Reproducible Research with Stata
using version control, GitHub, and MarkDoc

E. F. Haghis

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.
Figure 4: once you’re done with coding, commit the changes and push them to GitHub
Figure 5: viewing the history of changes
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/haghish/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
```

```
<table>
<thead>
<tr>
<th>repository</th>
<th>Author</th>
<th>Install</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MarkDoc</td>
<td>haghish</td>
<td>Install</td>
<td>A literate programming package for Stata which develops dynamic documents, slides, and help files in various formats homepage: <a href="http://haghish.com/markdoc">http://haghish.com/markdoc</a> Hits:49 Stars:5 Lang:Stata (Depend)</td>
</tr>
</tbody>
</table>
```
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

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.
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.

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

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 \texttt{github query} command followed by the \texttt{username/repository}.
. github query hagish/markdoc

<table>
<thead>
<tr>
<th>Version</th>
<th>Release Date</th>
<th>Install</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.8.8</td>
<td>2016-11-16</td>
<td>Install</td>
</tr>
<tr>
<td>3.8.7</td>
<td>2016-11-10</td>
<td>Install</td>
</tr>
<tr>
<td>3.8.6</td>
<td>2016-11-10</td>
<td>Install</td>
</tr>
<tr>
<td>3.8.5</td>
<td>2016-10-16</td>
<td>Install</td>
</tr>
<tr>
<td>3.8.4</td>
<td>2016-10-13</td>
<td>Install</td>
</tr>
<tr>
<td>3.8.3</td>
<td>2016-10-03</td>
<td>Install</td>
</tr>
<tr>
<td>3.8.2</td>
<td>2016-10-01</td>
<td>Install</td>
</tr>
<tr>
<td>3.8.1</td>
<td>2016-09-29</td>
<td>Install</td>
</tr>
<tr>
<td>3.8.0</td>
<td>2016-09-24</td>
<td>Install</td>
</tr>
<tr>
<td>3.7.9</td>
<td>2016-09-20</td>
<td>Install</td>
</tr>
<tr>
<td>3.7.8</td>
<td>2016-09-19</td>
<td>Install</td>
</tr>
<tr>
<td>3.7.7</td>
<td>2016-09-18</td>
<td>Install</td>
</tr>
<tr>
<td>3.7.6</td>
<td>2016-09-13</td>
<td>Install</td>
</tr>
<tr>
<td>3.7.5</td>
<td>2016-09-08</td>
<td>Install</td>
</tr>
<tr>
<td>3.7.4</td>
<td>2016-09-07</td>
<td>Install</td>
</tr>
<tr>
<td>3.7.3</td>
<td>2016-09-06</td>
<td>Install</td>
</tr>
<tr>
<td>3.7.2</td>
<td>2016-09-05</td>
<td>Install</td>
</tr>
<tr>
<td>3.7.0</td>
<td>2016-08-23</td>
<td>Install</td>
</tr>
<tr>
<td>3.6.9</td>
<td>2016-08-16</td>
<td>Install</td>
</tr>
<tr>
<td>3.6.7</td>
<td>2016-02-27</td>
<td>Install</td>
</tr>
</tbody>
</table>
- 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 confirms 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/haghish/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 a learning curve.
- Using **HTML** and **LaTeX** for documentation is only popular among nerds.
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.
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 haghash/markdoc

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

  . github query haghash/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
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
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/