A Computing Workflow for Reproducible Results

Scott Long

University of Kentucky

March 12, 2020

Slides at jsl.faculty.indiana.edu/ftp/

© 2020 J. Scott Long

wf-cwf UKy 2020-03-09 rc1.docx

Computing workflow & author reproducibility

1. A computing workflow is a set of strategies to efficiently and accurately obtain results that can be reproduced.
2. This workflow allows *you* to reproduce *your* results using *your* data and *your* analysis scripts.

Seems easy

1. Researchers often have difficulty reproducing their own results.
   - o Files are missing, scripts don't run or give different results.
2. AJPS requires authors to submit datasets and scripts before review.
   - o Of 200 submissions, less than 5% confirmed the paper's results.

Assessing author reproducibility

1. Select a paper *you* wrote last year or last week.
2. Re-run the scripts and *exactly* reproduce your findings.
3. Ask a colleague to reproduce your results using your files.

Examples of failures

- o A retraction when a dataset was lost.
- o Two horrid weeks reproducing results to avoid a retraction.
- o A dissertation delayed 18 months to reproduce the results.
- o Collaborators using different datasets with the same name.
- o Inconsistent findings when the 'same' variable was created differently in different scripts.
- o A 743-line program did not reproduce any of the paper's results.
- o And many more.

Computing workflow (CWF)

Among possible workflows, here's how I chose mine.

Criteria for selecting strategies

Primary criterion

Reproducible	The results are robust and can be confirmed.

Secondary criteria

Accurate	Minor and retraction-worthy errors are reduced.
Efficient	Your work gets finished.
Scalable	Procedures works with small and large projects.
Adaptable	Different computing environments can be used.
Accessible	Can be used by researchers with different skill sets.

Components of the computing workflow

Script files
: Script files are used.

Posting
: Files that produce results are *never* changed.

Dual workflow
: Separate workflows are used for data management and data analysis.

Run order naming
: Scripts are named to run in alphabetical order.

Effective names and labels
: Variable names and labels document results.

Reproduction package
: Scripts and datasets are shared so others can reproduce results.

No workflow is perfect

1. No workflow is optimal for every situation.
2. Some of my strategies may seem unnecessary and irritating.
   - o Some strategies prevent rare, but devastating problems.
   - o Fixing problems takes more time than preventing them.
3. Make exceptions reluctantly.
   - o "This time I can ignore posting because..."

Scripts

Scripts are text files that control your software.

- o Also called command files, do-files, syntax files, code files, programs…

Why use scripts?

1. Scripts are the easiest way to obtain reproducible results.
   - o Friends don't let friends point and click.
2. Scripts document *precisely* what you did.
3. Scripts simplify correcting errors and making revisions.

Effective scripts

Robust scripts

Robust scripts produce identical results on a different computer.

Legible

Legible scripts are clear to you and others.

How to write robust scripts

Perfectly robust scripts are impossible, but there is a lot you can do.

1. The script controls everything that affects the results

Robust scripts…

- o Load all datasets that are used
- o Set software parameters that affect results.
- o Do not use results left in memory by other scripts.

Problem: Using variables left in memory

1. `script1.do` uses `income` to create `incsquared` but does *not* save it to a dataset.
2. `script2.do` assumes `incsquared` is in memory.
   - o If `incsquared` is not in memory, the script does not run.
   - o If the wrong `incsquared` is in memory, the results are wrong.

Unknown seed for random numbers

1. Random numbers are generated using a *seed*.
2. Results can vary greatly with different seeds.
3. To be robust, the script should set the seed.

```
Models for Diabetes
           Random     Random
         | Sample 1   Sample 2
---------+--------------------
  female |  0.733***      NS
     bmi |  1.101***   1.067***
   white |  0.505***   0.518***
     age |  1.282***   1.262***
   agesq |  0.998***   0.999***
hsdegree |  0.780***   0.720***
  weight |      NS     1.006***
  height |      NS        NS
---------+----------------------
Legend: * p<.1; ** p<.05; *** p<.01
```

2. Results should not depend on your computing environment

Do not assume that others have your computing environment.

Problem: Hard coding directories

1. Drive and directory names change.
   - o LANs periodically change drive letters or names.
   - o Different computers use different drive letters or names.
   - o People use different directory names.
2. Robust scripts assume datasets are in the software's *default directory*.
   - o To this: `use hrs3`
   - o Not this: `use d:\user\jslong\groups\work\hrs3`
3. Editors returned a paper without review with this note:

   Your script includes `setwd("i:\user\jslong\")`.
   Please resubmit the code without an absolute path.

3. Results depend on the software used

To reproduce results, you need either:

- o Specific software that produces the same results.
- o Exact details on how computations were made.

Problem: User written commands must be installed

Others might not have the user written packages you use.

```
. mchange k5
command mchange is unrecognized
r(199);
```

Problem: Software changes

1. New versions of software can give different results.
   - o Algorithms change
   - o Syntax changes
2. Statistical packages disappear and packages drop commands.
3. File formats are no longer supported.

Solutions

1. Document the software used.
   ```
   // Stata 16 / Windows IC-64 / 8 cores / born 11 Dec 2019
   // Packages spost13, coefplot, gologit2 installed 09 Feb 2020
   ```
2. Preserve the software you used.
   - o Tarball the software used for each paper.
   - o At the least, backup user written packages.
3. Save data in multiple formats.

Testing if a script is robust

1. Run it on another computer.
2. Ask a colleague to test your reproduction package (discussed later).

Legible scripts

1. Legible scripts increase accuracy and efficiency.
   - o You don't want to reproduce incorrect results.
2. Legible scripts are easier to understand.
   - o This helps others understand what you did.
   - o It helps you find errors and fix scripts.

What makes a script legible?

1. Alignment, indentation, and spacing.
2. Command lines that are visible.
3. Comments.
4. Limited use of abbreviations.
5. Effective variable names and labels.

1. Alignment and indentation

Example 1: Formatting similar commands

Without alignment

```
rename dev origin
rename major jobchoice
rename HE parented
rename interest goals
rename score testscore
rename sgbt sgstd
```

With alignment

```
rename  dev       origin
rename  major     jobchoice
rename  HE        parented
rename  interest  goals
rename  score     testscore
rename  sgbt      sgstd
rename  restrict  restrictions
```

Example 2: Aligning variables

Without indentation

```
   logit y var01 var02
 var03 var04 var05 var06
   var07 var08 var09 var10 var11 var12
  var13 var14 var15
```

With indentation

```
logit y var01 var02 var03 var04 var05 var06 var07
        var08 var09 var10 var11 var12 var13 var14 var15
```

2. Commands are visible

1. Commands should be visible on the display you use.
2. When sharing files, use 80 columns or less.

Problem: Lines that are off screen

1. The command

   ```
   mlogit jobchoice income origin prestigepar aptitude siblings friends scale1std demands interestlvl jobgoal scale3 scale2_std motivation parented city female, noconstant basecat(1)
   ```

   was truncated to

   ```
   mlogit jobchoice income origin prestpar aptitude siblings friends scale1std…
   ```

   and gave peculiar results.
2. When I finally made the script legible, the problem was obvious.

   ```
   mlogit jobchoice
      income origin prestige aptitude siblings friends demands interest
      scale1std scale2std scale3 jobgoal motivation parented city female,
      noconstant basecat(1)
   ```

3. Effective comments

- o Comments help others to understand what you did.
- o They help you when you revisit the script later.

Examples

```
// Degree variables indicate enrollment NOT completion

// Sample represents those in the paid labor force

// !! PRELIMINARY ANALYSIS partial data as of 2005-01-17.

// See Kepler AmStat 2018 for justification of cluster adjustment

// Surprising result verified 2020-03-02
```

4. Limit abbreviations

This is a valid: `l a l in 11/l`

This is legible: `list age lfp in 11/last`

5. Effective variable names and labels

1. Uninformative names lead to errors.

   ```
   logit R00215 R01834 R21119 R032712
   ```

2. Variable labels start with the critical information.

   ```
   tcdoc     How important is it to go to ...
   tcmhprof  How important is it to go to ...
   tcpsy     How important is it to go to ...
   tcfam     How important is it to turn t...
   tcfriend  How important is it to turn t...
   ```

3. Truncated value labels are definitely a problem.

   ```
        R is |     Q15 Would let X care for children
     female? | Definitel  Probably   Probably   Definitel  |     Total
   ----------+---------------------------------------------+----------
        Male |        41         99        155        197  |       492
      Female |        73         98        156        215  |       542
   ```

Legible output

Mistake 1: truncation on the right

```
           |                                        Years of education
Occupation |         3          6          7          8          9         10      ......
-----------+-------------------------------------------------------------------......
    Menial |         0          2          0          0          3          1      ......
           |      0.00       6.45       0.00       0.00       9.68       3.23      ......
-----------+-------------------------------------------------------------------......
   BlueCol |         1          3          1          7          4          6      ......
           |      1.45       4.35       1.45      10.14       5.80       8.70      ......
-----------+-------------------------------------------------------------------......
     Craft |         0          3          2          3          2          2      ......
           |      0.00       3.57       2.38       3.57       2.38       2.38      ......
-----------+-------------------------------------------------------------------......
  WhiteCol |         0          0          0          1          0          1      ......
           |      0.00       0.00       0.00       2.44       0.00       2.44      ......
-----------+-------------------------------------------------------------------......
      Prof |         0          0          1          1          0          0      ......
           |      0.00       0.00       0.89       0.89       0.00       0.00      ......
-----------+-------------------------------------------------------------------......
     Total |         1          8          4         12          9         10      ......
           |      0.30       2.37       1.19       3.56       2.67       2.97      ......
```

Mistake 2: wrapped lines

```
           |                                         Years of education
Occupation |         3          6          7          8          9         10
     11         12         13 |     Total
-----------+------------------------------------------------------------------
------------------------------+----------
    Menial |         0          2          0          0          3          1
     3         12          2 |         31
           |      0.00       6.45       0.00       0.00       9.68       3.23
   9.68      38.71       6.45 |    100.00
-----------+------------------------------------------------------------------
------------------------------+----------
   BlueCol |         1          3          1          7          4          6
      5         26          7 |         69
           |      1.45       4.35       1.45      10.14       5.80       8.70
   7.25      37.68      10.14 |    100.00
-----------+------------------------------------------------------------------
------------------------------+----------
     Craft |         0          3          2          3          2          2
      7         39          7 |         84
           |      0.00       3.57       2.38       3.57       2.38       2.38
   8.33      46.43       8.33 |    100.00
-----------+------------------------------------------------------------------
------------------------------+----------
```

Mistake 3: printing markup language

```
           {txt}{c |}                   Years of education
Occupation {c |}         3          6          7          8          9 {c |}     Total
{hline 11}{c +}{hline 55}{c +}{hline 10}
    Menial {c |}{res}         0          2          0          0          3 {txt}{c
|}{res}         31
{txt}   BlueCol {c |}{res}         1          3          1          7          4
{txt}{c |}{res}         69
{txt}     Craft {c |}{res}         0          3          2          3          2
{txt}{c |}{res}         84
{txt}  WhiteCol {c |}{res}         0          0          0          1          0
{txt}{c |}{res}         41
{txt}      Prof {c |}{res}         0          0          1          1          0
{txt}{c |}{res}        112
{txt}{hline 11}{c +}{hline 55}{c +}{hline 10}
     Total {c |}{res}         1          8          4         12          9 {txt}{c
|}{res}        337

           {txt}{c |}                   Years of education
Occupation {c |}        10         11         12         13         14 {c |}     Total
{hline 11}{c +}{hline 55}{c +}{hline 10}
    Menial {c |}{res}         1          3         12          2          7 {txt}{c
|}{res}         31
{txt}   BlueCol {c |}{res}         6          5         26          7          3
{txt}{c |}{res}         69
```

Refine your scripts

1. After writing scripts, let them age.
2. Later,
   - o Verify accuracy
   - o Improve legibility
   - o Refine comments
   - o Confirm the script is robust

The posting principle

If you don't have the files, you can't reproduce the results.

Posting is defined by two simple rules

The share rule

Only share results *after* the files used are posted.

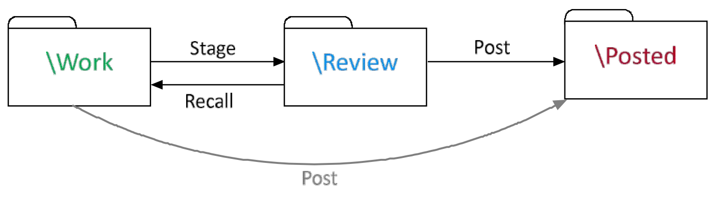
The no change rule

Never change a posted file.

Why posting?

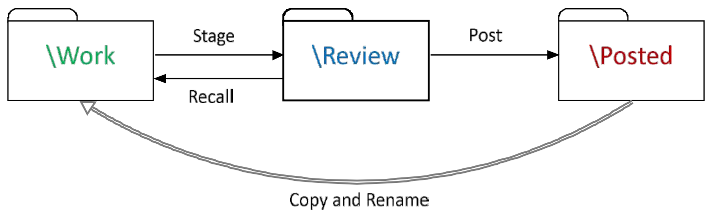
1. To reproduce results, you need the *unmodified* files.
2. You cannot mistakenly change critical files.
3. You won't have files with the same name and different content.

Files are moved, not copied



1. Scripts are developed in `\Work`.
2. Files are staged when they appear correct but need to be verified.
3. Files are recalled for verification or revision.
4. After verification, files are posted before they are shared.

Correcting posted files



1. `data1.do` creates `hsb2.dta` and the files are posted.
2. Later you discover that `data1.do` created `var5` incorrectly.
3. Copy `\Posted\data1.do` to `\Work\data1V2.do`
4. Then revise `data1V2.do` to create `var5V2` saved in `hsb2V2.dta`.
5. The V2 files are then posted.

Posting is essential for reproducibility

Primary benefits

1. Posting prevents losing critical files.
2. It prevents files with the ‘right’ name but the wrong content.

Other benefits

1. You know where critical files are located.
2. The progression from `\Work` to `\Review` to `\Posted` reduces errors.
3. The posting progressions documents a file’s status.
   - This helps immensely when projects are interrupted.
4. Backups are easier since you know which files are critical.

You agree, but...

“The datasets are *exactly* the same except I changed the married variable.”

“I was *sure* you hadn’t used the dataset yet, so I changed it.”

Dual workflows for data and analysis

- Data management is more exacting, less exciting, and more fundamental.
- Analysis sparks creativity that encourages poor data management.

The data workflow

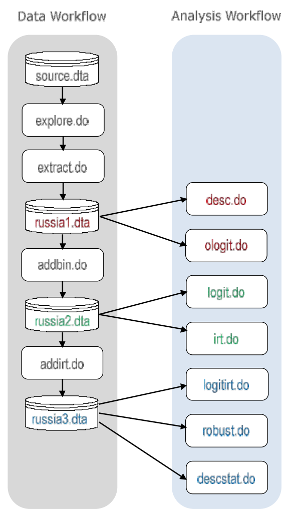
- Variables are created, verified, labeled, and documented.
- Observations are selected.
- Datasets are saved and documented.

The analysis workflow

- Datasets are used, but *not* created.
- Variables are analyzed, but *not* created.

Example of a dual workflow

Data workflow



Analysis workflow

Advantages of a dual workflow

1. Errors from careless data management are reduced.
   - Analysis scripts cannot create a variable with the same name differently.
   - You are more likely to properly name, label and document new variables.
2. Conflicting data management is less likely in collaborations.

Disadvantages

1. You must create a dataset when you *know* it will be used once.
2. It is annoying to interrupt data analysis to create a dataset.
3. Your paper won’t be on retractionwatch.com.

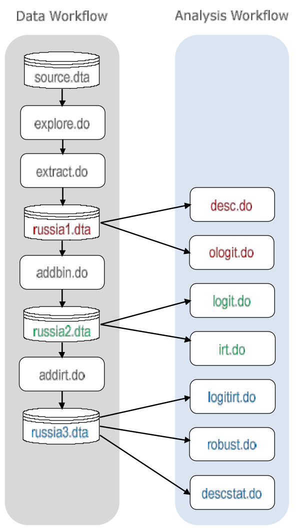
Reducing the opportunity costs

1. Before analysis begins, write a *cloning script* to create a *working dataset*.
   - This is a clone of the latest analysis dataset.
2. The working dataset is used by analysis scripts.
3. When a new variable is needed, the cloning script is revised.
4. Analysis scripts use the updated working dataset.
5. When analysis is complete, the cloning script is refined and verified.
6. Analysis scripts are re-run.

Run order naming

To determine the order in which scripts should be run you can:

1. Rely on memory.
2. Check documentation.
3. Trial and error.
4. Examine each script.
5. Use a master script.
6. Use run order naming.



Run order naming

Scripts are named to run in alphabetical order.

Do scripts need to be run in a specific order?

1. *Data scripts* must be run in a specific order.
2. *Analysis scripts* can be run in any order if the datasets exist.
   - o Run order names indicate the sequence of analyses.

Example using run order naming

When sorted alphabetically in your file manager, which names are easier to use?

Without run order names	With run order names
`addbin.do`	`data1-explore.do`
`addirt.do`	`data2-extract.do`
`desc.do`	`data3-addbin.do`
`explore.do`	`data4-addirt.do`
`extract.do`	`stat1-desc.do`
`irt.do`	`stat2-ologit.do`
`logit.do`	`stat3-logit.do`
`logitirt.do`	`stat4-irt.do`
`ologit.do`	`stat5-logitirt.do`
`robust.do`	`stat6-robust.do`
`sumstat.do`	`stat7-sumstat.do`

Correcting mistakes with run order naming

When you find an error, you know which scripts are affected.

Upstream scripts cannot be affected by the error.
Downstream scripts could be affected by the error.

Example of error correction

1. `data1.do` – `data9.do` create datasets `cwh1.dta` – `cwh9.dta`.
2. I find an error in `data6.do` which created `cwh6.dta`.
3. Upstream scripts `data1.do` – `data5.do` and `cwh1.dta` – `cwh5.dta` are fine.
4. Since downstream scripts and datasets could be affected, I revise `data6V2.do` – `data9V2.do` to create `cwh6V2.dta` – `cwh6V2.dta`.

A reproduction package

A *reproduction* package allows you and others to reproduce your results.

1. *Documentation* about software, variable construction, sample selection, and data sources.
2. *Analysis scripts* that produce all data driven results.
3. *Datasets* that at the minimum allow analysis scripts to run.

Minimum data requirements for reproducibility

1. The variables and observations used by the analysis scripts.

Data requirements for full verification and robustness checks

1. Source datasets or information on how to obtain them.
2. Data scripts that transform source datasets into the analysis datasets.

Constraints on providing data

1. Some datasets have restricted use policies that must not honored.
2. Researchers are reluctant to share data they are still using.

When should you create a reproduction package?

1. While writing a paper, anticipate creating a reproduction package.
   - o Keep track of all data driven conclusion.
   - o Keep track of every variable and observation used.
2. *Before* a draft is shared, create a reproduction package.
   - o Even if you do not distribute it, you might find errors.
   - o In papers, I only use results that are contained in the reproduction package.
   - o This makes revisions much simpler.
3. *Before* submitting a paper for review, update the package.
   - o Some journals only request the files when the paper is accepted.
   - o Others ask for them when findings are challenged.

Conclusions

1. The computing workflow
   - o Supports reproducibility
   - o Increases efficiency *in the long run*
   - o Prevents errors
   - o Make errors easier to fix
2. Posting and robust scripts are essential for reproducibility.
3. Other strategies improve efficiency and accuracy.
   - o Nobody wants to reproduce the wrong answer.
   - o Everybody wants something to reproduce.
4. The workflow has other advantages.
   - o Revising a paper is faster.
   - o Collaborative projects are easier to coordinate.
   - o Backups are easier since all critical files and only critical files are in `\Posted`.
5. Stress is reduced.

Thank you!

Questions?

References

Christensen, G., J. Freese & E. Miguel. 2019. *Transparent and Reproducible Social Science Research.* University of California Press.

Long, J.S. 2009. *The Workflow of Data Analysis Using Stata*. Stata Press.

Nuijten, M.B. 2020. Checking Robustness in 4 Steps. Project TIER Spring 2020 Webcast Series