Syntax

Whether to trace execution of programs

set trace { on | off }

Show # levels in tracing nested programs

set tracedepth #

Whether to show the lines after macro expansion

set traceexpand { on | off } [, permanently]

Whether to display horizontal separator lines

set tracesep { on | off } [, permanently]

Whether to indent lines according to nesting level

set traceindent { on | off } [, permanently]

Whether to display nesting level

set tracenumber { on | off } [, permanently]

Highlight pattern in trace output

set tracehilite "pattern" [, word]

Description

set trace on traces the execution of programs for debugging. set trace off turns off tracing after it has been set on.

set tracedepth specifies how many levels to descend in tracing nested programs. The default is 32000, which is equivalent to \( \infty \).

set traceexpand indicates whether the lines before and after macro expansion are to be shown. The default is on.

set tracesep indicates whether to display a horizontal separator line that displays the name of the subroutine whenever a subroutine is entered or exited. The default is on.
set traceindent indicates whether displayed lines of code should be indented according to the nesting level. The default is on.

set tracenumber indicates whether the nesting level should be displayed at the beginning of the line. Lines in the main program are preceded with 01; lines in subroutines called by the main program, with 02; etc. The default is off.

set tracehilite causes the specified pattern to be highlighted in the trace output.

**Options**

permanently specifies that, in addition to making the change right now, the traceexpand, traceseq, traceindent, and tracenumber settings be remembered and become the default settings when you invoke Stata.

word highlights only tokens that are delimited by nonalphanumeric characters. These would include tokens at the beginning or end of each line that are delimited by nonalphanumeric characters.

**Remarks and examples**

The set trace commands are extremely useful for debugging your programs.

Example 1

Stata does not normally display the lines of your program as it executes them. With set trace on, however, it does:

```stata
. program list simple
simple:
    1. args msg
    2. if `"msg"'=="hello" {
       3. display "you said hello"
       4. }
    5. else display "you did not say hello"
    6. display "good-bye"
. set trace on
. simple
begin simple —
    - args msg
    - if `"msg"'=="hello" {
       = if `"hello" '{
       display "you said hello"
       }
    - else display "you did not say hello"
you did not say hello
    - display "good-bye"
good-bye
end simple —
. set trace off
```

Lines that are executed are preceded by a dash. The line is shown before macro expansion, just as it was coded. If the line has any macros, it is shown again, this time preceded by an equal sign and with the macro expanded, showing the line exactly as Stata sees it.

In our simple example, Stata substituted nothing for `msg`, as we can see by looking at the macro-expanded line. Because nothing is not equal to “hello”, Stata skipped the display of “you said hello”, so a dash did not precede this line.
Stata then executed lines 5 and 6. (They are not reshown preceded by an equal sign because they contained no macros.)

To suppress the printing of the macro-expanded lines, type `set traceexpand off`.

To suppress the printing of the trace separator lines,

```
begin simple
end simple
```

type `set tracesep off`.

The output from our program is interspersed with the lines that caused the output. This can be greatly useful when our program has an error. For instance, we have written a more useful program called `myprog`. Here is what happens when we run it:

```
. myprog mpg, prefix("new")
invalid syntax
r(198);
```

We did not expect this, and, look as we will at our program code, we cannot spot the error. Our program contains many lines of code, however, so we have no idea even where to look. By setting `trace` on, we can quickly find the error:

```
. set trace on
. myprog mpg, prefix("new")
--- begin myprog ---
- version 13
- syntax varname, [Prefix(string)]
- local newname "'prefix' varname'
  = local newname "new"
invalid syntax
--- end myprog ---
```

The error was close to the top—we omitted the closing quote in the definition of the local `newname` macro.

**Technical note**

If you are looking for a command similar to `set trace` for use in Mata, see `mata set matalnum` in [M-3] *mata set*.

**Example 2**

`set tracedepth`, `set tracesep`, `set traceindent`, and `set tracenumber` are useful when debugging nested programs. Imagine that we have a program called `myprog1`, which calls `myprog2`, which then calls a modified version of our `simple` program from example 1.

With the default settings, we get:

```
. program list _all
simple2:
 1.      args msg
 2.      if ""msg""==""hello"" {
 3.         display "you said hello"
 4.       }
```
myprog2:
  1. args msg
  2. simple2 "'msg'"
  3. display "good"

myprog1:
  1. args msg
  2. myprog2 "'msg'"
  3. display "bye"

.set trace on
.myprog1 hello

begin myprog1
  - args msg
  - myprog2 "'msg'
    = myprog2 "'hello'
      - args msg
      - simple2 "'msg'
        = simple2 "'hello'
          - args msg
          - if "'msg'"=="hello" {
            = if "'hello'"=="hello" {
              - display "you said hello"
            }
            - else {
              display "you did not say hello"
            }
          }
          - display "good"
        end simple2
      end myprog2
    end myprog2
  end myprog1

.set trace off
.To see the nesting level for each line, you could use set tracenumber on.
.set trace on
.set tracenumber on
.myprog1 hello

begin myprog1
  01 - args msg
  01 - myprog2 "'msg'
    = myprog2 "'hello'
      02 - args msg
      02 - simple2 "'msg'
        = simple2 "'hello'
          03 - args msg
          03 - if "'msg'"=="hello" {
            = if "'hello'"=="hello" {
              - display "you said hello"
            }
            03 - else {
              you said hello
            }
          }
          03 - else {
            display "you did not say hello"
          }
        end simple2
      end myprog2
    end myprog2
  end myprog1

If you are interested only in seeing a trace of the first two nesting levels, you could set `tracedepth` to 2. If you did not want each nested level to be indented in the trace output, you could set `traceindent` off.
display "you did not say hello"
}
end simple2

- display "good"
good
end myprog2

- display "bye"
bye
end myprog1

. set traceindent on
. set trace off

Also see

[P] program — Define and manipulate programs
[R] query — Display system parameters
[R] set — Overview of system parameters
[U] 18 Programming Stata