How can I append (stack) a large number of files?

Authored by

stats writer

July 1, 2024

Appending, or stacking, a large number of files refers to the process of adding multiple files onto each other, creating a single file containing all the data from the individual files. This can be done by using a computer program or command that allows for the merging of files, such as a text editor or a specific file management tool. The process involves selecting the files to be stacked and then choosing the appropriate command or function to merge them together. This method is commonly used for organizing and consolidating large amounts of data into a more manageable and organized format. By appending files, users can efficiently handle and access a significant amount of data without having to open or manage multiple individual files.

How can I append (stack) a large number of files? | Stata FAQ

Large datasets sometimes come as a series of smaller datasets each containing information from a subset of cases. For example, one dataset for each U.S. State, or one dataset per year for data over a series of years. The Stata command append can be used to combine datasets quite easily, but if one has a large number of datasets, it can be time consuming to write code to append each dataset to a master dataset. In addition to being time consuming, it is also very easy to make errors when performing this sort of task. This page shows how the process can be streamlined and made less error prone using a few Stata

commands. The first step is to
create a list of the file names in a text file. Then the file read command
can be used to process every item on the list. We will show how to combine files
already in Stata format, and how to perform the same task with datasets that
come as text files.

Reading in and combining Stata data (.dta) files.

The first step is to create a list of files that need to be read in. To do
this we need to start with all of the datasets in a single directory. Once the
files are all in the same directory, you
can use dos commands to create a list of all of the files with a certain
extension in that directory. You can do this from within Stata, since Stata is
capable of relaying commands to your
operating system. You can execute any (valid) command to your operating system
in Stata by beginning the line with either "!" (without the quotation marks) or

the word "shell" (without the quotation marks). The first line of syntax below
moves us to the directory where all of the files we want to merge are stored. The second line of code uses
the ! to tell Stata that we want to execute a command in dos, then it
uses the dir command with options to create a list of all files with the
extension .dta
in the current directory, and saves that list to d:filelist.txt.

```
cd d:mydir
! dir *.dta /a-d /b >d:filelist.txt
```

On a Mac the command is:

```
cd d:mydir
! ls *.dta >filelist.txt
```

Assuming the directory d:mydir contained a
Stata data file for each of the
50 U.S. States, the file filelist.txt should contain a list of 50

**Stata data files, one for each state. The file looks like this (with states
omitted to save space):**

**dataset_ak.dta
dataset_al.dta
dataset_ar.dta
<lines omitted>
dataset_wi.dta
dataset_wv.dta
dataset_wy.dta**

**We will use Stata's file commands to read the list from filelist.txt.
Before we begin, it is useful to know a few things about the commands we will be
using. All commands for dealing with external files have a similar structure. They
start with the word file, followed by the type of operation
we want to perform (in this case of the first line of syntax below, open the file), then we see whatever name that we have given the
file within Stata (also known as a handle, in this case**

myfile), followed by something else, what this something else is
will depend on what we are doing with the file. You have to specify the name of the file
(in our example it is called myfile), because it is possible to have more than
one file open at a time, meaning that we need some way to tell Stata which file
we are referring to with any given command.

Before we begin coming the files, we probably want to increase the memory
Stata allows, so that we will not run out of space to store the observations we
are adding. The amount of memory necessary to append all the files depends on
the number and size of the files we wish to combine. The size of memory allowed
is determined by your operating system. Below we set the memory to 500 megabytes
using the set memory command, the m following 500 specifies
the units, by default Stata assumes the memory is given in kilobytes.

Now we are ready to go about combining the files. The first of code line below instructs Stata to open a file (file open), and tells Stata that we will refer to this file as myfile. The word using followed by a file path and name tells Stata where the text file saved. In the options (i.e. after the comma) we specify read which tells Stata that we are going to be reading (i.e., getting information from, but not adding information to) the file. Although it appears in the options section of the command, it is necessary to specify read or some other action, in order to open a file. The second line of syntax reads the first line of the file we have named myfile.

When Stata reads a text file, unless we tell it to do otherwise, it reads a single line and stops. Next time we tell Stata to read from the file, it will read the next line, and so forth working its way through the file. When Stata "reads" a line from a file, it copies the information in

**that line, and places
it in a local macro, which the user can then access. We tell Stata
to read a line from the file we have called myfile by starting with the
command file read myfile and then giving Stata the name of the local macro
where we want Stata to store the information from the file, in this case, line.**

```
file open myfile using "d:filelist.txt", read
file read myfile line
```

**Now we can start combining the datasets. We will open each of the state
datasets, and append them to a dataset called master_data.dta. The syntax to do
this uses the information from the local macro line. A macro is a type of
placeholder, we can place a piece of information (e.g., a numeric value or a
string of text) into, and, whenever we want to use it, we can put in the name of
the macro, rather than having to type in the actual piece**

of information
contained in the macro. Stata uses a ` (it is on the same key as the ~ character, at the top of your keyboard) followed by
the name of the macro, and then a ' (a normal apostrophe or single quote) to represent whatever you want
to fill in (e.g., `line'). When it interprets the command Stata replaces the `line'
with whatever is stored in the local macro line. Since this is the first dataset we have
processed we need to create the file master_data, the second line of syntax below does this by saving the dataset as
master_data.
The replace option after the comma tells Stata that if it encounters another file called master_data.dta in the current directory, that it should replace that file
with the current file. Finally, the third
line of syntax below reads the next line from myfile.

```
use `line'
save master_data, replace
file read myfile line
```

Now we are ready to go on and read in the rest of our files. However, there are a few things you need to know as background in order for this process to make sense. We know that when we use the file read command, Stata reads lines one a time, and that each time we ask it to read a line, Stata reads the next line in the file. But what does Stata do when it reaches the end of the file? Each time Stata reads a line from a file, it returns that line in the local macro we specify (in this example line), but it also returns something else, a returned result called "r(eof)". (See our Stata FAQ How can I access information stored after I run a command in Stata (returned results)? for more information on returned results.) As long as we are not at the end of the file, r(eof) is equal to zero, when the end of the file is reached, Stata sets r(eof) equal to one, and the local macro specified in the command (in our case line) will be empty.

The

code below uses r(eof). In the lines above, we have just read the second line of the file, so now we have the information from the second line of myfile in the local macro line, and r(eof) is equal to zero (since we are not at the end of the file).

In the first line of syntax below, the command while tells Stata to repeat whatever is inside the curly braces for as long as the statement following it is true. In this case the statement is r(eof)==0, so, as long as r(eof) is equal to zero, that is, as long as we are not at the end of the file, Stata should keep repeating whatever is in the curly brackets. In the first line of syntax between brackets, the append command tells Stata that we want to add observations from another dataset to the working dataset. The using `line' tells Stata that we want to add the observations from the dataset named in the local macro line. The second line of syntax in the curly brackets reads the next line from the file myfile and places it in the local macro line just as we did before. Stata will

continue
reading lines and appending datasets until it reaches the end of myfile
(when r(eof)==1)
then it will move on to whatever is next in the .do file. When it is done, you
should have one dataset, master_data.dta, that contains the cases from each of
your data files.

```
while r(eof)==0 {
append using `line'
file read myfile line
}
```

When we are done reading from it, we should close myfile. After we
close the file, we will not be able to read from it until we open it again. This isn't
necessary, but it is a good practice.

```
file close myfile
```

We probably also want to save our new data file,

master_data.dta now that
we have appended all the files.

save master_data, replace

To make it easier for you to cut and paste into a .do file, here is all the
syntax (except for the commands to your operating system) in a single block:

```
file open myfile using d:filelist.txt, read

file read myfile line
use `line'
save master_data, replace

file read myfile line
while r(eof)==0 { /* while you're not at the end of the file */
append using `line'
file read myfile line
}
file close myfile
save master_data, replace
```

Reading in, and combining data from ASCII (text) files

In the previous example we read in Stata data files. However, datasets often
come as ASCII (text) files. In this example we add the steps to convert the files from
comma separated text files to a Stata data file and then add them to the master
dataset. As we do this, we will also add a variable which identifies which
dataset the file came from. This example uses the insheet command which
requires that there be one case per line and that the data be either comma or
tab separated. This will cover many cases, but if you have data in some other
format, for example fixed format data, you will need to modify the syntax to
accommodate that.

As we did above, we first create a list of files in the directory.
This time, instead of getting a list of all files with the extension .dta, we
will get all files with the extension .txt, by using *.txt in

the second line of
syntax below. Note that we do not want to place the file filelist.txt in the
same directory as our data files, if we do, and we run this command again, the
list of data files will include filelist.txt, which is not actually a data file.
We avoid this by placing filelist.txt in another directory (i.e., d:).

```
cd d:mydir
! dir *.txt /a-d /b >d:filelist.txt
```

Assuming the directory d:mydir contained a
text file for each of the
50 U.S. States, the file filelist.txt should contain a list of 50
text files, one for each state. The file looks like this (with states
omitted to save space):

```
dataset_ak.txt
dataset_al.txt
dataset_ar.txt
```

<lines omitted>
dataset_wi.txt
dataset_wv.txt
dataset_wy.txt

As in the example above, the first line below opens the file using the
file open command,
giving it the name myfile. While the second line of syntax reads the
first line from the file we have named myfile.

```
file open myfile using "d:filelist.txt", read
file read myfile line
```

Now we want to read in the first of our datasets. In addition to formatting
requirements for insheet mentioned above, this example, assumes that the
first line of each of the text files contains the names of the variables. The
first line of code below uses drop _all to drop all cases from the
dataset, effectively clearing the dataset, but without

**some of the other effects
of the using the command clear (insheet requires an empty dataset). In the
second line of syntax below, the insheet command tells Stata that we want to read in an ASCII dataset. The third line of syntax below saves the dataset we have
just read in as a Stata data file, this file will have the same name as the
ASCII datasets plus the file extension .dta,
(e.g., dataset_ak.txt.dta) this is a
slightly odd looking file name, but, it should not cause any problems. The next
(fourth) line of syntax generates a new variable in our dataset called
state_id. We use the string function substr( ) toset the value
of this variable to the two letter state id. The function substr( ),
extracts a portion of a string (a substring). substr( ) takes three
arguments, separated by a commas, the first argument is the string we want
to use enclosed in quotation marks (i.e. the contents of the local macro**

line). The next argument is the place in the string where we
want to start extracting information, in this case, we want to start with the
9th character in the string. The third argument is the number of characters we
want to use, in this case, the two letter state id, so the third argument is
2. For example, substring("dataset_ak.txt",9,2) starts with
the 9th character ("a") and takes it and the next character ("k") for a total of
two, and returns "ak".

```
drop _all
insheet using `line', comma names
save `line'.dta, replace
generate state_id = substr("`line'",9,2)
```

We don't yet have a master dataset, so we will create one using the first
line of syntax below. The second line of syntax below reads the next line in

myfile.

save master_data.dta, replace
file read myfile line

The syntax below works similarly to the syntax for the previous example. The
first line tells Stata to perform the commands within the curly brackets for as
long as r(eof)==0, that is, for as long as we are not at the end
filelist.txt. The second through fourth lines repeat what we did just above.
Specifically, the second
line of syntax below uses insheet to read in a
comma separated data file. The third line saves the new dataset as a Stata data
file, and the forth line creates a variable called state_id that will
contain the two letter state abbreviation. The next line of syntax, append using master_data.dta
appends the dataset master_data.dta to our current dataset, in other words, it
adds the cases from the current dataset to the dataset

master_data. The
following line saves the dataset in memory as master_data.dta, replacing the old
version of this file. Moving to the next line of syntax, we use drop _all to drop
all observations from the current dataset. Finally, as above, we read in the next line of myfile using the command file read.

```
while r(eof)==0 {
insheet using `line', comma names
save `line'.dta, replace
generate state_id = substr("`line'",9,2)
append using master_data.dta
save master_data.dta, replace
drop _all
file read myfile line
}
```

Now we should have a one Stata dataset for each state, as well as a single
dataset, master_data.dta that contains observations from each state along with a
variable identifying which dataset each observation

came from.

To make it easier for you to cut and paste into a .do file, here is all the syntax (except for the commands to your operating system) in a single block:

```
file open myfile using d:filelist_text.txt, read
file read myfile line
insheet using `line', comma names

gen state_id = substr("`line'",9,2)
save `line'.dta, replace
save master_data.dta, replace

drop _all

file read myfile line
while r(eof)==0 {
insheet using `line', comma names
gen state_id = substr("`line'",9,2)
save `line'.dta, replace
append using master_data.dta
save master_data.dta, replace
drop _all
```

```
file read myfile line
}
```

See also