

R & GIS: Geospatial Plotting

Florian Endel* Peter Filzmoser**

* FFG IFEDH project, Student at Vienna University of Technology
(e-mail: florian@endel.at).

** Department of Statistics and Probability Theory, Vienna University
of Technology (e-mail: P.Filzmoser@tuwien.ac.at)

Abstract: Examples of spatial data within the R environment and the combination of R with data sets, spatial tools, libraries and other software products, which are common in real life environments, are provided in this paper.

Beginning with the setup of a new project using Git and an online repository, a Document build by Sweave - a combination of R and L^AT_EX - is explained. Once the fundamental setup is working rudimentarily, the import of data and (geo-) spatial information from different sources is shown. Finally some examples of spatial plots using the *sp* package in R are included.

Additionally some tools like "integrated development environments" (IDE), which may be supportive during the daily work and help newcomers learning the presented techniques, are mentioned and specific programs are recommended.

This paper itself is build using the described technologies and therefore illustrates all applied methods.

Keywords: R, Sweave, L^AT_EX, Spatial Data, Git

1. INTRODUCTION

Building a sophisticated report including spatial data, plots and usefule information (mostly) takes some effort.

Vast online resources and big data sets may overstrain even modern desktop computers, especially if common and established methods and routines need to scale up for new information. Working in groups or spatial distributed teams, changing requirements and specification, copyright issues, reproduction of foreign (or even own) results and updating reports using new data sets are common problems during the lifetime of a project. Additionally the amount of time, manpower and funding may be sometimes quite short.

The adaption of technologies presented in this paper may help overcome some of these challenges. Although there are great resources beginning with basic tutorials up to sophisticated books for every delineated software product, useful example of the whole workflow are still quite rare. Therefore the practical utilization of one single technique may be well documented, but the implementation and combination of several software tools during ongoing projects may still be challanging.

Additionally the awareness of great open source (Open Source Initiative: Definition of Open Source <http://www.opensource.org/osd.html>) software products, which are definitely ready to be used on a regular basis, especially the power of "The R Project for Statistical Computing", should be extended.

This paper itself is build using the described technologies and therefore illustrates all applied methods. To render the understanding and reproduction of the document considerable simple and provide an example of the methods used, all sourcecode is available online.

2. STRUCTURE

The structure of this report is tailored towards a possible workflow of a (simple) research project.

Beginning with the setup of a (online) repository to track changes and manage contributions of different team members, the structure of a Sweave document is explained. Using different data sets, partly provided within R packages, partly imported from other sources, the creation of geospatial plots is illustrated.

It should be obvious, that only a basic introduction and different examples can be provided in this document. Further details about this report can be looked up in the online repository. Additonally some useful online ressources and other references are included.

3. SETUP OF THE PROJECT

Before exploring the data and writing the report can be started, git has to be setup. On the official homepage of the git project, the following short definition is provided¹

Git is a free & open source, distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

* Forschungsförderungsgesellschaft (FFG) COIN Project <http://www.ffg.at/coin-programmlinie-kooperation-und-netzwerke>

¹ <http://git-scm.com/> [2012-01-28]

Every Git clone is a full-fledged repository with complete history and full revision tracking capabilities, not dependent on network access or a central server. Branching and merging are fast and easy to do.

After installing the git software (binaries are available on the mentioned website and in the repositories of almost every linux distribution), a folder for the new project is created on the local file system.

If there is absolutely no possibility to install the official git compilation, a pure Java implementation of git called JGit (jgit homepage: <http://eclipse.org/jgit/>) can be used. JGit does not need any installation or extended administrative rights, be a current "Java Runtime Environment" (JRE) has to be installed. Besides the EGit plugin of the Eclipse "Integrated Development Environment" (IDE) there are hardly any additional (graphical) tools which are able to utilize JGit's functionality. Therefore users are bound to Eclipse or the command line interface.

Git also provides a graphical frontend called "Git Gui". Many tasks ranging from basic management of local and remote repositories up to complex merging can be achieved with it. Especially for novice git users Git Gui may provide a more even learning curve than the command line interface (CLI).

Additionally there are some other graphical user interfaces (GUI) which simplify the usage of git. Eclipse IDE (<http://www.eclipse.org/>) includes the EGit (<http://eclipse.org/egit/>) plugin, a graphical git frontend based on the mentioned JGit implementation.

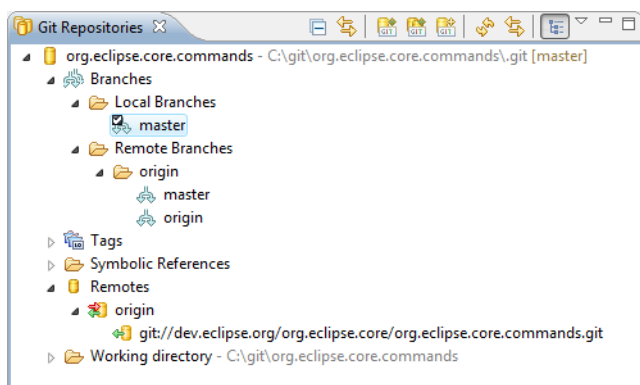


Fig. 1. git repositories presented by EGit / JGit in Eclipse IDE
screenshot provided by <http://eclipse.org/egit/> [2012-01-29]

Besides different, mostly platform dependent tools, SmartGit represents a current and very functional GUI, which is available on all popular operating system. Although SmartGit is sold commercially, a free version for non-commercial projects offered.

Additionally RStudio, a quite new development environment for R supports git (and subversion) directly since January 2012.

3.1 git init & clone

Now the local git repository can be set up by invoking the command line command `git init`. The `git clone` command can be used for creating an exact copy of an available remote repository.

How to clone (copy) the source of this paper, including the whole changelog, to a local folder and thereby create a local repository as an exact replica of the online version, is shown in line 1 of *Listing 1*. After the download has finished, the document can be compiled using the `pdflatex` command.

Note that the example in *Listing 1* shows how to compile the \LaTeX file generated by R and Sweave (both are described in chapter 4). It is necessary to run `pdflatex` two times to produce all references and listings correctly.

Although the implementation of the sources takes place on different linux distributions, the compilation should flawlessly work on other operating systems (OS). While the tools themselves are used in the same way on all popular systems, some details - like changing the directory using a terminal and setting up the search path to the binaries - may be different. If the creation of the PDF-file does not work on a specific setup, a prefabricated version is also located in the git repository.

Listing 1. Clone git repository of this paper and compile the included \LaTeX file

```
1 git clone git://github.com/FlorianEndel/
   Mathmod2012.git
2 cd Mathmod2012
3 pdflatex
   Mathmod2012_R_Geospatial_Plotting.tex
4 pdflatex
   Mathmod2012_R_Geospatial_Plotting.tex
```

After creating a new local git repository or cloning remote sources, everything is ready for editing the contents. Git does not anticipate specific software tools like special editors or file browsers. Any file in the repository can be edited using any favorite program. Solely the folder (including all contents) `.git` has to be ignored. Otherwise the local repository may be damaged.

3.2 git commit, pull, push

All changes can be permanently added to git's history (the changelog) using `git commit` followed by a summary of all changes made (line 1 in listing 2). A committed version of the whole project can be restored later on and users are able to survey all changes over time. This may be quite useful if something breaks during the implementation of a new feature or adjustment of existing code. Also try and error approaches may be applied, because resetting a previous state of the whole project is quite easy and fast. If the status of the whole project at a specific point in time is necessary, for example to compare published and updated versions or include comments from a review process, activating a previous commit using `git checkout [commit-id]` may be feasible.

After committing changes to the *local* repository, it is possible to merge the local version with the remote repository (if any exists and is configured) and provide all alterations to other team members or a public audience. Before someone pushes the local version to a (central) git server, all changes made by other people have to be downloaded and included into the own *branch* of the projects source. Downloading and merging can easily be carried out using *git fetch* and *git merge*. The command *git pull* is a combination of the two previous steps.

If there are no conflicting changes of the source code between the remote and the local versions, git is normally able to merge everything on its own flawlessly. Sooner or later every team faces the problem of conflicting and failing merges. The solution for such issues is not straight forward. Because the official documentation of git (<http://git-scm.com/documentation>) and a quick google.com search may be most helpful, there is no explanation provided within this paper.

Especially during manual merging of conflicting source files, a graphical interface may be very helpful.

Just as the (automatic or manual) merging is finished, it is highly recommended to compile the Sweave and L^AT_EX file again to be sure that no new errors were introduced. When the merging and testing procedure is successfully finished, the local changes can be loaded to another remote repository using *git push*.

As uploading (pushing) needs the right to write on a remote location, more configuration is normally necessary. As there are several possibilities to authenticate to a remote location, including different additional tools, signation and encryption, no general explanation can be provided.

The source of this paper is hosted by the free (git) provider github.com. As shown in Listing 1, it is quite easy to copy (clone) a project from their webserver. Uploading (pushing) needs some extra configuration which is explained in the excellent tutorial provided on help.github.com (pushing: help.github.com/remotes/#pushing).

Listing 2. pull and push of a local git repository without merge conflicts

```
1 git commit -m 'Summary of changes made'
2 git pull
3 pdflatex
   Mathmod2012_R_Geospatial_Plotting.tex
4 pdflatex
   Mathmod2012_R_Geospatial_Plotting.tex
5 # everything alright?
6 git push
```

3.3 git: further features

There are much more useful features of git waiting to be explored and used. Yet again official documentation on <http://git-scm.com/documentation> and practical tutorials, for example from help.github.com, are recommended.

In the following listing, the most important 'advanced' features are summarized (in alphabetical order):

- add** *git add* is used to add newly created files (within the folder managed by git) to the current repository.
- branch** Different branches of the same codebase can be edited independent from another. In fact, every local clone of a remote repository represents a (remote tracking) branch. Branches may be combined using *merge*.
- checkout** Switching between different branches (and even create new branches) can be established using *git checkout*. Even previously committed versions can be activated.
- log** Using the command *git log* all previous commit messages of the currently activated branch are shown.
- rebase** Rebase allows to easily change a series of commits, reordering, editing, or squashing commits together into a single commit. It is highly recommended that manipulating the history is performed with great care, because this is one of the rare possibilities to actually loose changes or (under special circumstances) even destroy the repository.
- reset** The *git reset* command is one of the most complicated functions of git. Be aware that using *reset* is one of the rare possibilities to actually loose changes! Resetting files or complete (committed) versions of a branch may be quite useful. A deeper understanding of git's different states - unstaged, staged, committed - is necessary to effectively use *reset*. A short and informative introduction to the 'normal workflow' of git can be found here: <http://learn.github.com/p/normal.html>.
- status** *git status* shows the status of the currently activated branch. Most notably new and changed files which may be involved in the next commit are listed.
- tag** Tags are special signs of committed versions. For example a specific revision of the repository of this paper (listing 1) is marked (tagged) as published.

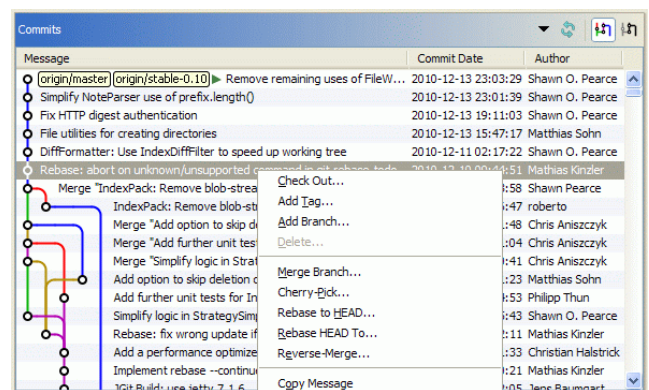


Fig. 2. example for a git changelog and history of commits visualized with Smartgit
screenshot provided by <http://www.syntevo.com/smartgit> [2012-01-29]

4. L^AT_EX, R & SWEAVE

Once the git repository is set up, the Sweave document has to be arranged. In the following paragraphs, involved technologies are briefly introduced.

Some central properties are shared by all of them. Availability under a open source license, platform independence, multilingualism, lots of documentation and a strong and proven community willing to help are the most important characteristics of these

4.1 R

R (R Development Core Team, 2011) is known as an upcoming statistical environment.

As an open source successor of S, it is known to be a powerful and flexible programming language and computing environment. Especially everyone's ability to utilize and contribute to the very rich environment of additional packages hosted mainly on "The Comprehensive R Archive Network" are good reasons to give R a try. More advanced users (spelling "user" and "developer" with capital 'R's is common in the R community) may use different extensions in addition to the variety of basic packages a normal installation offers. Most of the more sophisticated packages provide worthwhile introductions and even extensive literature.

R is mainly used by a command line interface. Although this - in comparison to current graphical user interfaces outdated - way to communicate with a computer program got a quite steep learning curve, practical advantages induce many users to learn and stick with it. Mainly the ease of combining playful exploration of datasets and automatic reproduction of any steps taken using script- or batch-files are advantageous.

Also the following examples, using special packages providing methods for spatial data, may be executed on a simple terminal, a graphical frontend to R, a remote server or inside a Sweave document. This flexibility enables the usage of the same R source code within a database², a local PC and even servers and cloud services.

Of course there are also some graphical user interfaces which may support the usage of R. The most popular and beginner friendly solutions are Eclipse with the StatET plugin and first of all the lately published RStudio (<http://rstudio.org/>). Both are open source and platform independent solution, which offer many other advantages like project management, integration of version management (git), support for Sweave and a client/server separation.

While Eclipse provides considerably more additional plugins and possibilities, RStudio can be highly recommended to all users.

4.2 Sweave

Sweave is defined in the official manual (Leisch, 2002) as follows:

Sweave provides a flexible framework for mixing text and S code for automatic document generation. A single source file contains both documentation text and S code, which are then woven into a final document

As explained above, R is an open source implementation of S. Therefore both language may be interchangeable.

In 2002, Sweave was presented in the former "R Journal" (<http://journal.r-project.org/>) called "R News" (official archive of R News: <http://cran.r-project.org/doc/Rnews/>). In the first part of this introduction, Sweave is summarized briefly (Leisch, 2002):

The purpose of Sweave is to create dynamic reports, which can be updated automatically if data or analysis change. Instead of inserting a prefabricated graph or table into the report, the master document contains the R code necessary to obtain it. When run through R, all data analysis output (tables, graphs, ...) is created on the fly and inserted into a final L^AT_EX document. The report can be automatically updated if data or analysis change, which allows for truly reproducible research.

Normally Sweave files got the file extension .Rd. After executing R CMD Sweave followed by the filename of the Sweave document, all R-code is extracted from the document and executed using R. The results are merged with the L^AT_EX code of the .Rd file and saved to as .tex file with the same filename. Afterwards the common compilation using latex, pdflatex, bibtex, and so on can be performed.

4.3 simple Sweave document

This subsection shows the basic structure of a Sweave document. As this paper is written using L^AT_EX and R, another working example can be found in the git repository, described in Listing 1 on page 2.

At the beginning (line 1 to 5) of Listing 3, a very simple L^AT_EX-Document is shown. On line 6 an R-environment is declared using the "noweb" syntax, which is described in Ramsey (1994); Johnson and Johnson (2000). Between `<label1, echo=false, results=hide>` and the @-sign, R source code can be located. In the header of the environment are some (local) settings and the name, which is displayed during the compilation process, located. The *results* setting defines how output generated by R should be handled by the L^AT_EX processor and *echo* configures whether messages from R should be included in the output.

More details about configuration of the noweb-environment of Sweave can be found in the official manual (Leisch, 2011).

Some additional functionalities are shown in the example. In addition to the whole declaration, any appearance of `\Sexpr{#R-Code}` is also executed by R and any output is included in the document. This may be quite useful if single values of an calculation should be inserted into continuous text.

`\SweaveInput{filename}` can be used to include another Sweave file. Although importing additional source code can be realized with R, partitioning a document to several Sweave files may be handy.

² for example *plr* (<http://www.joeconway.com/plr/> [2012-01-30]) for PostgreSQL (Group, T.P.G.D., 2011)

All single R environments which are executed in the same process share one global namespace. Also libraries, like `xtable` (loaded on line 7), which eases the creation of \LaTeX tables out of data stored in R, are available throughout the document.

Listing 3. basic structure of a Sweave document

```

1 \documentclass[a4paper]{article}
2 % additional \LaTeX configuration
3
4 \begin{document}
5
6 <<label1, echo=false, results=hide>>=
7 library(xtable)
8 # R code following
9 @
10
11 <<LaTeX_Output, echo=hide, results=tex>>=
12 # R code generating \LaTeX
13 @
14
15 <<<>>=
16 # empty header, standard settings
17 @
18
19 Normal \LaTeX code and a single line from
20 \Sexpr{\print{'R'}}.
21
22 \SweaveInput{./includes/config}
23
24 \end{document}

```

5. GEOSPATIAL PLOTTING

Finally the most important preparations are ready and some geospatial plots generated using R can be shown.

6. CONCLUSION

REFERENCES

- Group, T.P.G.D. (2011). *PostgreSQL 9.0 Official Documentation - Volume I. The SQL Language*. Fultus Corporation.
- Johnson, A.L. and Johnson, B.C. (2000). Literate programming using noweb. *Linux Journal*, 42(42), 1–12. URL <ftp://ftp.ssc.com/pub/lj/listings/issue42/2188.tgz>.
- Leisch, F. (2011). *Sweave User Manual*.
- Leisch, F. (2002). Sweave: Dynamic generation of statistical reports using literate data analysis. In W. Härdle and B. Rönz (eds.), *Compstat 2002 — Proceedings in Computational Statistics*, 575–580. Physica Verlag, Heidelberg. URL <http://www.stat.uni-muenchen.de/~leisch/Sweave>. ISBN 3-7908-1517-9.
- R Development Core Team (2011). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL <http://www.R-project.org/>. ISBN 3-900051-07-0.
- Ramsey, N. (1994). Literate programming simplified. *IEEE Software*, 11(5), 97–105. URL <http://doi.ieeecomputersociety.org/10.1109/52.311070>.