repscan

Automated detection of Stata commands linked to common reproducibility failures

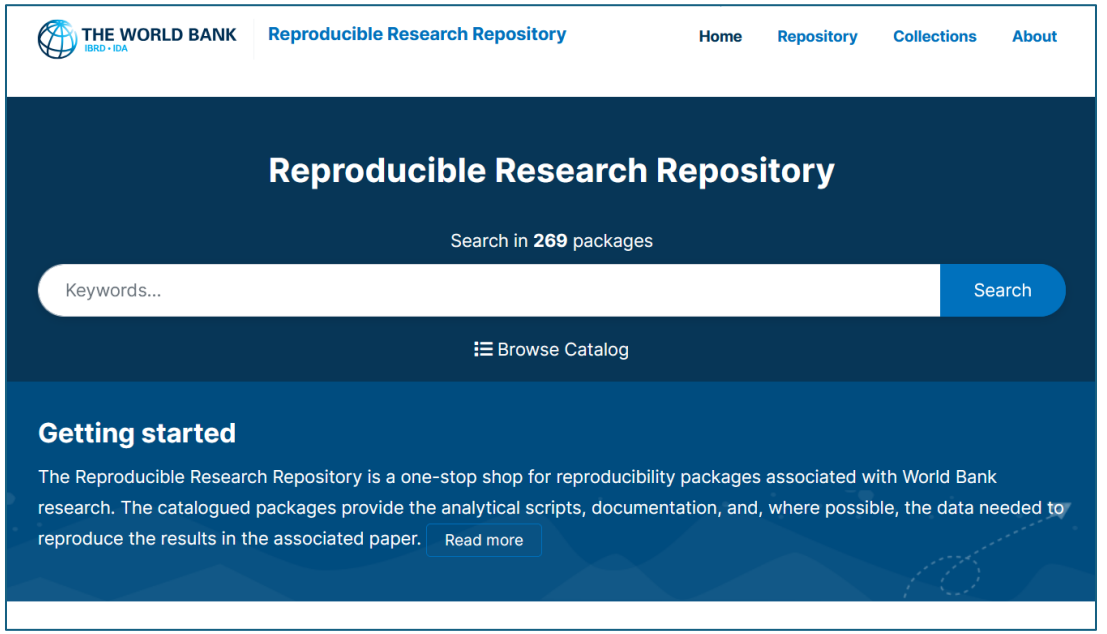
Luis Eduardo San Martin
World Bank – Development Impact Department

THE WORLD BANK
IBRD • IDA | WORLD BANK GROUP
Development Economics

Reproducible Research Repository

Some Initial Context

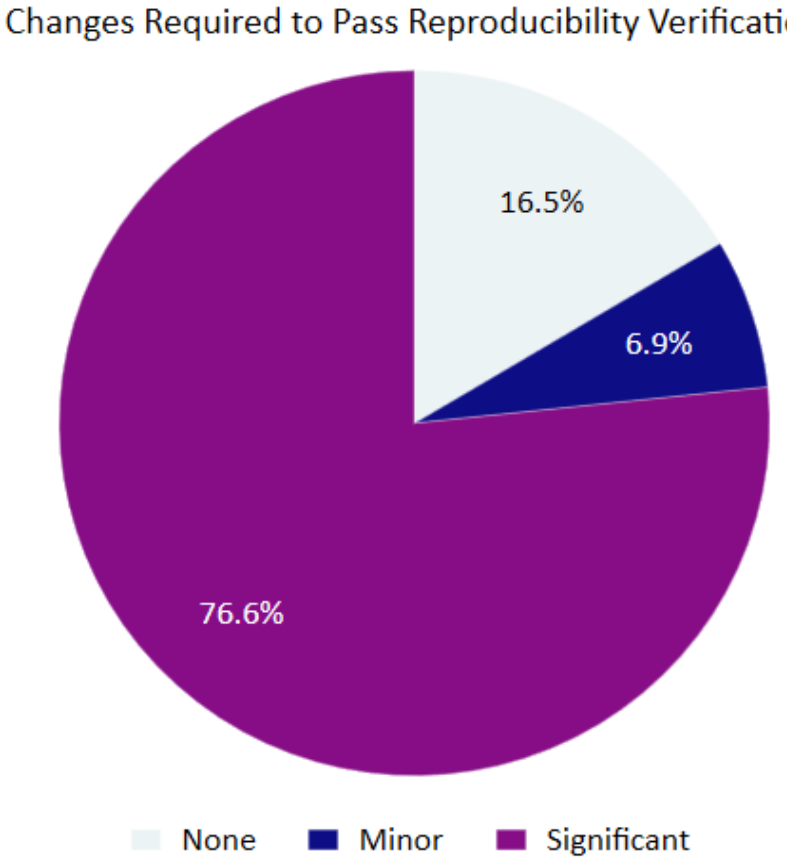
- Our team reviews and publishes reproducibility packages of development economics papers
- We started in September 2023 and have published 269
- 80% use Stata



https://reproducibility.worldbank.org

Reproducibility remains a challenge

Of the 300+ **reproducibility packages** we've reviewed, only **17%** were fully reproducible without changes.



In line with what other institutions see…

similar rates are reported by the **American Economic Journal Data Editor**, where **most packages also require updates.**

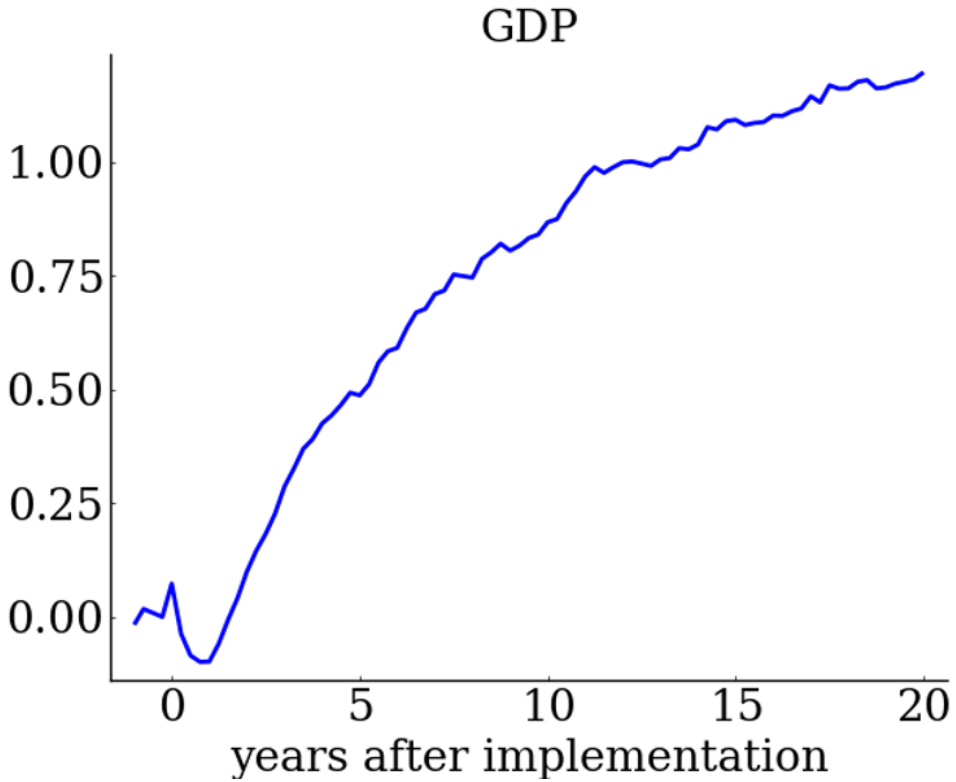
TABLE 1—RECOMMENDATIONS

Response option	Frequency
Accept	48
Accept with changes	241
Conditional accept	32
Revise and resubmit	5

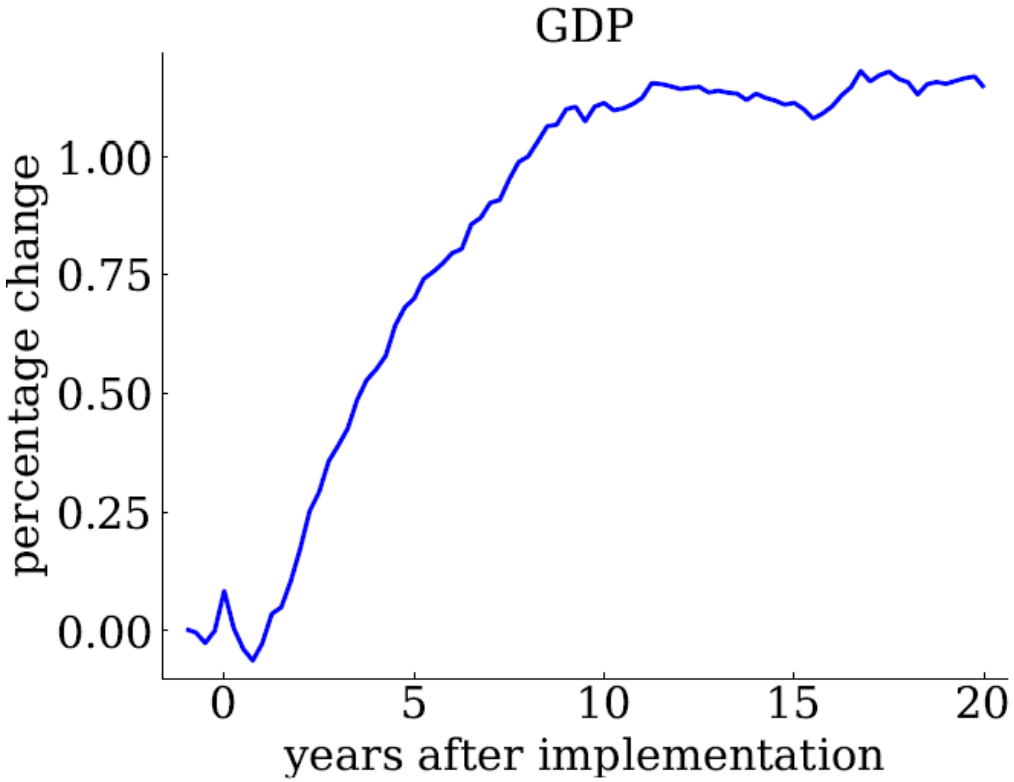
Vilhuber, Lars. 2023. "Report of the AEA Data Editor." *AEA Papers and Proceedings*, 113: 850-63.DOI: 10.1257/pandp.113.850

The Specific Problem

Result of first run



Second run



Motivation

- Our observation is that most common types of reproducibility issues come from:
  - Uncontrolled randomness
  - Sorting observations by non-unique combinations of variables

- Most cases relate to the same commands and functions:
  - `sort`
  - `bysort`
  - `runiform()`
  - Etc.

Motivation

- Then, it should be possible to identify **common reproducibility issues** by detecting the use of these functions, without executing the code

- Indeed, that's what `repscan` does! By flagging these cases for programmers to carefully review them

repscan

- repscan is a quick do-file scanner that flags functions than *can compromise* reproducibility

- It goes line by line through a do-file and doesn't require to run the scanned code

Command

```
repscan "randomization.do"
```

Installation

- `repscan` is a command in the package `repkit` – Stata tools for reproducible research

- It's available now with:

```
net install repkit, from(https://raw.githubusercontent.com/worldbank/repkit/dev/src)
```

- Will be part of the official `repkit` distribution soon (on SSC)

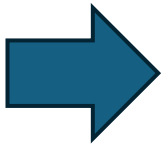
How it works

randomization.do

```
9
10  ****************************
11  ** Treatment randomization **
12  ****************************
13
14  ** Generating random numbers
15  gen rand = runiform()
16
```

Command

```
repscan "randomization.do"
```



```
Scanning do-file C:\Users\wb532468\OneDrive - WBG\Desktop/randomization.do:

 _____________________________________________________________________
|                                                                     |
|                              Issue                                  |
|_____________________________________________________________________|

  Line 15: using runiform() without setting a random seed first
_____________________________________________________________________
See repscan's help article for an explanation of each issue.

.
```

Basic Functionality

This will flag commands we've identified to have a *higher probability* of breaking reproducibility in final code outputs.

- Using `runiform()` without previously setting a random seed
- Many-to-many merges:

  `merge m:m`
- Forced drop of duplicates

  `duplicates drop` *`varlist`*`, force`

Command

```
repscan "bad.do"
```

```
 ------------------------------------------------------------------------------
|                                                                              |
|                                    Issue                                     |
|______________________________________________________________________________|

  Line 5: using runiform() without setting a random seed first
  Line 18: Using many-to-many merge
  Line 21: forced drop of duplicates

------------------------------------------------------------------------------
See repscan's help article for an explanation of each issue.

.
```

Why these commands?

- Using `runiform()` without previously setting a random seed will yield different random numbers every time

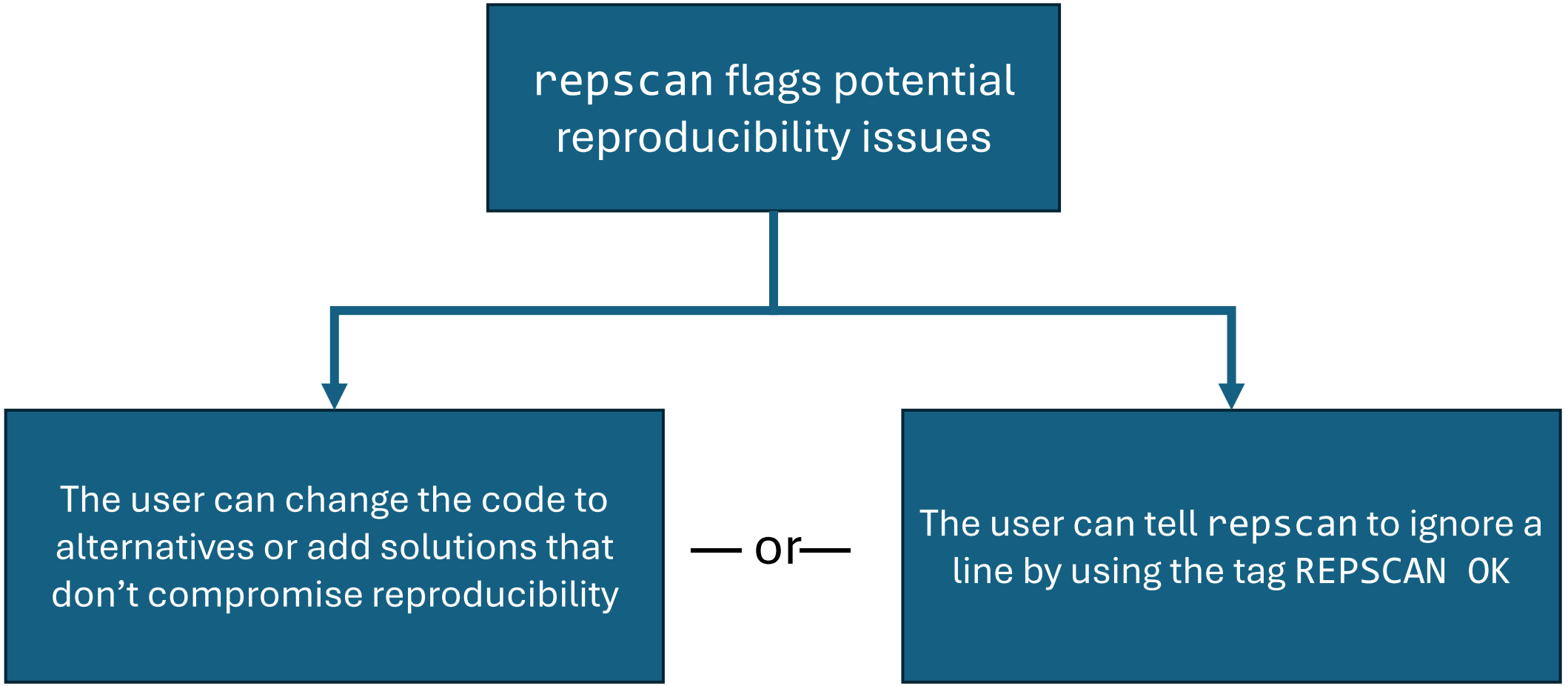
Why these commands?

- Using `runiform()` without previously setting a random seed will yield different random numbers every time
- Many-to-many merges match observations based on (1) having the same values of key variables and (2) the observation's position in their dataset (first in main with first in using, second with second, and so on). Reproducing the resulting dataset depends on consistent sorting in the main and using datasets.

Why these commands?

- Using `runiform()` without previously setting a random seed will yield different random numbers every time
- Many-to-many merges match observations based on (1) having the same values of key variables and (2) the observation's position in their dataset (first in main with first in using, second with second, and so on). Reproducing the resulting dataset depends on consistent sorting in the main and using datasets.
- A forced drop of duplicates keeps only the first observation where the values of `varlist` are repeated. Reproducing the results also depends on a consistent sorting

Intended use of `repscan`



Ignoring lines with REPSCAN OK

- Occasionally, the use of these commands will be acceptable for the user
- We can use the tag REPSCAN OK for that; repscan will skip those lines

construct-indicators.do

```
21
22
23    ** I'm sure this line is fine
24    duplicates drop, force // REPSCAN OK
25
```

```
. repscan "construct-indicators.do"
Scanning do-file construct-indicators.do:

 ___________________________________________________
|                                                   |
|                      Issue                        |
|___________________________________________________|


___________________________________________________
See repscan's help article for an explanation of each issue.

.
```

Complete Functionality

Command

```
repscan "bad.do", complete
```

This will **also** flag commands with a *lower probability* of breaking reproducibility.

Complete Functionality

Command

```
repscan "bad.do", complete
```

This will **also** flag commands with a *lower probability* of breaking reproducibility.

- **`sort` and `bysort`:** if sorting is not unique, the results may vary

Complete Functionality

Command

```
repscan "bad.do", complete
```

This will **also** flag commands with a *lower probability* of breaking reproducibility.

- **`sort` and `bysort`:** if sorting is not unique, the results may vary
- **`set sortseed`:** results might differ between Stata editions (MP, SE) and with a different number of processors used

Complete Functionality

Command

```
repscan "bad.do", complete
```

This will **also** flag commands with a *lower probability* of breaking reproducibility.

- **`sort` and `bysort`:** if sorting is not unique, the results may vary
- **`set sortseed`:** results might differ between Stata editions (MP, SE) and different number of processors used
- **`reclink`:** tie-breaking when multiple match candidates are equally likely are sort dependent

Complete Functionality

Command

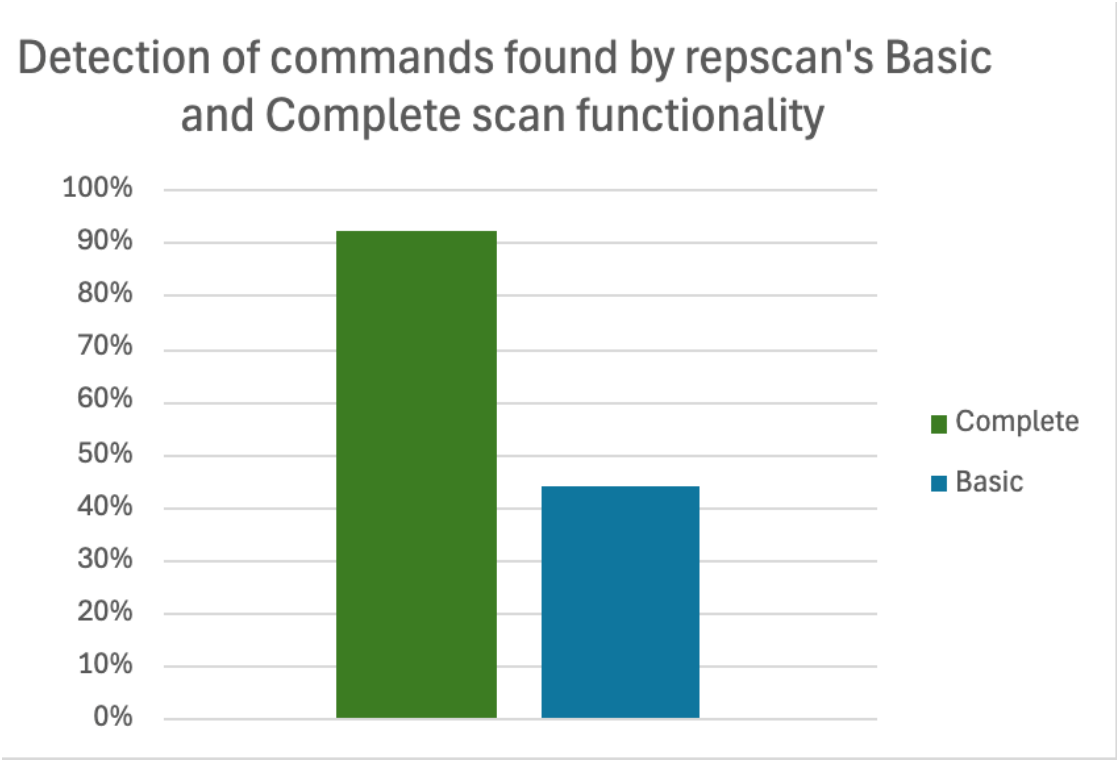
```
repscan "bad.do", complete
```

This will **also** flag commands with a *lower probability* of breaking reproducibility.

- **`sort` and `bysort`:** if sorting is not unique, the results may vary
- **`set sortseed`:** results might differ between Stata editions (MP, SE) and different number of processors used
- **`reclink`:** tie-breaking when multiple matches are equally likely are sort dependent
- **Setting a random seed (`set seed`) without setting the version:** random number generation sometimes changes between Stata versions; omitting the version might break longer-term reproducibility

`repscan` effectiveness

- We tried `repscan` on a sample of reproducibility packages with unstable outputs
- It's effective: the basic mode flags an issue in 44% of cases
- The complete mode flags in 92% of cases



Next steps

- Currently working on:
  - Vignettes with more detailed explanations of the issues and how to address them in `repkit`'s site
- Next steps:
  - More robust capture of issues—some abbreviated forms of these commands are not detected now
  - Recursiveness: allow `repscan` to detect when a sub do-file is run in the code so it can run on main do-files recursively
- Want to contribute?
  - Report a bug or desired new feature https://github.com/worldbank/repkit/issues
  - Fork our GH repository and contribute directly with a pull request

Thank you! Gracias!

Luis Eduardo San Martin
lsanmartin@worldbank.org

Team email:
reproducibility@worldbank.org

THE WORLD BANK
IBRD • IDA | WORLD BANK GROUP
Development Economics

Reproducible Research Repository