The Stata Journal (*yyyy*) *vv*, Number *ii*, pp. 1–10

A Workflow for Changing Working Directories and Managing Projects

Draft – Comments Welcome

Scott Long
Indiana University
Bloomington, IN/USA
jslong@iu.edu

2017-09-11

Abstract. An effective way to manage multiple projects is to use a different working directory for each project. This article describes the `makecd` command which creates commands that quickly change the working directory. These commands can also create global macros that customize the environment for each project and automatically run commands after changing the working directory. The `listcd` command provides a point and click interface for changing working directories.

Acknowledgements Trent Mize, Nic Bussberg, and Long Doan gave me valuable suggestions.

Keywords: st0001, `workingdir` package, `dropcd`, `listcd`, `makecd`, working directory, managing projects

Note To install the commands in this article, `search workingdir` and follow the links to install the commands.

Using a different working directory for each project is critical for an effective workflow. Indeed, changing the working directory is often the first thing people do after starting Stata. Sadly, changing working directories in Stata is like having a sports car that can only be started by opening the trunk to insert the key. You can run `cd` after entering a sometimes lengthy path name or use the menu to click through your directory structure. On a Mac, Stata opens in the working directory when you last exited Stata. For Windows and Unix, the default working directory is selected during installation with no simple way to change the default. The `workingdir` package contains commands that simplify changing working directories and let you create an environment associated with each working directory.

The `workingdir` package is organized around the idea of a project. A project is a distinct activity such as a writing a paper, taking a class, or developing a Stata command. The `makecd` command automatically creates (i.e., makes) commands, called `cd`*project* commands that let you change the working directory by entering a simple command name, rather than typing a lengthy path name or clicking through your directory structure. For example, you can easily create the command `cdpaper` that changes to the directory for a paper you are writing, while the command `cdclass`

© *yyyy* StataCorp LP

changes to the directory for your class. The `listcd` command lists your `cd`*project* commands along with information describing each project. Clicking on a command name runs the command. For many users being able to easily change working directories is the only feature of the `workingdir` package that will be used. However, for those who want more control over the way work is completed for each project, you can define global macros that specify paths where datasets are located and automatically run commands after changing the working directory. The `workingdir` package includes two primary commands:

`makecd` creates `cd`*project* commands that change the working directory, create global macros with information about each project, and optionally run commands when projects are switched. These commands are saved in your PERSONAL directory.

`listcd` displays an annotated list of `cd`*project* commands. You can click on a command name to change the working directories or run a `cd`*project* command from the Command window. It also provides a simple way to delete `cd`*project* commands.

Non-programmers can use these commands without knowing anything about Stata programming. Programmers can customize the commands to add other features. These commands were inspired by the `fastcd` package by Winters (2002), Baum's ideas about using globals macros create an environment for a research project (Baum 2016, 78-79), and Long's simple commands for changing the working directories (Long 2009, 111-112,117-118).

I begin by illustrating the most basic features for changing the working directory and show how these can be used to change Stata's initial working directory. The next section explains more advanced features that allow you to create robust and portable do-files that use global macros to specify the location of datasets and that can automatically execute commands when you change projects.

1 Basic features for changing projects

A simple example shows you how to create commands to move among directories. Suppose that a paper your are writing uses working directory `d:/dropbox/active/paper/work`, while a class you are taking uses `d:/dropbox/active/class/work`.[1] To create the command `cdpaper` to change the working directory for this paper, I start by changing my working directory to that location using either the *Change working directory* dialog or the `cd` command. For example,

```
. cd d:/dropbox/active/paper/work
d:/dropbox/active/paper/work
```

Then I create the `cdpaper` command:

```
. makecd paper
command cdpaper saved in your PERSONAL directory
```

1. I use forward slashes rather than a backslashes since these works in all operating systems.

`makecd` saves the file `cdpaper.ado` in my PERSONAL directory. To find out where that directory is, run `sysdir`. Next, I change to the working directory for my class and create the `cdclass` command:

```
. cd d:/dropbox/active/class/work
d:/dropbox/active/class/work

. makecd class
command cdclass saved in your PERSONAL directory
```

I can now use these commands to move between directories. To change to the directory for my paper,

```
. cdpaper
d:/dropbox/active/paper/work
```

where the command echoes the working directory that was set. To change to the directory for my class,

```
. cdclass
d:/dropbox/active/class/work
```

I can create `cd`*project* commands for all of my projects. To list these commands,

```
. listcd
      cdcda - d:/dropbox/active/cda2017/work/
    cdclass - d:/dropbox/class/work/
  cdcouples - d:/boxsync/kinsey/couples/work/
     cddesk - c:/users/jslong/desktop/statawork/
    cdpaper - d:/dropbox/active/paper/work/
    cdstart - d:/statastart/
 cdworkflow - d:/dropbox/active/workflow/work/
```

Command names on the left are shown in blue. If I click on a name, the command is executed. (This feature is not available when running Stata in console mode.) I can also run commands from the Command window or include them in do-files. As the number of `cd`*project* commands increases, scanning the list of directories becomes awkward. A more efficient approach is to use the `note()` option when creating a command:

```
. makecd paper, note(Groups paper with SAM) replace
command cdpaper saved in your PERSONAL directory
```

After creating several commands using notes,

```
. listcd
      cdcda - Stat 503 CDA
    cdclass - d:/dropbox/class/work/
  cdcouples - Couples 3 paper
     cddesk - Desktop
    cdpaper - Groups paper with SAM
    cdstart - d:/statastart/
 cdworkflow - Workflow 2nd
```

For commands that were not created with the `note()` option, the path is shown. Even when notes are available, you can list the directories by running `listcd, dir`.

You can delete a `cd`*project* command by deleting `cd`*project*`.ado` in PERSONAL. Easier yet,

```
. listcd, delete

      cdcda - Stat 503 CDA (delete)
    cdclass - d:/dropbox/class/work/ (delete)
  cdcouples - Couples 3 paper (delete)
     cddesk - Desktop (delete)
    cdpaper - Groups paper with SAM (delete)
    cdstart - d:/statastart/ (delete)
 cdworkflow - Workflow 2nd (delete)
```

and click on `delete` which is shown in blue.

Controlling the starting working directory

There is no easy way to change the working directory in which Stata starts, hereafter referred to as the *starting directory*. One solution is to run `listcd` as soon as Stata starts and click on the project where you want to start. Alternatively, you can modify your `profile.do` which is run automatically each time Stata is started. To find where `profile.do` is located, run

```
findfile profile.do, path(STATA;BASE;SITE;PERSONAL;PLUS)
```

If the file is not found, you need to create it. To determine the best location for the file, check *Stata Installation Guide* or *Getting Stata with Stata* for your operating system. You can find these guides by running `help getting started`.

You could add a `cd` command to `profile.do` to set the starting directory. For example,

```
cd d:/statastart
```

This works, but if you want to change the starting directory, you must edit `profile.do`. Since I want Stata to open in my most active project, and since that project changes frequently, editing `profile.do` is cumbersome. An easier solution is to use `makecd` to create the command `cdstart` to change to the working directory you want to start in. Then, add `cdstart` to `profile.do` so that it is run each time Stata starts. To change the startup directory, simply replace `cdstart` with a version that opens up the directory I want. For example, to make `d:/statastart` the starting directory:

```
cd d:/statastart
makecd start, replace
```

If I am working on a paper whose working directory is set with `cdpaper`, I run:

```
cdpaper
makecd start, replace
```

The next time I start Stata, it opens in the directory for my paper.

Even if Stata opens the the directory where I am usually working, I often work on other projects. To make it easy to change directories immediately after I start Stata, I add `listcd` to `profile.do`:

```
cdstart
listcd
```

`cdstart` opens in my preferred working directory and then `listcd` shows me other locations I can chose. To prevent an error if the `workingdir` package was not installed or if I haven't created `cdstart`, I can use the more robust commands:

```
capture noisily cdstart
capture noisily listcd
```

This will capture errors without ending the do-file, but also shows the output if an error does not occur.

2 Advanced features for managing projects

`cd`*project* commands can set global macros with paths where datasets are located (section 2.1). While I prefer to keep my datasets in the working directory, if you keep datasets in locations other than your working directory, these globals let you create robust do-files. You can also have a `cd`*project* command automatically run a command when you change projects (section 2.2). While I don't use these features often, for some projects they are very useful. Feel free to skip these sections if these are not features you need.

2.1 Robust specifications of data paths

The `use` command loads a dataset from the current working directory unless a path has been specified. This path can be on your computer, on a LAN, or an URL. To make do-files portable, you should not hard code these paths since they might not be valid on another computer. For example, if your do-file includes `use d:/datasets/binlfp4, clear`, it will not run on a computer with a different directory structure. Accordingly, I prefer to keep datasets in the working directory and use the robust command `use binlfp4, clear`. Sometimes, however, it is necessary or convenient to have datasets in other locations. For example, I might access data over the web or share datasets with collaborators on a shared directory on a LAN. Or, I might prefer to have datasets located someplace other than the working directory. If this is the case, you can create robust do-files by using a global macro created outside of the do-file that has the path where datasets are located. `makecd` can create `cd`*project* commands that define global macros with data paths, where each project can have different paths.

An example illustrates how to use globals that contain paths. I create a global with the path:

```
global S_cddata "d:/datasets/"
```

The global is created outside of my do-file since I want the do-file to run *without changes* on other computers. My do-file loads a dataset with the command

```
use "${S_cddata}binlfp4", clear
```

When the global `S_cddata` is expanded, the command is interpreted as

```
use "d:/datasets/binlfp4", clear
```

If the global was not defined, the command is interpreted as

```
use "binlfp4", clear
```

and Stata would look for the data in the working directory. When I work on a computer with a different file structure, my do-file will run if I first create the global `S_cddata` with the correct path for that computer. For example,

```
global S_cddata "c:/users/jslong/documents/datasets/"
```

The ending slash in the path is important. If the slash was missing, I would need to load the dataset with the command

```
use "${S_cddata}/binlfp4", clear
```

where I added a `/` before the dataset name. While this works, an error occurs if the global is not defined since the command is interpreted as

```
use "/binlfp4", clear
```

The `makecd` command lets your make `cd`*project* commands that define three global macros containing data locations. The `data(`*path*`)` option creates the global `S_cddata` with the first data path. For example,

```
makecd paper, data(d:/datasets/source)
```

`makecd` automatically adds an ending slash and converts `\` to `/` since slashes work with all operating systems while back slashes only work for Windows. The `data2(`*path*`)` option adds a second data path saved in the global `S_cddata2`. For example,

```
makecd paper, data(d:/datasets/source) data2(d:/datasets/derived)
```

Two data paths would be useful if you keep source datasets in one directory and datasets you create in another. You can create the global `S_cdurl` with a web address by using the `url(`*address*`)` option. For example,

```
makecd paper, url(http://www.indiana.edu/~jslsoc/stata/spex_data)
```

You can add all of the paths at the same time. For example,

```
makecd paper, data(d:/datasets/source) data2(d:/datasets/derived) ///
    url(http://www.indiana.edu/~jslsoc/stata/spex_data) replace
```

If you do not want `makecd` to add an ending slash, include the `noslash` option.

After running a `cd`*project* that includes data locations, a do-file can use or save data from multiple locations. For example,

```
use "${S_cddata}mrozsource", clear
  (output omitted)
save "${S_cddata2}binlfp5", replace
  (output omitted)
use ${S_cdurl}nomocc4, clear
  (output omitted)
use "couart4", clear // load data from working directory
  (output omitted)
```

Quotes are included in case a path contains a space.

2.2 Further customization of your research environment

The `autorun(`*command*`)` option specifies a command that is run by `cd`*project* after the working directory is changed and the global macros are defined. For example, to list the datasets in the working directory:

```
makecd paper, auto(dir *.dta)
```

When I run `cdpaper`,

```
. cdpaper
d:/dropbox/active/paper/work
. dir *.dta
 338.2M  11/16/16 14:36  34802-0001-Data-compressed.dta
  64.4M  11/16/16 14:34  gss2015-extract01.dta
```

Or, I could automatically load a dataset:

```
makecd paper, auto(use 34802-0001-Data-compressed, clear)
```

Then,

```
. cdpaper
d:/dropbox/active/paper/work
. use 34802-0001-Data-compressed, clear
(General Social Survey, 1972-2012 [Cumulative File])
```

where `cdpaper` ran `use 34802-0001-Data-compressed, clear`. If you want to automatically run more than one command, create a do-file in the project's working directory, say `paper-setup.do`, with the commands you want to run to initiate the project. For example,

```
makecd paper, autorun(do paper-setup)
```

Finally, you can create another global macro with another data path or other information that I have not anticipated. Option `user(`*string*`)` creates the global `S_cduser`

that contains a string. Unlike the `data()`, `data2()`, and `url()` options, an ending slash is not automatically added to the string. If you want `S_cduser` to contain a path, be sure to include and ending slash.

3 Command syntax for workflow commands

The syntax for each command is given, followed by a brief discussion of Stata programming features used by these commands. The `dropcd` is a utility for deleting `cd`*project* commands if you do not want to use the `listcd` interface.

3.1 makecd: creating cdproject commands

`makecd` creates a `cd`*project*`.ado` file that is saved in the PERSONAL directory. When no options are specified, the only thing that the `cd`*project* command does is change the working directory to the directory that was active when `makecd` was run. The syntax is:

`makecd` *project* [`,` `note(`*string*`)` `data(`*data-path*`)` `data2(`*data-path*`)` `url(`*url*`)`
`user(`*string*`)` `autorun(`*command*`)` `details` `noslash` `replace`]

where *project* is the mnemonic for your project. Commands created by `makecd` begin with `cd`, but you do not need to include `cd` in *project*.

Options

`note(`*string*`)` describes the project. Notes are shown by `listcd` to help you remember what each command does.

`data(`*data-path*`)` is a path where datasets are located which is stored in the global macro `S_cddata`, with an ending `/` added to the path unless the `noslash` option is used. You can load a dataset from this path with `use ${S_cddata}`*filename* or save a dataset with `save ${S_cddata}`*filename*.

`data2(`*data-path*`)` is a second path where datasets are located which is stored in the global `S_cddata2`. An ending `/` is added to the path unless the `noslash` option is used. You can load a dataset from this path with `use ${S_cddata2}`*filename* or save a dataset with `save ${S_cddata2}`*filename*.

`url(`*web-address*`)` is a web address which is stored in the global `S_cdurl`. An ending `/` is added unless the `noslash` option is used. You can load datasets with `use ${S_cdurl}`*filename*.

`user(`*string*`)` is any string, such as another data directory or URL. The string is stored in the global `S_cduser`. No ending slash is added.

`autorun(`*command*`)` is a command to be run by `cd`*project* after the the global macros

are created and the working directory is changed. Examples are `auto(dir *.dta)`) and `auto(do project-startup)`.

`details` lists the globals created when `cd`*project* is run.

`noslash` prevents an ending `/` from being added to the data path or URL.

`replace` overwrite `cd`*project*`.ado` if it exists.

Globals created by `cd`*project* commands are *not* removed by `macro drop _all`. To remove them you must run `macro drop` *name*, such as `macro drop S_cddata` or `macro drop S_cd*`.

3.2 The listcd command

`listcd` lists the `cd`*project* commands in PERSONAL along with the note or working directory for each project. If you click on the name of a command, which is shown in blue, the command is executed. The syntax is:

`listcd` [`,` `directories` `details` `delete`]

If the working directory for a command is not found, a warning is given.

Options

`details` displays the `S_cd` globals created by each command.

`directories` displays the directory being set even if a note is defined.

`delete` displays `(delete)` in blue; if you click on `delete`, the ado file for that command is deleted.

3.3 The dropcd command

`dropcd` delete `cd`*project*`.ado` files in PERSONAL.

`dropcd cd`*project* | `_all`

The file `cd`*project*.ado is deleted from the PERSONAL directory.

3.4 Programming features used by the workflow package

The `workingdir` commands depends on several features of Stata.

Dynamically creating ado files Non-programmers are unlikely to learn how to program simply to make it easier to change working directories. Programmers are unlikely to want to write such mundane commands. `makecd` uses `file write` to create

`cd`*project*`.ado` with the commands for changing the work directory, creating globals, and automatically running commands.

The PERSONAL directory The `cd`*project*`.ado` files are saved in the PERSONAL directory so that they will run regardless of your current working directory. If PERSONAL is not defined or the directory specified in PERSONAL does not exist, `makecd` exists with an error. Run `sysdir` to determine where your PERSONAL is located. If no directory is listed or it points to a directory that does not exist, you can set the directory with `sysdir set`; enter `help sysdir` for details.

System global macros It is generally better if your do-files do not depend on globals created outside of the do-file. If they do, the do-file might not run correctly if those globals do not exist, making it difficult to reproduce your analyses. Accordingly, I recommend including `macro drop _all` at the start of each do-file. To prevent this command from deleting the global macros created by `cd`*project* commands, these globals are system globals which are ignored by `macro drop _all`. To be a system global, the name must begin with `S_`. The following globals are created `cd`*project* commands:

`S_cdauto`	User specified command run by `cd`*project*.
`S_cdcmd`	Name of `cd`*project* command.
`S_cddata`	Path for datasets.
`S_cddata2`	Second path for datasets.
`S_cdnote`	Note describing `cd`*project*.
`S_cdurl`	Web address.
`S_cduser`	Any string.
`S_cdwd`	Working directory set by `cd`*project*.

To list the content of these globals, you can run `macro dir`. Or, run `listcd, details` to list all `cd`*project* commands along with the global macros they create. Running `cd`*project*`, details` lists the globals created by the command.

4 Conclusions

I hope these commands are useful for developing an efficient workflow that supports reproducible results. Hopefully, the functionality of these commands will find its way into a future release of Stata.

About the authors

Scott Long started writing these commands while teaching a course on reproducible results at the ICPSR Summer Program.

5 References

Baum, C. F. 2016. *An introduction to Stata programming*, vol. 2. 2nd ed. Stata Press College Station.

Long, J. S. 2009. *The Workflow of Data Analysis Using Stata*. College Station, TX: Stata Press.

Winters, N. 2002. fastcd: module to automate changing directories.