The Stata Journal (2018-08-08 draft) *vv*, Number *ii*, pp. 1–11

Commands for Changing the Working Directory

Draft II – Comments Welcome

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

2018-08-10

Abstract. If you work on multiple projects, you need an easy way to change the working directory. This article describes the `savecd` command which automatically creates `cd`*project* commands which lets you change your working directory by running `cd`*project*. `listcd` lists your `cd`*project* commands with a point and click interface for changing directories. Optionally, these commands can be used to customize the environment for each project.

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

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

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

If you work on multiple projects, changing the working directory is often the first you 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 use `cd` which requires you to type a sometimes lengthy path name or use the menu to click through your directory structure. On a Mac, Stata opens in the working directory that was used when you last exited Stata. With Windows and Unix, the default working directory is selected during installation with no simple way to change the default. The `workingdir` package provides commands that simplify changing working directories. For many users, this will be the only feature of the package that they need. For more advanced applications, the package also lets you customize the environment for each project. These commands were inspired by Winters (2002)'s `fastcd` package, Baum's ideas about using global macros to create an environment for a research project (Baum 2016, 78-79), and Long's simple commands for changing

© 2018-08-08 draft StataCorp LP

the working directories (Long 2009, 111-118).

The `workingdir` package is organized around the idea of a project. A project is a distinct activity such as a writing a paper or taking a class. The `savecd` command automatically creates commands, called `cd`*project* commands, that change the working directory. For example, if you are in the working directory used for a research paper, `savecd paper` creates the command `cdpaper`. Running `cdpaper` changes to the the working directory used for that paper. The `listcd` command lists your `cd`*project* commands along with the path for the working directory or a note describing the project. Clicking on the name runs the command. Many users will only use the commands to quickly change their working directory. For those who want more control over the way work is completed for each project, however, you can define global macros that specify paths where datasets are located and automatically run commands after changing the working directory.

Section 1 illustrates basic features for changing the working directory and shows how these can be used to change working directory where Stata opens. Section 2 explains 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. Section 3 provides the full syntax of each command along with some technical information. For Stata programmers, section 4 discusses how these commands were written.

1 Creating commands to change the working directory

A simple example illustrates how to create commands to switch among working directories. Suppose that `d:/active/paper/work` is the working directory used for a paper, while `d:/active/class/work` is used for a class.[1] To create the command `cdpaper` that changes to the working directory for the 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:/active/paper/work
d:/active/paper/work
```

To create the `cdpaper` command:

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

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

`savecd` saves `cdpaper.ado` to my PERSONAL directory. To create the command `cdclass` to change to the working directory for my class:

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

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

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

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

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

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

To list all of my `cd`*project* commands:

```
. listcd

      cdcda  d:/active/cda2017/work/ - drop
    cdclass  d:/class/work/ - drop
  cdcouples  y:/KI/research/couples/work/ - drop
     cddesk  c:/users/jslong/desktop/statawork/ - drop
    cdpaper  d:/active/paper/work/ - drop
    cdstart  d:/statastart/ - drop
 cdworkflow  d:/active/workflow/work/ - drop
```

The command names on the left are shown in blue. When you click on the name, the command is executed. If you click on `drop` on the right, also shown in blue, that command is dropped. You 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 describe each project with a note. For example:

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

where the `replace` indicates that if `cdpaper.ado` exists, replace it with the new command. After creating several commands using notes,

```
. listcd

      cdcda  Stat 503 CDA - drop
    cdclass  d:/class/work/ - drop
  cdcouples  Couples 3 paper - drop
     cddesk  Desktop - drop
    cdpaper  Groups paper with SAM - drop
 cdworkflow  WFDAUS - drop
```

For commands created without the `note()` option, the path is shown. Even when notes are available, you can list the paths instead by running `listcd, dir`.

1.1 If cdproject commands are not found

Suppose you run a command and obtain an error, such as:

```
. cdapaper
command cdapaper is unrecognized
r(199);
```

This would occur if you created the `cd`*personal* commands on different computer. If that is the case, copy the `cd`*personal*`.ado` files to the PERSONAL on the computer you are using. To find the location of this directory, run `adopoth`.

A second possibility is that your PERSONAL directory does not exist or it is not on the adopath. The adopath is the set of folders where Stata looks for commands, such as `cdpaper.ado`. Run `adopath` to check that a PERSONAL directory is on the adopath. If it is not, Enter `help adopath` for details on how to add the PERSONAL directory to the adopath.

1.2 Controlling the starting working directory

Stata does not provide an easy way to change the working directory in which Stata starts. For simplicity, I refer to this as the *starting directory*. One solution is to set the starting directory in your `profile.do`, a do-file that is run automatically each time Stata opens. 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`.

To have Stata start in the directory you want, you can add a `cd` command to `profile.do`. For example,

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

The problem is that if you want to change the starting directory, something I do when I change my focus to a new project, you have to edit `profile.do`.

An easier solution is to use `savecd` to create `cdstart` to change to the starting directory you want to use. For example, suppose I want to start the working directory for a paper that I am writing. I change to that directory and create `cdstart`:

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

. savecd start, replace
command cdstart saved in your PERSONAL directory
```

Then, I add `cdstart` to `profile.do` so that the command is run each time Stata starts. To change the starting directory, I replace `cdstart` with a version that opens up the directory I want. For example, make my class the starting directory:

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

. savecd start, replace
command cdstart saved in your PERSONAL directory
```

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

Even if Stata opens the the directory where I usually work, I might want to switch to a different projects. To make this easy, I add `listcd` to `profile.do`:

```
cdstart
listcd
```

`cdstart` opens in my starting directory and then `listcd` shows me other locations I can choose. To prevent an error if the `workingdir` package was not installed or if I have not created `cdstart`, I use the more robust commands:

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

`capture` capture errors without ending the do-file, while `noisily` prevents `capture` from suppressing the output from the command. After these two commands are added to `profile.do`, I no longer need to edit that file to change my starting directory.

2 Advanced features for managing projects

`cd`*project* commands can also create global macros with paths where datasets are located (section 2.1) and automatically run a command when you change projects (section 2.2). This allows `cd`*project* commands to create the project environment discussed by Baum (2016, 78-79). While I don't use these features often, for some projects they are very useful. If all you want to do is quickly change your working directory, you can skip the rest of this section.

2.1 Robust specification of data paths

The `use` command loads a dataset from the current working directory unless a path is specified. This path can be on your computer, on a LAN, or an URL. To make do-files portable, you should not hard code paths since they might not be valid on another computer or at a later time on your computer. For example, if your do-file includes the command `use d:/datasets/binlfp4, clear`, it will not run on a computer with a different directory structure. To avoid this problem, I 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 need to access data over the web or share datasets with collaborators using a shared directory on the LAN. Or, I might prefer to keep datasets for all of my projects in `d:/datasets/`.

When datasets are not in the working directory, you can create robust do-files by using a global macro that contains the path where datasets are located. For example, if in `d:/datasets/`, I create the global with this path:

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

Two things are important. First, This global is created *outside* of my do-file since I want the do-file to run on other computers without without any changes. If I

created the global inside the do-file, I would have to change the do-file. Second, the name of the global begins with `S_` which makes it a *system global*. These globals are not dropped by `macro drop _all`. This is important since you want the global to persist even if the do-file that use it includes `macro drop _all`.

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 `S_cddata` was not defined, the `use` command is interpreted as

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

and Stata looks for the dataset in the working directory. When I work on a computer with a different directory 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 critical. Here's why. If the slash is missing, I would load the dataset with the command

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

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

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

The `savecd` command can create `cd`*project* commands that define three global macros that contain data locations. The `data(`*path*`)` option creates the global `S_cddata` with the first data path. For example,

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

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

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

Two data paths are useful if you keep source datasets in one directory and datasets you create in another. For example, after running `cdpaper` that includes data locations, a do-file can use or save data in different locations. For example,

```
use "${S_cddata}mrozsource", clear
```
(*output omitted*)

```
save "${S_cddata2}binlfp5", replace
```
(*output omitted*)

Quotes are included in case a path contains a space. You can create the global `S_cdurl` with a web address by using the `url(`*address*`)` option. For example,

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

2.2 Further customization of the project 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:

```
savecd 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:

```
savecd 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,

```
savecd 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.

3.1 savecd: creating cdproject commands

`savecd` creates a `cd`*project*`.ado` file that is saved in the PERSONAL directory. If 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 `savecd` was run. The syntax is:

`savecd` *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 `savecd` begin with `cd`, but you do not need to include `cd` in *project*.

Options

`note(`*string*`)` describes the project. These notes are shown by `listcd` to document the project. To make adding notes faster, you can use the alternative syntax where everything after the project name and before the comma is the note, so that `savecd paper My project description` is equivalent to

`savecd, note(paper My project description)`.

`data(`*data-path*`)` is a path where datasets are located which is stored in the global macro `S_cddata`. An ending `/` is 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 drop these globals, you must run `macro drop` *name*, such as `macro drop S_cddata` or `macro drop S_cd*`.

3.2 listcd: point and click interface for cdproject commands

`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. `- drop` is shown in blue to the right of the command name and the path/note. If you click on `drop` for `cd`*project*, the file `cd`*project*`.ado` in PERSONAL is deleted.

The syntax is:

`listcd` [*file-specification* `,` `dir`ectories `d`etails]

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

Options

file-specification is a valid Mac, Unix, or Windows file specification to select the `cd`*project* commands that are listed. For example, `listcd *tch*` lists commands that have `tch` in the project name.

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

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

4 Programming features used by the workflow package

While non-programmers can use these commands without knowing anything about Stata programming, programmers can easily customize these commands. If you want to do this, there are two points to keep in mind.

Writing cdproject.ado files `savecd` uses `file write` to create `cd`*project*`.ado`. While these commands are straight forward, you need to be careful with the quotes in commands like this:

```
file write `adohandle´ _col(5) `"global S_cdcmd   "`adoname´""´
```

System global macros The `cd`*project* commands created by `savecd` use global macros to hold information about the project. To prevent these globals from being dropped by `macro drop _all`, the `cd`*project* commands use system globals which are not dropped. remove by `macro drop _all`. 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*.

`listcd, details` lists each `cd`*project* command along with the associated global macros. `cd`*project*`, details` lists the globals for that command.

5 Conclusions

I hope these commands are useful for developing an efficient workflow that supports reproducible results.

About the authors

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

6 References

Baum, C. F. 2016. *An introduction to Stata programming*. 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.