The Stata Journal (2010)
10, Number 3, pp. 503–504

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 *varlist*s, 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 *varlist*s 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

Cox, N. J. 2010. Speaking Stata: Finding variables. *Stata Journal* 10: 281–296.