11 Language syntax

Contents

11.1 Overview

With few exceptions, the basic Stata language syntax is

[`by` *varlist*`:`] *command* [*varlist*] [`=`*exp*] [`if` *exp*] [`in` *range*] [*weight*] [`,` *options*]

where square brackets distinguish optional qualifiers and options from required ones. In this diagram, *varlist* denotes a list of variable names, *command* denotes a Stata command, *exp* denotes an algebraic expression, *range* denotes an observation range, *weight* denotes a weighting expression, and *options* denotes a list of options.

11.1.1 varlist

Most commands that take a subsequent *varlist* do not require that you explicitly type one. If no *varlist* appears, these commands assume a *varlist* of `_all`, the Stata shorthand for indicating all the variables in the dataset. In commands that alter or destroy data, Stata requires that the *varlist* be specified explicitly. See [U] **11.4 varname and varlists** for a complete description.

Some commands take a *varname*, rather than a *varlist*. A *varname* refers to exactly one variable. The `tabulate` command requires a *varname*; see [R] **tabulate oneway**.

▷ Example 1

The `summarize` command lists the mean, standard deviation, and range of the specified variables. In [R] **summarize**, we see that the syntax diagram for summarize is

`summarize` [*varlist*] [*if*] [*in*] [*weight*] [`,` *options*]

Farther down on the manual page is a table summarizing `options`, but let's focus on the syntax diagram itself first. Because everything except the word `summarize` is enclosed in square brackets, the simplest form of the command is "`summarize`". Typing `summarize` without arguments is equivalent to typing `summarize _all`; all the variables in the dataset are summarized. Underlining denotes the shortest allowed abbreviation, so we could have typed just `su`; see [U] **11.2 Abbreviation rules**.

The table that defines *options* looks like this:

options	Description
Main	
`detail`	display additional statistics
`meanonly`	suppress the display; calculate only the mean; programmer's option
`format`	use variable's display format
`separator(#)`	draw separator line after every # variables; default is `separator(5)`

Thus we learn we could also type, for instance, `summarize, detail` or `summarize, detail format`.

As another example, the `drop` command eliminates variables or observations from a dataset. When dropping variables, its syntax is

`drop` *varlist*

`drop` has no option table because it has no options.

In fact, nothing is optional. Typing `drop` by itself would result in the error message "varlist or in range required". To drop all the variables in the dataset, we must type `drop _all`.

Even before looking at the syntax diagram, we could have predicted that *varlist* would be required—`drop` is destructive, so Stata requires us to spell out our intent. The syntax diagram informs us that *varlist* is required because *varlist* is not enclosed in square brackets. Because `drop` is not underlined, it cannot be abbreviated.

◁

11.1.2 by varlist:

The by *varlist*: prefix causes Stata to repeat a command for each subset of the data for which the values of the variables in *varlist* are equal. When prefixed with by *varlist*:, the result of the command will be the same as if you had formed separate datasets for each group of observations, saved them, and then gave the command on each dataset separately. The data must already be sorted by *varlist*, although by has a sort option; see **[U] 11.5 by varlist: construct** for more information.

▷ Example 2

Typing summarize marriage_rate divorce_rate produces a table of the mean, standard deviation, and range of marriage_rate and divorce_rate, using all the observations in the data:

```
. use https://www.stata-press.com/data/r18/census12
(1980 Census data by state)
. summarize marriage_rate divorce_rate
    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
marriage_r~e |         50    .0133221    .0188122   .0074654   .1428282
divorce_rate |         50    .0056641    .0022473   .0029436   .0172918
```

Typing by region: summarize marriage_rate divorce_rate produces one table for each region of the country:

```
. sort region
. by region: summarize marriage_rate divorce_rate
-------------------------------------------------------------------------------
-> region = N Cntrl
    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
marriage_r~e |         12    .0099121    .0011326   .0087363   .0127394
divorce_rate |         12    .0046974    .0011315   .0032817   .0072868
-------------------------------------------------------------------------------
-> region = NE
    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
marriage_r~e |          9    .0087811     .001191   .0075757   .0107055
divorce_rate |          9     .004207    .0010264   .0029436   .0057071
-------------------------------------------------------------------------------
-> region = South
    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
marriage_r~e |         16    .0114654    .0025721   .0074654   .0172704
divorce_rate |         16     .005633    .0013355   .0038917   .0080078
-------------------------------------------------------------------------------
-> region = West
    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
marriage_r~e |         13    .0218987    .0363775   .0087365   .1428282
divorce_rate |         13    .0076037    .0031486   .0046004   .0172918
```

The dataset must be sorted on the by variables:

```
. use https://www.stata-press.com/data/r18/census12
(1980 Census data by state)
. by region: summarize marriage_rate divorce_rate
not sorted
r(5);
. sort region
. by region: summarize marriage_rate divorce_rate
(output appears)
```

We could also have asked that by sort the data:

```
. by region, sort: summarize marriage_rate divorce_rate
(output appears)
```

by *varlist*: can be used with most Stata commands; we can tell which ones by looking at their syntax diagrams. For instance, we could obtain the correlations by region, between marriage_rate and divorce_rate, by typing by region: correlate marriage_rate divorce_rate.

◁

❑ Technical note

The *varlist* in by *varlist*: may contain up to 120,000 variables with Stata/MP, 32,767 variables with Stata/SE, or 2,048 variables with Stata/BE; these are the maximum allowed in the dataset. For instance, if we had data on automobiles and wished to obtain means according to market category (market) broken down by manufacturer (origin), we could type by market origin: summarize. That *varlist* contains two variables: market and origin. If the data were not already sorted on market and origin, we would first type sort market origin.

❑

❑ Technical note

The *varlist* in by *varlist*: may contain string variables, numeric variables, or both. In the example above, region is a string variable, in particular, a str7. The example would have worked, however, if region were a numeric variable with values 1, 2, 3, and 4, or even 12.2, 16.78, 32.417, and 152.13.

❑

11.1.3 if exp

The if *exp* qualifier restricts the scope of a command to those observations for which the value of the expression is *true* (which is equivalent to the expression being nonzero; see **[U] 13 Functions and expressions**).

▷ Example 3

Typing summarize marriage_rate divorce_rate if region=="West" produces a table for the western region of the country:

```
. summarize marriage_rate divorce_rate if region == "West"
    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
marriage_r~e |         13    .0218987    .0363775   .0087365   .1428282
divorce_rate |         13    .0076037    .0031486   .0046004   .0172918
```

The double equal sign in `region=="West"` is not an error. Stata uses a *double* equal sign to denote equality testing and one equal sign to denote assignment; see **[U] 13 Functions and expressions**.

A command may have at most one `if` qualifier. If you want the summary for the West restricted to observations with values of `marriage_rate` in excess of 0.015, do *not* type `summarize marriage_rate divorce_rate if region=="West" if marriage_rate>.015`. Instead type

```
. summarize marriage_rate divorce_rate if region == "West" & marriage_rate > .015
    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
marriage_r~e |          1    .1428282           .   .1428282   .1428282
divorce_rate |          1    .0172918           .   .0172918   .0172918
```

You may not use the word *and* in place of the symbol "`&`" to join conditions. To select observations that meet one condition *or* another, use the "`|`" symbol. For instance, `summarize marriage_rate divorce_rate if region=="West" | marriage_rate>.015` summarizes all observations for which `region` is West *or* `marriage_rate` is greater than 0.015.

◁

▷ Example 4

`if` may be combined with `by`. Typing `by region: summarize marriage_rate divorce_rate if marriage_rate>.015` produces a set of tables, one for each region, reflecting summary statistics on `marriage_rate` and `divorce_rate` among observations for which `marriage_rate` exceeds 0.015:

```
. by region: summarize marriage_rate divorce_rate if marriage_rate > .015
-------------------------------------------------------------------------------
-> region = N Cntrl
    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
marriage_r~e |          0
divorce_rate |          0
-------------------------------------------------------------------------------
-> region = NE
    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
marriage_r~e |          0
divorce_rate |          0
-------------------------------------------------------------------------------
-> region = South
    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
marriage_r~e |          2    .0163219    .0013414   .0153734   .0172704
divorce_rate |          2    .0061813    .0025831   .0043548   .0080078
-------------------------------------------------------------------------------
-> region = West
    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
marriage_r~e |          1    .1428282           .   .1428282   .1428282
divorce_rate |          1    .0172918           .   .0172918   .0172918
```

The results indicate that there are no states in the Northeast and North Central regions for which marriage_rate exceeds 0.015, whereas there are two such states in the South and one state in the West.

◁

11.1.4 in range

The in *range* qualifier restricts the scope of the command to a specific observation range. A range specification takes the form $\#_1[/\#_2]$, where $\#_1$ and $\#_2$ are positive or negative integers. Negative integers are understood to mean "from the end of the data", with -1 referring to the last observation. The implied first observation must be less than or equal to the implied last observation.

The first and last observations in the dataset may be denoted by f and l (lowercase letter), respectively. F is allowed as a synonym for f, and L is allowed as a synonym for l. A range specifies absolute observation numbers within a dataset. As a result, the in qualifier may not be used when the command is preceded by the by *varlist*: prefix; see **[U] 11.5 by varlist: construct**.

▷ Example 5

Typing summarize marriage_rate divorce_rate in 5/25 produces a table based on the values of marriage_rate and divorce_rate in observations 5–25:

```
. summarize marriage_rate divorce_rate in 5/25
    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
marriage_r~e |         21    .0093941    .0012851   .0075757     .01293
divorce_rate |         21    .0045305    .0011273   .0029436   .0072868
```

This is, admittedly, a rather odd thing to want to do. It would not be odd, however, if we substituted list for summarize. If we wanted to see the states with the 10 lowest values of marriage_rate, we could type sort marriage_rate followed by list marriage_rate in 1/10.

Typing summarize marriage_rate divorce_rate in f/l is equivalent to typing summarize marriage_rate divorce_rate—all observations are summarized.

◁

▷ Example 6

Typing summarize marriage_rate divorce_rate in 5/25 if region == "South" produces a table based on the values of the two variables in observations 5–25 for which the value of region is South:

```
. summarize marriage_rate divorce_rate in 5/25 if region == "South"
    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
marriage_r~e |          4    .0108187    .0016621   .0089399     .01293
divorce_rate |          4    .0051821    .0009356   .0043054   .0063596
```

The ordering of the in and if qualifiers is not significant. The command could also have been specified as summarize marriage_rate divorce_rate if region == "South" in 5/25.

◁

▷ Example 7

Negative `in` ranges can be useful with `sort`. For instance, we have data on automobiles and wish to list the five with the highest mileage ratings:

```
. use https://www.stata-press.com/data/r18/auto
(1978 automobile data)
. sort mpg
. list make mpg in -5/l
```

	make	mpg
70.	Toyota Corolla	31
71.	Plym. Champ	34
72.	Subaru	35
73.	Datsun 210	35
74.	VW Diesel	41

◁

11.1.5 =exp

= *exp* specifies the value to be assigned to a variable and is most often used with `generate` and `replace`. See **[U] 13 Functions and expressions** for details on expressions and [D] **generate** for details on the `generate` and `replace` commands.

▷ Example 8

Expression	Meaning
`generate newvar=oldvar+2`	creates a new variable named `newvar` equal to `oldvar`+2
`replace oldvar=oldvar+2`	changes the contents of the existing variable `oldvar`
`egen newvar=rank(oldvar)`	creates `newvar` containing the ranks of `oldvar` (see [D] **egen**)

◁

11.1.6 weight

weight indicates the weight to be attached to each observation. The syntax of *weight* is

[*weightword*=*exp*]

where you actually type the square brackets and where *weightword* is one of

weightword	Meaning
`weight`	default treatment of weights
`fweight` *or* `frequency`	frequency weights
`pweight`	sampling weights
`aweight` *or* `cellsize`	analytic weights
`iweight`	importance weights

The underlining indicates the minimum acceptable abbreviation. Thus `weight` may be abbreviated `w` or `we`, etc.

▷ Example 9

Before explaining what the different types of weights mean, let's obtain the population-weighted mean of a variable called `median_age` from data containing observations on all 50 states of the United States. The dataset also contains a variable named `pop`, which is the total population of each state.

```
. use https://www.stata-press.com/data/r18/census12
(1980 Census data by state)
. summarize median_age [weight=pop]
(analytic weights assumed)
```

Variable	Obs	Weight	Mean	Std. dev.	Min	Max
median_age	50	225907472	30.11047	1.66933	24.2	34.7

In addition to telling us that our dataset contains 50 observations, Stata informs us that the sum of the weight is 225,907,472, which was the number of people living in the United States as of the 1980 census. The weighted mean is 30.11. We were also informed that Stata assumed that we wanted "analytic" weights. ◁

`weight` is each command's idea of what the "natural" weights are and is one of `fweight`, `pweight`, `aweight`, or `iweight`. When you specify the vague `weight`, the command informs you which kind it assumes. Not every command supports every kind of weight. A note below the syntax diagram for a command will tell you which weights the command supports.

Stata understands four kinds of weights:

1. `fweights`, or frequency weights, indicate duplicated observations. `fweights` are always integers. If the `fweight` associated with an observation is 5, that means there are really 5 such observations, each identical.
2. `pweights`, or sampling weights, denote the inverse of the probability that this observation is included in the sample because of the sampling design. A `pweight` of 100, for instance, indicates that this observation is representative of 100 subjects in the underlying population. The scale of these weights does not matter in terms of estimated parameters and standard errors, except when estimating totals and computing finite-population corrections with the `svy` commands; see [SVY] **Survey**.
3. `aweights`, or analytic weights, are inversely proportional to the variance of an observation; that is, the variance of the jth observation is assumed to be σ^2/w_j, where w_j are the weights. Typically, the observations represent averages, and the weights are the number of elements that gave rise to the average. For most Stata commands, the recorded scale of `aweights` is irrelevant; Stata internally rescales them to sum to N, the number of observations in your data, when it uses them.
4. `iweights`, or importance weights, indicate the relative "importance" of the observation. They have no formal statistical definition; this is a catch-all category. Any command that supports `iweights` will define how they are treated. They are usually intended for use by programmers who want to produce a certain computation.

See [U] **20.24 Weighted estimation** for a thorough discussion of weights and their meaning.

❑ Technical note

When you do not specify a weight, the result is equivalent to specifying `[fweight=1]`. ❑

11.1.7 options

Many commands take command-specific options. These are described along with each command in the *Reference* manuals. Options are indicated by typing a comma at the end of the command, followed by the options you want to use.

▷ Example 10

Typing `summarize marriage_rate` produces a table of the mean, standard deviation, minimum, and maximum of the variable `marriage_rate`:

```
. summarize marriage_rate
    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
marriage_r~e |         50    .0133221    .0188122   .0074654   .1428282
```

The syntax diagram for `summarize` is

`summarize` [*varlist*] [*if*] [*in*] [*weight*] [, *options*]

followed by the option table

options	Description
Main	
`detail`	display additional statistics
`meanonly`	suppress the display; calculate only the mean; programmer's option
`format`	use variable's display format
`separator(#)`	draw separator line after every # variables; default is `separator(5)`

Thus the options allowed by `summarize` are `detail` or `meanonly`, `format`, and `separator()`. The shortest allowed abbreviations for these options are `d` for `detail`, `mean` for `meanonly`, `f` for `format`, and `sep()` for `separator()`; see **[U] 11.2 Abbreviation rules**.

Typing `summarize marriage_rate, detail` produces a table that also includes selected percentiles, the four largest and four smallest values, the skewness, and the kurtosis.

```
. summarize marriage_rate, detail
                      marriage_rate
-------------------------------------------------------------
      Percentiles      Smallest
 1%     .0074654       .0074654
 5%     .0078956       .0075757
10%     .0080043       .0078956       Obs                  50
25%     .0089399       .0079079       Sum of wgt.          50

50%     .0105669                      Mean           .0133221
                       Largest       Std. dev.      .0188122
75%     .0122899       .0146266
90%     .0137832       .0153734       Variance       .0003539
95%     .0153734       .0172704       Skewness       6.718494
99%     .1428282       .1428282       Kurtosis       46.77306
```

◁

Some commands have options that are required. For instance, the `ranksum` command requires the `by(`*groupvar*`)` option, which identifies the grouping variable. A *groupvar* is a specific kind of *varname*. It identifies to which group each observation belongs.

❑ Technical note

Once you have typed the *varlist* for the command, you can place options anywhere in the command. You can type `summarize marriage_rate divorce_rate if region=="West", detail`, or you can type `summarize marriage_rate divorce_rate, detail, if region=="West"`. You use a second comma to indicate a return to the command line as opposed to the option list. Leaving out the comma after the word `detail` would cause an error because Stata would attempt to interpret the phrase `if region=="West"` as an option rather than as part of the command.

You may not type an option in the middle of a *varlist*. Typing `summarize marriage_rate, detail, divorce_rate` will result in an error.

Options need not be specified contiguously. You may type `summarize marriage_rate divorce_rate, detail, if region=="South", noformat`. Both `detail` and `noformat` are options.

❑

❑ Technical note

Most options are toggles—they indicate that something either is or is not to be done. Sometimes it is difficult to remember which is the default. The following rule applies to all options: if *option* is an option, then `no`*option* is an option as well, and vice versa. Thus if we could not remember whether `detail` or `nodetail` were the default for `summarize` but we knew that we did not want the detail, we could type `summarize, nodetail`. Typing the `nodetail` option is unnecessary, but Stata will not complain.

Some options take *arguments*. The Stata `kdensity` command has an `n(#)` option that indicates the number of points at which the density estimate is to be evaluated. When an option takes an argument, the argument is enclosed in parentheses.

Some options take more than one argument. In such cases, arguments should be separated from one another by commas. For instance, you might see in a syntax diagram

`saving(`*filename*`[ , replace ])`

Here `replace` is the (optional) second argument. *Lists*, such as lists of variables (varlists) and lists of numbers (numlists), are considered to be one argument. If a syntax diagram reported

`powers(`*numlist*`)`

the list of numbers would be one argument, so the elements would not be separated by commas. You would type, for instance, `powers(1 2 3 4)`. In fact, Stata will tolerate commas here, so you could type `powers(1,2,3,4)`.

Some options take string arguments. `regress` has an `eform()` option that works this way—for instance, `eform("Exp Beta")`. To play it safe, you should type the quotes surrounding the string, although it is not required. If you do not type the quotes, any sequence of two or more consecutive blanks will be interpreted as one blank. Thus `eform(Exp  beta)` would be interpreted the same as `eform(Exp beta)`.

❑

11.1.8 numlist

A *numlist* is a list of numbers. Stata allows certain shorthands to indicate ranges:

Numlist	Meaning
`2`	just one number
`1 2 3`	three numbers
`3 2 1`	three numbers in reversed order
`.5 1 1.5`	three different numbers
`1 3 -2.17 5.12`	four numbers in jumbled order
`1/3`	three numbers: 1, 2, 3
`3/1`	the same three numbers in reverse order
`5/8`	four numbers: 5, 6, 7, 8
`-8/-5`	four numbers: −8, −7, −6, −5
`-5/-8`	four numbers: −5, −6, −7, −8
`-1/2`	four numbers: −1, 0, 1, 2
`1 2 to 4`	four numbers: 1, 2, 3, 4
`4 3 to 1`	four numbers: 4, 3, 2, 1
`10 15 to 30`	five numbers: 10, 15, 20, 25, 30
`1 2:4`	same as 1 2 to 4
`4 3:1`	same as 4 3 to 1
`10 15:30`	same as 10 15 to 30
`1(1)3`	three numbers: 1, 2, 3
`1(2)9`	five numbers: 1, 3, 5, 7, 9
`1(2)10`	the same five numbers, 1, 3, 5, 7, 9
`9(-2)1`	five numbers: 9, 7, 5, 3, and 1
`-1(.5)2.5`	the numbers −1, −.5, 0, .5, 1, 1.5, 2, 2.5
`1[1]3`	same as 1(1)3
`1[2]9`	same as 1(2)9
`1[2]10`	same as 1(2)10
`9[-2]1`	same as 9(−2)1
`-1[.5]2.5`	same as −1(.5)2.5
`1 2 3/5 8(2)12`	eight numbers: 1, 2, 3, 4, 5, 8, 10, 12
`1,2,3/5,8(2)12`	the same eight numbers
`1 2 3/5 8 10 to 12`	the same eight numbers
`1,2,3/5,8,10 to 12`	the same eight numbers
`1 2 3/5 8 10:12`	the same eight numbers

`poisson`'s `constraints()` option has syntax `constraints(`*numlist*`)`. Thus you could type `constraints(2 4 to 8)`, `constraints(2(2)8)`, etc.

11.1.9 datelist

A *datelist* is a list of dates or times and is often used with graph options when the variable being graphed has a date format. For a description of how dates and times are stored and manipulated in Stata, see **[U] 25 Working with dates and times**. Calendar dates, also known as `%td` dates, are recorded in Stata as the number of days since 01jan1960, so 0 means 01jan1960, 1 means 02jan1960, and 16,541 means 15apr2005. Similarly, −1 means 31dec1959, −2 means 30dec1959, and −16,541 means 18sep1914. In such a case, a datelist is a list of dates, as in

```
15apr1973 17apr1973 20apr1973 23apr1973
```

or it is a first and last date with an increment between, as in

```
17apr1973(3)23apr1973
```

or it is a combination:

```
15apr1973 17apr1973(3)23apr1973
```

Dates specified with spaces, slashes, or commas must be bound in parentheses, as in

```
(15 apr 1973) (april 17, 1973)(3)(april 23, 1973)
```

Evenly spaced calendar dates are not especially useful, but with other time units, even spacing can be useful, such as

```
1999q1(1)2005q1
```

when `%tq` dates are being used. `1999q1(1)2005q1` means every quarter between 1999q1 and 2005q1. `1999q1(4)2005q1` would mean every first quarter.

To interpret a datelist, Stata first looks at the format of the related variable and then uses the corresponding date-to-numeric translation function. For instance, if the variable has a `%td` format, the `td()` function is used to translate the date; if the variable has a `%tq` format, the `tq()` function is used; and so on. See *Typing dates into expressions* in [D] **Datetime**.

11.1.10 Prefix commands

Stata has a handful of commands that are used to prefix other Stata commands. `by` *varlist*`:`, discussed in section [U] **11.1.2 by varlist:**, is in fact an example of a prefix command. In that section, we demonstrated `by` using

```
by region: summarize marriage_rate divorce_rate
```

and later,

```
by region, sort: summarize marriage_rate divorce_rate
```

and although we did not, we could also have demonstrated

```
by region, sort: summarize marriage_rate divorce_rate, detail
```

Each of the above runs the `summarize` command separately on the data for each region.

`by` itself follows standard Stata syntax:

`by` *varlist*[`,` *options*]`:` ...

In `by region, sort: summarize marriage_rate divorce_rate, detail`, `region` is `by`'s varlist and `sort` is `by`'s option, just as `marriage_rate` and `divorce_rate` are `summarize`'s varlist and `detail` is `summarize`'s option.

by is not the only prefix command, and the full list of such commands is

Prefix command	Description
by	run command on subsets of data
collect	run command and collect results to include in a table
frame	run command on the data in a specified frame
statsby	same as by, but collect statistics from each run
rolling	run command on moving subsets and collect statistics
bootstrap	run command on bootstrap samples
jackknife	run command on jackknife subsets of data
permute	run command on random permutations
simulate	run command on manufactured data
svy	run command and adjust results for survey sampling
mi estimate	run command on multiply imputed data and adjust results for multiple imputation (MI)
bayes	fit a Bayesian regression model
fmm	fit a finite mixture model
nestreg	run command with accumulated blocks of regressors and report nested model comparison tests
stepwise	run command with stepwise variable inclusion/exclusion
xi	run command after expanding factor variables and interactions; for most commands, using factor variables is preferred to using xi (see [U] 11.4.3 Factor variables)
fp	run command with fractional polynomials of one regressor
mfp	run command with multiple fractional polynomial regressors
capture	run command and capture its return code
noisily	run command and show the output
quietly	run command and suppress the output
version	run command under specified version

The last group—capture, noisily, quietly, and version—deal with programming Stata and, for historical reasons, capture, noisily, and quietly allow you to omit the colon, so one programmer might code

```
quietly regress ...
```

and another

```
quietly: regress ...
```

All the other prefix commands require the colon. In addition to the corresponding reference manual entries, you may want to consult Baum (2016) for a richer discussion of prefix commands.

11.2 Abbreviation rules

Stata allows abbreviations. In this manual, we usually avoid abbreviating commands, variable names, and options to ensure readability:

```
. summarize myvar, detail
```

Experienced Stata users, on the other hand, tend to abbreviate the same command as

```
. sum myv, d
```

As a general rule, command, option, and variable names may be abbreviated to the shortest string of characters that uniquely identifies them.

This rule is violated if the command or option does something that cannot easily be undone; the command must then be spelled out in its entirety.

Also, a few common commands and options are allowed to have even shorter abbreviations than the general rule would allow.

The general rule is applied, without exception, to variable names.

11.2.1 Command abbreviation

The shortest allowed abbreviation for a command or option can be determined by looking at the command's syntax diagram. This minimal abbreviation is shown by underlining:

```
generate
append
rotate
run
```

If there is no underlining, no abbreviation is allowed. For example, `replace` may not be abbreviated, the underlying reason being that `replace` changes the data.

`rename` can be abbreviated `ren`, `rena`, or `renam`, or it can be spelled out in its entirety.

Sometimes short abbreviations are also allowed. Commands that begin with the letter *d* include `decode`, `describe`, `destring`, `dir`, `discard`, `display`, `do`, and `drop`, which suggests that the shortest allowable abbreviation for `describe` is `desc`. However, because `describe` is such a commonly used command, you may abbreviate it with the single letter `d`. You may also abbreviate the `list` command with the single letter `l`.

The other exception to the general abbreviation rule is that commands that alter or destroy data must be spelled out completely. Two commands that begin with the letter *d*, `discard` and `drop`, are destructive in the sense that, once you give one of these commands, there is no way to undo the result. Therefore, both must be spelled out.

The final exceptions to the general rule are commands implemented as ado-files. Such commands may not be abbreviated. Ado-file commands are external, and their names correspond to the names of disk files.

11.2.2 Option abbreviation

Option abbreviation follows the same logic as command abbreviation: you determine the minimum acceptable abbreviation by examining the command's syntax diagram. The syntax diagram for `summarize` reads, in part,

```
summarize ..., detail format
```

The `detail` option may be abbreviated `d`, `de`, `det`, ..., `detail`. Similarly, option `format` may be abbreviated `f`, `fo`, ..., `format`.

The `clear` and `replace` options occur with many commands. The `clear` option indicates that even though completing this command will result in the loss of all data in memory, and even though the data in memory have changed since the data were last saved on disk, you want to continue. `clear` must be spelled out, as in `use newdata, clear`.

The `replace` option indicates that it is okay to save over an existing dataset. If you type `save mydata` and the file `mydata.dta` already exists, you will receive the message "file mydata.dta already exists", and Stata will refuse to overwrite it. To allow Stata to overwrite the dataset, you would type `save mydata, replace`. `replace` may not be abbreviated.

❑ Technical note

`replace` is a stronger modifier than `clear` and is one you should think about before using. With a mistaken `clear`, you can lose hours of work, but with a mistaken `replace`, you can lose days of work.

❑

11.2.3 Variable-name abbreviation

- Variable names may be abbreviated to the shortest string of characters that uniquely identifies them given the data currently loaded in memory.

  If your dataset contained four variables, `state`, `mrgrate`, `dvcrate`, and `dthrate`, you could refer to the variable `dvcrate` as `dvcrat`, `dvcra`, `dvcr`, `dvc`, or `dv`. You might type `list dv` to list the data on `dvcrate`. You could not refer to the variable `dvcrate` as `d`, however, because that abbreviation does not distinguish `dvcrate` from `dthrate`. If you were to type `list d`, Stata would respond with the message "ambiguous abbreviation". (If you wanted to refer to *all* variables that started with the letter *d*, you could type `list d*`; see **[U] 11.4 varname and varlists**.)

- The character `~` may be used to mean that "zero or more characters go here". For instance, `r~8` might refer to the variable `rep78`, or `rep1978`, or `repair1978`, or just `r8`. (The `~` character is similar to the `*` character in **[U] 11.4 varname and varlists**, except that it adds the restriction "and only one variable matches this specification".)

  Above, we said that you could abbreviate variables. You could type `dvcr` to refer to `dvcrate`, but, if there were more than one variable that started with the letters `dvcr`, you would receive an error. Typing `dvcr` is the same as typing `dvcr~`.

11.2.4 Abbreviations for programmers

Stata has several useful commands and functions to assist programmers with abbreviating and unabbreviating command names and variable names.

Command/function	Description
unab	expand and unabbreviate standard variable lists
tsunab	expand and unabbreviate variable lists that may contain time-series operators
fvunab	expand and unabbreviate variable lists that may contain time-series operators or factor variables
unabcmd	unabbreviate command name
novarabbrev	turn off variable abbreviation
varabbrev	turn on variable abbreviation
set varabbrev	set whether variable abbreviations are supported
abbrev(*s*,*n*)	string function that abbreviates *s* to *n* display columns
abbrev(*s*,*n*)	Mata variant of above that allows *s* and *n* to be matrices

11.3 Naming conventions

A name is a sequence of 1 to 32 letters (A–Z, a–z, and any Unicode letter), digits (0–9), and underscores (_).

Programmers: Local macro names can have no more than 31 characters in the name; see [U] **18.3.1 Local macros**.

Stata reserves the following names:

```
alias       _n          _r_p
_all        _N          _r_se
_b          _pi         _r_ub
byte        _pred       _r_z
_coef       _r_b        _r_z_abs
_cons       _rc         _se
double      _r_ci       _skip
float       _r_cri      str#
if          _r_crlb     strL
in          _r_crub     using
int         _r_df       with
long        _r_lb
```

You may not use these reserved names for your variables.

The first character of a name must be a letter or an underscore (macro names are an exception; they may also begin with a digit). We recommend, however, that you not begin your variable names with an underscore. All of Stata's built-in variables begin with an underscore, and we reserve the right to incorporate new *_variables* freely.

Stata respects case; that is, myvar, Myvar, and MYVAR are three distinct names.

Most objects in Stata—not just variables—follow this naming convention.

11.4 varname and varlists

A *varlist* is a list of variable names. The variable names in a *varlist* refer either exclusively to new (not yet created) variables or exclusively to existing variables. A *newvarlist* always refers exclusively to new (not yet created) variables. Similarly, a *varname* refers to one variable, either existing or not yet created. A *newvar* always refers to one new variable.

Sometimes a command will refer to a varname in another way, such as "*groupvar*". This is still a varname. The different name is used to give you an extra hint about the purpose of that variable. For example, a *groupvar* is the name of a variable that defines groups within your data. Other common ways of referring to a *varname* or *varlist* in Stata are

depvar, which means the dependent variable for an estimation command;

indepvars, which means a *varlist* containing the independent variables for an estimation command;

xvar, which means a continuous real variable, often plotted on the x axis of a graph;

yvar, which means a variable that is a function of an *xvar*, often plotted on the y axis of a graph;

clustvar, which means a numeric variable that identifies the cluster or group to which an observation belongs;

panelvar, which means a numeric variable that identifies panels in panel data, also known as cross-sectional time-series data; and

timevar, which means a numeric variable with a `%td`, `%tc`, or `%tC` format.

11.4.1 Lists of existing variables

In lists of existing variable names, variable names may be repeated.

▷ Example 11

If you type `list state mrgrate dvcrate state`, the variable `state` will be listed twice, once in the leftmost column and again in the rightmost column of the list.

◁

Existing variable names may be abbreviated as described in [U] **11.2 Abbreviation rules**. You may also use "`*`" to indicate that "zero or more characters go here". For instance, if you suffix `*` to a partial variable name (for example, `sta*`), you are referring to all variable names that start with that letter combination. If you prefix `*` to a letter combination (for example, `*rate`), you are referring to all variables that end in that letter combination. If you put `*` in the middle (for example, `m*rate`), you are referring to all variables that begin and end with the specified letters. You may put more than one `*` in an abbreviation.

▷ Example 12

If the variables `poplt5`, `pop5to17`, and `pop18p` are in our dataset, we may type `pop*` as a shorthand way to refer to all three variables. For instance, `list state pop*` lists the variables `state`, `poplt5`, `pop5to17`, and `pop18p`.

If we had a dataset with variables `inc1990`, `inc1991`, ..., `inc1999` along with variables `incfarm1990`, ..., `incfarm1999`; `pop1990`, ..., `pop1999`; and `ms1990`, ..., `ms1999`, then `*1995` would be a shorthand way of referring to `inc1995`, `incfarm1995`, `pop1995`, and `ms1995`. We could type, for instance, `list *1995`.

In that same dataset, typing `list i*95` would be a shorthand way of listing `inc1995` and `incfarm1995`.

Typing `list i*f*95` would be a shorthand way of listing to `incfarm1995`.

◁

`~` is an alternative to `*`, and really, it means the same thing. The difference is that `~` indicates that if more than one variable matches the specified pattern, Stata will complain rather than substituting all the variables that match the specification.

▷ Example 13

In the previous example, we could have typed `list i~f~95` to list `incfarm1995`. If, however, our dataset also included variable `infant1995`, then `list i*f*95` would list both variables and `list i~f~95` would complain that `i~f~95` is an ambiguous abbreviation.

◁

You may use `?` to specify that one character goes here. Remember, `*` means zero or more characters go here, so `?*` can be used to mean one or more characters goes here, `??*` can be used to mean two or more characters go here, and so on.

▷ Example 14

In a dataset containing variables `rep1`, `rep2`, . . . , `rep78`, `rep?` would refer to `rep1`, `rep2`, . . . , `rep9`, and `rep??` would refer to `rep10`, `rep11`, . . . , `rep78`.

◁

You may place a dash (`-`) between two variable names to specify all the variables stored between the two listed variables, inclusive. You can determine storage order by using `describe`; it lists variables in the order in which they are stored.

▷ Example 15

If the dataset contains the variables `state`, `mrgrate`, `dvcrate`, and `dthrate`, in that order, typing `list state-dvcrate` is equivalent to typing `list state mrgrate dvcrate`. In both cases, three variables are listed.

◁

11.4.2 Lists of new variables

In lists of *new variables*, no variable names may be repeated or abbreviated.

You may specify a dash (`-`) between two variable names that have the same letter prefix and that end in numbers. This form of the dash notation indicates a range of variable names in ascending numerical order.

For example, typing `input v1-v4` is equivalent to typing `input v1 v2 v3 v4`. Typing `infile state v1-v3 ssn using rawdata` is equivalent to typing `infile state v1 v2 v3 ssn using rawdata`.

Many commands that require a specific number of new variables also allow the new variables to be specified using the *stub*`*` notation. For example, if you are using `predict` to generate four new variables, you could type `predict pred*` to create new variables `pred1`, `pred2`, `pred3`, and `pred4`.

You may specify the storage type before the variable name to force a storage type other than the default. The numeric storage types are `byte`, `int`, `long`, `float` (the default), and `double`. The string storage types are `str#`, where # is replaced with an integer between 1 and 2045, inclusive, representing the maximum length of the string, or `strL`. See **[U] 12 Data**.

For instance, the list `var1 str8 var2 var3` specifies that `var1` and `var3` be given the default storage type and that `var2` be stored as a `str8`—a string whose maximum length is eight bytes.

The list `var1 int var2 var3` specifies that `var2` be stored as an `int`. You may use parentheses to bind a list of variable names. The list `var1 int(var2 var3)` specifies that both `var2` and `var3` be stored as `int`s. Similarly, the list `var1 str20(var2 var3)` specifies that both `var2` and `var3` be stored as `str20`s. The different storage types are listed in **[U] 12.2.2 Numeric storage types** and **[U] 12.4 Strings**.

▷ Example 16

Typing `infile str2 state str10 region v1-v5 using mydata` reads the `state` and `region` strings from the file `mydata.raw` and stores them as `str2` and `str10`, respectively, along with the variables `v1` through `v5`, which are stored as the default storage type `float` (unless we have specified a different default with the `set type` command).

Typing `infile str10(state region) v1-v5 using mydata` would achieve almost the same result, except that the `state` and `region` values recorded in the data would both be assigned to `str10` variables. (We could then use the `compress` command to shorten the strings. See [D] **compress**; it is well worth reading.)

◁

❑ Technical note

You may append a colon and a *value label name* to numeric variables. (See **[U] 12.6 Dataset, variable, and value labels** for a description of value labels.) For instance, `var1 var2:myfmt` specifies that the variable `var2` be associated with the value label stored under the name `myfmt`. This has the same effect as typing the list `var1 var2` and then subsequently giving the command `label values var2 myfmt`.

The advantage of specifying the value label association with the colon notation is that value labels can then be assigned by the current command; see [D] **input** and [D] **infile (free format)**.

❑

▷ Example 17

Typing `infile int(state:stfmt region:regfmt) v1-v5 using mydata, automatic` reads the state and region data from the file `mydata.raw` and stores them as `int`s, along with the variables `v1` through `v5`, which are stored as the default storage type.

In our previous example, both state and region were strings, so how can strings be stored in a numeric variable? See **[U] 12.6 Dataset, variable, and value labels** for the complete answer. The colon notation specifies the name of the value label, and the `automatic` option tells Stata to assign unique numeric codes to all character strings. The numeric code for state, which Stata will make up on the fly, will be stored in the `state` variable. The mapping from numeric codes to words will be stored in the value label named `stfmt`. Similarly, regions will be assigned numeric codes, which are stored in `region`, and the mapping will be stored in `regfmt`.

If we were to `list` the data, the `state` and `region` variables would look like strings. `state`, for instance, would appear to contain things like `AL`, `CA`, and `WA`, but actually it would contain only numbers like 1, 2, 3, and 4.

◁

11.4.3 Factor variables

Factor variables are extensions of varlists of existing variables. When a command allows factor variables, in addition to typing variable names from your data, you can type factor variables, which might look like

`i.`*varname*

`i.`*varname*`#i.`*varname*

`i.`*varname*`#i.`*varname*`#i.`*varname*

`i.`*varname*`##i.`*varname*

`i.`*varname*`##i.`*varname*`##i.`*varname*

Factor variables create indicator variables from categorical variables and are allowed with most estimation and postestimation commands, along with a few other commands.

Consider a variable named `group` that takes on the values 1, 2, and 3. Stata command `list` allows factor variables, so we can see how factor variables are expanded by typing

```
. list group i.group in 1/5
```

	group	1.group	2.group	3.group
1.	1	1	0	0
2.	1	1	0	0
3.	2	0	1	0
4.	2	0	1	0
5.	3	0	0	1

There are no variables named `1.group`, `2.group`, and `3.group` in our data; there is only the variable named `group`. When we type `i.group`, however, Stata acts as if the variables `1.group`, `2.group`, and `3.group` exist. `1.group`, `2.group`, and `3.group` are called virtual variables. `1.group` is a virtual variable equal to 1 when `group` $= 1$, and 0 otherwise. `2.group` is a virtual variable equal to 1 when `group` $= 2$, and 0 otherwise. `3.group` is a virtual variable equal to 1 when `group` $= 3$, and 0 otherwise.

The categorical variable to which factor-variable operators are applied must contain nonnegative integers.

❑ Technical note

We said above that `3.group` equals 1 when `group` $= 3$ and equals 0 otherwise. We should have added that `3.group` equals missing when `group` contains missing. To be precise, `3.group` equals 1 when `group` $= 3$, equals system missing (`.`) when `group` $\geq$ `.`, and equals 0 otherwise.

❑

❑ Technical note

We said above that when we typed `i.group`, Stata acts as if the variables `1.group`, `2.group`, and `3.group` exist, and that might suggest that the act of typing `i.group` somehow created the virtual variables. That is not true; the virtual variables always exist.

In fact, `i.group` is an abbreviation for `1.group`, `2.group`, and `3.group`. In any command that allows factor variables, you can specify virtual variables. Thus the listing above could equally well have been produced by typing

```
. list group 1.group 2.group 3.group in 1/5
```

#.varname is defined as equal to 1 when *varname* = *#*, equal to system missing (`.`) when *varname* ≥ `.`, and equal to 0 otherwise. Thus `4.group` is defined even when `group` takes on only the values 1, 2, and 3. `4.group` would be equal to 0 in all observations. Referring to `4.group` would not produce an error such as "virtual variable not found".

❑

When factor-variable operators are used in a regression command, one of the categories is chosen as a base category. If we type

```
. regress y i.group
```

this is equivalent to typing

```
. regress y 1b.group 2.group 3.group
```

`1b.group` is different from the other virtual variables. The `b` is a marker indicating base value. `1b.group` is a virtual variable equal to 0. We can see this by typing

```
. list group i.group in 1/5
```

	group	1.group	2.group	3.group
1.	1	1	0	0
2.	1	1	0	0
3.	2	0	1	0
4.	2	0	1	0
5.	3	0	0	1

When the `i.group` collection is included in a linear regression, virtual variable `1b.group` drops from the estimation because it does not vary; thus the coefficients on `2.group` and `3.group` would measure the change from `group` = 1. Hence, the term base value.

11.4.3.1 Factor-variable operators

`i.group` is called a factor variable, although more correctly, we should say that `group` is a categorical variable to which factor-variable operators have been applied. There are five factor-variable operators:

Operator	Description
`i.`	unary operator to specify indicators
`c.`	unary operator to treat as continuous
`o.`	unary operator to omit a variable or indicator
`#`	binary operator to specify interactions
`##`	binary operator to specify full-factorial interactions

When you type `i.group`, it forms the indicators for the distinct values of `group`. We will usually say this more briefly as `i.group` forms indicators for the levels of `group`, and sometimes we will abbreviate the statement even more and say `i.group` forms indicators for `group`.

The `c.` operator means continuous. We will get to that below.

The `o.` operator specifies that a continuous variable or an indicator for a level of a categorical variable should be omitted. For example, `o.age` means that the continuous variable `age` should be omitted, and `o2.group` means that the indicator for `group` = 2 should be omitted.

`#` and `##`, pronounced cross and factorial cross, are operators for use with pairs of variables.

`i.group#i.sex` means to form indicators for each combination of the levels of `group` and `sex`.

`group#sex` means the same thing, which is to say that use of `#` implies the `i.` prefix.

`group#c.age` (or `i.group#c.age`) means the interaction of the levels of `group` with the continuous variable `age`. This amounts to forming `i.group` and then multiplying each level by `age`. We already know that `i.group` expands to the virtual variables `1.group`, `2.group`, and `3.group`, so `group#c.age` results in the collection of variables equal to `1.group*age`, `2.group*age`, and `3.group*age`. `1.group*age` will be `age` when `group` = 1, and 0 otherwise. `2.group*age` will be `age` when `group` = 2, and 0 otherwise. `3.group*age` will be `age` when `group` = 3, and 0 otherwise.

In a regression of `y` on `age` and `group#c.age`, `group` = 1 will again be chosen as the base value of `group`. Thus `group#c.age` expands to `1b.group*age`, `2.group*age`, and `3.group*age`. `1b.group*age` will be zero because `1b.group` is zero, so it will be omitted. `2.group*age` will measure the change in the age coefficient for `group` = 2 relative to the base group, and `3.group*age` will measure the change for `group` = 3 relative to the base.

Here are some more examples of use of the operators:

Factor specification	Result
`i.group`	indicators for levels of `group`
`i.group#i.sex`	indicators for each combination of levels of `group` and `sex`, a two-way interaction
`group#sex`	same as `i.group#i.sex`
`group#sex#arm`	indicators for each combination of levels of `group`, `sex`, and `arm`, a three-way interaction
`group##sex`	same as `i.group i.sex group#sex`
`group##sex##arm`	same as `i.group i.sex i.arm group#sex group#arm sex#arm group#sex#arm`
`sex#c.age`	two variables—`age` for males and 0 elsewhere, and `age` for females and 0 elsewhere; if `age` is also in the model, one of the two virtual variables will be treated as a base
`sex##c.age`	same as `i.sex age sex#c.age`
`c.age`	same as `age`
`c.age#c.age`	`age` squared
`c.age#c.age#c.age`	`age` cubed

Several factor-variable terms are often specified in the same varlist, such as

```
. regress y  i.sex i.group sex#group age sex#c.age
```

or, equivalently,

```
. regress y  sex##group sex##c.age
```

11.4.3.2 Base levels

When we typed `i.group` in a regression command, `group` = 1 became the base level. When we do not specify otherwise, the smallest level becomes the base level.

You can specify the base level of a factor variable by using the `ib.` operator. The syntax is

Base operator[a]	Description
`ib#.`	use # as base, # = value of variable
`ib(##).`	use the #th ordered value as base[b]
`ib(first).`	use smallest value as base (default)
`ib(last).`	use largest value as base
`ib(freq).`	use most frequent value as base
`ibn.`	no base level

[a] The `i` may be omitted. For instance, you can type `ib2.group` or `b2.group`.
[b] For example, `ib(#2).` means to use the second value as the base.

Thus, if you want to use `group` = 3 as the base, you can type `ib3.group`. You can type

```
. regress y  i.sex ib3.group sex#ib3.group age sex#c.age
```

or you can type

```
. regress y  i.sex ib3.group sex#group age sex#c.age
```

That is, you only have to set the base once. If you specify the base level more than once, it must be the same base level. You will get an error if you attempt to change base levels in midsentence.

If you type `ib3.group`, the virtual variables become `1.group`, `2.group`, and `3b.group`.

Were you to type `ib(freq).group`, the virtual variables might be `1b.group`, `2.group`, and `3.group`; `1.group`, `2b.group`, and `3.group`; or `1.group`, `2.group`, and `3b.group`, depending on the most frequent group in the data.

11.4.3.3 Setting base levels permanently

You can permanently set the base level by using the `fvset` command; see [R] **fvset**. For example,

```
. fvset base 3  group
```

sets the base for `group` to be 3. The setting is recorded in the data, and if the dataset is resaved, the base level will be remembered in future sessions.

If you want to set the base group back to the default, type

```
. fvset base default  group
```

If you want to set the base levels for a group of variables to be the largest value, you can type

```
. fvset base last  group sex arm
```

See [R] **fvset** for details.

Base levels can be temporarily overridden by using the `ib.` operator regardless of whether they are set explicitly.

11.4.3.4 Selecting levels

Typing `i.group` specifies virtual variables `1b.group`, `2.group`, and `3.group`. Regardless of whether you type `i.group`, you can access those virtual variables. You can, for instance, use them in expressions and `if` statements:

```
. list if 3.group
  (output omitted)
. generate over_age = cond(3.group, age-21, 0)
```

Although throughout this section we have been typing `#.group` such as `3.group` as if it is somehow different from `i.group`, the complete, formal syntax is `i3.group`. You are allowed to omit the `i`. The point is that `i3.group` is just a special case of `i.group`; `i3.group` specifies an indicator for the third level of `group`, and `i.group` specifies the indicators for all the levels of `group`. Anyway, the above commands could be typed as

```
. list if i3.group
  (output omitted)
. generate over_age = cond(i3.group, age-21, 0)
```

Similarly, the virtual variables `1b.group`, `2.group`, and `3.group` more formally would be referred to as `i1b.group`, `i2.group`, and `i3.group`. You are allowed to omit the leading `i` whenever what appears after is a number or a `b` followed by a base specification.

You can select a range of levels—a range of virtual variables—by using the `i(`*numlist*`).`*varname*. This can be useful when specifying the model to be fit using estimation commands. You may not omit the `i` when specifying a numlist.

Examples	Description
`i2.cat`	single indicator for `cat` = 2
`2.cat`	same as `i2.cat`
`i(2 3 4).cat`	three indicators, `cat` = 2, `cat` = 3, and `cat` = 4; same as `i2.cat i3.cat i4.cat`
`i(2/4).cat`	same as `i(2 3 4).cat`
`2.cat#1.sex`	a single indicator that is 1 when `cat` = 2 and `sex` = 1 and is 0 otherwise
`i2.cat#i1.sex`	same as `2.cat#1.sex`

Rather than selecting the levels that should be included, you can specify the levels that should be omitted by using the `o.` operator. When you use `io(`*numlist*`).`*varname* in a command, indicators for the levels of *varname* other than those specified in *numlist* are included. When omitted levels are specified with the `o.` operator, the `i.` operator is implied, and the remaining indicators for the levels of *varname* will be included.

Examples	Description
`io2.cat`	indicators for levels of `cat`, omitting the indicator for `cat` = 2
`o2.cat`	same as `io2.cat`
`io(2 3 4).cat`	indicators for levels of `cat`, omitting three indicators, `cat` = 2, `cat` = 3, and `cat` = 4
`o(2 3 4).cat`	same as `io(2 3 4).cat`
`o(2/4).cat`	same as `io(2 3 4).cat`
`o2.cat#o1.sex`	indicators for each combination of the levels of `cat` and `sex`, omitting the indicator for `cat` = 2 and `sex` = 1

11.4.3.5 Applying operators to a group of variables

Factor-variable operators may be applied to groups of variables by using parentheses. You may type, for instance,

```
i.(group sex arm)
```

to mean `i.group i.sex i.arm`.

In the examples that follow, variables group, sex, arm, and cat are categorical, and variables age, wt, and bp are continuous:

Examples	Expansion
i.(group sex arm)	i.group i.sex i.arm
group#(sex arm cat)	group#sex group#arm group#cat
group##(sex arm cat)	i.group i.sex i.arm i.cat group#sex group#arm group#cat
group#(c.age c.wt c.bp)	group#c.age group#c.wt group#c.bp
group#c.(age wt bp)	same as group#(c.age c.wt c.bp)

Parentheses can shorten what you type and make it more readable. For instance,

```
. regress y  i.sex i.group sex#group age sex#c.age c.age#c.age sex#c.age#c.age
```

is easier to understand when written as

```
. regress y  sex##(group c.age c.age#c.age)
```

11.4.3.6 Using factor variables with time-series operators

Factor-variable operators may be combined with the L. and F. time-series operators, so you may specify lags and leads of factor variables in time-series applications. You could type iL.group or Li.group; the order of the operators does not matter. You could type L.group#L.arm or L.group#c.age.

Examples include

```
. regress y b1.sex##(i(2/4).group cL.age cL.age#cL.age)
. regress y 2.arm#(sex#i(2/4)b3.group cL.age)
. regress y 2.arm##cat##(sex##i(2/4)b3.group cL.age#c.age) c.bp
>           c.bp#c.bp c.bp#c.bp#c.bp sex##c.bp#c.age
```

11.4.3.7 Video examples

Introduction to factor variables in Stata, part 1: The basics

Introduction to factor variables in Stata, part 2: Interactions

Introduction to factor variables in Stata, part 3: More interactions

11.4.4 Time-series varlists

Time-series varlists are a variation on varlists of existing variables. When a command allows a time-series varlist, you may include time-series operators. For instance, `L.gnp` refers to the lagged value of variable `gnp`. The time-series operators are

Operator	Meaning
`L.`	lag x_{t-1}
`L2.`	2-period lag x_{t-2}
...	
`F.`	lead x_{t+1}
`F2.`	2-period lead x_{t+2}
...	
`D.`	difference $x_t - x_{t-1}$
`D2.`	difference of difference $x_t - x_{t-1} - (x_{t-1} - x_{t-2}) = x_t - 2x_{t-1} + x_{t-2}$
...	
`S.`	"seasonal" difference $x_t - x_{t-1}$
`S2.`	lag-2 (seasonal) difference $x_t - x_{t-2}$
...	

Time-series operators may be repeated and combined. `L3.gnp` refers to the third lag of variable `gnp`. So do `LLL.gnp`, `LL2.gnp`, and `L2L.gnp`. `LF.gnp` is the same as `gnp`. `DS12.gnp` refers to the one-period difference of the 12-period difference. `LDS12.gnp` refers to the same concept, lagged once.

`D1.` = `S1.`, but `D2.` ≠ `S2.`, `D3.` ≠ `S3.`, and so on. `D2.` refers to the difference of the difference. `S2.` refers to the two-period difference. If you wanted the difference of the difference of the 12-period difference of `gnp`, you would write `D2S12.gnp`.

Operators may be typed in uppercase or lowercase. Most users would type `d2s12.gnp` instead of `D2S12.gnp`.

You may type operators however you wish; Stata internally converts operators to their canonical form. If you typed `ld2ls12d.gnp`, Stata would present the operated variable as `L2D3S12.gnp`.

In addition to using *operator#*, Stata understands *operator*`(`*numlist*`)` to mean a set of operated variables. For instance, typing `L(1/3).gnp` in a varlist is the same as typing `L.gnp L2.gnp L3.gnp`. The operators can also be applied to a list of variables by enclosing the variables in parentheses; for example,

```
. use https://www.stata-press.com/data/r18/gxmpl1
. list year L(1/3).(gnp cpi)
```

	year	L. gnp	L2. gnp	L3. gnp	L. cpi	L2. cpi	L3. cpi
1.	1989	.	.	.	.	.	.
2.	1990	5837.9	.	.	124	.	.
3.	1991	6026.3	5837.9	.	130.7	124	.
4.	1992	6367.4	6026.3	5837.9	136.2	130.7	124
5.	1993	6689.3	6367.4	6026.3	140.3	136.2	130.7
6.	1994	7098.4	6689.3	6367.4	144.5	140.3	136.2
7.	1995	7433.4	7098.4	6689.3	148.2	144.5	140.3
8.	1996	7851.9	7433.4	7098.4	152.4	148.2	144.5

The parentheses notation may be used with any operator. Typing `D(1/3).gnp` would return the first through third differences.

The parentheses notation may be used in operator lists with multiple operators, such as `L(0/3)D2S12.gnp`.

Operator lists may include up to one set of parentheses, which may enclose a *numlist*; see **[U] 11.1.8 numlist**.

The time-series operators `L.` and `F.` may be combined with factor variables. If we want to lag the indicator variables for the levels of the factor variable `region`, we would type `iL.region`. We could also say that we are specifying the level indicator variables for the lag of the region variables. They are equivalent statements.

The numlists and parentheses notation from both factor varlists and time-series operators may be combined. For example, `iL(1/3).region` specifies the first three lags of the level indicators for `region`. If `region` has four levels, this is equivalent to typing `i1L1.region i2L1.region i3L1.region i4L1.region i1L2.region i2L2.region i3L2.region i4L2.region i1L3.region i2L3.region i3L3.region i4L3.region`. Pushing the notation further, `i(1/2)L(1/3).(region education)` specifies the first three lags of the level 1 and level 2 indicator variables for both `region` and `education`.

❑ Technical note

The `D.` and `S.` time-series operators may not be combined with factor variables because such combinations could have two meanings. `iD.a` could be the level indicators for the difference of the variable `a` from its prior period, or it could be the level indicators differenced between the two periods. These are generally not the same values, nor even the same number of indicators. Moreover, they are rarely interesting.

❑

Before you can use time-series operators in varlists, you must set the time variable by using the `tsset` command:

```
. list l.gnp
time variable not set
r(111);
. tsset time
  (output omitted)
. list l.gnp
  (output omitted)
```

See [TS] **tsset**. The time variable must take on integer values. Also, the data must be sorted on the time variable. `tsset` handles this, but later you might encounter

```
. list l.mpg
not sorted
r(5);
```

Then type `sort time` or type `tsset` to reestablish the order.

The time-series operators respect the time variable. `L2.gnp` refers to $\texttt{gnp}_{t-2}$, regardless of missing observations in the dataset. In the following dataset, the observation for 1992 is missing:

```
. use https://www.stata-press.com/data/r18/gxmpl2
. list year gnp l2.gnp, separator(0)
```

	year	gnp	L2. gnp	
1.	1989	5837.9	.	
2.	1990	6026.3	.	
3.	1991	6367.4	5837.9	
4.	1993	7098.4	6367.4	← note, filled in correctly
5.	1994	7433.4	.	
6.	1995	7851.9	7098.4	

Operated variables may be used in expressions:

```
. generate gnplag2 = l2.gnp
(3 missing values generated)
```

Stata also understands cross-sectional time-series data. If you have cross sections of time series, you indicate this when you `tsset` the data:

```
. tsset country year
```

See [TS] **tsset**. In fact, you can type that, or you can type

```
. xtset country year
```

`xtset` is how you set panel data just as `tsset` is how you set time-series data and here the two commands do the same thing. Some panel datasets are not cross-sectional time series, however, in that the second variable is not time, so `xtset` also allows

```
. xtset country
```

See [XT] **xtset**.

11.4.4.1 Video example

Time series, part 3: Time-series operators

11.5 by varlist: construct

`by` *varlist*`:` *command*

The `by` prefix causes *command* to be repeated for each distinct value or combination of values of the variables in *varlist*. *varlist* may contain numeric, string, or a mixture of numeric and string variables. (*varlist* may not contain time-series operators.)

`by` is an optional prefix to perform a Stata command separately for each group of observations where the values of the variables in the *varlist* are the same.

During each iteration, the values of the system variables `_n` and `_N` are set in relation to the first observation in the by-group; see [U] **13.7 Explicit subscripting**. The `in` *range* qualifier cannot be used with `by` *varlist*`:` because ranges specify absolute rather than relative observation numbers.

❑ Technical note

The inability to combine `in` and `by` is not really a constraint because `if` provides all the functionality of `in` and a bit more. If you wanted to perform *command* for the first three observations in each of the by-groups, you could type

```
. by varlist: command if _n<=3
```

❑

The results of *command* would be the same as if you had formed separate datasets for each group of observations, `saved` them, `used` each separately, and issued *command*.

▷ Example 18

We provide some examples using `by` in [U] **11.1.2 by varlist:** above. We demonstrate the effect of `by` on `_n`, `_N`, and explicit subscripting in [U] **13.7 Explicit subscripting**.

`by` requires that the data first be sorted. For instance, if we had data on the average January and July temperatures in degrees Fahrenheit for 420 cities located in the Northeast and West and wanted to obtain the averages, `by region`, across those cities, we might type

```
. use https://www.stata-press.com/data/r18/citytemp3, clear
(City temperature data)
. by region: summarize tempjan tempjuly
not sorted
r(5);
```

Stata refused to honor our request because the data are not sorted by `region`. We must either `sort` the data by `region` first (see [D] **sort**) or specify `by`'s `sort` option (which has the same effect):

```
. by region, sort: summarize tempjan tempjuly
```

```
-> region = NE
    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
     tempjan |        164    27.88537    3.543096        16.6       31.8
    tempjuly |        164       73.35    2.361203        66.5       76.8

-> region = N Cntrl
    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
     tempjan |        284    21.69437    5.725392         2.2       32.6
    tempjuly |        284    73.46725    3.103187        64.5       81.4

-> region = South
    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
     tempjan |        250     46.1456    10.38646        28.9         68
    tempjuly |        250     80.9896     2.97537          71       87.4

-> region = West
    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
     tempjan |        256    46.22539    11.25412          13       72.6
    tempjuly |        256    72.10859    6.483131        58.1       93.6
```

◁

▷ Example 19

Using the same data as in the example above, we estimate regressions, by `region`, of average January temperature on average July temperature. Both temperatures are specified in degrees Fahrenheit.

```
. by region: regress tempjan tempjuly
```

```
-> region = NE
      Source |       SS           df       MS      Number of obs   =       164
-------------+----------------------------------   F(1, 162)       =    479.82
       Model |  1529.74026         1  1529.74026   Prob > F        =    0.0000
    Residual |  516.484453       162  3.18817564   R-squared       =    0.7476
-------------+----------------------------------   Adj R-squared   =    0.7460
       Total |  2046.22471       163  12.5535258   Root MSE        =    1.7855

------------------------------------------------------------------------------
     tempjan |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
    tempjuly |    1.297424   .0592303    21.90   0.000     1.180461    1.414387
       _cons |   -67.28066   4.346781   -15.48   0.000    -75.86431     -58.697
------------------------------------------------------------------------------

-> region = N Cntrl
      Source |       SS           df       MS      Number of obs   =       284
-------------+----------------------------------   F(1, 282)       =    115.89
       Model |  2701.97917         1  2701.97917   Prob > F        =    0.0000
    Residual |  6574.79175       282  23.3148644   R-squared       =    0.2913
-------------+----------------------------------   Adj R-squared   =    0.2887
       Total |  9276.77092       283  32.7801093   Root MSE        =    4.8285

------------------------------------------------------------------------------
     tempjan |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
    tempjuly |   .9957259   .0924944    10.77   0.000     .8136589    1.177793
       _cons |  -51.45888   6.801344    -7.57   0.000    -64.84673   -38.07103
------------------------------------------------------------------------------

-> region = South
      Source |       SS           df       MS      Number of obs   =       250
-------------+----------------------------------   F(1, 248)       =     95.17
       Model |  7449.51623         1  7449.51623   Prob > F        =    0.0000
    Residual |  19412.2231       248  78.2750933   R-squared       =    0.2773
-------------+----------------------------------   Adj R-squared   =    0.2744
       Total |  26861.7394       249  107.878471   Root MSE        =    8.8473

------------------------------------------------------------------------------
     tempjan |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
    tempjuly |     1.83833   .1884392     9.76   0.000     1.467185    2.209475
       _cons |     -102.74   15.27187    -6.73   0.000    -132.8191   -72.66089
------------------------------------------------------------------------------
```

```
-> region = West
      Source |       SS           df       MS      Number of obs   =       256
-------------+----------------------------------   F(1, 254)       =      2.84
       Model |  357.161728         1  357.161728   Prob > F        =    0.0932
    Residual |  31939.9031       254   125.74765   R-squared       =    0.0111
-------------+----------------------------------   Adj R-squared   =    0.0072
       Total |  32297.0648       255  126.655156   Root MSE        =    11.214

------------------------------------------------------------------------------
     tempjan |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
    tempjuly |   .1825482   .1083166     1.69   0.093    -.0307648    .3958613
       _cons |    33.0621   7.84194     4.22   0.000     17.61859     48.5056
------------------------------------------------------------------------------
```

The regressions show that a 1-degree increase in the average July temperature in the Northeast corresponds to a 1.3-degree increase in the average January temperature. In the West, however, it corresponds to a 0.18-degree increase, which is only marginally significant.

◁

❑ Technical note

`by` has a second syntax that is especially useful when you want to play it safe:

`by` *varlist*$_1$ `(`*varlist*$_2$`):` *command*

This says that Stata is to verify that the data are sorted by *varlist*$_1$ *varlist*$_2$ and then, assuming that is true, perform *command* by *varlist*$_1$. For instance,

```
. by subject (time): generate finalval = val[_N]
```

By typing this, we want to create new variable `finalval`, which contains, in each observation, the final observed value of `val` for each subject in the data. The final value will be the last value if, within subject, the data are sorted by time. The above command verifies that the data are sorted by `subject` and `time` and then, if they are, performs

```
. by subject: generate finalval = val[_N]
```

If the data are not sorted properly, an error message will instead be issued. Of course, we could have just typed

```
. by subject: generate finalval = val[_N]
```

after verifying for ourselves that the data were sorted properly, as long as we were careful to look.

`by`'s second syntax can be used with `by`'s `sort` option, so we can also type

```
. by subject (time), sort: generate finalval = val[_N]
```

which is equivalent to

```
. sort subject time
. by subject: generate finalval = val[_N]
```

❑

See Mitchell (2020, chap. 8) for numerous examples of processing groups using the `by:` construct. Also see Cox (2002).

11.6 Filenaming conventions

Some commands require that you specify a *filename*. Filenames are specified in the way natural for your operating system:

Windows	Unix	Mac
`mydata`	`mydata`	`mydata`
`mydata.dta`	`mydata.dta`	`mydata.dta`
`c:mydata.dta`	`~friend/mydata.dta`	`~friend/mydata.dta`
`"my data"`	`"my data"`	`"my data"`
`"my data.dta"`	`"my data.dta"`	`"my data.dta"`
`myproj\mydata`	`myproj/mydata`	`myproj/mydata`
`"my project\my data"`	`"my project/my data"`	`"my project/my data"`
`C:\analysis\data\mydata`	`~/analysis/data/mydata`	`~/analysis/data/mydata`
`"C:\my project\my data"`	`"~/my project/my data"`	`"~/my project/my data"`
`..\data\mydata`	`../data/mydata`	`../data/mydata`
`"..\my project\my data"`	`"../my project/my data"`	`"../my project/my data"`

We strongly discourage using Unicode characters beyond plain ASCII in filenames because different operating systems use different UTF encodings for Unicode characters. For example, because Linux encodes filenames in UTF-8 and Windows encodes them in UTF-16, the file may become unusable after it has been transferred from one system to another if it contains Unicode characters beyond plain ASCII.

In most cases, where *filename* is a file that you are loading, *filename* may also be a URL. For instance, we might specify `use https://www.stata-press.com/data/r18/nlswork`.

All operating systems allow blanks in filenames, and so does Stata. However, if the filename includes a blank, you must enclose the filename in double quotes. Typing

```
. save "my data"
```

would create the file `my data.dta`. Typing

```
. save my data
```

would be an error.

Usually (the exceptions being `copy`, `dir`, `ls`, `erase`, `rm`, and `type`), Stata automatically provides a file extension if you do not supply one. For instance, if you type `use mydata`, Stata assumes that you mean `use mydata.dta` because `.dta` is the file extension Stata normally uses for data files.

Stata provides the following default file extensions that are used by various commands:

.ado	automatically loaded do-files
.dct	text data dictionary
.do	do-file
.dta	Stata dataset file format
.dtas	Stata frameset file format
.dtasig	datasignature file
.gph	graph
.grec	Graph Editor recording (text format)
.irf	impulse–response function datasets
.log	log file in text format
.mata	Mata source code
.mlib	Mata library
.mmat	Mata matrix
.mo	Mata object file
.raw	text-format data
.smcl	log file in SMCL format
.stbcal	business calendars
.ster	saved estimates
.sthlp	help file
.stjson	Stata collection results, labels, and styles
.stpr	project file
.stptrace	parameter-trace file; see [MI] **mi ptrace**
.stsem	SEM Builder file
.stswm	spatial weighting matrix
.stswp	Do-file Editor backup (swap) file
.stxer	ancillary file to .ster when using lasso commands
.sum	checksum files to verify network transfers

You do not have to name your data files with the .dta extension—if you type an explicit file extension, it will override the default. For instance, if your dataset was stored as myfile.dat, you could type use myfile.dat. If your dataset was stored as simply myfile with no file extension, you could type the period at the end of the filename to indicate that you are explicitly specifying the null extension. You would type use myfile. to use this dataset.

❑ Technical note

Stata also uses other file extensions. These files are of interest only to advanced programmers or are for Stata's internal use. They are

.class	class file for object-oriented programming; see [P] **class**
.dlg	dialog resource file
.idlg	dialog resource include file
.ihlp	help include file
.key	search's keyword database file
.maint	maintenance file (for Stata's internal use only)
.mnu	menu file (for Stata's internal use only)
.pkg	user-site package file
.plugin	compiled addition (DLL)
.scheme	control file for a graph scheme
.style	graph style file
.toc	user-site description file

❑

11.6.1 A special note for Mac users

Have you seen the notation `myfolder/myfile` before? This notation is called a path and describes the location of a file or folder (also called a directory).

You do not have to use this notation if you do not like it. You could instead restrict yourself to using files only in the current folder. If that turns out to be too restricting, Stata for Mac provides enough menus and buttons that you can probably get by. You may, however, find the notation convenient. If you do, here is the rest of the definition.

The character `/` is called a path delimiter and delimits folder names and filenames in a path. If the path starts with no path delimiter, the path is relative to the current folder.

For example, the path `myfolder/myfile` refers to the file `myfile` in the folder `myfolder`, which is contained in the current folder.

The characters `..` refer to the folder containing the current folder. Thus `../myfile` refers to `myfile` in the folder containing the current folder, and `../nextdoor/myfile` refers to `myfile` in the folder `nextdoor` in the folder containing the current folder.

If a path starts with a path delimiter, the path is called an absolute path and describes a fixed location of a file or folder name, regardless of what the current folder is. The leading `/` in an absolute path refers to the root directory, which is the main hard drive from which the operating system is booted. For example, the path `/myfolder/myfile` refers to the file `myfile` in the folder `myfolder`, which is contained in the main hard drive.

11.6.2 A shortcut to your home directory

Stata understands `~` to mean your home directory. Thus, you can refer to a dataset named `mydata.dta` in a subdirectory named `mydir` within your home directory by referring to the path

```
~\mydir\mydata.dta
```

in Stata for Windows or by referring to the path

```
~/mydir/mydata.dta
```

in Stata for Mac or Stata for Unix.

11.7 References

Baum, C. F. 2016. *An Introduction to Stata Programming*. 2nd ed. College Station, TX: Stata Press.

Buis, M. L. 2020. Stata tip 135: Leaps and bounds. *Stata Journal* 20: 244–249.

Cox, N. J. 2002. Speaking Stata: How to move step by: step. *Stata Journal* 2: 86–102.

——. 2009. Stata tip 79: Optional arguments to options. *Stata Journal* 9: 504.

——. 2023. Stata tip 151: Puzzling out some logical operators. *Stata Journal* 23: 293–297.

Cox, N. J., and C. B. Schechter. 2019. Speaking Stata: How best to generate indicator or dummy variables. *Stata Journal* 19: 246–259.

——. 2023. Stata tip 152: if and if: When to use the if qualifier and when to use the if command. *Stata Journal* 23: 589–594.

Daniels, L., and N. Minot. 2020. *An Introduction to Statistics and Data Analysis Using Stata*. Thousand Oaks, CA: Sage.

Kolev, G. I. 2006. Stata tip 31: Scalar or variable? The problem of ambiguous names. *Stata Journal* 6: 279–280.

Mitchell, M. N. 2020. *Data Management Using Stata: A Practical Handbook*. 2nd ed. College Station, TX: Stata Press.

Ryan, P. 2005. Stata tip 22: Variable name abbreviation. *Stata Journal* 5: 465–466.