NAME

autoexpect − generate an Expect script from watching a session

SYNOPSIS

autoexpect [*args*] [*program args...*]

INTRODUCTION

autoexpect watches you interacting with another program and creates an Expect script that reproduces your interactions. For straightline scripts, autoexpect saves substantial time over writing scripts by hand. Even if you are an Expect expert, you will find it convenient to use autoexpect to automate the more mindless parts of interactions. It is much easier to cut/paste hunks of autoexpect scripts together than to write them from scratch. And if you are a beginner, you may be able to get away with learning nothing more about Expect than how to call autoexpect.

The simplest way to use autoexpect is to call it from the command line with no arguments. For example:

```
% autoexpect
```

By default, autoexpect spawns a shell for you. Given a program name and arguments, autoexpect spawns that program. For example:

```
% autoexpect ftp ftp.cme.nist.gov
```

Once your spawned program is running, interact normally. When you have exited the shell (or program that you specified), autoexpect will create a new script for you. By default, autoexpect writes the new script to "script.exp". You can override this with the −f flag followed by a new script name.

The following example runs "ftp ftp.cme.nist.gov" and stores the resulting Expect script in the file "nist".

```
% autoexpect −f nist ftp ftp.cme.nist.gov
```

It is important to understand that autoexpect does not guarantee a working script because it necessarily has to guess about certain things − and occasionally it guesses wrong. However, it is usually very easy to identify and fix these problems. The typical problems are:

- Timing. A surprisingly large number of programs (rn, ksh, zsh, telnet, etc.) and devices (e.g., modems) ignore keystrokes that arrive "too quickly" after prompts. If you find your new script hanging up at one spot, try adding a short sleep just before the previous send.

  You can force this behavior throughout by overriding the variable "force_conservative" near the beginning of the generated script. This "conservative" mode makes autoexpect automatically pause briefly (one tenth of a second) before sending each character. This pacifies every program I know of.

  This conservative mode is useful if you just want to quickly reassure yourself that the problem is a timing one (or if you really don't care about how fast the script runs). This same mode can be forced before script generation by using the −c flag.

  Fortunately, these timing spots are rare. For example, telnet ignores characters only after entering its escape sequence. Modems only ignore characters immediately after connecting to them for the first time. A few programs exhibit this behavior all the time but typically have a switch to disable it. For example, rn's −T flag disables this behavior.

  The following example starts autoexpect in conservative mode.

  ```
  autoexpect −c
  ```

The −C flag defines a key to toggle conservative mode. The following example starts autoexpect (in non-conservative mode) with ^L as the toggle. (Note that the ^L is entered literally - i.e., enter a real control-L).

autoexpect −C ^L

The following example starts autoexpect in conservative mode with ^L as the toggle.

autoexpect −c −C ^L

- Echoing. Many program echo characters. For example, if you type "more" to a shell, what autoexpect actually sees is:

  you typed 'm',
  computer typed 'm',
  you typed 'o',
  computer typed 'o',
  you typed 'r',
  computer typed 'r',
  ...

  Without specific knowledge of the program, it is impossible to know if you are waiting to see each character echoed before typing the next. If autoexpect sees characters being echoed, it assumes that it can send them all as a group rather than interleaving them the way they originally appeared. This makes the script more pleasant to read. However, it could conceivably be incorrect if you really had to wait to see each character echoed.

- Change. Autoexpect records every character from the interaction in the script. This is desirable because it gives you the ability to make judgements about what is important and what can be replaced with a pattern match.

  On the other hand, if you use commands whose output differs from run to run, the generated scripts are not going to be correct. For example, the "date" command always produces different output. So using the date command while running autoexpect is a sure way to produce a script that will require editing in order for it to work.

  The −p flag puts autoexpect into "prompt mode". In this mode, autoexpect will only look for the the last line of program output − which is usually the prompt. This handles the date problem (see above) and most others.

  The following example starts autoexpect in prompt mode.

  autoexpect −p

  The −P flag defines a key to toggle prompt mode. The following example starts autoexpect (in non-prompt mode) with ^P as the toggle. Note that the ^P is entered literally - i.e., enter a real control-P.

  autoexpect −P ^P

  The following example starts autoexpect in prompt mode with ^P as the toggle.

  autoexpect −p −P ^P

OTHER FLAGS

The **−quiet** flag disables informational messages produced by autoexpect.

The **−Q** flag names a quote character which can be used to enter characters that autoexpect would otherwise consume because they are used as toggles.

The following example shows a number of flags with quote used to provide a way of entering the toggles literally.

```
autoexpect −P ^P −C ^L −Q ^Q
```

STYLE

I don’t know if there is a "style" for Expect programs but autoexpect should definitely not be held up as any model of style. For example, autoexpect uses features of Expect that are intended specifically for computer-generated scripting. So don’t try to faithfully write scripts that appear as if they were generated by autoexpect. This is not useful.

On the other hand, autoexpect scripts do show some worthwhile things. For example, you can see how any string must be quoted in order to use it in a Tcl script simply by running the strings through autoexpect.

SEE ALSO

"Exploring Expect: A Tcl-Based Toolkit for Automating Interactive Programs" by Don Libes, O’Reilly and Associates, January 1995.

AUTHOR

Don Libes, National Institute of Standards and Technology