GitHub: A Better Place for Developing, Maintaining, and Hosting Statistical Software

E. F. Haghish
Department of Medical Psychology and Medical Sociology
University of Gttingen

haghish@med.uni-goettingen.de

Abstract. The popularity of GitHub is growing, among not only software developers, but also statisticians and data scientists. This manuscript discusses why GitHub is a good place for developing, maintaining, and collaborating on statistical software. Furthermore, the manuscript introduces the `github` module version `2.0` for Stata, which facilitates building, searching, installing, and managing statistical packages hosted on GitHub. The module also provides programs for mining statistical repositories hosted on GitHub and SSC. Further suggestions are made to enhance the practice of developing and hosting statistical packages on GitHub as well as using them for data analysis.

Keywords: version control, social coding, social computing, statistics software, data mining, github

1 Introduction

GitHub[1] is a social coding site that offers plenty of features for collaboration on software such as tracking issues, documentation platform, managing tasks, and version control (Chacon and Straub 2014). Within a few years after its launch in 2008, GitHub has become not only the largest host for Git repositories with over 28 million developers (GitHub 2018c) and 29 million public repositories (GitHub 2018b), but also the largest code hosting site in the world (Gousios et al. 2014b).

Such a sharp growing trend can also be seen for public Stata repositories on GitHub. As shown in figure 1, from 2013 to 2018, there has been a rapid growth in the number of Stata repositories and over 3300 repositories and 450 installable users-written packages have been publicly published. The resurgence in GitHub popularity for hosting Stata packages underscores the necessity for a module that facilitates accessing and managing them. In this manuscript, I introduce the `github` package, which provides a handful of tools for searching, installing, building, and managing packages that are hosted on GitHub as well as programs for mining GitHub repositories. In addition, I point out the pros and cons of using GitHub for developing and hosting statistical packages. I start by arguing why GitHub is a good place for developing statistical software

[1] `www.github.com`

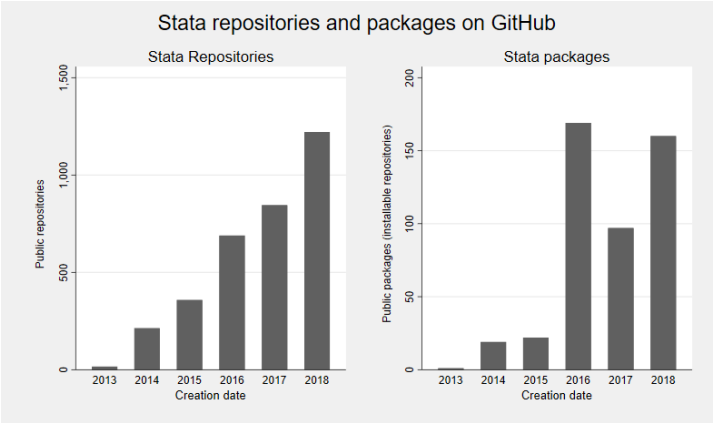


Figure 1: Public Stata repositories and packages on GitHub by year of creation

and reveal a few problems that GitHub can ameliorate for Stata user-written software.

1.1 Developing statistical software on GitHub

At the first glance, GitHub appears to be a multi-faceted platform with a variety of industry-standard services that expedite software development, maintenance, and documentation. The preeminent advantage of GitHub, nonetheless, is its social nature (Thung et al. 2013). GitHub is essentially a social network for gregarious developers to broadcast their coding exercise, follow others' activities, audit a repository, discover recent projects, and collaborate (Dabbish et al. 2012b; Vasilescu et al. 2013). Subsequently, the pro-social characteristics of GitHub promotes project dissemination by attracting attention and building reputation (Jiang et al. 2013), ergo, peer-reviewing the code by plenty of programmers.

The question, however, is that how GitHub can turn programming into a collective activity? At its core, GitHub is a combination of Git (Torvalds and Hamano 2010) – a popular Distributed Version Control (DVC) system – with a social media, which enables code-integration (collaboration) through a two-level hierarchical network (Cross et al. 2002). Those who have writing privileges to a repository belong to the upper-level hierarchy and can make *direct* contributions to the project. In contrast, in an *indirect* involvement, developers who don't have writing privilege, *fork* (clone) (Jiang et al. 2017) the main repository, rework a part of the code, and then submit a *pull request* to be reviewed by the project maintainers. If accepted, the change will be incorporated in the repository. Such a practice – known as *pull-based development model* (Barr et al. 2012; Gousios et al. 2014a) – permits anyone to view, fork, and contribute to any public repository on GitHub (Vasilescu et al. 2014). The pull-based development model relies on a DVC for monitoring individual inputs, preserving

the progress history, supporting backup to any development period or versions, and blending individual contributions into a project (Tichy 1985; Hammack et al. 2002; Fischer et al. 2003).

There is a strive for more transparency in scientific computations (Miguel et al. 2014) and GitHub is a step toward this end. On the one hand, the publicly available development history of a project, i.e. how it has evolved in time and who committed to it, elevates its transparency (Dabbish et al. 2012a). On the other hand, the project dissemination aspect of GitHub boosts the transparency, enabling anyone on the Internet to be a potential peer-reviewer (Jiang et al. 2013; Dabbish et al. 2012b). Studies have shown that GitHub users take such information into account when evaluating a repository (Dabbish et al. 2012a; Jiang et al. 2017; Marlow et al. 2013).

Despite the success of Open Source Software (OSS), there are many challenges associated with the OSS development. For example, a lack of structured documentation and communication, which are required for a teamwork (Vasilescu et al. 2014). Moreover, every time a *pull request* is made to update the project, one of the project's core-team members should review and integrate the suggested changes to the code or documentation (Gousios et al. 2015). Evidently, the more complex the project, the larger such obstacles (El Emam et al. 2001). Nevertheless, generally speaking, such problems are less probably for Stata packages, because the majority does not have an intricate structure. To evaluate this assertion, all of Stata packages published on Boston College Statistical Software Components (SSC) and GitHub until April 30th 2019 were mined and analyzed (see section 4 and 7.1). The resulted data sets included 475 packages hosted on GitHub and 2723 on SSC. Analyzing their content revealed that, respectively, 49% and 70% the modules hosted on GitHub and SSC are comprised of a single script file. In the same vein, 85% of packages on GitHub and SSC, included four and two script files only. This implies that the majority of Stata packages has a relatively simple structure.

1.2 Hosting statistical software on GitHub

The SSC archive has been the primary landing point for the majority of Stata packages[2]. Packages published on SSC are indexed by the `search` command after a while. Although, this convenience comes with several critical drawbacks that might make GitHub a superior alternative:

1. SSC does not require specifying nor reports any information about version and license of hosted packages.

2. SSC only hosts the latest release of the package, without archiving the previous releases. Along with lack of version specification, this can pose threats to research reproducibility.

3. There is no mechanism to cite and install dependencies on SSC (see section

[2]as shown in figure 1, since 2016, GitHub has received a viable credibility from Stata community for hosting packages

2.4.1), which cannot be accomplished without archiving previous releases. Otherwise, relying on other user-written software without declaring[3] the required version would be a sloppy programming practice.

GitHub knows none of these limits. Stata packages hosted on GitHub can be downloaded as 1) *development version*, which is often the main branch of the repository and 2) as a *stable release*. The latter is of significant importance, because it is prepared and documented for the installation. Releasing a stable version on GitHub is as simple as a few mouse clicks. More importantly, all previous releases are archived, and remain accessible. For every release, the maintainer has to declare a unique version and optionally, can write a report about that release. In addition, not only GitHub can host the software, but also its documentation, via a version-controlled *Wiki*. Users can fork the Wiki, update it, and make a pull-request, just as done for updating a program.

However, the primary obstacle with retrieving packages from GitHub is that Stata's `search` command cannot look for packages hosted on GitHub. Moreover, installing a package using Stata's `net install` command – using the repository URL – can only setup the development versions. In the following sections, I introduce `github` module version `2.0` that aims to solve all of these limitations. I will explain how to select a license, declare a version, cite and automatically install dependencies, and revert to a particular version of a software, along with its required dependencies.

❑ **Technical note**

The `github` package includes the `pkgzip` (type `help pkgzip` for documentation) command, which can download any package from SSC as a zip file, a feature lacking by the SSC. It also obtains the release date of the package and renames the zip-file accordingly. The publication date is not a replacement for a software version, but at least provides a clue about the package updates. For example, to obtain a package named `xxx` from SSC, type the following code, which would download *xxx_yyyymmdd.zip* in your working directory. Users relying on SSC packages for data analysis are recommended to download and archive the packages along with their analysis code.

```
. pkgzip xxx
```

❑

2 The `github` module

The `github` package provides a comprehensive toolbox for working with Stata packages that are hosted on GitHub. The module itself does not offer any command for interacting with a version control software. Instead, it facilitates:

1. Searching GitHub API, Stata.com, and other web sources for Stata packages.

[3]This is indeed a universal practice. Remember that Stata programs also confirm the required Stata version using `version` command

2. Installing a development, a stable, or an archived stable version of a package, along with its dependencies.

3. Managing and updating installed packages

4. Introducing a graphical user interface (GUI) and for building packaging files (i.e. the *pkg* and *toc* files) to make a GitHub repository installable.

5. Mining Stata repositories on GitHub and SSC.

2.1 Installation

The `github` module is hosted on GitHub only and can be installed by typing:

```
. net install github, from("https://haghish.github.io/github/")
```

2.2 Syntax

The general syntax of the `github` package is summarized below. The behavior of the package changes based on the given *subcommand*, which are summarized in table 1.

`github` [*`subcommand`*] [`...`] [`, options`]

Table 1: Summary of `github`'s subcommands

Subcommands	Description
Essential	
`list`	expedites managing packages installed with `github`
`search`	looks for packages or repositories via GitHub API
`install`	installs a package along with its dependencies
`uninstall`	removes a package from Stata
`update`	updates a package to the latest version
Supplementary	
`version`	returns the version of an installed package
`query`	returns all archived stable versions of a package
`check`	tests whether a repository is an installable Stata package

Apart from `list` that doesn't take any other argument and `search`, which can take multiple keywords, other subcommands are followed by either a *packagename* or a *username/repository*. For the latter, the combination of the *username/repository* provides the URL address to access the GitHub repository and is required by the `install`, `check`, and `query` subcommands.

2.3 Searching Stata repositories

The `github search` command contacts the GitHub API to search for Stata packages using one or multiple keywords. The complete syntax is as follows:

`github search` *Keywords* [`, language(str) in(str) all net local`]

The options help narrowing down or expanding the search scope. For example, you may limit the results to repositories where the majority of the code is written in the Stata language[4] (default) or expand the search by including results from Stata's `search` command. These options are further explained below:

`language` specifies the programming language of interest. The default is `"Stata"`, which only shows repositories where the majority of the source code is written with Stata. Specify `"all"` (or the name of a programming language) to search for repositories written in other programming languages.

`in` specifies the search scope. The default is `"name, description"`, which searches the repository names and their description. Alternative search scopes are `"name"`, `"description"`, `"readme"`, or `"all"`. The latter, includes results from repository's name, description, and its *README.md* file, if exists [5].

`all` asserts that repositories that are not installable Stata packages should also be included in the output. In other words, if the repository does not include the *packagename.pkg* and *stata.toc* to make it installable, it will be included in the search results. By default, only installable repositories are shown in the output.

`net` indicates that a general package search should be performed, including results from the `search` command. If this option is specified, `github search` will make a complete list of the relevant packages on GitHub as well as other web sources. Subsequently, it investigates the search output and notes their publication date. This will provide a hint regarding the version of the packages published on different platforms.

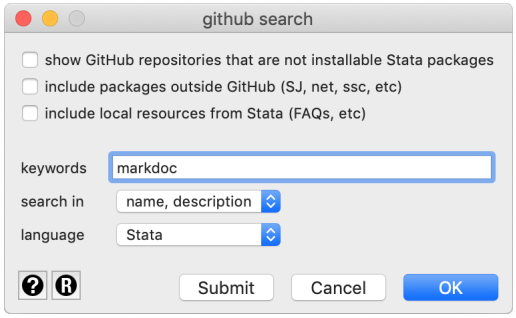
`local` appends the results of a local Stata search (e.g. help files, FAQs, Stata Journal manuscripts, etc.) in the `github search` results.

[4] GitHub automatically detects the main programming language of a repository using the linguist library (GitHub 2018a). However, consider that the majority of the code of a Stata package might be written in another programming language such as Java, Python, etc., and thus, the repository language might be miss-specified.

[5] GitHub recommends including a *README.md* file in every repository and further describe the repository

The package incorporates a search GUI with all options above, which can be launched by typing `db github`. For example, to search for the `markdoc` (Haghish 2016) package, one can execute `github search markdoc` command or alternatively, type the package name as the keywords in the GUI:

```
. db github
```



Repository	Username	Install	Description
markdoc	haghish	Install *11246k*	A literate programming package for Stata which develops dynamic documents, slides, and help files in various formats homepage http://haghish.com/markdoc updated on 2019-05-27 **Fork:**17 **Star:**37 **Lang:**Stata (dependency)

Figure 2: `github` search GUI and its output table

The command presents useful information in the search output. For example, it prints the latest update of the package, its homepage (if specified), and its main programming language. It also checks whether the repository is an installable Stata package and if so, an *Install* text appears in the output table that allows installing the package with a mouse click. Furthermore, the search command examines the package dependencies and displays a link to the *dependency.do* file, if found in the repository (see section 2.4.1), as shown in the lower-right corner of the output above.

Another way to look for Stata packages is to search keywords. For example, searching for the *literate programming* keywords would show any installable repository that includes both or any of these words in their repository name or description:

```
. github search literate programming
```

Repository	Username	Install	Description
markdoc	haghish	Install *11235k*	A literate programming package for Stata which develops dynamic documents, slides, and help files in various formats homepage http://haghish.com/markdoc updated on 2019-01-10 **Fork:**16 **Star:**38 **Lang:**Stata (dependency)
weaver	haghish	Install *1341k*	A Stata Log System in HTML or LaTeX for Dynamic Document and literate programming in Stata homepage http://www.haghish.com/weaver updated on 2018-11-04 **Fork:**2 **Star:**2 **Lang:**Stata (dependency)
literate_stata	mcallaghan	Install *541k*	No description, website, or topics provided. updated on 2016-02-23 **Fork:**1 **Star:**0 **Lang:**Stata

Figure 3: `Literate programming` search output

2.4 Installing GitHub repositories

The `query`, `check`, and `install` subcommands are to some extent related to package installation. The `query` subcommand displays the archived stable releases of a repository and allows installing an older release of a package, if desired (see section 5). The `check` subcommand tests whether the repository is an installable Stata package. Finally, the `install` subcommand downloads and installs the package and its dependencies, if there are any. These subcommands require the username and the repository name and follow a similar syntax, which is shown below.

`github` [*subcommand*] *username/repository* [`, stable` `version(`*str*`)`]

The `install` subcommand takes two options of `stable` and `version(`*str*`)`, which are further explained below:

`stable` installs the latest stable release of a software and is generally recommended for installing packages from GitHub

`version(`*str*`)` specifies a particular version (release tags) to be installed.

For example, let assume you would like to install the `rcall` package (Haghish 2019) from GitHub, which interfaces R within Stata interactively. The package is hosted on `github.com/haghish/rcall`, but if you did not the *username/repository* address, you could simply search for it as shown below and obtain the Username and Repository name from the search output.

```
. github search rcall
```

Repository	Username	Install	Description
rcall	haghish	Install *1106k*	Seamless interactive R in Stata. rcall allows communicating data sets, matrices, variables, and scalars between Stata and R conveniently homepage http://www.haghish.com/packa~p updated on 2019-02-28 **Fork:**12 **Star:**27 **Lang:**Stata (dependency)
stata-rcallst~t	luispfons~a	Install *19k*	Call R's stringdist package from Stata using rcall updated on 2019-04-24 **Fork:**1 **Star:**0 **Lang:**Stata
stata-rcallco~e	luispfons~a	Install *48k*	Call R's countrycode package from Stata using rcall updated on 2019-04-15 **Fork:**0 **Star:**1 **Lang:**Stata (dependency)

In this example, there are three Stata packages mentioning *rcall* in their repository name or description. A closer inspection reveals that two of the packages are built upon `rcall`, which is shown at the top of the results output. Using the Username and Repository name with the `check` subcommand confirms that the repository is an installable Stata package.

```
. github check haghish/rcall
toc file was found
pkg file was found
(the repository is installable)
```

In the same fashion, using the `query` subcommand, the archived stable releases of the package are displayed:

```
. github query haghish/rcall
```

Version	Release Date	Install
2.4.1	2018-11-01	Install
2.3.0	2018-03-02	Install
2.2.3	2017-12-06	Install
2.1.2	2017-10-10	Install
...	...	...
1.0.3	2016-07-15	Install

Finally, we can use the `install` subcommand to install the Stata package. It is recommended to use the `stable` option, to install the latest stable release of the package, if available. If no stable version is released by the repository maintainer, the command notifies you and installs the development version instead.

```
. github install haghish/rcall, stable
```

```
checking rcall consistency and verifying not already installed...
installing into /Users/haghish/Library/Application Support/Stata/ado/plus/...
installation complete.

Checking package dependencies
installing rcall package dependencies:


. /***
> Installing package dependency
> ============================
>
> The following R packages are required by rcall. rcall attempts to detect R
> Statistical Software on your system automatically and install the dependency
> R packages. If the installation fails, read the rcall help file and install
> the dependencies manually.
> ***/
.
. rcall_check

. rcall: install.packages("readstata13", repos="http://cran.us.r-project.org")
```

The `rcall` package has a dependency, which was also evident from the search output. I will address declaring package dependencies in the following section.

❑ **Technical note**

As shown from the returned results of the `query` subcommand, the last stable version of the `rcall` package to date is `2.4.1`, released on `2018-11-01`. We can install this version by clicking on the `install` link in the output table. Doing so will apply the `version` option to install a particular release of the package:

```
. github install haghish/rcall, version(2.4.1)
```

By avoiding the `stable` or `version` options or install a package using the `net install` command, the development version of a package is installed. Nonetheless, installing a stable release is generally recommended.

❑

2.4.1 Package dependencies

Package dependency is rather an unaddressed problem with Stata, particularly because SSC does not provide any procedure for declaring and installing dependencies. As a result, developers apply workaround tricks to check whether the required packages are installed. For example, they may search for a script file from the required dependency package. If the file was not found on the machine, an error is returned notifying the user that a dependency is missing. With such a clumsy workflow, developers have no control over the versions of the dependencies and that can often lead to bigger problems in a long run. From a different perspective, this problem motivates the developers to include all of their codes within a single package and avoid modulating their code into separate packages that can be used by other developers.

The `github` package provides a solution to this problem, allowing automatic installation of package dependencies. When a package is installed, the `github` command looks for a file named `dependency.do` within the repository and if

found, it executes it. The dependency file may use the `github install` command along with the `version` option to require a particular stable release.

2.5 Handling installed packages

The `list`, `update` and `uninstall` subcommands, as their names suggest, carry out the package management tasks. The general syntax of these subcommands is as follows:

`github` [*subcommand*] [*packagename*]

These subcommands only require the *packagename*. Perhaps the handiest of these subcommands is the `list`, which lists the installed packages and checks if there is an update available for any of them:

```
. github list
```

Date	Name	Version	user/repository	Latest release	
13 May 2019	github	1.9.7	haghish/github	1.9.7	
20 Dec 2018	markdoc	4.4.0	haghish/markdoc	4.4.5	(update)
20 Dec 2018	md2smcl	1.4	haghish/md2smcl	1.4	
23 Nov 2018	rcall	2.4.1	haghish/rcall	2.5.0	(update)
13 Mar 2019	statax	1.8	haghish/statax	1.8	
13 Mar 2019	weaver	3.4.3	haghish/weaver	3.4.3	

In the output above, the currently installed version and the latest release of the software is shown. If there is a newer release available, a clickable *update* text appears which allows updating the package. Alternatively, the same actions can be done using the `update` subcommand, followed by the package name. For example, to update the `rcall`, type:

```
. github update rcall
```

Finally, to remove the package, type:

```
. github uninstall rcall
```

❑ Technical note

When a package is installed using `github install`, its information is stored in an internal database. The internal database is named *github.dta* and is automatically created in the `plus\g\` directory. Anytime a package is installed, updated, or removed within the `github` command, the database is also updated accordingly. It is recommended that the users avoid using the `ado` command for removing or updating packages installed with `github` command, to keep the database consistent.

❑

3 Building package installation files

To install a repository, Stata demands package installation files encapsulating information necessary for installing and managing the package. For instance, package name, list of installation and/or ancillary files, publication date, author name, software description, etc. This information is stored within two files, named *stata.toc* and *packagename.pkg* and are created manually within a text editor. For detailed explanation about these two files, type `help usersite`.

The `github` package offers a structured framework as well as a GUI for building the installation files. The minimum required information are:

1. Name
2. A short title
3. Version
4. License
5. Author name
6. Email or contact information
7. Installation and/or ancillary files

The installation files can be script files, help files, or generally any file that should be copied to the user's machine within the installation. To select the installation or ancillary files, browse to the package directory and click on the files that should be installed, while holding the CTRL key (CMD on Mac). To launch the GUI, type:

```
. db make
```

The GUI calls the `make`[6] command to create the *packagename.pkg* and *stata.toc* files. Optionally, it can create two additional files, which are *README.md* and *make.do*, using the typed information. The latter preserves is code for re-building the package. Generating this file is recommended, especially if the package documentation is generated by `markdoc` (Haghish 2016) package, a procedure that is thoroughly discussed elsewhere (Haghish in preparation).

3.1 Example of building a Stata package

To demonstrate how a Stata package can be built using the `make` GUI, I have prepared a simple Stata package named `echo`. The package includes an ado-file named *echo.ado* and a help file named *echo.sthlp*. As the name suggests, the program only echoes a given string character. To follow the example, forking the repository using the following URL:

- `https://github.com/haghish/echo.git`

Next, change the working directory to where the forked repository is stored and then, launch the `make` GUI to fill in the inquired information, as shown in figure 4.

[6] for documentation type `help make`

```
. db make
```

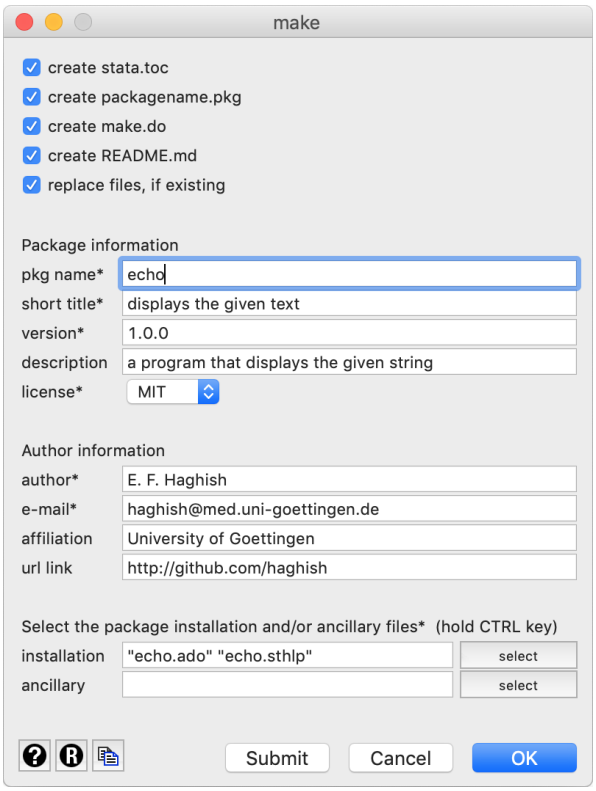


Figure 4: Example of information required for building the package installation

For this example, there are no ancillary files. Nevertheless, we have two installation files, an ado-file and a sthlp-file, that both should be selected. Next, executing the GUI will create the package installation files, which are *echo.pkg*, *stata.toc*. Uploading these files to the GitHub repository would suffice to make the repository installable. I also specified two additional files to be created, *make.do* and *README.md*. The generated files are shown in figure 5.

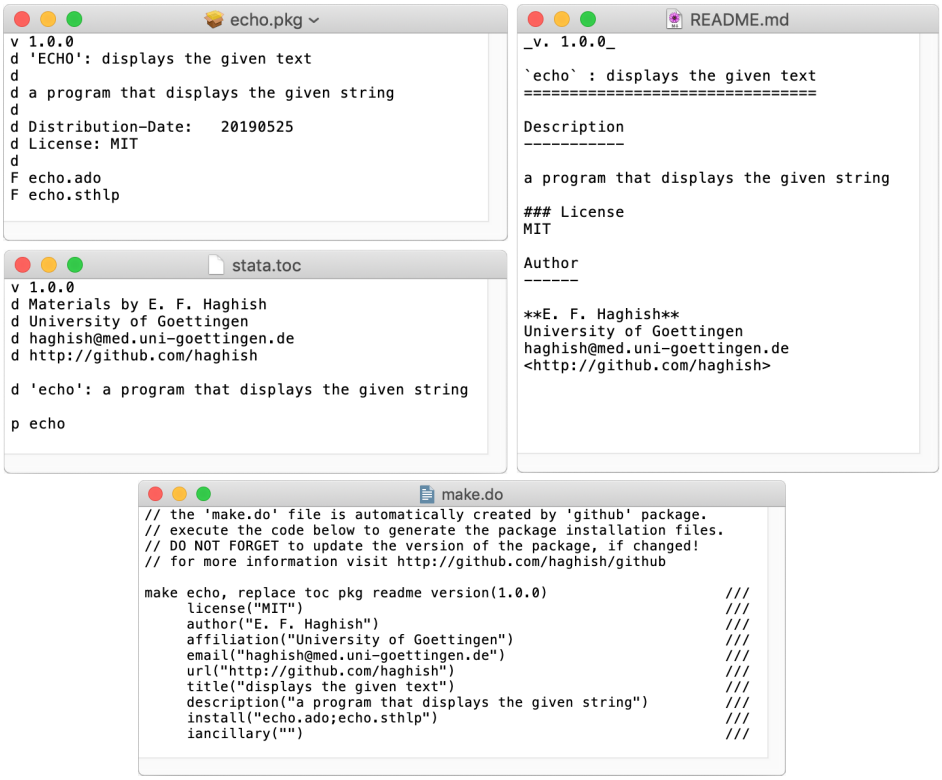


Figure 5: Creating installation and supplementary files with the `make` GUI

❑ **Technical note**

The interested reader can inspect the `github`[7], `markdoc`[8], and `rcall`[9] repositories for real-world examples of building package installations with the `make` command. All of these repositories include a *make.do* file to build the package, as well as the documentation and help files.

❑

4 Mining Stata packages on GitHub

The GitHub API limits the search results. However, it also allows searching for repositories based on creation date. This feature can be used to conduct a consecutive search for Stata repositories within a short time frame in order to mine Stata repositories. Next, the `github check` command can be applied to examine which of the repositories are installable Stata packages. The code for listing all Stata repositories is included in the Appendix.

[7] `https://github.com/haghish/github`
[8] `https://github.com/haghish/markdoc`
[9] `https://github.com/haghish/rcall`

The results of mining process are stored in three data sets. The *archive.dta* includes a list of all GitHub repositories that are written in Stata language as well as all repositories that mention Stata in their repository name or description. The *gitget.dta* is a subset of the *archive.dta* that is installable via Stata. Finally, the *githubfilese.dta*, lists all the installable files associated with each Stata package on GitHub. The *gitget.dta* and *githubfiles.dta* data sets are installed within the `github` package. However, the *archive.dta* data set is available from the main branch of the repository[10].

These data sets can bring new features to the `github` package. For example, the address information i.e. *username/repository* stored in *gitget.dta* can be used to simplify package installation, by requiring the package name only. The `gitget` command (type `help gitget`), which is a wrapper for `github install` and takes the same options, explores this idea. For example, to install the latest stable version of `markdoc` package, you can type:

```
. gitget markdoc, stable
```

The complete list of data packages recognized by the `gitget` command can be found in *gitget.md*[11] file in the repository. However, `gitget` is not a recommended alternative for software installation, because it is not updated in real-time. Another potential application of mining Stata packages is searching for a specific file-name, using `findfile` subcommand. Knowing that Stata does not allow different script files with identical names to coexist, programmers can examine their filenames to ensure similar names are not used by other packages. For example, let us search for any script file among Stata packages on GitHub that has the "dy"in its name, which yields the following results:

```
. github findfile dyn
```

File	Package	Repository
dynpandoc.ado	dynpandoc	huapeng01016/StataMarkdown
dynpandoc.sthlp	dynpandoc	huapeng01016/StataMarkdown
dynamicPage1.mata	sdp2016	wbuchanan/stataDataVizSDP2016

5 Discussion

From the 90s, the emergence of the internet made collaborations between geographically apart developers possible (Raymond 1999). This was of a particular importance for open source software development, which survives on volunteers contributing on their spare time (Alexander Hars 2002; Raymond 1999; Raymond and Enterprises 1997). In contrast to the early skepticism (Lewis 1999), collaborating on open source software has become a social norm, as implied by the enormous community of sites such as GitHub and Stackoverflow (Vasilescu

[10] `https://github.com/haghish/github/raw/master/archive.dta`
[11] `https://github.com/haghish/github/blob/master/gitget.md`

et al. 2013). Similarly, the community of Stata developers on social coding platforms is fast-growing and to this date, GitHub has become a home to hundreds of Stata package and thousands of Stata repositories. In fact, I am expecting the number of hosted Stata packages on GitHub to surpass SSC in a few years.

In this manuscript, I tried to fill the gap between Stata developers on GitHub and the rest of the community by providing a comprehensive software toolbox for searching, building, installing, and managing Stata packages from GitHub. In the same vein, I argued in favor of using GitHub for developing, maintaining, documenting, and even hosting statistical package. Besides, I warned the reader of some shortcoming of SSC archive because of discarding software version, license, dependency declaration, and most importantly, lack of archiving previous releases. Despite SSC, The Comprehensive R Archive Network (CRAN) (Claes et al. 2014) archives all the previous stable versions and yet, it also includes its hosted packages on Github [12] and archives the different versions via GitHub releases. Such a practice enables users to quickly navigate through different versions of a program and trace the changed codes across stable releases. A similar move from SSC would be very welcomed and effectively solves all the issue I have remarked.

The `github` package addressed SSC shortcomings, allowing access to previous releases of a software as well as automatizing the dependency installation. It also eases searching for Stata packages hosted on GitHub and allows installing any of the previous stable releases along with their dependencies.

There is a saying that goes "too many cooks spoil the broth."In my opinion, however, when it comes to coding and computational transparency, the more chefs participate, the better. I underscored that the primary benefit of GitHub is not its convenient and helpful software tools, but its community of experts. Its huge community makes GitHub a truly better place for developing, maintaining, and hosting statistical software and a step forward toward computational transparency.

6 References

Alexander Hars, S. O. 2002. Working for free? Motivations for participating in open-source projects. *International Journal of Electronic Commerce* 6(3): 25–39.

Angulo, M. A., and O. Aktunc. 2018. Using GitHub as a Teaching Tool for Programming Courses .

Barr, E. T., C. Bird, P. C. Rigby, A. Hindle, D. M. German, and P. Devanbu. 2012. Cohesive and isolated development with branches. In *International Conference on Fundamental Approaches to Software Engineering*, 316–331. Springer.

Chacon, S., and B. Straub. 2014. *Pro git*. Apress.

[12] https://github.com/cran

Claes, M., T. Mens, and P. Grosjean. 2014. On the maintainability of CRAN packages. In *Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), 2014 Software Evolution Week - IEEE Conference on*, 308–312.

Cross, R., S. P. Borgatti, and A. Parker. 2002. Making invisible work visible: Using social network analysis to support strategic collaboration. *California management review* 44(2): 25–46.

Dabbish, L., C. Stuart, J. Tsay, and J. Herbsleb. 2012a. Social coding in GitHub: transparency and collaboration in an open software repository. 1277–1286. ACM.

———. 2012b. Leveraging transparency. *IEEE software* 30(1): 37–43.

El Emam, K., S. Benlarbi, N. Goel, and S. N. Rai. 2001. The confounding effect of class size on the validity of object-oriented metrics. *IEEE Transactions on Software Engineering* 27(7): 630–650.

Feliciano, J., M.-A. Storey, and A. Zagalsky. 2016. Student experiences using GitHub in software engineering courses: a case study. In *Proceedings of the 38th International Conference on Software Engineering Companion*, 422–431. ACM.

Fischer, M., M. Pinzger, and H. Gall. 2003. Populating a release history database from version control and bug tracking systems. In *Software Maintenance, 2003. ICSM 2003. Proceedings. International Conference on*, 23–32. IEEE.

GitHub. 2018a. Linguist. URL `https://github.com/github/linguist`.

———. 2018b. Repository search. URL `https://github.com/search?q=is:public`.

———. 2018c. User search. URL `https://github.com/search?q=type%3Auser&type=Users`.

Gousios, G., M. Pinzger, and A. v. Deursen. 2014a. An exploratory study of the pull-based software development model. In *Proceedings of the 36th International Conference on Software Engineering*, 345–355. ACM.

Gousios, G., B. Vasilescu, A. Serebrenik, and A. Zaidman. 2014b. Lean GHTorrent: GitHub data on demand. 384–387. ACM.

Gousios, G., A. Zaidman, M.-A. Storey, and A. Van Deursen. 2015. Work practices and challenges in pull-based development: the integrator's perspective. In *Proceedings of the 37th International Conference on Software Engineering-Volume 1*, 358–368. IEEE Press.

Gruber, J. 2004. Markdown: Syntax. URL `http://daringfireball.net/projects/markdown/syntax`.

Haghish, E. F. 2016. markdoc: Literate Programming in Stata. *Stata Journal* 16(4): 964–988.

———. 2019. Seamless interactive language interfacing between R and Stata. *The Stata Journal* 19(1): 61–82. URL `https://doi.org/10.1177/1536867X19830891`.

———. in preparation. Software Documentation with markdoc 5.0. *Stata Journal* .

Hammack, S. G., L. O. Jundt, J. M. Lucas, and A. P. Dove. 2002. Version control and audit trail in a process control system. US Patent 6,449,624.

Jiang, J., D. Lo, J. He, X. Xia, P. S. Kochhar, and L. Zhang. 2017. Why and how developers fork what from whom in GitHub. *Empirical Software Engineering* 22(1): 547–578. URL `https://doi.org/10.1007/s10664-016-9436-6`.

Jiang, J., L. Zhang, and L. Li. 2013. Understanding project dissemination on a social coding site. 132–141.

Lewis, T. 1999. The open source acid test. *Computer* 32(2): 128–127.

Marlow, J., L. Dabbish, and J. Herbsleb. 2013. Impression formation in online peer production: activity traces and personal profiles in github. 117–128. ACM.

Miguel, E., C. Camerer, K. Casey, J. Cohen, K. M. Esterling, A. Gerber, R. Glennerster, D. P. Green, M. Humphreys, G. Imbens, et al. 2014. Promoting transparency in social science research. *Science* 343(6166): 30–31.

Raymond, E. 1999. The cathedral and the bazaar. *Knowledge, Technology & Policy* 12(3): 23–49.

Raymond, E. S., and T. Enterprises. 1997. The cathedral and the bazaar.

Thung, F., T. F. Bissyande, D. Lo, and L. Jiang. 2013. Network structure of social coding in GitHub. 323–326. IEEE.

Tichy, W. F. 1985. RCS—a system for version control. *Software: Practice and Experience* 15(7): 637–654.

Torvalds, L., and J. Hamano. 2010. Git: Fast version control system. *URL http://git-scm.com* .

Vasilescu, B., V. Filkov, and A. Serebrenik. 2013. Stackoverflow and github: Associations between software development and crowdsourced knowledge. 188–195. IEEE.

Vasilescu, B., S. Van Schuylenburg, J. Wulms, A. Serebrenik, and M. G. van den Brand. 2014. Continuous integration in a social-coding world: Empirical evidence from GitHub 401–405.

Zagalsky, A., J. Feliciano, M.-A. Storey, Y. Zhao, and W. Wang. 2015. The emergence of github as a collaborative platform for education. In *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing*, 1906–1917. ACM.

About the authors

Haghish is a statistician at the Department of Medical Psychology and Medical Sociology, University of Gttingen, Gttingen, Germany.

7 Appendix

The `github` package includes commands for mining Stata packages on SSC as well as GitHub.

7.1 Mining SSC packages

The `sscminer` command creates a data set that list all packages hosted on SSC and also specifies the installable files that are included within each package. Optionally, it also allows you to download all packages and store them in subdirectories. Finally, it also creates a zip-file for each package, based on the release date. The command below generates a data set named *archive.dta*, which analyzes the SSC packages. To download and archive all of the packages, add the `download` option. For more information, read the command help file by typing `help sscminer`.

```
. sscminer, save("archive.dta")
```

7.2 Mining GitHub packages

Mining Stata packages on GitHub is s complex process, because there might be Stata packages that are not recognized to be written in Stata language. Furthermore, GitHub API has a limit in the search results and thus, the mining process has to be carried out step by step, based on the frequency of the search results. The `githublistpack` program (for documentation type `help githublistpack`) carries out a consecutive search for Stata repositories and packages. In the code below, I search for 1) repositories with Stata language, 2) repositories in all languages that mention the keyword "Stata" within their repository name, description, or *README.md* file, and combine the results. The `githublistpack` command automatically examines the repositories to see whether they are installable Stata packages. Next, I will loop through each installable Stata package to see whether it includes a dependency file, as specified with *dependency.do* file (see section 2.4.1), and then, generate the `gitget.dta` data set, which is used by the `gitget` command.

```
// mining repositories in Stata language
githublistpack , language(Stata) append replace all in(all)  ///
    perpage(100) save("archive1") duration(1)

// mining stata-related repositories in all languages
githublistpack stata, language(all) append all in(all)       ///
    perpage(100) replace save("archive2") duration(1)

// merging the data sets
use "archive2.dta", clear
append using "archive1.dta"
duplicates drop address, force
drop _merge
saveold "archive.dta", replace

// checking for package dependency
```

```
use "archive.dta", clear
capture drop dependency
generate dependency = .

local j 0
local last = _N
forval N = 1/`last' {
  if installable[`N'] == 1 {
    local j = `j' + 1
    local address : di address[`N']
    capture githubdependency `address'
    if `r(dependency)' == 1 {
      display as txt "`N'/`last'"
      replace dependency = 1 in `N'
    }
  }
}

saveold "archive.dta", replace

// generating gitget data set
use "archive.dta", clear
keep if installable == 1
saveold "gitget.dta", replace
```