The *Stata Journal* publishes reviewed papers together with shorter notes or comments, regular columns, book reviews, and other material of interest to Stata users. Examples of the types of papers include 1) expository papers that link the use of Stata commands or programs to associated principles, such as those that will serve as tutorials for users first encountering a new field of statistics or a major new technique; 2) papers that go “beyond the Stata manual” in explaining key features or uses of Stata that are of interest to intermediate or advanced users of Stata; 3) papers that discuss new commands or Stata programs of interest either to a wide spectrum of users (e.g., in data management or graphics) or to some large segment of Stata users (e.g., in survey statistics, survival analysis, panel analysis, or limited dependent variable modeling); 4) papers analyzing the statistical properties of new or existing estimators and tests in Stata; 5) papers that could be of interest or usefulness to researchers, especially in fields that are of practical importance but are not often included in texts or other journals, such as the use of Stata in managing datasets, especially large datasets, with advice from hard-won experience; and 6) papers of interest to those who teach, including Stata with topics such as extended examples of techniques and interpretation of results, simulations of statistical concepts, and overviews of subject areas.

For more information on the *Stata Journal*, including information for authors, see the webpage

[http://www.stata-journal.com](http://www.stata-journal.com)

The *Stata Journal* is indexed and abstracted in the following:

- CompuMath Citation Index®
- Current Contents/Social and Behavioral Sciences®
- RePEc: Research Papers in Economics
- Science Citation Index Expanded (also known as SciSearch®)
- Scopus
- Social Sciences Citation Index®

**Copyright Statement:** The *Stata Journal* and the contents of the supporting files (programs, datasets, and help files) are copyright © by StataCorp LP. The contents of the supporting files (programs, datasets, and help files) may be copied or reproduced by any means whatsoever, in whole or in part, as long as any copy or reproduction includes attribution to both (1) the author and (2) the *Stata Journal*.

The articles appearing in the *Stata Journal* may be copied or reproduced as printed copies, in whole or in part, as long as any copy or reproduction includes attribution to both (1) the author and (2) the *Stata Journal*.

Written permission must be obtained from StataCorp if you wish to make electronic copies of the insertions. This precludes placing electronic copies of the *Stata Journal*, in whole or in part, on publicly accessible websites, file servers, or other locations where the copy may be accessed by anyone other than the subscriber.

Users of any of the software, ideas, data, or other materials published in the *Stata Journal* or the supporting files understand that such use is made without warranty of any kind, by either the *Stata Journal*, the author, or StataCorp. In particular, there is no warranty of fitness of purpose or merchantability, nor for special, incidental, or consequential damages such as loss of profits. The purpose of the *Stata Journal* is to promote free communication among Stata users.

The *Stata Journal* (ISSN 1536-867X) is a publication of Stata Press. Stata, Mata, NetCourse, and Stata Press are registered trademarks of StataCorp LP.
Stata tip 91: Putting unabbreviated varlists into local macros

Nicholas J. Cox
Department of Geography
Durham University
Durham, UK
n.j.cox@durham.ac.uk

Within interactive sessions, do-files, or programs, Stata users often want to work with varlists, lists of variable names. For convenience, such lists may be stored in local macros. Local macros can be directly defined for later use, as in

```
.local myx "weight length displacement"
.regress mpg `myx`
```

However, users frequently want to put longer lists of names into local macros, spelled out one by one so that some later command can loop through the list defined by the macro. Such varlists might be indirectly defined in abbreviations using the wildcard characters * or ?. These characters can be used alone or can be combined to express ranges. For example, specifying * catches all variables, *TX* might define all variables for Texas, and *200? catches the years 2000–2009 used as suffixes.

In such cases, direct definition may not appeal for all the obvious reasons: it is tedious, time-consuming, and error-prone. It is also natural to wonder if there is a better method. You may already know that foreach (see [P] foreach) will take such wildcarded varlists as arguments, which solves many problems.

Many users know that pushing an abbreviated varlist through describe or ds is one way to produce an unabbreviated varlist. Thus

```
.describe, varlist
```

is useful principally for its side effect of leaving all the variable names in r(varlist). ds is typically used in a similar way, as is the user-written findname command (Cox 2010).

However, if the purpose is just to produce a local macro, the method of using describe or ds has some small but definite disadvantages. First, the output of each may not be desired, although it is easily suppressed with a quietly prefix. Second, the modus operandi of both describe and ds is to leave saved results as r-class results. Every now and again, users will be frustrated by this when they unwittingly overwrite r-class results that they wished to use again. Third, there is some inefficiency in using either command for this purpose, although you would usually have to work hard to measure it.
The solution here is to use the `unab` command; see [P] unab. `unab` has just one restricted role in life, but that role is the solution here. `unab` is billed as a programming command, but nothing stops it from being used interactively as a simple tool in data management. The simple examples

```
. unab myvars : *
. unab TX : *TX*
. unab twenty : *200?
```

show how a local macro, named at birth (here as `myvars`, `TX`, and `twenty`), is defined as the unabbreviated equivalent of each argument that follows a colon. Note that using wildcard characters, although common, is certainly not compulsory.

The word “unabbreviate” is undoubtedly ugly. The help and manual entry do also use the much simpler and more attractive word “expand”, but the word “expand” was clearly not available as a command name, given its use for another purpose.

This tip skates over all the fine details of `unab`, and only now does it mention the sibling commands `tsunab` and `fvunab`, for use when you are using time-series operators and factor variables. For more information, see [P] unab.

**Reference**