

Reproducible Research with Stata

using version control, GitHub, and MarkDoc

E. F. Haghish

Nov. 17th, 2016

Reproducible Analysis

- Overview
- Definition

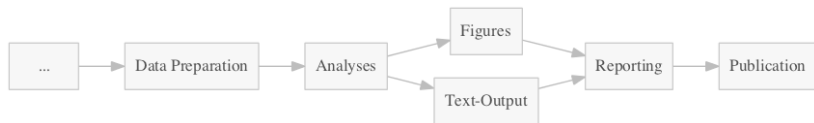


Figure 1: Reproducible Analysis

to do so, we will need:

- the same version of software, data, and code, (and the same OS, depending on the software)
- and a literate programming software

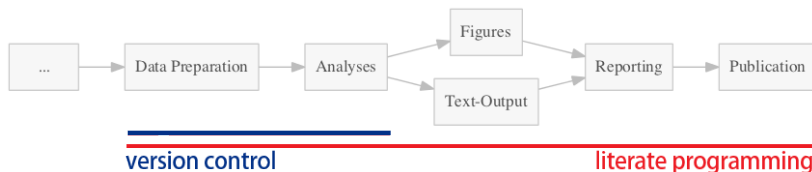


Figure 2: version control and literate programming also imply coding the analysis

- the analysis should be reproduced with **identical** software
- we should be able to access the software **without** requesting it from the author.
 - the data, code, and software should be accessible publicly
 - all versions of the software used for running the analysis should be accessible.
- archiving older versions becomes crucial. For example, Statistical Software Component (SSC) does not archive different versions of a package, in contrast to CRAN
- for developing computational programs, version control becomes much more important for fixing bugs and cooperating on the software

Concerns about package archiving

While the idea and importance of archiving versions is clear, some users may have concerns such as:

- 1 having access to different versions of a software might cause confusion for users, making them install old software
- 2 that can cause confusion for users from where they should install their software?
- 3 some would argue that we simply don't need to make archives of older software because there is no use in that
- 4 software update fixes bugs. what is the point of using previous versions if we knew they are buggy?
- 5 what is the point of reproducing the same results, using the same software version, when we know they are bugged?

GitHub for Stata community

GitHub is a general platform that is used for variety of purposes:

- 1 sharing data
- 2 sharing code
- 3 developing and collaborating software
- 4 hosting software for R, Stata, . . .
- 5 archiving software versions
- 6 documenting software, using GitHub Wiki
- 7 reading code within browser

Learning GitHub

- Using GitHub has a learning curve
- Using the GitHub desktop can considerably eliminate the learning curve.
- GitHub has a desktop GUI for Windows and Mac. Linux users have several third-party software options
 - I recommend **SmartGit** for Linux users
- When using GitHub, you still write and update your code in your computer. Once you have made a change, you can register your commit on your machine (via the App or command-line), and when you are through, you can push it to the repository on GitHub website. Therefore the workflow for programming does not change much.

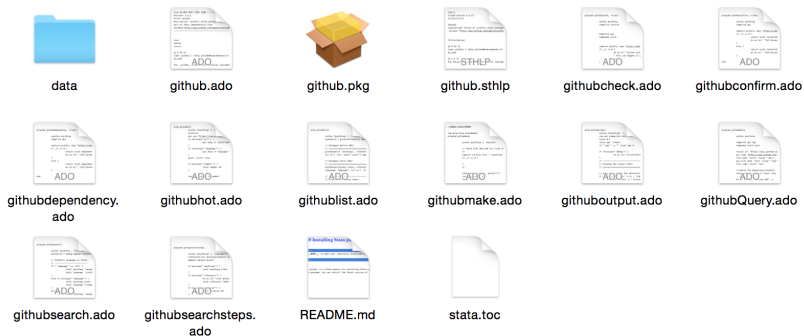


Figure 3: a screenshot of the `github` package on my local drive, where programming takes place

The screenshot shows the Stata IDE's Git interface. On the left, a sidebar lists repositories under 'GitHub' and 'Other'. The 'github' repository is selected. The main window displays a diff for the 'github.ado' file, comparing the current state to 'github.ado'. The diff shows a change in the version number from 1.2.2 to 1.2.3. Below the diff, a commit message is entered: 'the documentation of the package was updated for version 1.2.3'. The 'Commit and Sync master' button is circled in red.

Filter Repositories

Compare

Sync

GitHub

- CodeMap
- colorcode
- convertGraph
- diagram
- github
- MarkDoc
- neat
- Rcall
- ST516
- Statax
- Weaver
- webimage

Other

- AP

1 Change

github.ado

@@ -1,5 +1,5 @@

1 1 /** DO NOT EDIT THIS LINE

2 -Version: 1.2.2

2 +Version: 1.2.3

3 3 Title: github

4 4 Description: installs Stata packages with a particular version

5 5 (release) as well as their dependencies from

1.2.3

the documentation of the package was updated for version 1.2.3

Commit and Sync master

Figure 4: once you're done with coding, commit the changes and push them to GitHub

The screenshot displays the GitHub interface for the repository 'haghigh/github'. The top navigation bar shows the repository name, a dropdown menu for 'master', and buttons for 'No Uncommitted Changes' and 'History'. A 'Pull Request' button is also visible. The left sidebar contains a 'Filter Repositories' search bar and a list of repositories, with 'github' selected. The main content area shows the commit history for the 'master' branch. The commit history table lists the following commits:

Commit Hash	Version	Author	Time Ago	Changes
1.2.2	1.2.2	haghigh	8 hours ago	3 +
1.2.2	1.2.2	haghigh	8 hours ago	3 +
1.2.2	1.2.2	haghigh	8 hours ago	3 +
1.2.1	1.2.1	haghigh	8 hours ago	3 +
1.2.1	1.2.1	haghigh	8 hours ago	3 +
1.2.1	1.2.1	haghigh	8 hours ago	3 +
1.2.1	1.2.1	haghigh	8 hours ago	3 +
1.2.1	1.2.1	haghigh	8 hours ago	9 +
1.2.0	1.2.0	haghigh	1 day ago	2 +

The diff view for the selected commit shows changes to the 'README.md' file. The diff is color-coded: red for deletions and green for additions. The text in the diff is as follows:

```

4 | 4 | ---
5 | 5 | - __NOTE__: to make your repository installable, you need
  |   |   __package_name.pkg__ and __stata.toc__ files. The [__MarkDoc
  |   |   Package__](https://github.com/haghigh/MarkDoc) can __automatically__
  |   |   build these files for you, making your package ready to be
  |   |   installable from any platform. I strongly recommend you to use
  |   |   __MarkDoc__ for writing your package documentation (sthlp) files and
  |   |   creating the __pkg__ and __toc__ files, because the package updates
  |   |   all of these files automatically.
  |   |
  |   | + __NOTE__: to make your repository installable, you need
  |   |   __package_name.pkg__ and __stata.toc__ files. The [__MarkDoc
  |   |   Package__](https://github.com/haghigh/MarkDoc) can __automatically__
  |   |   build these files for you, making your package ready to be
  |   |   installable from any platform. I strongly recommend you to use
  |   |   __MarkDoc__ for writing your package documentation (sthlp) files and
  |   |   creating the __pkg__ and __toc__ files, because the package updates
  |   |   all of these files automatically. It is noteworthy that there are
  |   |   many Stata modules on GitHub which are not installable. Make sure
  |   |   your repository won't be one of them! It is noteworthy that the
  |   |   __github__ package favors repositories that are installable.
  |   |   Therefore, by making your package installable, you will receive such
  
```

Figure 5: viewing the history of changes

The github package

- It's similar to the `ssc` command in Stata. But it is used for searching, installing, and uninstalling Stata packages from GitHub.
- The package can be installed from GitHub using:

```
. net install github, from("https://raw.githubusercontent.com/haghighi/github/master/")
```

- such a command is usually required for installing any Stata package on GitHub. But `github` command makes life easier in many ways

Examples

- let's search for a package named `markdoc` on GitHub
- using `github search` command followed by the keyword
- this searches first for all repositories named `markdoc` that have **Stata** as their language and are installable packages (have the **pkg** and **toc** files in the repository)
- the output shows a description of the package, along with its **dependencies** which will be installed automatically

```
. github search markdoc
```

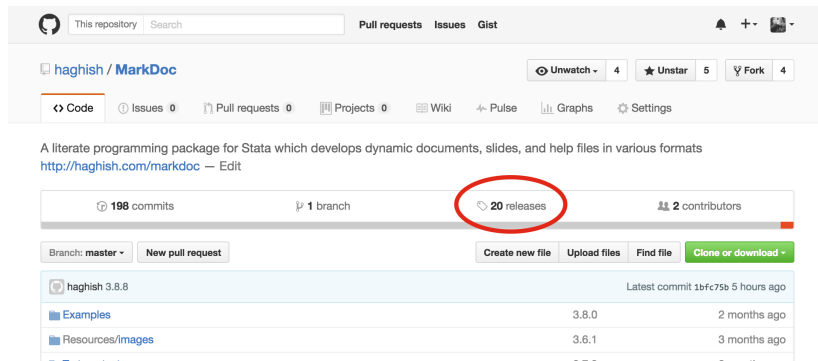
```
-----
 repository   Author      Install  Description
-----
 MarkDoc      haghish    Install  A literate programming package for Stata
              3937k      which develops dynamic documents, slides,
              and help files in various formats
              homepage: http://haghish.com/markdoc
              Hits:49   Stars:5   Lang:Stata   (Depend)
-----
```

- The `github` command allows you to specify the dependencies of the package and install them automatically after the package.
- the dependencies are simply a file named **dependency.do** that includes the code for installing a **particular version of the package** or alternatively, **the latest version of it**. But it allows the user to define a particular version of the dependencies, to ensure the package works as expected by the author and recent development of the dependency packages do not yield unexpected results
- You can install the package with a mouse click or, type the `github install` followed by **username/repository** names:

```
. github install haghish/markdoc
```

- executing the command shows that `markdoc` installs `weaver` package and `weaver` package installs another package called `statax` which is its own dependency
- having the option to install dependencies, allows the authors to break their packages into pieces, which allows others to rely on the smaller pieces in their programs. Having the option `version`, makes it safe to use a particular version of the package.
- that also means more citations

The versions are in fact GitHub releases, which are so easy to make

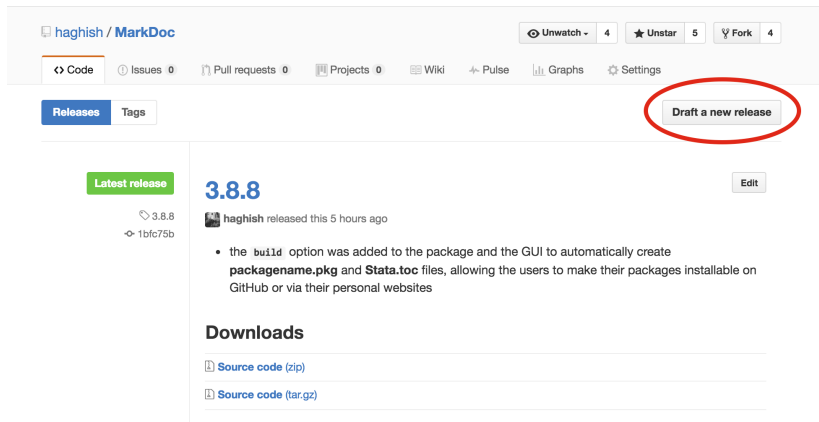


The screenshot shows the GitHub repository page for `haghish / MarkDoc`. The repository description is "A literate programming package for Stata which develops dynamic documents, slides, and help files in various formats" with a link to <http://haghish.com/markdoc>. The repository statistics show 198 commits, 1 branch, 20 releases (highlighted with a red circle), and 2 contributors. The 'Releases' tab is selected, and the following table is visible:

haghish 3.8.8	Latest commit 1bfc75b 5 hours ago
Examples	3.8.0 2 months ago
Resources/images	3.6.1 3 months ago
Resources/...	3.7.0 3 months ago

Figure 6: Viewing the software releases on GitHub

clicking on the releases button will open a page where all the previous releases are listed, the fixed bugs are explained, and you can download the old as well as the newest source code



The screenshot shows the GitHub interface for the repository 'haghigh / MarkDoc'. At the top, there are navigation buttons for 'Code', 'Issues', 'Pull requests', 'Projects', 'Wiki', 'Pulse', 'Graphs', and 'Settings'. Below these, there are buttons for 'Releases' and 'Tags'. The 'Releases' button is highlighted in blue. To the right of the 'Releases' button, there is a button labeled 'Draft a new release', which is circled in red. Below the 'Releases' button, there is a section for the latest release, version 3.8.8, released 5 hours ago. The release description includes a list of changes, such as the addition of the 'build' option to the package and the GUI to automatically create 'packagename.pkg' and 'Stata.toc' files. Below the release description, there is a 'Downloads' section with links for 'Source code (zip)' and 'Source code (tar.gz)'.

Figure 7: Creating a new release

Excellent! This tag will be created from the target when you publish this release.

Target: **master**

Release title

Write Preview Markdown supported

- The following bugs were fixed
- The following features were added

Attach files by dragging & dropping, [selecting them](#), or pasting from the clipboard.

Attach binaries by dropping them here or [selecting them](#).

This is a pre-release
We'll point out that this release is identified as non-production ready.

Publish release Save draft

Tagging suggestions

It's common practice to prefix your version names with the letter v. Some good tag names might be v1.0 or v2.3.4.

If the tag isn't meant for production use, add a pre-release version after the version name. Some good pre-release versions might be v0.2-alpha or v5.9-beta.3.

Semantic versioning

If you're new to releasing software, we highly recommend reading about [semantic versioning](#).

Figure 8: publishing the new release

Accessing releases via Stata

- Once a new release is made on GitHub or the package master is updated, the new version becomes available for all users instantly.
- You can view all of the available versions using the `github query` command followed by the **username/repository**

```
. github query haghish/markdoc
```

Version	Release Date	Install
3.8.8	2016-11-16	Install
3.8.7	2016-11-10	Install
3.8.6	2016-11-10	Install
3.8.5	2016-10-16	Install
3.8.4	2016-10-13	Install
3.8.3	2016-10-03	Install
3.8.2	2016-10-01	Install
3.8.1	2016-09-29	Install
3.8.0	2016-09-24	Install
3.7.9	2016-09-20	Install
3.7.8	2016-09-19	Install
3.7.7	2016-09-18	Install
3.7.6	2016-09-13	Install
3.7.5	2016-09-08	Install
3.7.4	2016-09-07	Install
3.7.3	2016-09-06	Install
3.7.2	2016-09-05	Install
3.7.0	2016-08-23	Install
3.6.9	2016-08-16	Install
3.6.7	2016-02-27	Install

- Clicking on the `install` text would install any of the previous versions
- Alternatively we can use the `version(tag)` option to install any version. The *tag* is the version that we specify for each release. For example, version 3.8.7 of MarkDoc (old version) can be installed as follows:

```
. github install haghish/markdoc, version(3.8.7)
```

- the same procedure can be used in the **dependency.do** file to install a particular version of a package

Other github subcommands

- you can uninstall a package, which only requires the repository name

```
. github uninstall markdoc
```

- you can check whether a repository is installable? This will confirm that the *packagename.pkg* and the **stata.toc** files exist in the repository. The **github search** command also carries out this process and only shows the **install** text if the package is installable

```
. github check haghish/markdoc
stata.toc file was found
pkg file was found
haghish/markdoc is installable
```

- You can view the Stata packages that are popular and you have plenty of options to search different repositories:
- try:

```
. github hot  
. github hot, n(30)  
. github hot, all  
. github hot, all language(Python)
```

- the data is available on GitHub:
<https://raw.githubusercontent.com/haghighi/github/master/data/archive.dta>
- you can build a fresh archive of Stata repositories on GitHub anytime. and it takes about 10 minutes to be executed. The command will create a dataset with the given name.

```
. github list stata, language(all) in(all) all save(archive) append
```

Literate Programming

- Reproducible documentation
- Idea



Figure 9: Literate Programming Process

- The main idea is to make the code more readable and well-written by preparing it for others to read and comprehend it. The document is **only a byproduct**.
- Literate programming **must not be reduced to generating dynamic document!**
- It is meant to:
 - 1 make reading and comprehending source code and data analysis code easier by including the documentation
 - 2 make the analysis and documentation reproducible
 - 3 make writing documentation easier

- The documentation is written inside the code file to make them more understandable, therefore the **readability** of the code is central to literate programming paradigm
- the markup language used for documentation should be as simple as possible, to avoid unnecessary complications in the source code
- the markup language should not impose learning curve
- using **HTML** and **LaTeX** for documentation is only popular among nerds

Workflow

- The same workflow is used in statistics for data analysis
- The documentation is specified with an especial notation
 - Therefore, the source code is not directly sourceable
 - The most popular programs only include the Weave process.

Workflow

This workflow can be improved in a variety of ways:

- Interactive literate programming
- Real-time weaving
- Supporting different markup languages
- Supporting all documentation features required by statisticians
- Producing documents in various formats, using the same source
- Thinking about procedures to improve the readability of the code
- Keep the source code sourceable

Making literate programming convenient

- Real-time document update in different formats
 - Docx, html, latex, PDF, slides, etc.
- A simple GUI interface to facilitate working with the packages
- Support for LaTeX mathematical notations in all document formats
- Automated process for capturing, saving, and including figures in the document
- Creating automatic layouts for markup languages (Improving readability)
- Ensuring that the code files remain sourceable

MarkDoc package

- **MarkDoc** is a general purpose literate programming package for Stata
- it supports **Markdown**, **LaTeX**, and **HTML**
- it provides a holistic approach to reproducible documentation of produce various formats from **the same source**
 - Microsoft Office Docx
 - OpenOffice ODT
 - LaTeX
 - HTML
 - Beamer slides
 - Web-based HTML slides
 - epub
 - Stata help files (sthlp)
 - Stata package vignette in all formats mentioned above

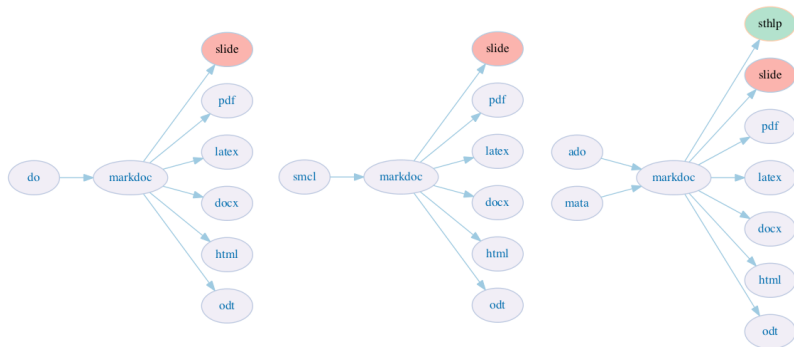


Figure 10: MarkDoc includes several engines for processing **do**, **smcl**, **ado**, and **mata** files

MarkDoc's features

- MarkDoc supports interactive workflow
 - using the **smcl** log as the source code allows you to view the document in any of the formats after making a change, without re-executing the whole code
 - using the **do** file as the source requires re-executing the whole source code for generating the document
 - using **ado** and **mata** code, which are used for programming, only extract the documentation for generating Stata help files and package vignettes.
- Script files written for MarkDoc will remain sourceable in Stata, i.e. the analysis can be executed as usual in Stata because the documentation are written as a special comment format, only meaningful to MarkDoc

MarkDoc's features

- has a very simple method for capturing and including graphs in the dynamic document
- supports writing dynamic text
- supports creating dynamic tables conveniently
- it defined markers for keeping the output document simple, while preserving all of the analysis code and results
 - 1 `//ON` and `//OFF` for activating and deactivating the results of a code chunk
 - 2 `/**/` for hiding a command
 - 3 `***` for hiding output
 - 4 `//IMPORT filename` for importing documentation from external files, which helps to keep the do-files clean

Installation

- **MarkDoc** is hosted on SSC and can be installed by:

```
. github install haghish/markdoc
```

- Since Feb 2016, all of the releases of MarkDoc are accessible via GitHub:

```
. github query haghish/markdoc
```

- an analysis that is used by older versions of MarkDoc will be reproducible, because you can install older versions of MarkDoc to generate the documentation

Documentation

- The documentation is written as comment, between `/**` and `*/` comment signs. For example, your `do` file could look like:

```
. stata command

/**
Markdown heading 1
=====

Markdown heading 2
-----

documentation text.
***/

. stata command
```

- **Markdown** syntax is explained on GitHub Wiki, which holds the manual of the package: `https://github.com/haghish/MarkDoc/wiki/Markdown-tutorial`
- examples of MarkDoc package can be found on GitHub as well: `https://github.com/haghish/MarkDoc/tree/master/Examples`

Learning MarkDoc

- MarkDoc comes with a graphical user interface which makes learning the package much easier.
- Typing `db markdoc` opens the dialog box
- The dialog box has 3 independent tabs, each work with a particular engine, and has its own options.

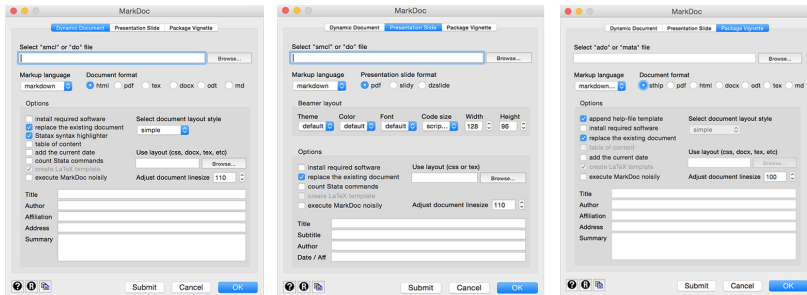


Figure 11: MarkDoc GUI

Applications in research

Reproducible research software can improve analysis transparency by:

- Documenting the process of data analysis
- Allowing the whole analysis to be reproduced step by step
- Embedding the interpretation
- Automatizing reporting the results and eliminating untraceable human errors
- Combining the results, figures, and all of the interpretations in a publication-ready document that ideally should be available in various document formats (Docx, LaTeX, PDF, OpenOffice ODT).

Applications in education

- Reproducible research tools can be used by students to:
 - Create notes for themselves within statistical packages
 - Document their code
 - Read and comprehend code that is written by others in the same fashion
 - Practice data analysis in a more disciplined way

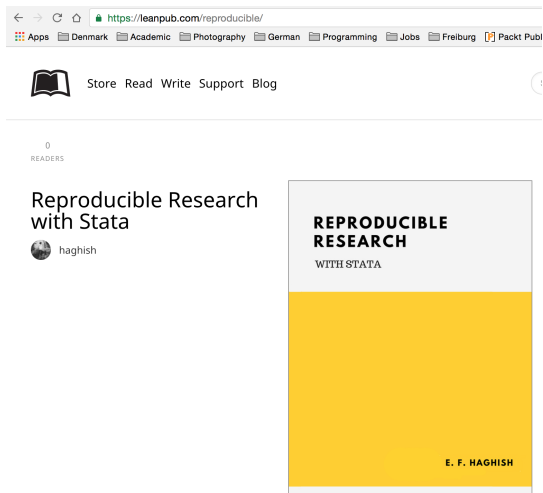
Applications in education

- Reproducible research tools are very useful tools for teaching statistics at any level for teachers, students, and programmers
 - Teachers can use them for creating educational materials using the same “**source files**”. The source files can be used to produce:
 - Presentation slides
 - Handouts
 - HTML documents for websites or blog posts
 - eBooks
 - Produce documents with different levels of documentation
 - e.g. slides and detailed handouts can be produced from the same source by specifying how and to what extent the documentations should be included
 - The documentation can be reused in other formats, which encourages practicing literate programming

Applications in statistical programming

- Programmers can get benefit from this paradigm for documenting their own code
 - Making it more comprehensible for others
 - Encouraging others to read their code
 - Using the documentation to produce various documentation formats
 - Help files, package vignettes, etc.
 - Reading

If you found this talk interesting, you can expect a book that touches on the topics of this presentation in much more details and examples: <https://leanpub.com/reproducible/>



The screenshot shows a web browser window with the URL <https://leanpub.com/reproducible/>. The browser's address bar and tabs are visible. The website header includes a book icon and navigation links: Store, Read, Write, Support, and Blog. Below the header, it indicates 0 READERS. The main content area features the book title "Reproducible Research with Stata" and the author's name "haghish" next to a small globe icon. To the right is a large image of the book cover, which has a white top half and a yellow bottom half. The text on the cover reads "REPRODUCIBLE RESEARCH WITH STATA" and "E. F. HAGHISH".