES

The expect and reply strings may contain escape sequences. All of the sequences are legal in the reply string. Many are legal in the expect. Those which are not valid in the expect sequence are so indicated.

'' Expects or sends a null string. If you send a null string then it will still send the return character. This sequence may either be a pair of apostrophe or quote characters.

\b represents a backspace character.

\c Suppresses the newline at the end of the reply string. This is the only method to send a string without a trailing return character. It must be at the end of the send string. For example, the sequence hello\c will simply send the characters h, e, l, l, o. *(not valid in expect.)*

\d Delay for one second. The program uses sleep(1) which will delay to a maximum of one second. *(not valid in expect.)*

\K Insert a BREAK *(not valid in expect.)*

\n Send a newline or linefeed character.

\N Send a null character. The same sequence may be represented by \0. *(not valid in expect.)*

\p Pause for a fraction of a second. The delay is 1/10th of a second. *(not valid in expect.)*

\q Suppress writing the string to the SYSLOG file. The string ?????? is written to the log in its place. *(not valid in expect.)*

\r Send or expect a carriage return.

\s Represents a space character in the string. This may be used when it is not desirable to quote the strings which contains spaces. The sequence 'HI TIM' and HI\sTIM are the same.

\t Send or expect a tab character.

\T Send the phone number string as specified with the *−T* option *(not valid in expect.)*

\U Send the phone number 2 string as specified with the *−U* option *(not valid in expect.)*

**** Send or expect a backslash character.

\ddd Collapse the octal digits (ddd) into a single ASCII character and send that character. *(some characters are not valid in expect.)*

^C Substitute the sequence with the control character represented by C. For example, the character DC1 (17) is shown as ^Q. *(some characters are not valid in expect.)*

ENVIRONMENT VARIABLES

Environment variables are available within chat scripts, if the *−E* option was specified in the command line. The metacharacter *$* is used to introduce the name of the environment variable to substitute. If the substitution fails, because the requested environment variable is not set, *nothing* is replaced for the variable.

TERMINATION CODES

The *chat* program will terminate with the following completion codes.

0 The normal termination of the program. This indicates that the script was executed without error to the normal conclusion.

1 One or more of the parameters are invalid or an expect string was too large for the internal buffers. This indicates that the program as not properly executed.

2 An error occurred during the execution of the program. This may be due to a read or write operation failing for some reason or chat receiving a signal such as SIGINT.

3 A timeout event occurred when there was an *expect* string without having a "−subsend" string. This may mean that you did not program the script correctly for the condition or that some unexpected event has occurred and the expected string could not be found.

4 The first string marked as an *ABORT* condition occurred.

5 The second string marked as an *ABORT* condition occurred.

6 The third string marked as an *ABORT* condition occurred.

7 The fourth string marked as an *ABORT* condition occurred.

... The other termination codes are also strings marked as an *ABORT* condition.

Using the termination code, it is possible to determine which event terminated the script. It is possible to decide if the string "BUSY" was received from the modem as opposed to "NO DIAL TONE". While the first event may be retried, the second will probably have little chance of succeeding during a retry.

SEE ALSO

Additional information about *chat* scripts may be found with UUCP documentation. The *chat* script was taken from the ideas proposed by the scripts used by the *uucico* program.

uucico(1), uucp(1)

COPYRIGHT