Title

stata.com

split — Split string variables into parts

Syntax

```
split strvar [if] [in] [, options]
```

options	Description
Main	
`generate(`*stub*`)`	begin new variable names with *stub*; default is *strvar*
`parse(`*parse_strings*`)`	parse on specified strings; default is to parse on spaces
`limit(`#`)`	create a maximum of # new variables
`notrim`	do not trim leading or trailing spaces of original variable
Destring	
`destring`	apply `destring` to new string variables, replacing initial string variables with numeric variables where possible
`ignore("`*chars*`")`	remove specified nonnumeric characters
`force`	convert nonnumeric strings to missing values
`float`	generate numeric variables as type `float`
`percent`	convert percent variables to fractional form

Menu

Data > Create or change data > Other variable-transformation commands > Split string variables into parts

Description

`split` splits the contents of a string variable, *strvar*, into one or more parts, using one or more *parse_strings* (by default, blank spaces), so that new string variables are generated. Thus `split` is useful for separating "words" or other parts of a string variable. *strvar* itself is not modified.

Options

Main

`generate(`*stub*`)` specifies the beginning characters of the new variable names so that new variables *stub*`1`, *stub*`2`, etc., are produced. *stub* defaults to *strvar*.

`parse(`*parse_strings*`)` specifies that, instead of using spaces, parsing use one or more *parse_strings*. Most commonly, one string that is one punctuation character will be specified. For example, if `parse(,)` is specified, then `"1,2,3"` is split into `"1"`, `"2"`, and `"3"`.

You can also specify 1) two or more strings that are alternative separators of "words" and 2) strings that consist of two or more characters. Alternative strings should be separated by spaces. Strings that include spaces should be bound by `" "`. Thus if `parse(, " ")` is specified, `"1,2 3"` is also split into `"1"`, `"2"`, and `"3"`. Note particularly the difference between, say, `parse(a b)` and `parse(ab)`: with the first, `a` and `b` are both acceptable as separators, whereas with the second, only the string `ab` is acceptable.

`limit(#)` specifies an upper limit to the number of new variables to be created. Thus `limit(2)` specifies that, at most, two new variables be created.

`notrim` specifies that the original string variable not be trimmed of leading and trailing spaces before being parsed. `notrim` is not compatible with parsing on spaces, because the latter implies that spaces in a string are to be discarded. You can either specify a parsing character or, by default, allow a `trim`.

Destring

`destring` applies `destring` to the new string variables, replacing the variables initially created as strings by numeric variables where possible. See [D] **destring**.

`ignore()`, `force`, `float`, `percent`; see [D] **destring**.

Remarks and examples

stata.com

`split` is used to split a string variable into two or more component parts, for example, "words". You might need to correct a mistake, or the string variable might be a genuine composite that you wish to subdivide before doing more analysis.

The basic steps applied by `split` are, given one or more separators, to find those separators within the string and then to generate one or more new string variables, each containing a part of the original. The separators could be, for example, spaces or other punctuation symbols, but they can in turn be strings containing several characters. The default separator is a space.

The key string functions for subdividing string variables and, indeed, strings in general, are `strpos()`, which finds the position of separators, and `substr()`, which extracts parts of the string. (See [D] **functions**.) `split` is based on the use of those functions.

If your problem is not defined by splitting on separators, you will probably want to use `substr()` directly. Suppose that you have a string variable, `date`, containing dates in the form `"21011952"` so that the last four characters define a year. This string contains no separators. To extract the year, you would use `substr(date,-4,4)`. Again suppose that each woman's obstetric history over the last 12 months was recorded by a `str12` variable containing values such as `"npppppppbnn"`, where `p`, `b`, and `n` denote months of pregnancy, birth, and nonpregnancy. Once more, there are no separators, so you would use `substr()` to subdivide the string.

`split` discards the separators, because it presumes that they are irrelevant to further analysis or that you could restore them at will. If this is not what you want, you might use `substr()` (and possibly `strpos()`).

Finally, before we turn to examples, compare `split` with the `egen` function `ends()`, which produces the head, the tail, or the last part of a string. This function, like all `egen` functions, produces just one new variable as a result. In contrast, `split` typically produces several new variables as the result of one command. For more details and discussion, including comments on the special problem of recognizing personal names, see [D] **egen**.

split can be useful when input to Stata is somehow misread as one string variable. If you copy and paste into the Data Editor, say, under Windows by using the clipboard, but data are space-separated, what you regard as separate variables will be combined because the Data Editor expects comma- or tab-separated data. If some parts of your composite variable are numeric characters that should be put into numeric variables, you could use destring at the same time; see [D] **destring**.

```
. split var1, destring
```

Here no generate() option was specified, so the new variables will have names var11, var12, and so forth. You may now wish to use rename to produce more informative variable names. See [D] **rename**.

You can also use split to subdivide genuine composites. For example, email addresses such as tech-support@stata.com may be split at "@":

```
. split address, p(@)
```

This sequence yields two new variables: address1, containing the part of the email address before the "@", such as "tech-support", and address2, containing the part after the "@", such as "stata.com". The separator itself, "@", is discarded. Because generate() was not specified, the name address was used as a stub in naming the new variables. split displays the names of new variables created, so you will see quickly whether the number created matches your expectations.

If the details of individuals were of no interest and you wanted only machine names, either

```
. egen machinename = ends(address), tail p(@)
```

or

```
. generate machinename = substr(address, strpos(address,"@") + 1,.)
```

would be more direct.

Next suppose that a string variable holds names of legal cases that should be split into variables for plaintiff and defendant. The separators could be " V ", " V. ", " VS ", and " VS. ". (We assume that any inconsistency in the use of uppercase and lowercase has been dealt with by the string function upper(); see [D] **functions**.) Note particularly the leading and trailing spaces in our detailing of separators: the first separator is " V ", for example, not "V", which would incorrectly split "GOLIATH V DAVID" into "GOLIATH ", " DA", and "ID". The alternative separators are given as the argument to parse():

```
. split case, p(" V " " V. " " VS " " VS. ")
```

Again with default naming of variables and recalling that separators are discarded, we expect new variables case1 and case2, with no creation of case3 or further new variables. Whenever none of the separators specified were found, case2 would have empty values, so we can check:

```
. list case if case2 == ""
```

Suppose that a string variable contains fields separated by tabs. For example, import delimited leaves tabs unchanged. Knowing that a tab is char(9), we can type

```
. split data, p(‘=char(9)’) destring
```

p(char(9)) would not work. The argument to parse() is taken literally, but evaluation of functions on the fly can be forced as part of macro substitution.

Finally, suppose that a string variable contains substrings bound in parentheses, such as (1 2 3) (4 5 6). Here we can split on the right parentheses and, if desired, replace those afterward. For example,

```
. split data, p(")")
. foreach v in `r(varlist)' {
        replace `v' = `v' + ")"
. }
```

Stored results

`split` stores the following in `r()`:

Scalars

`r(nvars)`	number of new variables created
`r(varlist)`	names of the newly created variables

Acknowledgments

`split` was written by Nicholas J. Cox of the Department of Geography at Durham University, UK, and coeditor of the *Stata Journal*, who in turn thanks Michael Blasnik of M. Blasnik & Associates for ideas contributed to an earlier jointly written program.

Also see

[D] **destring** — Convert string variables to numeric variables and vice versa

[D] **egen** — Extensions to generate

[D] **functions** — Functions

[D] **rename** — Rename variable

[D] **separate** — Create separate variables