Checking reproducibility on Day T

Lars Vilhuber

2024-10-14

A PDF of this presentation is available here.


BBC one

Day T

Day T-1

Introduction

This document describes a few **possible** steps to self-check replication packages before submitting them to a journal. It is not meant to be exhaustive, and it is not meant to be prescriptive. There are **many ways to construct** a replication package, and many more to check that it works.

Computational Empathy

The key ingredient is what I call “**computational empathy**” - thinking about what an unknown person attempting to reproduce the results in your paper might face, **what they might know and assume**, and more importantly, **what they might not know** or know to assume. While the replication package might very well run on your computer, that is by no means evidence that it will run on someone else’s computer.

Prerequisites

In what follows, we will assume that the replicator satisfies the following conditions:

- they are familiar with their own **operating system** (not yours!)
- they are somewhat familiar with **how to run code** in the “dominant” programming language in your field (but…)
- they are **probably not familiar** with cutting edge methods of running software that you might be using (but want to learn!)
- they likely have some experience with **how social scientists write code** and prepare data, but may not have your years of experience

TL;DR

Techy lingo for “**too long, didn’t read**”. A summary of the most important takeaways will be at the top of each section.

Targets

We want to check that

- [ ] your code runs without problem, after all the debugging.
- [ ] it actually produces all the outputs
- [ ] your code runs without manual intervention, and with low effort.
- [ ] your code generates a log file that you can inspect, and that you could share with others.
- [ ] it will run on somebody else’s computer

Why

Unpredictable things happening to your computing environment:

- Software updates
- Change of employer
- Sudden need for a new computer

Run it all again

The very first test is that your code must run, beginning to end, top to bottom, without error, and ideally without any user intervention. This should in principle (re)create all figures, tables, and numbers you include in your paper.

TL;DR

This is pretty much the most basic test of reproducibility. If you cannot run your code, you cannot reproduce your results, nor can anybody else. So just re-run the code.

Exceptions

Code runs for a very long time

What happens when some of these re-runs are very long? See later in this chapter for how to handle this.

Making the code run takes YOU a very long time

While the code, once set to run, can do so on its own, *you* might need to spend a lot of time getting all the various pieces to run.

This should be a warning sign:

If it takes you a long time to get it to run, or to manually reproduce the results, it **might take others even longer**.[1]

Furthermore, it may suggest that you **haven’t been able to re-run** your own code very often, which can be indicate **fragility** or even **lack of reproducibility**.

Takeaways

- [x] your code runs without problem, after all the debugging.
- [ ] your code runs without manual intervention, and with low effort
- [ ] it actually produces all the outputs
- [ ] your code generates a log file that you can inspect, and that you could share with others.
- [ ] it will run on somebody else's computer

Why is this not enough?

Does your code run without manual intervention?

Automation and robustness checks, as well as efficiency.

Can you provide evidence that you ran it?

Generating a log file means that you can inspect it, and you can share it with others. Also helps in debugging, for you and others.

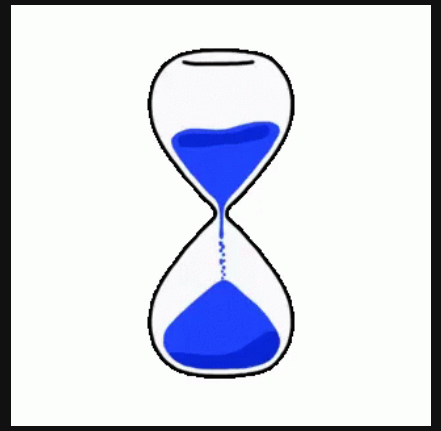
Will it run on somebody else's computer?

Running it again does not help:

- because it does not guarantee that somebody else has all the software (including packages!)
- because it does not guarantee that all the directories for input or output are there
- because many intermediate files might be present that are not in the replication package
- because you might have run things out of sequence, or relied on previously generated files in ways that won't work for others
- because some outputs might be present from test runs, but actually fail in this run

Hands-off running: Creating a controller script

Did it take you a long time to run everything again?



⏳

Let’s ramp it up a bit.

- Your code must run, **beginning to end**, top to bottom, without error, and **without any user intervention**.
- This should in principle (re)create all **figures**, **tables**, and **in-text numbers** you include in your paper.

Seem trivial?

> Out of **8280** replication packages in ~20 top econ journals, only **2594** (**31.33%**) had a main/controller script.[2]

TL;DR

- Create a “main” file that runs all the other files in the correct order.
- Run this file, without user intervention.
- It should run without error.

Creating a main or master script

In order to be able to enable “hands-off running”, the **main (controller) script is key**. I will show here a few simple examples for single-software replication packages. We will discuss more complex examples in one of the next chapters.

Examples

Stata

Set the root directory (dynamically)

```
* main.do
global rootdir : pwd
* Run the data preparation file
do $rootdir/01_data_prep.do
* Run the analysis file
do $rootdir/02_analysis.do
* Run the table file
do $rootdir/03_tables.do
* Run the figure file
do $rootdir/04_figures.do
* Run the appendix file
do $rootdir/05_appendix.do
```

Stata

Call the various files that consitute your complete analysis.

```
* main.do
global rootdir : pwd
* Run the data preparation file
do $rootdir/01_data_prep.do
* Run the analysis file
do $rootdir/02_analysis.do
* Run the table file
do $rootdir/03_tables.do
* Run the figure file
do $rootdir/04_figures.do
* Run the appendix file
do $rootdir/05_appendix.do
```

Notes

- The use of `do` (instead of `run` or even `capture run`) is best, as it will show the code that is being run, and is thus more transparent to you and the future replicator.

Notes

- Run this using the right-click method (Windows) or from the terminal (macOS, Linux):

```
cd /where/my/code/is
stata-mp -b do main.do
```

where `stata-mp` should be replaced with `stata` or `stata-se` depending on your licensed version.

R

Set the root directory (using `here()` or `rprojroot()`).

```r
# main.R
## Set the root directory
# If you are using Rproj files or git
rootdir <- here::here()
# or if not
# rootdir <- getwd()
## Run the data preparation file
source(file.path(rootdir, "01_data_prep.R")
       echo = TRUE)
## Run the analysis file
source(file.path(rootdir, "02_analysis.R"),
       echo = TRUE)
## Run the table file
source(file.path(rootdir, "03_tables.R"), e
## Run the figure file
source(file.path(rootdir, "04_figures.R"),
## Run the appendix file
source(file.path(rootdir, "05_appendix.R")
```

R

Call each of the component programs, using `source()`.

```r
# main.R
## Set the root directory
# If you are using Rproj files or git
rootdir <- here::here()
# or if not
# rootdir <- getwd()
## Run the data preparation file
source(file.path(rootdir, "01_data_prep.R")
       echo = TRUE)
## Run the analysis file
source(file.path(rootdir, "02_analysis.R"),
       echo = TRUE)
## Run the table file
source(file.path(rootdir, "03_tables.R"), e
## Run the figure file
source(file.path(rootdir, "04_figures.R"),
## Run the appendix file
source(file.path(rootdir, "05_appendix.R")
```

Notes for R

The use of `echo=TRUE` is best, as it will show the code that is being run, and is thus more transparent to you and the future replicator.

Notes for R

Even if you are using Rstudio, run this using the terminal method in Rstudio for any platform, or from the terminal (macOS, Linux):

```
cd /where/my/code/is
R CMD BATCH main.R
```

Do not use `Rscript`, as it will not generate enough output!

Other examples

For examples for **Julia, Python, MATLAB,** and **multi-software scripts**, see the full text.

Takeaways

- [x] your code runs without problem, after all the debugging.
- [x] your code runs without manual intervention, and with low effort
- [ ] it actually produces all the outputs
- [ ] your code generates a log file that you can inspect, and that you could share with others.
- [ ] it will run on somebody else's computer

Are your figures missing?

Do you usually **right-click and save** the figures?

Or **copy-paste** then into a Word document?

Hands-off running: Automatically saving figures

Say you have 53 figures and 23 tables, the latter created from 161 different specifications. That makes for a lot of work when re-running the code, if you haven’t automated the saving of said figures and tables.

TL;DR

- Save all figures using commands, rather than manually.
- It’s easy.

Saving figures programmatically

In order to be able to enable “hands-off running”, saving figures (and tables) automatically is important. I will show here a few simple examples for various statistical programming languages.

Stata

After having created the graph (“`graph twoway`”, “`graph bar`”, etc.), simply add “`graph export "name_of_file.graph-extension", replace`”. Many formats are available, as required by journal requirements.

```
1  sysuse auto
2  graph twoway (lfitci mpg disp) (scatter mpg disp)
3  graph export "path/to/figure1.png"
```

For more information, see https://www.stata.com/manuals/g-2graphexport.pdf.

R

Methods vary, but the two main ones are redefining the graphics device for the desired format, or using the `ggsave` wrapper.

```
1
2 library(ggplot2)
3 library(here)
4 figures <- here::here("figures")
5
6 ggp <- ggplot(mtcars,aes(mpg,disp)) +
7         geom_point()+
8         stat_smooth(method = "lm",geom="smooth" )
9 ggsave(ggp,file.path(figures,"figure1.png"))
```

(for more information, see https://ggplot2.tidyverse.org/reference/ggsave.html)

For more examples

Python, MATLAB, other R methods - see the full text.

> In every programming language, this is simple!

Same for tables

Learn how to save tables in robust, reproducible ways. Do not try to copy-paste from console!

Stata

`esttab` or `outreg2`, also `putexcel`. For fancier stuff, treat tables as data, use `regsave` or `export excel` to manipulate.

R

`xtable`, `stargazer`, others.

Takeaways

- [x] your code runs without problem, after all the debugging.
- [x] your code runs without manual intervention, and with low effort
- [x] it actually produces all the outputs
- [ ] your code generates a log file that you can inspect, and that you could share with others.
- [ ] it will run on somebody else's computer

Next...

I want to get ahead of the game…

Somebody will ask “but I have confidential data…”

So what happens when...

You cannot share a file

The file no longer exists on the internet

The code takes ages to run

How can you show that you actually ran the code?

The question of how to provide access confidential data is a separate issue, with no simple solution.

Creating log files

In order to document that you have actually run your code, a log file, a transcript, or some other evidence, may be useful. It may even be required by certain journals.

TL;DR

- Log files are a way to document that you have run your code.
- In particular for code that runs for a very long time, or that uses data that cannot be shared, log files may be the only way to document basic reproducibility.

Overview

- Most statistical software has ways to keep a record that it has run, with the details of that run.
- Some make it easier than others.
- You may need to instruct your code to be “verbose”, or to “log” certain events.
- You may need to use a command-line option to the software to create a log file.

In almost all cased, the generated log files are simple text files, without any formatting, and can be read by any text editor (e.g., Visual Studio Code, Notepad++, etc.).

If not, ensure that they are (avoid *Stata SMCL* files, for example).

Creating log files explicitly

We start by describing how to explicitly generate log files as part of the statistical processing code.

Stata

Start by creating a directory for the log files.

```
1  global logdir "${rootdir}/logs"
2  cap mkdir "$logdir"
3  local c_date = c(current_date)
4  local cdate = subinstr("`c_date'", " ", "_", .)
5  local c_time = c(current_time)
6  local ctime = subinstr("`c_time'", ":", "_", .)
7  local logname = "`cdate'-`ctime'-`c(username)'"
8  local globallog = "$logdir/logfile_`logname'.log"
9  log using "`globallog'", name(global) replace text
```

Stata

Add code to capture date, time, and who ran the code.

```
global logdir "${rootdir}/logs"
cap mkdir "$logdir"
local c_date = c(current_date)
local cdate = subinstr("`c_date'", " ", "_", .)
local c_time = c(current_time)
local ctime = subinstr("`c_time'", ":", "_", .)
local logname = "`cdate'-`ctime'-`c(username)'"
local globallog = "$logdir/logfile_`logname'.log"
log using "`globallog'", name(global) replace text
```

Stata

Create a logfile, giving it a name so it does not get closed.

```
1  global logdir "${rootdir}/logs"
2  cap mkdir "$logdir"
3  local c_date = c(current_date)
4  local cdate = subinstr("`c_date'", " ", "_", .)
5  local c_time = c(current_time)
6  local ctime = subinstr("`c_time'", ":", "_", .)
7  local logname = "`cdate'-`ctime'-`c(username)'"
8  local globallog = "$logdir/logfile_`logname'.log"
9  log using "`globallog'", name(global) replace text
```

Python

Create a wrapper that will capture the calls for any function

```python
from datetime import datetime
def track_calls(func):
    def wrapper(*args, **kwargs):
        with open('function_log.txt', 'a') as f:
            timestamp = datetime.now().strftime('%Y
            f.write(f"[{timestamp}] Calling {func._
        result = func(*args, **kwargs)
        return result
    return wrapper

# Usage
@track_calls
def my_function(x, y,default="TRUE"):
    return x + y

my_function(1, 2,default="false")
```

Python

Activate the wrapper

```python
from datetime import datetime
def track_calls(func):
    def wrapper(*args, **kwargs):
        with open('function_log.txt', 'a') as f:
            timestamp = datetime.now().strftime('%
            f.write(f"[{timestamp}] Calling {func._
        result = func(*args, **kwargs)
        return result
    return wrapper

# Usage
@track_calls
def my_function(x, y,default="TRUE"):
    return x + y

my_function(1, 2,default="false")
```

Python

Ideally, capture the output

```python
# Usage
@track_calls
def my_function(x, y,default="TRUE"):
    return x + y

my_function(1, 2,default="false")
# Output
# [2024-12-15 12:05:37] Calling my_function with ar
```

Notes

- More examples
- While some software (Stata, MATLAB) will create log files that contain **commands and output**, others (R, Python) by default will create log files that contain **only output**.

Creating log files automatically

An alternative (or complement) to creating log files explicitly is to use native functionality of the software to create them. This usually is triggered when using the command line to run the software, and thus may be considered an **advanced topic.** The examples below are for Linux/macOS, but similar functionality exists for Windows.

Stata

To automatically create a log file, run Stata from the command line with the `-b` option:

```
1  stata -b do main.do
```

which will create a file `main.log` in the same directory as `main.do`.

> - For this to work, the filename cannot include spaces.
> - On Windows, follow instructions here.

R

To automatically create a log file, run R from the command line using the `BATCH` functionality, as follows:

```
1 R CMD BATCH infile [outfile]
```

> On Windows, you may need to include the full path of R:

```
C:\Program Files\R\R-4.1.0\bin\R.exe CMD BATCH infile [outfile]
```

R

To automatically create a log file, run R from the command line using the `BATCH` functionality, as follows:

```
1  R CMD BATCH main.R
```

`outfile` is omitted. This will create a file `main.Rout` in the same directory as `main.R`.

R

If you prefer a different name for the output file, you can specify it.

```
1 R CMD BATCH main.R main.$(date +%F-%H:%M:%S).Rout
```

which will create a second-precise date-time stamped log file.

R

If you want to prevent R from saving or restoring its environment (by default, `R CMD BATCH` does both), you can specify the `--no-save` and `--no-restore` options.

```
1  R CMD BATCH --no-save --no-restore main.R main.$(date +%F-%H:%M:%S).Rout
```

R

The most output, and the least “acquired” information, is obtained by running the following command:

```
R CMD BATCH --debugger --verbose --vanilla main.R main.$(date +%F-%H:%M:%S)
```

R

If there are other commands, such as `sink()`, active in the R code, the `main.Rout` file will not contain some output.

R

> To see more information, check the manual documentation by typing `?BATCH` (or `help(BATCH)`) from within an R interactive session. Or by typing `R CMD BATCH --help` from the command line.

MATLAB

To automatically create a log file, run MATLAB from the command line as follows:

```
matlab -nodisplay -r "addpath(genpath('.')); main" -logfile matlab.log
```

MATLAB

A similar command on Windows would be:

```
start matlab -nosplash -minimize -r "addpath(genpath('.'));main" -logfil
```

Julia, Python

In order to capture screen output in Julia and Python, on Unix-like system (Linux, macOS), the following can be run:

```
julia main.jl | tee main.log
```

which will create a log file with everything that would normally appear on the console using the `tee` command.

Julia, Python

In order to capture screen output in Julia and Python, on Unix-like system (Linux, macOS), the following can be run:

```
python main.py | tee main.log
```

which will create a log file with everything that would normally appear on the console using the `tee` command.

Takeaways

- [x] your code runs without problem, after all the debugging.
- [x] your code runs without manual intervention, and with low effort
- [x] it actually produces all the outputs
- [x] your code generates a log file that you can inspect, and that you could share with others.
- [ ] it will run on somebody else's computer

Environments

TL;DR

- Search paths and environments are key concepts to create **portable**, **reproducible** code, by **isolating** each project from others.
- Methods exist in all (statistical) programming languages

What is an environment?

From the renv documentation:

- **Isolated**: Installing a new or updated package for one project won't break your other projects, and vice versa.
- **Portable**: Easily transport your projects from one computer to another, *even across different platforms*.
- **Reproducible**: Records the exact package versions you depend on, and ensures those exact versions are the ones that get installed wherever you go.

What software supports environments?

- R: `renv` package
- Python: `venv` or `virtualenv` module
- Julia: `Pkg` module

“But I use Stata!”

Hold on...

Understanding search paths

Generically, all “environments” simply modify where the specific **software searches** (the “search path”) for its components, and in particular any supplementary components (packages, libraries, etc.).[3]

Search paths

- R: `.libPaths()`

```
> .libPaths()
[1] "C:/Users/lv39/AppData/Local/R/win-library/4.3"
[2] "C:/Users/lv39/AppData/Local/Programs/R/R-4.3.2
```

Search paths

- R: `.libPaths()`
- Python: `sys.path`

```
>>> import sys
>>> from pprint import pprint
>>> pprint(sys.path)
['',
 'C:\\Users\\lv39\\AppData\\Local\\Programs\\Python
 'C:\\Users\\lv39\\AppData\\Local\\Programs\\Python
 'C:\\Users\\lv39\\AppData\\Local\\Programs\\Python
 'C:\\Users\\lv39\\AppData\\Local\\Programs\\Python
 'C:\\Users\\lv39\\AppData\\Local\\Programs\\Python
```

Search paths

- R:
  `.libPaths()`
- Python:
  `sys.path`
- Julia:
  `DEPOT_PATH`
  (Julia docs)

```
julia> DEPOT_PATH
3-element Vector{String}:
 "C:\\Users\\lv39\\.julia"
 "C:\\Users\\lv39\\.julia\\juliaup\\julia-1.10.0+0.
 "C:\\Users\\lv39\\.julia\\juliaup\\julia-1.10.0+0.
```

“Yes, but what about Stata?”

> We now have the **ingredients** to understand (project) environments in Stata.

Environments in Stata

TL;DR

- Creating virtual environments in Stata is feasible
- Doing so stabilizes the code, and makes it more transportable

Search paths in Stata

In Stata, we typically do not talk about environments, but the same basic structure applies: Stata searches along a set order for its commands.

Search paths in Stata

Some commands are built into the executable (the software that is opened when you click on the Stata icon), but most other internal, and all external commands, are found in a search path.

The `sysdir` directories

The default set of directories which can be searched, from a freshly installed Stata, can be queried with the `sysdir` command, and will look something like this:

```
1  sysdir
```

```
1     STATA:  C:\Program Files\Stata18\
2      BASE:  C:\Program Files\Stata18\ado\base\
3      SITE:  C:\Program Files\Stata18\ado\site\
4      PLUS:  C:\Users\lv39\ado\plus\
5  PERSONAL:  C:\Users\lv39\ado\personal\
6  OLDPLACE:  c:\ado\
```

The `adopath` search order

The search paths where Stata looks for commands is queried by `adopath`, and looks similar, but now has an order assigned to each entry:

```
1
```

```
adopath
```

```
1    [1]   (BASE)       "C:\Program Files\Stata18\ado\base/"
2    [2]   (SITE)       "C:\Program Files\Stata18\ado\site/"
3    [3]                "."
4    [4]   (PERSONAL)   "C:\Users\lv39\ado\personal/"
5    [5]   (PLUS)       "C:\Users\lv39\ado\plus/"
6    [6]   (OLDPLACE)   "c:\ado/"
```

The path at work

To look for a command, Stata will look in the first directory, then the second, and so on, until it finds it. If it does not find it, it will return an error.

```
1  which reghdfe
```

```
command reghdfe not found as either built-in or ado-file
r(111);
```

Where are packages installed?

When we install a package (`net install`, `ssc install`)[4], only one of the (`sysdir`) paths is relevant: `PLUS`.

```
1    [1]   (BASE)       "C:\Program Files\Stata1
2    [2]   (SITE)       "C:\Program Files\Stata1
3    [3]                "."
4    [4]   (PERSONAL)   "C:\Users\lv39\ado\perso
5    [5]   (PLUS)       "C:\Users\lv39\ado\plus/
6    [6]   (OLDPLACE)   "c:\ado/"
```

Installing packages

```
1  ssc install reghdfe
2  which reghdfe
```

```
1  . ssc install reghdfe
2  checking reghdfe consistency and verifying not already install
3  installing into C:\Users\lv39\ado\plus\...
4  installation complete.
5
6  . which reghdfe
7  C:\Users\lv39\ado\plus\r\reghdfe.ado
8  *! version 6.12.3 08aug2023
```

Using environments in Stata

But the `(PLUS)` directory can be manipulated

```
1  * Set the root directory
2  global rootdir : pwd
3  * Define a location where we will hold all
4  global adodir "$rootdir/ado"
5  * make sure it exists, if not create it.
6  cap mkdir "$adodir"
7  * Now let's simplify the adopath
8  * - remove the OLDPLACE and PERSONAL paths
9  * - NEVER REMOVE THE SYSTEM-WIDE PATHS - ba
10 adopath - OLDPLACE
11 adopath - PERSONAL
12 * modify the PLUS path to point to our new
13 sysdir set PLUS "$adodir"
14 adopath ++ PLUS
15 * verify the path
16 adopath
```

Using environments in Stata

```
1  * Set the root directory
2  global rootdir : pwd
3  * Define a location where we will hold a
4  global adodir "$rootdir/ado"
5  * make sure it exists, if not create it
6  cap mkdir "$adodir"
7  * Now let's simplify the adopath
8  * - remove the OLDPLACE and PERSONAL pat
9  * - NEVER REMOVE THE SYSTEM-WIDE PATHS -
10 adopath - OLDPLACE
11 adopath - PERSONAL
12 * modify the PLUS path to point to our p
13 sysdir set PLUS "$adodir"
14 adopath ++ PLUS
15 * verify the path
16 adopath
```

```
1  . adopath
2    [1]  (PLUS)      "C:\Users\lv39\Documents/PROJECT123/ado/"
3    [2]  (BASE)      "C:\Program Files\Stata18\ado\base/"
4    [3]  (SITE)      "C:\Program Files\Stata18\ado\site/"
5    [4]               "."
```

Using environments in Stata

Let's verify again where the `reghdfe` package is:

```
1  which reghdfe
```

```
1  . which reghdfe
2  command reghdfe not found as either built-i
3  r(111);
```

Using environments in Stata

So it is no longer found. Why? Because we have removed the previous location (the old `PLUS` path) from the search sequence. It's as if it didn't exist.

Previously:

```
. which reghdfe
C:\Users\lv39\ado\plus\r\reghdfe.ado
*! version 6.12.3 08aug2023
```

```
. adopath
  [1]  (PLUS)      "C:\Users\lv39\Documents/PROJECT
  [2]  (BASE)      "C:\Program Files\Stata18\ado\ba
  [3]  (SITE)      "C:\Program Files\Stata18\ado\si
  [4]              "."
```

Installing packages when an environment is active

When we now install `reghdfe` again:

```
. ssc install reghdfe
checking reghdfe consistency and verifying not already installed...
installing into C:\Users\lv39\Documents\PROJECT123\ado\plus\...
installation complete.

. which reghdfe
C:\Users\lv39\Documents\PROJECT123\ado\plus\r\reghdfe.ado
*! version 6.12.3 08aug2023
```

We now see it in the **project-specific** directory, which we can distribute with the whole project.

Installing precise versions of Stata packages

Let's imagine we need an older version of `reghdfe`.

- In general, it is **not** possible in Stata to install an older version of a package in a straightforward fashion.
- You *may* have success with the Wayback Machine archive of SSC.

Package repositories

Most package repositories are versioned:

- R: CRAN, Bioconductor
- Python: PyPI
- Julia: “General” default Julia package registry.

Stata does not (as of 2024). **But** see the full site for one approach.

Takeaways

From the earlier desiderata of *environments*:

- ✅ **Isolated**: Installing a new or updated package for one project won't break your other projects, and vice versa.
- ✅ **Portable**: Easily transport your projects from one computer to another, *even across different platforms*.
- ❌ **Reproducible**: Records the exact package versions you depend on, and ensures those exact versions are the ones that get installed wherever you go.

Takeaways

- ☑ your code runs without problem, after all the debugging.
- ☑ your code runs without manual intervention, and with low effort
- ☑ it actually produces all the outputs
- ☑ your code generates a log file that you can inspect, and that you could share with others.
- ☑ ❓ it will run on somebody else's computer

Other methods

Non-technical means

- Use a new computer
- Have an undergraduate student run it
- Ask your office neighbor to run it

More technical means

- Virtual Machines
- Use of containers

Use of containers

- **Containers** are a way to simulate a “computer within a computer”, which can be used to run code in an isolated environment.
- They are relatively lightweight, and are *starting* to be used as part of replication packages in economics (but **only 0.13% of 8280 packages**…)

Use of containers

- They do not work in all situations, and require some **more advanced technical skills** (typically Linux, in addition to the statistical software).
- Using containers to test for reproducibility is easier, and should be considered as part of a toolkit.
- Several **online services** make such testing (and development) easy.

Last but not least

Confidential data

Do you know your rights?

TL;DR

- be able to separate the confidential data from other (to be made public) components
- all code must be available
- do not publish what you are not allowed to!

Permissions

These will be noted in the **data use agreement (DUA), license, or non-disclosure agreement (NDA)** that you **signed or clicked through** to obtain access to the data from the data provider.

> Careful: scraped or downloaded data that did not have an explicit license!

Keep in mind

Just because

- you (and the entire world) can **download the data**
- does **NOT** give you the (automatic) right to **re-publish** the data.

Permissions

- Do NOT **transfer or publish** data that you have no rights to transfer. Always carefully read your data use agreement (DUA), license, or non-disclosure agreement (NDA) that you signed.
- Do NOT upload restricted-access data to the journal's platform.
- DO structure the repository to take into the account the data that cannot be published.

Communicating restrictions

Whatever **restrictions** are imposed on the data typically convey to other replicators as well.

- Document them in the **public** README, in the section about “Data Availability and Provenance Statements.”

Consider

A project with confidential and public-use data.

```
README.pdf
code/
    main.do
    01_data_prep.do
    02_confidential_prep.do
    03_analysis.do
    04_figures.do
data/
   raw/
      cps0001.dat
   confidential/
      ssa.csv
   conf_analysis/
      confidential_combined.dta
```

Organize your project so you can exclude confidential data

Clearly separate the restricted from the open-access data, both in terms of the raw data as well as the processed data:

```
README.pdf
code/
    main.do
    01_data_prep.do
    02_confidential_prep.do
    03_analysis.do
    04_figures.do
data/
   raw/
      cps0001.dat
   confidential/
      ssa.csv
   conf_analysis/
      confidential_combined.dta
```

Strategy

When the replication package relies on confidential data that cannot be shared, or is shared under different conditions, you should

- preserve (archive) the confidential replication package
  - If the data cannot be removed from a secure enclave, they should nevertheless be archived wherever the confidential data are kept[5]
  - If the data can be shared, but are subject to access restrictions, follow this guide on creating a separate data deposit and, when creating the restricted deposit at ICPSR, follow these instructions on how to do so.

Strategy

Prepare a **confidential (partial) replication package** `project-confidential.zip`, contains the contents of `data/confidential` and possibly `data/conf_analysis`.

```
README.pdf
code/
    main.do
    01_data_prep.do
    02_confidential_prep.do
    03_analysis.do
    04_figures.do
data/
   raw/
      cps0001.dat
   confidential/
      ssa.csv
   conf_analysis/
      confidential_combined.dta
```

Strategy

Prepare a **non-confidential replication package** that contains **all code**, and **any data** that is not subject to publication controls

```
README.pdf
code/
    main.do
    01_data_prep.do
    02_confidential_prep.do
    03_analysis.do
    04_figures.do
data/
   raw/
      cps0001.dat
   confidential/
      ssa.csv
   conf_analysis/
      confidential_combined.dta
```

Strategy

Important: the package contains **all code**, including the code that is used to process the **confidential data!**

```
README.pdf
code/
    main.do
    01_data_prep.do
    02_confidential_prep.do
    03_analysis.do
    04_figures.do
data/
   raw/
      cps0001.dat
   confidential/
      ssa.csv
   conf_analysis/
      confidential_combined.dta
```

Strategy

- Ensure that replicators have detailed instructions (**README**) on how to combine the two packages
- Specify which (if any) of the results in their paper can be reproduced without the confidential data.

```
README.pdf
code/
    main.do
    01_data_prep.do
    02_confidential_prep.do
    03_analysis.do
    04_figures.do
data/
   raw/
      cps0001.dat
   confidential/
      ssa.csv
   conf_analysis/
      confidential_combined.dta
```

Questions

This presentation

- Github
- Presentation
- DOI 10.5281/zenodo.13928186 Archived versions

Content is License: CC BY-NC 4.0.

References

Footnotes

1. Source: Red Warning PNG Clipart, CC-BY.
2.

Results computed on Nov 26, 2023 based on a scan of replication packages conducted by Sebastian Kranz. 2023. “Economic Articles with Data”. https://ejd.econ.mathematik.uni-ulm.de/, searching for the words `main`, `master`, `makefile`, `dockerfile`, `apptainer`, `singularity` in any of the program files in those replication packages. Code not yet integrated into this presentation.

3.

Formally, this is true for operating systems as well, and in some cases, the operating system and the programming language interact (for instance, in Python).

4.

`net install` refererence. Strictly speaking, the location where ado packages are installed can be changed via the `net set ado` command, but this is rarely done