Title

stata.com

version — Version control

Description Syntax Options Remarks and examples Also see

Description

`version` with no arguments shows the current internal version number to which the command interpreter is set. It can be used interactively, in do-files, or in ado-files.

`version #` sets the command interpreter and other features such as random-number generators (RNGs) to version number #. `version #` is used to allow old programs to run correctly under more recent versions of Stata and to ensure that new programs run correctly under future versions of Stata.

`version #:` executes *command* under version # and then resets the version to what it was before the `version #: ...` command was given.

For information about external version control, see [R] **which**.

Syntax

Show version number to which command interpreter is set

`version`

Interactively or in do-files (but not in ado-files or programs defined by `program`)

Set command interpreter to version # and set other features such as RNGs to version #

`version` *#*

`version` *#*`:` *command*

In ado-files or programs (but not interactively or in do-files)

Set command interpreter to version #, but do not set other features such as RNGs to version #

`version` *#* [`, born(`*ddMONyyyy*`)`]

`version` *#* [`, born(`*ddMONyyyy*`)`]`:` *command*

Everywhere (interactively, in do-files, in ado-files, and in programs)

Set only the other features such as RNGs to version #

`version` *#*`, user`

`version` *#*`, user:` *command*

Options

`born(`*ddMONyyyy*`)` is rarely specified and indicates that the Stata executable must be dated *ddMONyyyy* (for example, 13Jul2009) or later. StataCorp and users sometimes write programs in ado-files that require the Stata executable to be of a certain date. The `born()` option allows us or the author of an ado-file to ensure that ado-code that requires a certain updated executable is not run with an older executable.

Generally all that matters is the version number, so you would not use the `born()` option. You use `born()` in the rare case that you are exploiting a feature added to the executable after the initial release of that version of Stata. See `help whatsnew` to browse the features added to the current version of Stata since its original release.

`user` causes `version` to backdate other features of Stata. For instance, the results of Stata's RNGs change—improve—with each version of Stata. The RNGs are said to be under user-version control rather than version control.

If you type `version` # interactively or in your do-files, Stata not only understands old syntax, it backdates (removes) improvements made after #, such as those to the RNGs. You do not have to specify the `user` option. The modern version of Stata will still produce the same results as it produced in the past.

Programmers: When you type `version` # in your programs and ado-files, Stata does not backdate the other improvements. If, for some reason, you want to force the other improvements to be backdated, specify the `user` option. Option `user` is seldom used except by those (say, developers at StataCorp) needing to test that Stata works properly.

Remarks and examples

stata.com

`version` ensures that programs written under an older release of Stata will continue to work under newer releases of Stata. If you do not write programs and if you use only the programs distributed by StataCorp, you can ignore `version`. If you do write programs, see **[U] 18.11.1 Version** for guidelines to follow to ensure compatibility of your programs with future releases of Stata.

❑ Technical note

When Stata is invoked, it sets its internal version number to the current version of Stata, which is 14.2 as of this writing. Typing `version` without arguments shows the current value of the internal version number:

```
. version
version 14.2
```

One way to make old programs work is to set the internal version number interactively to that of a previous release:

```
. version 9.0
. version
version 9.0
```

Now Stata's default interpretation of a program is the same as it was for Stata 9.0.

You cannot set the version to a number higher than the current version. For instance, because we are using Stata 14.2, we cannot set the version number to 14.7.

```
. version 14.7
this is version 14.2 of Stata; it cannot run version 14.7 programs
  (output omitted)
r(9);
```

❑

❑ Technical note

We strongly recommend that all ado-files and do-files begin with a `version` command. For programs (ado-files), the `version` command should appear immediately following the `program` command:

```
program myprog
        version 14.2
        (etc.)
end
```

❑

❑ Technical note

As of Stata 14, Stata's RNGs were improved, renamed, and restructured; see [R] **set seed**. The default RNG in Stata 14 is the 64-bit Mersenne Twister (`mt64`). Before Stata 14, the RNG was the 32-bit KISS (`kiss32`). If user version is 14, RNG results are based on `mt64`. If user version is less than 14, RNG results are based on `kiss32`. This also affects the results of commands that use the RNGs, such as `bootstrap`, `bsample`, and the `mi` suite of commands.

❑

❑ Technical note

Version control within a RNG is specified at the time the `set seed` command is given, not at the time the random-number generation function such as `rnormal()` is used. For instance, typing

```
. (assume version is set to be 11.2)
. set seed 123456789
. any_command ...
```

causes *any_command* to use the Stata 11.2 version of `rnormal()` even if *any_command* is an ado-file containing an explicit `version` statement setting the version to less than 11.2. This occurs because the version of `rnormal()` that is used was determined at the time the seed was set, and the seed was set under version 11.2 or later.

This works in both directions. Consider

```
. version 11.1: set seed 123456789
. any_command ...
```

In this case, *any_command* uses the older version of `rnormal()` because the seed was set under version 11.1, before `rnormal()` was updated. *any_command* uses the older version of `rnormal()` even if *any_command* itself includes an explicit `version` statement setting the version to 11.2 or later.

Thus both older and newer ado-files can use the newer or older `rnormal()`, and they can do so without modification. The only case in which you need to modify a do-file or ado-file is when it is older, it contains `set seed`, and you now want it to use the new `rnormal()`. In that case, find the `set seed` command in the do-file or ado-file,

```
version 10              // for example
...
set seed 123456789
...
```

and change it to read

```
version 10              // for example
...
version 11.2: set seed 123456789
...
```

You need to change only the one line.

Everything written above about prefixing `set seed` with a `version` is irrelevant if you are restoring the seed to a state previously obtained from `c(seed)`:

```
set rngstate X075bcd151f123bb5159a55e50022865700023e53
```

The string state `X075bcd151f123bb5159a55e50022865700023e53` includes the version number at the time the seed was set. Prefixing the above with `version`, whether older or newer, will do no harm but is unnecessary.

❑

For an up-to-date summary of version changes, see `help version`.

Also see

[P] **display** — Display strings and values of scalar expressions

[R] **which** — Display location and version for an ado-file

[U] **18.11.1 Version**