



The World's Largest Open Access Agricultural & Applied Economics Digital Library

This document is discoverable and free to researchers across the globe due to the work of AgEcon Search.

Help ensure our sustainability.

Give to AgEcon Search

AgEcon Search

<http://ageconsearch.umn.edu>

aesearch@umn.edu

*Papers downloaded from **AgEcon Search** may be used for non-commercial purposes and personal study only. No other use, including posting to another Internet site, is permitted without permission from the copyright owner (not AgEcon Search), or as allowed under the provisions of Fair Use, U.S. Copyright Act, Title 17 U.S.C.*

No endorsement of AgEcon Search or its fundraising activities by the author(s) of the following work or their employer(s) is intended or implied.

Developing, maintaining, and hosting Stata statistical software on GitHub

E. F. Haghish
PROMENTA Research Center
Department of Psychology
University of Oslo
Oslo, Norway
info@haghish.com

Abstract. The popularity of GitHub is growing, among not only software developers but also statisticians and data scientists. In this article, I discuss why social coding platforms such as GitHub are preferable for developing, documenting, maintaining, and collaborating on statistical software. Furthermore, I introduce the `github` command version 2.0 for Stata, which facilitates building, searching, installing, and managing statistical packages hosted on GitHub. I also provide a command for searching filenames in all Stata packages published on the Statistical Software Components Archive and GitHub to ensure unique filenames and package names, which is a common concern among Stata programmers. I make further suggestions to enhance the practice of developing and hosting statistical packages on GitHub as well as using them for data analysis.

Keywords: pr0073, github, version control, social coding, social computing, statistics software, data mining

1 Introduction

GitHub¹ 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 became 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 2019, there has been a rapid growth in the number of Stata repositories, and over 4,097 repositories and 749 installable community-contributed packages have been publicly published.² The resurgence in GitHub popularity for hosting Stata packages underscores the necessity for a command that facilitates accessing and managing them. In this article, I introduce the `github` package, which provides a handful of tools for searching, installing, building, and managing packages

1. <http://www.github.com>.

2. See Haghish (2019a) for the code and procedure of data mining GitHub and the Statistical Software Components (SSC) Archive.

that are hosted on GitHub. 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 and reveal a few problems that GitHub can ameliorate for Stata community-contributed software.

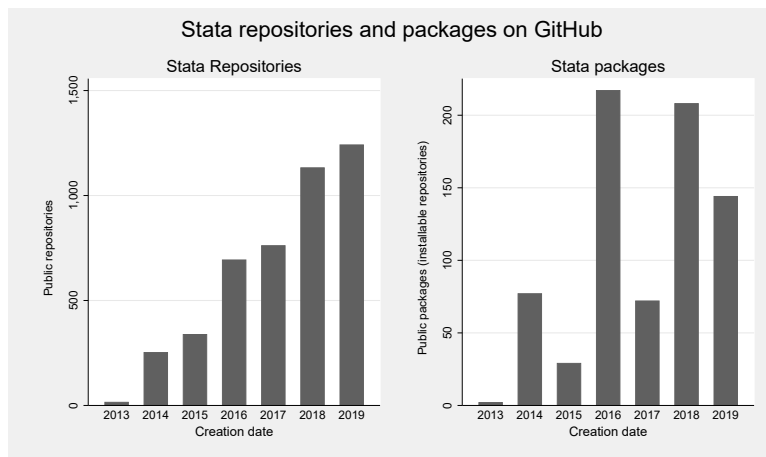


Figure 1. Public Stata repositories and packages on GitHub by year of creation

1.1 Developing statistical software on GitHub

At first glance, GitHub appears to be a multifaceted 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, Filkov, and Serebrenik 2013). Subsequently, the pro-social characteristics of GitHub promotes project dissemination by attracting attention and building reputation (Jiang, Zhang, and Li 2013), ergo peer reviewing the code by plenty of programmers.

The question, however, is, How can GitHub turn programming into a collective activity? At its core, GitHub is a combination of Git (Torvalds and Hamano 2010)—a popular distributed version control system—with a social media, which enables code integration (collaboration) through a two-level hierarchical network (Cross, Borgatti, and Parker 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 do not 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 a pull-based development model (Barr et al. 2012; Gousios, Pinzger, and Van Deursen 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 distributed version control 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, Pinzger, and Gall 2003).

There is a striving 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, that is, 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, Zhang, and Li 2013; Dabbish et al. 2012b). Studies have shown that GitHub users account for such information when evaluating a repository (Dabbish et al. 2012a; Jiang et al. 2017; Marlow, Dabbish, and Herbsleb 2013).

Despite the success of open-source software, there are many challenges associated with the open-source software development. For example, a lack of structured documentation and communication, which are required for 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 probable for Stata packages because most do not have an intricate structure. To evaluate this assertion, I mined and analyzed all Stata packages published on the Boston College Statistical Software Components (SSC) Archive and GitHub until January 1, 2019.³ The resulting datasets comprised 707 installable repositories (some of which included multiple Stata packages) hosted on GitHub and 2,807 packages hosted on the SSC Archive. Analyzing their content revealed that, respectively, 60% and 70% of the commands hosted on GitHub and the SSC Archive are composed of a single script file (that is, an ado-file or a Mata file). In the same vein, roughly 80% of packages on GitHub and 90% of packages on the SSC Archive included up to three script files only. This implies that most Stata packages have a relatively simple structure.

1.2 Hosting statistical software on GitHub

The SSC Archive has been the primary landing point for most Stata packages. Packages published on the SSC Archive are indexed by the `search` command after a while. But this convenience comes with several critical drawbacks that might make GitHub an excellent alternative:

1. The SSC Archive facilitates searching and installing a package. However, there is no way to download a package as a zip file for archiving it in a data-analysis project or inspecting it (see the technical note below).

3. The data and analysis code are available on <http://github.com/haghish/githubtools>.

2. The SSC Archive does not require specifying or report any information about version and license of hosted packages.
3. The SSC Archive hosts only the latest release of the package, without archiving the previous releases. Along with lack of version specification, this can pose threats to research reproducibility.
4. There is no mechanism to cite and install dependencies on the SSC Archive (see section 2.4), which cannot be accomplished without archiving previous releases either. Otherwise, relying on other community-contributed software without declaring⁴ 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) a development version, which is often the main branch of the repository, and 2) a stable release. The latter is significant 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, GitHub can host not only 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 they would do 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 set up only the main branch (which is most likely to be the development version). In the following sections, I introduce `github` command version 2.0, which 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 `ssczip`⁵ program can download any package from the SSC Archive as a zip file. 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.

□

1.3 Addressing the problem of unique filenames

Stata lacks modularized package loading, in contrast to R or Python. Therefore, all community-contributed script files are stored in shared directories. Thus, there is a

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

5. For installation, type `github install haghish/ssczip, stable`.

possibility of overwriting a script file if an identical filename is used in two installed packages. This problem can be ameliorated by checking whether a particular filename exists in other packages. The `github` package includes a command for searching through all known packages, helping developers to ensure unique filenames. Moreover, it includes a command for building Stata package installations that automatically examines the filenames (see section 3).

2 The `github` command

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

1. searching GitHub application programming interface (API), <http://www.stata.com>, and other web sources for Stata packages.
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 the building of packaging files (that is, the `pkg` and `toc` files) to make a GitHub repository installable.

2.1 Installation

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

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

The Stata programs that are used for mining GitHub and the SSC Archive are included in the `githubtools` repository and can be optionally installed as follows:

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

2.2 Syntax

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

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

Table 1. Summary of `github`'s subcommands

Subcommands	Description
Essential	
<code>list</code>	expedites managing packages installed with <code>github</code>
<code>search</code>	looks for packages or repositories via GitHub API
<code>install</code>	installs a package along with its dependencies
<code>uninstall</code>	removes a package from Stata
<code>update</code>	updates a package to the latest version
Supplementary	
<code>version</code>	returns the version of an installed package
<code>query</code>	returns all archived stable versions of a package
<code>check</code>	tests whether a repository is an installable Stata package
<code>findfile</code>	searches filenames in all Stata packages on the SSC Archive and GitHub

Apart from `list` (which does not 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(string) in(string) all net local
    save(filename) replace]
```

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 language⁶ (default) or expand the search by including results from Stata's **search** command. These options are further explained below:

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

in(*string*) specifies the search scope. The default is **in**("name, description"), which searches the repository names and their description. Alternative search scopes are "name", "description", "readme", or "all". The latter includes results from the repository's name, description, and its README.md file, if it exists.⁷

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 *package-name.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 the 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 (help files, FAQs, *Stata Journal* articles, etc.) in the **github search** results.

save(*filename*) stores the search results in a dataset.

replace replaces the existing dataset.

6. 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 misspecified.

7. GitHub recommends including a README.md file in every repository and further describing the repository.

The package incorporates a search GUI with all options above that can be launched by typing `db github`. For example, to search for the `markdoc` (Haghighi 2016) package, one can type `db github` and then type the package name as the keywords in the GUI (see figure 2):

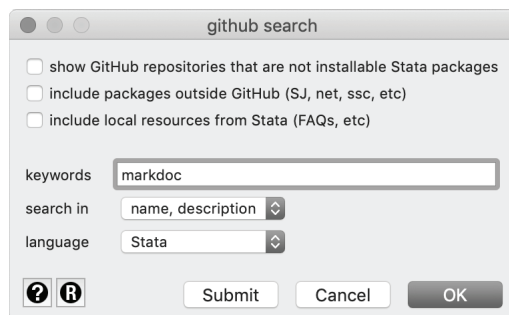


Figure 2. `github search` GUI

Alternatively, one can execute the `github search markdoc` command:

```
. github search markdoc
```

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

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; 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), 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 the repository name or description as shown below:

```
. 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

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 4). 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(string)]
```

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(string) specifies a particular version (release tags) to be installed.

For example, let us assume you would like to install the `rcall` package (Haghish 2019b) from GitHub, which interfaces R within Stata interactively. The `rcall` package is hosted on <http://github.com/haghish/rcall>, but if you did not know 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 Rs 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 Rs countrycode package from Stata using rcall updated on 2019-04-15 Fork:0 Star:1 Lang:Stata (dependency)

In this example, three Stata packages mention `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, with 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
(output omitted)		
1.0.3	2016-07-15	Install

Finally, we can use the `install` subcommand to install the Stata package. I recommend using 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()` option or installing a package using the `net install` command, one installs the development version of a package. Nonetheless, installing a stable release is generally recommended.

□

Package dependencies

Package dependency is a somewhat unaddressed problem with Stata, particularly because the SSC Archive does not provide any procedure for declaring and installing dependencies. Thus, developers apply work-around 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 the 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 require only the *packagename*. Perhaps the handiest of these subcommands is `list`, which lists the installed packages and checks whether 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 are shown. If there is a newer release available, a clickable `update` text appears that 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. I recommend that users, to keep the database consistent, avoid using the `ado` command for removing or updating packages installed with the `github` command. Nevertheless, the `github list` command examines whether any package has been removed without using the `github uninstall` command and updates the database accordingly.

□

3 github for developers

Stata developers can make use of two features of the `github` package. First and foremost, the package can search through all Stata packages on GitHub and the SSC Archive to find out whether a given filename is used elsewhere. Second, it provides a systematic framework for building and updating the package installation files. The latter greatly simplifies taking care of packaging and rebuilding a Stata software on GitHub and also automatically checks whether the package filenames exist elsewhere. These two features are described below.

3.1 Ensuring unique filenames

As mentioned in section 1.3, Stata community-contributed packages might overwrite one another if they include identical filenames. Having an archive of all known Stata packages and their filenames can guide programmers to check whether their filenames are unique. The mentioned database is named `githubfiles.dta` and is included within the `github` package. The code for building the database is available on <http://github.com/haghish/githubtools>.

The `github findfile` command can search through the database and return any file that includes the given string in its name. For example, let us search for any script file that has “mark” in its name, which yields the following results:

```
. github findfile mark
```

File	Package	Repository
mark_changes.hlp	mark_changes	SSC
mark_changes.ado	mark_changes	SSC
markdoc.sthlp	markdoc	haghigh/markdoc
markdoc_formal.docx	markdoc	haghigh/markdoc
markdoc.ado	markdoc	haghigh/markdoc
markdoccheck.ado	markdoc	haghigh/markdoc
markup.ado	markdoc	haghigh/markdoc
markdoc_stata.docx	markdoc	haghigh/markdoc
markdocpandoc.ado	markdoc	haghigh/markdoc
markdocversion.ado	markdoc	haghigh/markdoc
markdoc.dlg	markdoc	haghigh/markdoc
markdoc_simple.docx	markdoc	haghigh/markdoc
markdocstyle.ado	markdoc	haghigh/markdoc
markdoc_title.dlg	markdoc	haghigh/markdoc
markdoc_minimal.docx	markdoc	haghigh/markdoc
markdocwkhtmltopdf.ado	markdoc	haghigh/markdoc
markov.ado	markov	SSC
markov.hlp	markov	SSC
markstat.css	markstat	SSC
markstat.docx	markstat	SSC
markstat.ado	markstat	SSC
markstat.sthlp	markstat	SSC
markstats5.zip	markstat	SSC
marktouse.ado	marktouse	SSC
marktouse.hlp	marktouse	SSC
markdown.dlg	stmd	SSC

The output shows the files and packages that include “mark” in their name and whether they are hosted on GitHub or the SSC Archive.

3.2 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 ancillary files, publication date, author name, software description). This information is stored within two files, named `stata.toc` and `packagename.pkg` and 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 is the following:

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

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, while holding the *CTRL* key (*Command* on Mac), click on the files that should be installed. To launch the GUI, type

```
. db make
```

The GUI calls the `make`⁸ command (included within the `github` package) 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 includes the code for rebuilding the package installation. Generating this file is recommended, especially if the package documentation is generated by the `markdoc` (Haghish 2016) package, a procedure that is thoroughly discussed elsewhere (Haghish 2020). Simultaneously, the `make` command also examines whether each filename has been published elsewhere, whether on the SSC Archive or GitHub.

3.3 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 character string. To follow the example, fork 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 launch the `make` GUI to fill in the required information, as shown in figure 3.

8. For documentation, type `help make`.


```
. db make
```

The screenshot shows a 'make' dialog box with the following fields and options:

- ☒ create stata.toc
- ☒ create packagename.pkg
- ☒ create make.do
- ☒ create README.md
- ☒ replace files, if existing

Package information

pkg name*

short title*

version*

description

license*

Author information

author*

e-mail*

affiliation

url link

Select the package installation and/or ancillary files* (hold CTRL key)

installation

ancillary

Buttons: ? R [icon] Submit Cancel OK

Figure 3. 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 help file, that should both be selected. Next, executing the GUI will create the package installation files, which are `echo.pkg` and `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 4.

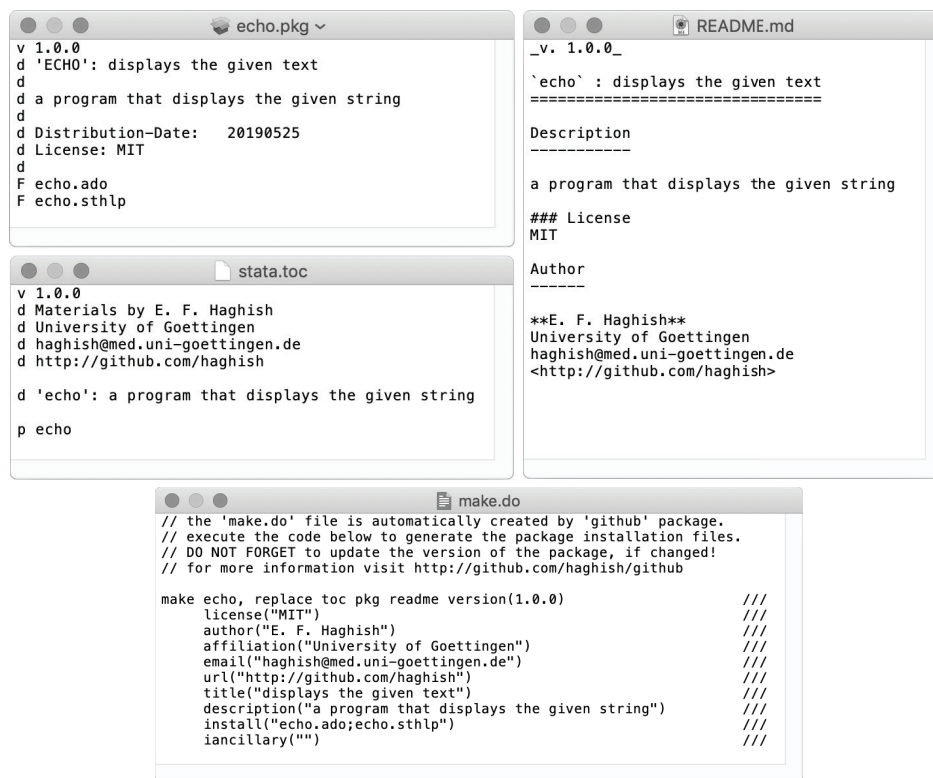


Figure 4. Creating installation and supplementary files with the `make` GUI

□ Technical note

The interested reader can inspect the `github`⁹, `markdoc`¹⁰, and `rcall`¹¹ 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 Discussion

In the '90s, the emergence of the Internet made collaborations between geographically separated developers possible (Raymond 1999). This was particularly important for open-source software development, which survives on volunteers contributing during their spare time (Hars and Ou 2002; Raymond 1999, 2001). In contrast to the early skep-

9. <https://github.com/haghish/github>.

10. <https://github.com/haghish/markdoc>.

11. <https://github.com/haghish/rcall>.

ticism (Lewis 1999), collaborating on open-source software has become a social norm, as implied by the enormous community of sites such as GitHub and Stack Overflow (Vasilescu, Filkov, and Serebrenik 2013). Similarly, the community of Stata developers on social coding platforms is growing fast, and GitHub has become a home to hundreds of Stata packages and thousands of Stata repositories.

In this article, 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 a statistical package. I also warned the reader of some shortcomings of the SSC Archive because of its issues with discarding software versions, licenses, dependency declarations, and, most importantly, lack of archiving previous releases. Unlike the SSC archive, the Comprehensive R Archive Network (CRAN) (Claes, Mens, and Grosjean 2014) archives all the previous stable versions and also includes its hosted packages on Github¹² 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 the SSC Archive would be welcomed and effectively solve all the issues I have listed.

The `github` package addressed the SSC Archive 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 one to install 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 involved, 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 good place for developing, maintaining, and hosting statistical software and stepping toward computational transparency.

5 Programs and supplemental materials

The `github` command is hosted on GitHub and can be installed by typing

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

6 References

- Barr, E. T., C. Bird, P. C. Rigby, A. Hindle, D. M. German, and P. Devanbu. 2012. Cohesive and isolated development with branches. In *Fundamental Approaches to Software Engineering. FASE 2012. Lecture Notes in Computer Science*, vol. 7212, ed. J. de Lara and A. Zisman, 316–331. Berlin: Springer. https://doi.org/10.1007/978-3-642-28872-2_22.

12. <https://github.com/cran>.

- Chacon, S., and B. Straub. 2014. *Pro Git*. 2nd ed. New York: Apress.
- Claes, M., T. Mens, and P. Grosjean. 2014. On the maintainability of CRAN packages. In *2014 Software Evolution Week—IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE)*, 308–312. Antwerp, Belgium: IEEE. <http://doi.org/10.1109/CSMR-WCRE.2014.6747183>.
- 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: 25–46. <https://doi.org/10.2307/41166121>.
- Dabbish, L., C. Stuart, J. Tsay, and J. Herbsleb. 2012a. Social coding in GitHub: Transparency and collaboration in an open software repository. In *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work*, 1277–1286. Seattle, WA: ACM. <https://doi.org/10.1145/2145204.2145396>.
- . 2012b. Leveraging transparency. *IEEE Software* 30: 37–43. <https://doi.org/10.1109/MS.2012.172>.
- 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: 630–650. <https://doi.org/10.1109/32.935855>.
- Fischer, M., M. Pinzger, and H. Gall. 2003. Populating a release history database from version control and bug tracking systems. In *International Conference on Software Maintenance, 2003. ICSM 2003. Proceedings*, 23–32. Amsterdam: IEEE. <https://doi.org/10.1109/ICSM.2003.1235403>.
- GitHub. 2018a. Linguist. <https://github.com/github/linguist>.
- . 2018b. Repository search. <https://github.com/search?q=is:public>.
- . 2018c. User search. <https://github.com/search?q=type%3Auser&type=Users>.
- Gousios, G., M. Pinzger, and A. Van Deursen. 2014a. An exploratory study of the pull-based software development model. In *Proceedings of the 36th International Conference on Software Engineering*, 345–355. Hyderabad, India: ACM. <https://doi.org/10.1145/2568225.2568260>.
- Gousios, G., B. Vasilescu, A. Serebrenik, and A. Zaidman. 2014b. Lean GHTorrent: GitHub data on demand. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, 384–387. Hyderabad, India: ACM. <https://doi.org/10.1145/2597073.2597126>.
- 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—vol. 1*, 358–368. Florence, Italy: IEEE. <https://doi.org/10.1109/ICSE.2015.55>.

- Haghighi, E. F. 2016. markdoc: Literate programming in Stata. *Stata Journal* 16: 964–988. <https://doi.org/10.1177/1536867X1601600409>.
- . 2019a. Mining Stata packages to confirm unique package names and file names. <https://github.com/haghighi/miningtools>.
- . 2019b. Seamless interactive language interfacing between R and Stata. *Stata Journal* 19: 61–82. <https://doi.org/10.1177/1536867X19830891>.
- . 2020. Software documentation with markdoc 5.0. *Stata Journal* 20: 336–362. <https://doi.org/10.1177/1536867X20931000>.
- Hammack, S. G., L. O. Jundt, J. M. Lucas, and A. P. Dove. 2002. Version control and audit trail in a process control system. U.S. Patent 6,449,624.
- Hars, A., and S. Ou. 2002. Working for free? Motivations for participating in open-source projects. *International Journal of Electronic Commerce* 6: 25–39. <https://doi.org/10.1080/10864415.2002.11044241>.
- 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: 547–578. <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. In *2013 20th Working Conference on Reverse Engineering (WCRE)*, 132–141. Koblenz, Germany: IEEE. <https://doi.org/10.1109/WCRE.2013.6671288>.
- Lewis, T. 1999. The open source acid test. *IEEE Computer* 32: 125–128. <https://doi.org/10.1109/2.745728>.
- Marlow, J., L. Dabbish, and J. Herbsleb. 2013. Impression formation in online peer production: Activity traces and personal profiles in GitHub. In *Proceedings of the 2013 Conference on Computer Supported Cooperative Work*, 117–128. San Antonio, TX: ACM. <https://doi.org/10.1145/2441776.2441792>.
- Miguel, E., C. Camerer, K. Casey, J. Cohen, K. M. Esterling, A. Gerber, R. Glennerster, D. P. Green, M. Humphreys, G. Imbens, D. Laitin, T. Madon, L. Nelson, B. A. Nosek, M. Petersen, R. Sedlmayr, J. P. Simmons, U. Simonsohn, and M. V. der Laan. 2014. Promoting transparency in social science research. *Science* 343: 30–31. <https://doi.org/10.1126/science.1245317>.
- Raymond, E. 1999. The cathedral and the bazaar. *Knowledge, Technology & Policy* 12: 23–49. <https://doi.org/10.1007/s12130-999-1026-0>.
- Raymond, E. S. 2001. *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*. Rev. ed. Sebastopol, CA: O'Reilly.
- Thung, F., T. F. Bissyande, D. Lo, and L. Jiang. 2013. Network structure of social coding in GitHub. In *2013 17th European Conference on Software Maintenance and Reengineering*, 323–326. Genova, Italy: IEEE. <https://doi.org/10.1109/CSMR.2013.41>.

- Tichy, W. F. 1985. RCS—A system for version control. *Software: Practice and Experience* 15: 637–654. <https://doi.org/10.1002/spe.4380150703>.
- Torvalds, L., and J. Hamano. 2010. Git: Fast version control system. <http://git-scm.com>.
- Vasilescu, B., V. Filkov, and A. Serebrenik. 2013. StackOverflow and GitHub: Associations between software development and crowdsourced knowledge. In *2013 International Conference on Social Computing*, 188–195. Alexandria, VA: IEEE. <https://doi.org/10.1109/SocialCom.2013.35>.
- Vasilescu, B., S. van Schuylenburg, J. Wulms, A. Serebrenik, and M. G. J. van den Brand. 2014. Continuous integration in a social-coding world: Empirical evidence from GitHub. In *Proceedings of the International Conference on Software Maintenance and Evolution*, 401–405. Washington, DC: IEEE. <https://doi.org/10.1109/ICSME.2014.62>.

About the author

E. F. Haghish is a researcher at the Department of Psychology, University of Oslo, Oslo, Norway.

A Appendix

The `miningtools` package (Haghish 2019a) includes commands for mining Stata packages on the SSC Archive as well as GitHub. These commands can build a complete list of Stata packages available via the SSC Archive and GitHub. The repository is available via <http://github.com/haghish/githubtools> and also includes code and instructions for mining Stata packages on the SSC Archive and GitHub. To install the `miningtools` package, type

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

The `github` package includes two mined datasets, `githubfiles.dta`, which includes the filenames of all Stata packages, and `gitget.dta`, which includes installable package names and their relevant information. Figure 1 was created using a mined dataset, and the code for reproducing the figure can be accessed from https://github.com/haghish/github/blob/master/article_code.do.