frlink — Link frames

Description

`frlink` creates and helps manage links between datasets in different frames. A link allows the variables in one frame to be accessed by another. See [D] **frames intro** if you do not know what a frame is.

Quick start

Create 1-to-1 linkage to frame `fr2` and match on variable `matchvar`

```
frlink 1:1 matchvar, frame(fr2)
```

Create many-to-1 linkage to frame `fr3`, matching variables `v1` and `v2` in the current frame to variables `x1` and `x2` in frame `fr3`, naming the linkage `lnk`

```
frlink m:1 v1 v2, frame(fr3 x1 x2) generate(lnk)
```

List names of linkages in current frame

```
frlink dir
```

Show details for linkage `lnk`

```
frlink describe lnk
```

Attempt to re-create linkage `lnk` after data have changed

```
frlink rebuild lnk
```

Eliminate linkage `lnk`

```
drop lnk
```

Syntax

Create linkage between current frame and another

```
frlink { 1:1 | m:1 } varlist1 , frame(frame2 [ varlist2 ]) [ generate(linkvar1) ]
```

List names of existing linkages

```
frlink dir
```

List details about existing linkage, and verify it is still valid

```
frlink describe linkvar2
```

Re-create existing linkage when data have changed or frames are renamed

```
frlink rebuild linkvar2 [ , frame(frame3) ]
```

Drop existing linkage (dropping the variable eliminates the linkage)

```
drop linkvar2
```

`1:1` and `m:1` indicate how observations are to be matched.

varlist$_1$ contains the match variables in the current frame, which we will call frame 1.

linkvar$_1$ is the name to be given to the new variable that `frlink` creates. The variable is added to the dataset in frame 1. The variable contains all the information needed to link the frames.

You specify the name for *linkvar*$_1$ using the `generate(`*linkvar*$_1$`)` option, or you let `frlink` name it for you. If `frlink()` chooses the name, the variable is given the same name as *frame*$_2$.

linkvar$_2$ is the name of an existing link variable.

`collect` is allowed with `frlink dir` and `frlink rebuild`; see **[U] 11.1.10 Prefix commands**.

Options

Options are presented under the following headings:

Options for frlink 1:1 and frlink m:1

`frame(`*frame*$_2$ [*varlist*$_2$]`)` specifies the name of the frame, *frame*$_2$, to which a linkage is created and optionally the names of variables in *varlist*$_2$ on which to match. If *varlist*$_2$ is not specified, the match variables are assumed to have the same names in both frames. `frame()` is required.

To create a link to a frame named `counties`, you can type

```
. frlink m:1 countyid, frame(counties)
```

This example omits specification of *varlist*$_2$, and it works when the match variable `countyid` has the same name in both frames. If the variable were named `cntycode`, however, in the other frame, you type

```
. frlink m:1 countyid, frame(counties cntycode)
```

The rule for matching observations is thus that `countyid` in the current frame equals `cntycode` in the other frame.

You can specify multiple match variables when necessary. For example, you want to match on county names in US data. County names repeat across the states, so you match on the combined county and state names by typing

```
. frlink m:1 countyname statename, frame(counties)
```

If the match variables had different names in frame `counties`, such as `county` and `state`, you type

```
. frlink m:1 countyname statename, frame(counties county state)
```

`generate(`*linkvar*$_1$`)` specifies the name of the new variable that will contain all the information needed to link the frames. This variable is added to the dataset in frame 1. This option is rarely used.

If this option is not specified, the link variable will then be named the same as the frame name specified in the `frame()` option.

Options for frlink rebuild

`frame(`*frame*$_3$`)` specifies a frame name that differs from the existing linkage. *frame*$_3$ is the new name of a frame linked by *linkvar*$_2$.

For instance, yesterday, you created a linkage named `george` to the data in the frame named `george` by typing

```
. frlink m:1 countyname statename, frame(george)
```

Today, you loaded the linked data into a frame named `counties`. To rebuild the linkage so that linkage `george` links to the data in frame `counties`, type

```
. frlink rebuild george, frame(counties)
```

If you also wish to rename the linkage to be `counties`, type

```
. rename george counties
```

Then you would have a linkage named `counties` to the data in the frame named `counties`.

Remarks and examples

Remarks are presented under the following headings:

Overview of the frlink command

`frlink 1:1` and `frlink m:1` create linkages between the current frame and another frame you specify. This adds a new variable to the current frame, known as the link variable. You can use the `frget` command to copy variables from the linked frame to the current frame and use the `frval()` function to use the other frame's variables in expressions. You can use the `fralias add` command to define aliases of variables from the linked frame in the current frame. An alias is a reference to a variable in another frame, similar to a copy, but uses very little memory. You cannot modify the observations in a alias variable.

Linkages are said to be named, but the name is in fact the name of the link variable that `frlink` creates.

`frlink dir` lists the names of existing linkages.

`frlink describe` *linkvar* displays details about the specified linkage. It also checks the validity of the link variable and, if there are problems, tells you how to fix it.

`frlink rebuild` *linkvar* re-creates the specified *linkvar*. If *linkvar* is invalid, `frlink rebuild` will fix it.

Type `drop linkvar` to delete linkages.

Everything you need to know about linkages

Here is everything you need to know in outline form:

What follows will turn you into an expert.

1. A linkage connects one frame to another. Here are the advantages.

   Create a linkage and you can access the variables in another frame using the `frval()` function and the commands `frget` and `fralias add`.

1.1 The `frval()` function. You can type

```
. generate rel_income = income / frval(counties, median_income)
```

`frval(counties, median_income)` returns the value of the `median_income` variable in frame `counties`. If the current frame contained data on people and the county frame contained data on counties (linked to with link variable `counties` in the current frame), the above would produce person income divided by the median income of the county in which he or she resides. See `frval()` in **[FN] Programming functions**.

1.2 The `frget` command. You can type

```
(1) . frget median_income, from(counties)
(2) . frget medinc = median_income, from(counties)
(3) . frget median_income pop, from(counties)
(4) . frget median_income pop attr*, from(counties)
(5) . frget median_income pop attr*, from(counties) prefix(c_)
```

and more ...

(1) copies `median_income` from frame `counties` into the data in the current frame.

(2) does the same but names the variable `medinc`.

(3) copies two variables.

(4) copies lots of variables.

(5) copies lots of variables and renames them to start with `c_`.

This is only a smattering of what `frget` can do. See [D] **frget**.

1.3 The `fralias add` command. You can type

```
(1) . fralias add median_income, from(counties)
(2) . fralias add medinc = median_income, from(counties)
(3) . fralias add median_income pop, from(counties)
(4) . fralias add median_income pop attr*, from(counties)
(5) . fralias add median_income pop attr*, from(counties) prefix(c_)
```

and more ...

(1) alias `median_income` from frame `counties` so that you can use its observations in the current frame.

(2) does the same but names the alias variable `medinc`.

(3) aliases two variables.

(4) aliases lots of variables.

(5) aliases lots of variables and renames them to start with `c_`.

This is only a smattering of what `fralias add` can do. See [D] **fralias**.

2. The `frlink` command creates linkages.

`frlink` creates a linkage from the current frame to the frame you specify.

```
. frlink ..., frame(counties)
```

3. Linkages are named.

   The command

   ```
   . frlink ..., frame(counties)
   ```

   creates a linkage named `counties` to the frame named `counties`.

   You can specify option `generate()` to give the linkage a different name. To create a linkage named `c` to the frame `counties`, type

   ```
   . frlink ..., frame(counties) generate(c)
   ```

4. A linkage is a variable added to the data.

   The entire physical manifestation of a linkage is the addition of a single variable to the dataset in the current frame. Typing

   ```
   . frlink ..., frame(counties)
   ```

   adds new variable `counties` to the dataset in the current frame.

   ```
   . frlink ..., frame(counties) generate(c)
   ```

   adds new variable `c` to the dataset in the current frame.

   The added variable is known as the "link variable", or *linkvar*.

5. Drop the link variable, remove the link.

   Because linkages are just a variable, if you drop the variable, you remove the link.

   ```
   . drop counties
   . drop c
   ```

6. Do not modify the contents of the link variable.

   If you modify the link variable's contents, you invalidate the linkage. If you are lucky, the next time you use the `frget` or `fralias add` command or the `frval()` function, they will detect the problem and issue an error. If not, they will simply produce incorrect results.

   ```
   . replace counties = ...          // Do not do this
   . replace c = ...                 // Do not do this
   ```

   If you accidentally modify the link variable's contents, use `frlink rebuild` to repair it.

   ```
   . frlink rebuild counties
   . frlink rebuild c
   ```

7. Linkages are formed based on equality of match variables.

   To construct a link to frame `counties`, type

   ```
   . frlink ..., frame(counties)
   ```

   The complete command would have the dots filled in. Part of what needs to appear in place of the dots are the match variables. A more complete version of the command is

   ```
   . frlink ... countyid, frame(counties)
   ```

   We specified one match variable, `countyid`.

   Linkages are formed by matching observations in the current frame to observations in the other frame when their match variables are equal.

In the example, the match variables are countyid in the current frame and countyid in the county frame. Observations are matched when the countyid variables are equal.

Let's unravel that. The data in the current frame are on people. countyid in the current frame records the county in which each person resides.

Meanwhile, the data in the county frame contains information on counties, such as a county's median income. Variable countyid in this frame records the county each observation describes.

Observations in the two frames are matched when the county in which a person resides equals the county being described. Once we have formed the linkage by typing

```
. frlink ... countyid, frame(counties)
```

if we then type

```
. generate rel_income = income / frval(counties, median_income)
```

we obtain the ratio of each person's income to the median income in the county in which he or she resides.

8. You can specify more than one match variable.

   We just considered the case of one match variable—countyid—in each of the frames:

   ```
   . frlink ... countyid, frame(counties)
   ```

   Let's imagine that instead of containing countyid, the datasets contain countyname. Substituting countyname for countyid might be insufficient to form the desired linkage:

   ```
   . frlink ... countyname, frame(counties)
   ```

   County names in the United States are repeated across states. Monroe County, for instance, exists in Florida, Mississippi, Texas, and other states. To link the frames, we need to match on both county and state names:

   ```
   . frlink ... countyname statename, frame(counties)
   ```

   Because county and state names, taken together, uniquely identify the locations, the order in which we specify them is irrelevant:

   ```
   . frlink ... statename countyname, frame(counties)
   ```

9. Match variables can be named differently in the two frames.

   When we type

   ```
   . frlink ... countyname statename, frame(counties)
   ```

   we are stating the variables countyname and statename appear in both frames. If the names are different in the two frames, specify the names used in the current frame following the frlink command, and specify the names used in the other frame in the frame() option, after the frame's name:

   ```
   . frlink ... countyname statename, frame(counties cnty usstate)
   ```

   countyname and statename are the variable names used in the current frame. The variables corresponding to them in frame counties are named cnty and usstate.

10. Match type: One-to-one or many-to-one matching.

    Consider the linkage created by

    ```
    . frlink ... countyid, frame(counties)
    ```

    The current frame contains data on persons, and the other frame—`counties`—contains data on counties.

    All that is needed to turn the above into a complete command is to replace the dots with a match type, which can be `1:1` or `m:1`. In this case, the match type should be `m:1`, and the full command is

    ```
    . frlink m:1 countyid, frame(counties)
    ```

    `m:1` stands for many-to-one matching. `m:1` means that is okay if more than one observation in the current frame matches the same observation in the other frame. We specify `m:1` because it is possible that multiple people in the current frame reside in the same county. If five people live in county 207, all five will match to the observation in frame `counties` that describes county 207.

    The alternative `1:1` means that at most one observation in the current frame can match an observation in the other frame. Specifying `1:1` would be appropriate for matching person data in the current frame with more data on him or her in the other frame. If persons were to be matched on `personid` and if the other frame were named `person2`, we type

    ```
    . frlink 1:1 personid, frame(person2)
    ```

    Matched would be persons in the current frame who also appeared in the second frame.

    If you think about it, `1:1` is a special case of `m:1`. `1:1` means at most one observation matches. `m:1` means one or more observations match. This means that, if

    ```
    . frlink 1:1 personid, frame(morepersons)
    ```

    forms the linkage you want, so will

    ```
    . frlink m:1 personid, frame(morepersons)
    ```

    So why specify `1:1`? We specify `1:1` so that `frlink` can issue an error message if the result is not `1:1`. When matching people's data to more data on the same people, if two people in the first frame matched the same observation in the second, that means

    P1. there is an error in the first dataset: the same person appears more than once in it; or

    P2. there is an error in variable `personid` in the first dataset: the `personid` variable contains the wrong value; or

    P3. we are not thinking clearly and should have specified `m:1` instead of `1:1`.

    You specify `1:1` so that the software can flag situations where the reality is different from your expectations. Then you fix your data or your thinking.

11. Linking can result in unmatched observations.

    Imagine that you have successfully executed

    ```
    . frlink m:1 countyid, frame(counties)
    ```

The result will be that each observation in the current frame will be matched or unmatched. Observations in the current frame are matched when the values of `countyid` are found in frame `counties`. The remaining observations, if any, are unmatched. Unmatched observations are not an error; they are a characteristic and perhaps a shortcoming of your datasets.

`frlink` tells you how many unmatched observations there are when you create the linkage.

Function `frval()` will subsequently return missing values for the unmatched observations. If you type

```
. generate relative_income = income/frval(linkvar, median_income)
```

variable `relative_income` would be missing (`.`) for the unmatched observations, the same as if unmatched observations were matched but contained `median_income==.`.

`frget` and alias variables created by `fralias add` behave similarly. `frget` sets the unmatched observations equal to missing in the copied variable. Alias variables return missing values for unmatched observations.

```
. frget median_income, from(counties)
. fralias add median_income, from(counties) prefix(a_)
```

In addition, the link variable in the current frame contains missing values for the unmatched observations. This is useful. How many observations in the current frame are unmatched? If you do not remember, type

```
. count if counties==.
```

You can look at the data for the unmatched observations.

```
. browse if counties==.
```

You can analyze the unmatched data.

```
. summarize if counties==.
```

If observations will be useful to you only when they are matched with county data, you can keep just the matched data by typing

```
. keep if counties!=.
```

12. Linkages are directional.

    We say that we link the current frame to another frame, but it's really the other way around. Data flow to the current frame from the other frame. If you have created the linkage

    ```
    . frlink m:1 countyid, frame(counties)
    ```

    then you can access data in frame `counties` from the current frame, but you cannot access data in the current frame from frame `counties`.

13. How to create nested linkages.

    Consider separate frames containing data on students, the schools they attend, and the counties in which the schools are located. Here is the setup:

    Current frame: `students.dta` containing variables for each student's ID, the ID of the schools he or she attends, and student characteristics.

    Frame `schools`: `schools.dta` containing each school's ID, the ID of the counties in which the schools are located, and school characteristics.

Frame counties: us_counties.dta containing each county's ID and county characteristics.

Here is how you load the datasets into the frames:

```
. frame create schools
. frame create counties
. use students
. frame schools: use schools
. frame counties: use us_counties
```

Here is how you link the frames:

```
. frlink m:1 schoolid, frame(schools)
. frget countyid, from(schools)
. frlink m:1 countyid, frame(counties)
```

The first command links students with the schools they attend.

The second command copies variable countyid from frame schools to the current frame.

The third command links students with the counties in which their schools are located.

The command that copied countyid into the current frame was necessary so that the students in the current frame could be linked to the county frame.

Said generically, if you have data in frames *A*, *B*, and *C*, you link frame *A* to *B* and link frame *A* to *C* to access all the data from *A*.

Said negatively, linkages are not transitive. Linking frame *A* to *B* and *B* to *C* is not sufficient to allow frame *A* to access all the data.

14. Saving and using linked frames.

You have created students-linked-to-county data:

```
. use students
. frame create counties
. frame counties: use us_counties
. frlink m:1 countyid, frame(counties)
```

To save the datasets so that you can use them later, you need only type

```
. save students, replace
```

It is necessary to save students.dta because it has a new variable in it, namely, the linkage variable counties. It is not necessary to save us_counties.dta because it has not changed.

That said, you might still wish to save both files:

```
. save students, replace
. frame counties: save us_counties, replace
```

The data in frame counties were not changed, but the sort order of the data changed. Linking sorts the linked-to frame on its match variables. We recommend you save both datasets.

To later load the data, you type

```
. use students
. frame create counties
. frame counties: use us_counties
```

You might want to put these lines in a do-file. You could call it `usestudents.do`. Then, whenever you wanted to load the data, all you need to do is type

```
. do usestudents
```

15. Do's and don'ts.

We start with the don'ts. There are only three:

Do not modify the contents of the link variable,

… but if you do, use `frlink rebuild` to fix it.

Do not rename the match variables in either frame,

… but if you do, drop the link variable, and use `frlink m:1` or `1:1` to link the frames again.

Do not drop the match variables from either frame,

… and if you do, we cannot help you.

Everything else is a *do*, but they come in two flavors. The first is *do* without qualifications. The second is also a *do*, but do it only if you follow it by typing `frlink rebuild`.

Here are the *do's* without qualifications:

Do drop the link variable. That's how you eliminate the link.

Do rename the link variable.

Do drop observations in the current frame.

Do add new variables in either frame.

Do modify or rename variables in either frame, with the exception of the link and the match variables.

And here are the *do's* with qualification, which is always the same: Type `frlink rebuild` afterward.

Do rebuild after adding observations in either or both frames.

Do rebuild after dropping observations in the linked frame.

Do rebuild after modifying the contents of the match variables in either or both frames.

And remember a rule that always applies:

It is always safe to type `frlink rebuild`.

If there is no problem, it will do nothing.

If there is a problem, it will fix it unless it cannot,

… then it explains why and do nothing to your data.

You are now an expert on linked frames.

Example 1: A typical m:1 linkage

File `persons.dta` contains data on people. Among its variables is `countyid`, containing the county code where each person resides.

File txcounty.dta contains data on Texas counties. Among its variables is countyid, the county code for the county that each observation describes.

Here is how we load and link the datasets:

```
. use https://www.stata-press.com/data/r19/persons
. frame create txcounty
. frame txcounty: use https://www.stata-press.com/data/r19/txcounty
(Median income in Texas counties)
. frlink m:1 countyid, frame(txcounty)
(all observations in frame default matched)
```

Linkages are for situations where you want to analyze the data in the current frame using variables from both frames.

Below, we create new variable relative_income in the current frame equal to income (in the current frame) divided by median_income (from the county frame):

```
. generate relative_income = income / frval(txcounty, median_income)
. summarize relative_income
    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
relative_i~e |         20    .5501545    .1090887    .352133   .7038001
```

If we wanted to use median_income from the county frame in a linear regression, we would use the frget command to add median_income to the current frame's data:

```
. frget median_income, from(txcounty)
. regress income ... median_income ...
```

We will not do that because persons.dta contains fictional values and is not worth the bother. But realize what would be possible if these datasets were real and contained more variables:

Get a variable:

```
frget median_income, from(txcounty)
```

Get a variable, but change its name:

```
frget medinc = median_income, from(txcounty)
```

Get a lot of variables:

```
frget median* nbus-pop, from(txcounty)
```

Get a lot of variables, but change their names to begin with c_:

```
frget median* nbus-pop, prefix(c_) from(txcounty)
```

See [D] **frget**.

A more memory-efficient option is to use fralias add to create aliases instead of copies. See [D] **fralias**.

How link variables work

frlink performs two actions when it creates a link:

1. It adds the link variable to the dataset in the current frame.
2. It sorts the dataset in the other frame by its match variables.

In the example above, this means that

1. `frlink` adds variable `txcounty` to the data in the current frame.
2. `frlink` sorts the data in frame `txcounty` by `countyid`. (It literally executes `frame txcounty: sort countyid`.)

Look at variable `txcounty` in the first observations of `persons.dta` in the current frame:

```
. list in 1/5
```

	personid	countyid	income	txcounty	relati~e
1.	1	5	30818	5	.7038001
2.	2	3	30752	3	.4225046
3.	3	2	29673	2	.5230381
4.	4	3	32115	3	.441231
5.	5	2	31189	2	.5497603

Each observation of variable `txcounty` contains the observation number in frame `txcounty` that matches the current observation. The above list says that

obs. 5 of frame `txcounty` matches obs. 1 of the current frame

obs. 3 of frame `txcounty` matches obs. 2 of the current frame

obs. 2 of frame `txcounty` matches obs. 3 of the current frame

obs. 3 of frame `txcounty` matches obs. 4 of the current frame

obs. 2 of frame `txcounty` matches obs. 5 of the current frame

… assuming the data in frame `txcounty` are sorted on `countyid`

Frame `txcounty` is the other frame. It is the other frame that must be sorted, not the data in the current frame.

Even so, the assumption is iffy. It is true after `frlink` creates the linkage because `frlink` itself sorts the data. And `fralias add`, `frget`, and `frval()` check the sort order before using the other frame's data so that accidents do not happen.

The only way things can go wrong are 1) if you change the contents of the link variable `txcounty` or 2) you drop or modify the match variable `countyid`. So do not do that.

Advanced examples

Example 1 showed you how linkages are usually used. We linked person data to county data. We could show you another example that links student data to school data and student data to county data, but it amounts to nothing more than example 1, done twice.

We have two more examples to show you, but we admit that they are advanced and abstruse.

The first is an example in which linkage shines, but the solution is seldom useful beyond the particular example shown.

The second concerns 1:1 linkages. If 1:1 is appropriate for your problem, you probably want to merge the datasets, not link them. You probably want to use merge, not frlink. On occasion, however, a situation arises where linkage is a better solution. We show you one and provide guidelines on how to identify other such situations.

Example 2: A complex m:1 linkage

We have a dataset on families and the file is named, naturally enough, family.dta. The dataset contains information on variables of interest, as all datasets do, but that is not what makes this dataset interesting, so the variables are simply named x1, x2, ..., x5. What makes this dataset interesting is that it contains observations on related adult people. It contains adult children, parents, and grandparents.

Such data are notoriously difficult to process and analyze.

In the dataset, every person is identified by a person ID, called a "pid". The data also contain the variables pid_m and pid_f, which are the pids for the person's mother and father, if they too are in the data. The oldest generation in the data has pid_m==. and pid_f==.

One person in the data is person number 14982. Here are the values of ID variables for 14982:

```
. list pid* if pid==14982
```

	pid	pid_m	pid_f
8.	14982	695966	933335

Variables pid_m and pid_f are the IDs of 14982's mother and father. The mother is 695966 and the father, 933335.

Here are the recorded ID variables for 695966, 14982's mother:

```
. list pid* if pid==695966
```

	pid	pid_m	pid_f
431.	695966	186484	238126

14982's maternal grandmother is 186484 and maternal grandfather, 238126.

Let's stay with the maternal side of the family. Here are the ID variables for 186484, 14982's maternal grandmother:

```
. list pid* if pid==186484
```

	pid	pid_m	pid_f
100.	186484	.	.

The grandmother's variables have missing values for her mother's and father's ID, so we cannot continue back further. Nonetheless, there are other people in this dataset just like 14982, people on whom we have their data, their parents' data, and their parents' parents' data.

frlink can link the data so that we have access to all of them. To do that, we will create six linkages, named

linkage name	meaning linkage to
f	father
m	mother
mm	mother's mother
mf	mother's father
fm	father's mother
ff	father's father

Once we have these six linkages, we will be able to access variables for the person, his or her parents, and their parents. We will be able to do that using the frval() function or the fralias add and frget commands.

If we wanted to access x1 using function frval(), we would do so with the following:

value of x1 desired	type
own value	x1
mother's value	frval(m, x1)
father's value	frval(f, x1)
mother's mother's value	frval(mm, x1)
mother's father's value	frval(mf, x1)
father's mother's value	frval(fm, x1)
father's father's value	frval(ff, x1)

If we wanted to copy all five variables of interest to the current frame, prefixed by their relationship, we would do so with the following:

value of x1-x5 desired	type
own value	x1-x5
mother's variables	frget x1-x5, from(m) prefix(m)
father's variables	frget x1-x5, from(f) prefix(f)
mother's mother's variables	frget x1-x5, from(mm) prefix(mm)
mother's father's variables	frget x1-x5, from(mf) prefix(mf)
father's mother's variables	frget x1-x5, from(fm) prefix(fm)
father's father's variables	frget x1-x5, from(ff) prefix(ff)

Instead, we can alias all five variables of interest to the current frame, prefixed by their relationship, with the following:

value of x1–x5 desired	type
own value	x1-x5
mother's variables	fralias add x1-x5, from(m) prefix(m)
father's variables	fralias add x1-x5, from(f) prefix(f)
mother's mother's variables	fralias add x1-x5, from(mm) prefix(mm)
mother's father's variables	fralias add x1-x5, from(mf) prefix(mf)
father's mother's variables	fralias add x1-x5, from(fm) prefix(fm)
father's father's variables	fralias add x1-x5, from(ff) prefix(ff)

If we combined all 5 variables of interest from all 7 sources, we would have a total of 35 variables of interest. We could form that dataset by typing just six commands. Before we can do any of this, we must build the linkages.

To build them, we start in the usual way. We load the data of interest into the current frame and load into the other frame the data we want to link:

```
. clear all
. use https://www.stata-press.com/data/r19/family
(Fictional family linkage data)
. frame create family
. frame family: use https://www.stata-press.com/data/r19/family // yes, the same data
(Fictional family linkage data)
```

We are in fact going to link family.dta to itself, but frlink requires that linkages be made from the current frame to the other frame. Nonetheless, we will be able to create all six linkages to that single frame.

To create the first two linkages, we type

```
. frlink m:1 pid_m, frame(family pid) generate(m)
(355 observations in frame default unmatched)
. frlink m:1 pid_f, frame(family pid) generate(f)
(355 observations in frame default unmatched)
```

Because we are linking people to people, your natural inclination might be that the matching needs to be 1:1. That was our inclination too, but when we tried, frlink complained that the data were m:1 and refused. It took us a minute to realize why. Some of the people in the data have the same mother or father.

We have shown you the commands to build the first two linkages. Four remain to be built. What is different about these four is that the current frame does not contain the necessary match variable. Think about forming the mm linkage, which is the maternal grandmother of a person in the current frame. We need a variable containing the ID of the current person's mother's mother or frval(m, pid_m). We could call the variable pid_mm, and construct it and the related match variables by typing

```
. generate pid_mm = frval(m, pid_m)
(539 missing values generated)
. generate pid_mf = frval(m, pid_f)
(539 missing values generated)
. generate pid_fm = frval(f, pid_m)
(539 missing values generated)
. generate pid_ff = frval(f, pid_f)
(539 missing values generated)
```

Alternatively, we could have obtained them by using the `frget` command:

```
frget pid_mm = pid_m, from(m)
frget pid_mf = pid_f, from(m)
frget pid_fm = pid_m, from(f)
frget pid_ff = pid_f, from(f)
```

It does not matter which we use.

Once we have the match variables, we can form the linkages:

```
. frlink m:1 pid_mm, frame(family pid) generate(mm)
(539 observations in frame default unmatched)
. frlink m:1 pid_mf, frame(family pid) generate(mf)
(539 observations in frame default unmatched)
. frlink m:1 pid_fm, frame(family pid) generate(fm)
(539 observations in frame default unmatched)
. frlink m:1 pid_ff, frame(family pid) generate(ff)
(539 observations in frame default unmatched)
```

At this point, we are basically done. We are interested, however, in the sample of people for whom data on their parents and grandparents are available. We can drop the other people from the data in the current frame.

```
. drop if pid_m ==. | pid_f ==.
(355 observations deleted)
. drop if pid_mm==. | pid_mf==.
(184 observations deleted)
. drop if pid_fm==. | pid_ff==.
(0 observations deleted)
. count                                 // number of observations remaining
  100
```

We now have our data ready for analysis.

What are the chances that an even 100 people would be left? They would be nil if this were real data. We manufactured these data, however, so there is no reason to continue to analyze variables `x1` through `x5`. They contain fictional values, and random.

Example 3: A 1:1 linkage, a simple solution to a hard problem

Most `1:1` cases are better handled by `merge`. Here is an exception.

You are working with hospital patient data, file `discharge1.dta`. The file contains variable `patientid` among other variables, and you receive additional data on the same patients, file `discharge2.dta`. Loading the two datasets into separate frames and linking them is easy to do.

```
. use https://www.stata-press.com/data/r19/discharge1, clear
. frame create discharge2
. frame discharge2: use https://www.stata-press.com/data/r19/discharge2
. frlink 1:1 patientid, frame(discharge2)
```

But should we be doing this at all? Would it not be better to merge `discharge1.dta` with `discharge2.dta`? It usually would be. It would be if you received the following note from George:

> Kathy: Here are new data on the 1,980 patients. The data contain the five variables that were previously omitted. – George.

`merge` will allow you to add these five new variables. Use it.

The note you received from George, however, reads

> Kathy: Here are the data on the 1,980 patients. You're not going to believe this, but even though they said there are five values that needed to be updated, they sent all the data again! Verify their claim, and tell me which variables they updated. – George.

This is a case for linking because you will not have to rename the 19 variables so that you can verify their claim. The link solution of handling George's request is easier:

```
. use https://www.stata-press.com/data/r19/discharge1, clear
(Fictional WA hospital discharges)
. frame create discharge2
. frame discharge2: use https://www.stata-press.com/data/r19/discharge2
(Fictional WA hospital discharges)
. frlink 1:1 patientid, frame(discharge2)
(all observations in frame default matched)
. foreach v of varlist patientid-proc3code {
  2.     quietly count if `v' != frval(discharge2, `v', discharge2)
  3.     if (r(N)!=0) {
  4.       display "`v': " r(N) " value(s) changed"
  5.     }
  6. }
sex: 1 value(s) changed
los: 1 value(s) changed
billed: 1 value(s) changed
diag1code: 1 value(s) changed
diag2code: 1 value(s) changed
```

It turns out that the updated data are just as it was represented to be.

These data had two features that made them a candidate for linking rather than merging:

1. The sample of interest was the observations in the original data, the data in the current frame, and
2. lots of variables in the two datasets had the same names, and we were interested in both sets of values.

Let's now think about other examples. Only some `1:1` problems will have feature 1. `1:1` matches in which you will subsequently analyze the merged data—`_merge==3` in `merge` speak—will all have feature 1.

Feature 2 arises less often. In the example, the new data updated the old. Linkages make comparing values easier when the names are the same. Linkages in general make it easier when variable names are the same, even when there is no reason to compare them. Imagine that both datasets contain a variable called `income`, but they are different measures of income. In the combined result, you want them both,

so you need to rename one of them. Now imagine that there are hundreds of variables and a handful share the same names across datasets even though they contain different concepts of whatever is being measured. Linkages make renaming them easy.

First, link the data:

```
. frlink personid, frame(newdata)
```

Then, try to copy all the variables:

```
. frget *, from(newdata)
```

The command will either work or tell you the variables that have the same name in both frames. Imagine that `frget` lists `income` and six other variables. You want to copy `income`, so you rename the variable:

```
. frame newdata: rename income farmincome
```

Now try again:

```
. frget *, from(newdata)
```

Of course the command does nothing but repeat the six variables that still have the same names in both frames. You review the list one last time and decide that you still do not care about those six variables. Then you type

```
. frget *, from(newdata) exclude(‘r(dups)’)
```

This time it works! When variables have the same name, in addition to listing them, `frget` saves their names in `r(dups)`. That is why we typed `frget *, from(newdata)` when we knew we had not yet resolved all the duplicate names. We wanted `frget` to set `r(dups)` so that we could next tell `frget` to copy all the variables, except `exclude(‘r(dups)’)`.

Now that we have gotten the variables of interest, we break the link:

```
. drop newdata
. frame drop newdata
```

The data in memory are now the same data that we could have coaxed `merge` into producing had we done everything right.

Stored results

`frlink m:1` and `frlink 1:1` store the following in `r()`

Scalars

`r(unmatched)`	# of observations in the current frame unable to be matched

`frlink dir` stores the following in `r()`:

Scalars

`r(n_vars)`	# of link variables

Macros

`r(vars)`	space-separated list of link-variable names

`frlink describe` stores nothing in `r()`.

`frlink rebuild` stores the following in `r()`:

Scalars

`r(unmatched)`	# of observations in the current frame unable to be matched

Also see

[D] **fralias** — Alias variables from linked frames

[D] **frget** — Copy variables from linked frame

[D] **frames intro** — Introduction to frames

[D] **merge** — Merge datasets