

Gitinfo Lua package*

Erik Nijenhuis <erik@xerdi.com>

14th September 2024

This file is maintained by **Xerdi**.
Bug reports can be opened at
<https://github.com/Xerdi/gitinfo-lua>.

Abstract

This project aims to display git project information in PDF documents. It's mostly written in Lua for executing the git commands, therefore making this package only applicable for `lualatex`. If `lualatex` isn't working for you, you could try `gitinfo2` instead. For `ℒATEX` it provides a set of standard macros for displaying basic information or setting the project directory, and a set of advanced macros for formatting commits and tags.

Contents

1 Usage	2	F	
1.1 Git	2	<code>\forgitauthors</code>	4
1.2 Lua \mathcal{L} A \mathcal{T} E \mathcal{X}	2	<code>\forgitcommit</code>	5
	2	<code>\forgittag</code>	5
	2	<code>\forgittagseq</code>	6
2 LaTeX Interface	3		
2.1 Package Options	3	G	
2.2 Basic macros	3	<code>\gitauthor</code>	4
2.3 Multiple Authors	4	<code>\gitcommit</code>	5
2.4 Commits	5	<code>\gitdate</code>	4
2.5 Tags	5	<code>\gitdirectory</code>	3
2.6 Changelog	6	<code>\gitemail</code>	4
	6	<code>\gittag</code>	6
3 Project Example	7	<code>\gitunsetdirectory</code>	4
		<code>\gitversion</code>	4

Index

D		I	
<code>\dogitauthors</code>	4	<code>\ifgitdirty</code>	4

*This document corresponds to package `gitinfo-lua` version 1.2.0 written on 2024-09-14.

1 Usage

For the package to work one should work, and only work, with Lua \TeX . Another prerequisite is that there is an available git repository either in the working directory, or somewhere else on your machine (see section 2.2).

1.1 Git

For this package to work at a minimum, there has to be an initialized Git repository, and preferably, at least with one commit. For example, the following minimal example should do the trick already:

Listing 1: Minimal Git setup

```
mkdir my_project
cd my_project
echo "# My Project" > README.md
git init && git commit -am "Init"
```

Then in order for the changelog to work, the project needs to contain either ‘lightweight-’ or ‘annotated’ tags. The main difference is that a lightweight tag takes no extra options, for example: `git tag 0.1`. See listing 12 for more examples on authoring and versioning with git.

1.2 Lua \TeX

For generating the document with \TeX one must make use of `lua latex` . For example, when having the main file ‘main.tex’:

Listing 2: Generating the document with \TeX

```
# Generate once
lua $\text{latex}$  -shell-escape main
# Generate and keep watching with LaTeXMK
latexmk -pvc -lua $\text{latex}$  -shell-escape main
```

Note that in both cases option `--shell-escape` is required. This is required for issuing git via the commandline. If using `--shell-restricted` mode, which is the default, make sure to add git to the CSV variable `shell_escape_commands` in either your `texmf.cnf` or using a Lua initialization script, like:

Listing 3: Lua initialization script

```
-- Global texconfig should already be available when executed with lua $\text{latex}$ 
local texconfig = texconfig or require('texconfig')

-- Use restricted shell_escape with git as only command.
-- Add others where needed, separated with a comma (no spaces in between)
texconfig.shell_escape = 'p'
texconfig.shell_escape_commands = 'git'
```

The Lua initialization script can be used as follows:

Listing 4: With Lua initialization script

```
lualatex --lua=gitinfo-lua-init.lua main
```

For using the script with `latexmk`, this can be achieved with the `-lualatex="COMMAND"` option or specifying the `$lualatex` command using a `latexmkrc` configuration file:

Listing 5: Overriding Lua \TeX on commandline

```
latexmk --lualatex --lualatex="lualatex --lua=gitinfo-lua-init.lua %0 %S" main
```

Listing 6: Overriding Lua \TeX in `latexmkrc`

```
$lualatex = "lualatex --lua=gitinfo-lua-init.lua %0 %S";
```

Keep in mind that both the Lua initialization script and `latexmkrc` need to be placed within the same directory as the main file.

When utilizing the continuous compilation option `-pvc` with `latexmk`, it's important to note that only committed changes will be detected, while tag changes, unfortunately, won't be recognized.

2 LaTeX Interface

2.1 Package Options

`\usepackage[<opts...>]{gitinfo-lua}` This package provides some options for default formatting purposes. The author sorting is one of them. If the options contain `contrib` the authors will be sorted based on their contributions; otherwise, the authors will be sorted alphabetically, which is the default option `alpha`. Another option is the `titlepage` option, which sets the `\author` and `\date` macros accordingly. By default, it sets the local git author, equivalent to option `author`. Pass the option `authors` to set all git authors of the project based on commit history instead.

Another option, more concerning directory management, `rootdir`, sets the current working directory to the root directory of the current project for all `git` commands that are executed, similar to what `\gitdirectory` does. If you're using recording of files, this option comes in handy when the main file is in a subdirectory of the project. Otherwise, if the root directory isn't set appropriately, you'll receive the warning 'warning: couldn't read HEAD from git project directory'.

2.2 Basic macros

By default, the main file's directory is used as git project directory. This directory can be manipulated with `\gitdirectory{<path>}`. The foremost difference between using the `rootdir` option and the `gitdirectory` macro, is that the macro can specify a git directory which is part of another project. The main reason for this macro to exist is its usage in

`\gitunsetdirectory` the project example in section 3. To undo an operation done with `\gitdirectory` and switch back to the main file’s directory, use `\gitunsetdirectory`.

`\gitversion` The current version can be display by using `\gitversion` and is equivalent to `git describe --tags --always`, working for both lightweight and annotated tags. For this project `\gitversion` results in 1.2.0. When the version is dirty it will be post fixed with `-<commit count>-<short ref>`. For example, when this paragraph was written, the version was displaying 0.0.1-14-gcc2bc30.

`\ifgitdirty` To test whether there are any pending changes in the local Git repository, use the `\ifgitdirty{<true code>}{<false code>}` macro. For this macro to work properly, be sure to add a `.gitignore`¹ file and include all \TeX based generated files.

`\gitdate` The `\gitdate` macro gets the most recent date from the git log. Meaning, the last ‘short date’ variant is picked from the last commit. This short date is formatted ISO based and is already suitable for use in packages like `isodate` for more advanced date formatting.

`\gitauthor` The author’s name and email can be accessed using `\gitauthor` and `\gitemail`. These `\gitemail` values are based on `git config user.name` and `git config user.email`.

2.3 Multiple Authors

`\dogitauthors` When projects having multiple authors, this package can help with the `\dogitauthors[<conj>]` and `\forgitauthors[<conj>]{<cname>}` macro. Where `\dogitauthors` executes a default formatting implementation of `\git@format@author` and `\forgitauthors` executes the given `<cname>` for every author available. The optional `<conj>` conjunction makes it possible to even integrate it further. For example, when setting the authors in pdfx, the conjunction would be `[\sep]`, so that the authors are properly separated in the document properties².

Listing 7: Formatting authors

```
1 \newcommand{\myauthorformat}[2]{#1 ↵  
  \href{mailto:#2}{#2}}  
2 \forgitauthors[\\]{\myauthorformat}  
3 % Or using standard format  
4 \dogitauthors[\\]
```

Results in

Alice <alice@example.com> Bob <bob@example.com>
--

This example is generated with the history of the `git-test-project` (see section 3) and is alphabetically sorted with package option `alpha`.

¹A proper example of a `.gitignore` file for \LaTeX : <https://raw.githubusercontent.com/github/gitignore/main/TeX.gitignore>

²See package documentation of pdfx: <https://ctan.org/pkg/pdfx>

2.4 Commits

For this section the git project of this document is used due to the fact that there are references to revisions. The test project's revisions change for every user, since they get recreated every time `test-scenario.sh` is executed (see section 3).

```
\gitcommit [format]{cname}{revision}
```

For displaying commit data `\gitcommit` can be used. The optional `format` takes variables separated by a comma. The default `format` is `h,an,ae,as,s,b`. The `cname` is a user defined command accepting every variable as argument.

Listing 8: Formatting a commit

```
1 \newcommand{\formatcommit}[3]{#1, by #2 on \printdate{#3}}
2
3 \gitcommit[s,an,as]{\formatcommit}{75dc036}
```

Results in

```
Add value escaping,
by Erik Nijenhuis on
23rd October 20233
```

Consult `man git-log` for possible format variables and omit the `%` for every variable.

```
\forgitcommit [format]{cname}{rev_spec, files={...}, flags={...}, cwd={...}}
```

For displaying multiple commits the `\forgitcommit` is used, which has the same arguments as `\gitcommit`, but only this time the `cname` is executed for every commit. The last argument, which originally only took a `rev_spec`, now also supports some additional 'named' arguments. The argument `files` takes a list of file names relative from the root of the git project. When `files` is given, all commits will be filtered out accordingly. Currently, for `flags`, only merges and no-merges are supported, which includes or excludes merge commits. The `cwd` option is like `\gitdirectory`, but only for this call.

Listing 9: Formatting commits

```
1 \newcommand{\formatcommits}[2]{\item #1\quad -#2}
2
3 \begin{itemize}
4   \forgitcommit[s,an]{\formatcommits}{75dc036...e51c481}
5 \end{itemize}
```

Results in

- Add value escaping
—Erik Nijenhuis
- Reimplement `for_commit`
—Erik Nijenhuis

2.5 Tags

In this section the `git-test-project` is used.

The tags are mostly useful for generating changelogs. For formatting tags, there's a `\forgittag` `\forgittag[format]{cname}`. Again, like `\forgitcommit` it takes a `format`, however, this time more complex, since the formatting options differ between `git log` and `git`

³`\printdate` from `isodate`: <https://www.ctan.org/pkg/isodate>

for-each-ref. For more info regarding these formatting options consult the man page of git-for-each-ref.

Listing 10: Formatting tags

```
1 \newcommand{\formattags}[2]{\item Version #1\type: #2}
2
3 \begin{itemize}
4   \forgittag[refname:short,objecttype]{formattags}
5 \end{itemize}
```

Results in

- Version 0.0.1
type: commit
- Version 0.1.0
type: tag

This example shows that the versions used are mixed. This is, of course, a horrible way to manage a project's version, though, we'll continue on with this hard objective. For example, if we wish to display the author of the lightweight and annotated tag, we can do so by specifying a format using the if-then-else feature of the format specification. The format would be: (taggername)(taggername)(authorname). Here the taggername will show up, or if not present, the authorname will be shown instead.

The default format specification is like the \forgitcommit format, but then again, some bit more complex:

```
refname:short,(taggername)(taggername,taggeremail,taggerdate:short)
(authorname,authoremail,authordate:short),subject,body
```

This is a robust example of getting all information, being it either a lightweight- or annotated tag.

`\forgittagseq` For displaying commits in between tags, there's a `\forgittagseq{<csname>}`. The `<csname>` takes exactly three arguments, namely, the `<current>`, `<next tag>` and `<rev spec>`. The last iteration gives an empty value for `<next tag>` and the `<rev spec>` is identical to `<current>`.

`\gittag` Afterward tag info can be fetched using the `\gittag[<format>]{<csname>}{<tag>}`. This macro takes the same formatting specification as `\fotgittag`. Beware of using `\gittag` for the `<next tag>` parameter in `\forgittagseq`.

All these macros put together are demonstrated in listing 11 (see next page).

2.6 Changelog

This example demonstrates the generation of a changelog. For simplicity's sake, every tag is displayed in a description environment's item and within an enumerate environment displaying commits in between.

Listing 11: Formatting a changelog

```
1 \section*{Change History}
2 \newcommand{\commitline}[1]{\item #1}
3 \newcommand{\formatversion}[3]{%
4   \item[#1]
```

```

5  \gittag[(taggerdate)(taggerdate:short)(authordate:short)]{printdate}{#1}
6  \begin{itemize}
7    \forgitcommit[s]{commitline}{#3}
8  \end{itemize}
9  }%
10 \begin{description}
11   \forgittagseq{formatversion}
12 \end{description}

```

Results in

Change History

0.1.0 6th August 2017

- Add gitignore
Get the TeX.gitignore from the gitignore repository and use it for this project.
From github

0.0.1 5th August 2017

- Add intro (README.md)
- Add readme

For displaying the tagline (see line 5) we use the existing `\printdate` macro of package `isodate`, which also takes exactly one argument For every version sequence the commits in between are displayed (see line 7), where the last sequence having the initial commit as second argument plays well with the `\forgitcommit` macro and makes it possible to show the whole sequence of history.

3 Project Example

This documentation uses an example project which gets created by the `git-scenario.sh` script (see listing 12). It creates some commits having dates in the past and different authors set. Lastly it creates a ‘lightweight-’ and ‘annotated’ tag.

To set up this scenario either do `make scenario` or execute `bash git-scenario.sh` in an initialized git repository. Keep in mind that when executing with Bash directly, you may need to specify the path to the Bash file.

Listing 12: `git-scenario.sh`

```

1  #!/bin/bash
2
3  set -e

```

```

4
5 set_author() {
6   git config user.name $1
7   git config user.email $2
8   git config committer.name $1
9   git config committer.email $2
10  git config author.name $1
11  git config author.email $2
12 }
13
14 alice() {
15   set_author 'Alice' 'alice@example.com'
16 }
17 bob() {
18   set_author 'Bob' 'bob@example.com'
19 }
20 charlie() {
21   set_author 'Charlie' 'charlie@example.com'
22 }
23
24 alice
25
26 echo "# My project" > README.md
27 git add README.md
28 git commit -m "Add readme" --date="2017-08-04 10:32"
29
30 bob
31
32 echo "
33 Another project by Alice and Bob." >> README.md
34 git add README.md
35 git commit -m "Add intro (README.md)" --date="2017-08-05 06:12"
36
37 alice
38
39 GIT_COMMITTER_DATE="2017-08-05 07:11" git tag 0.0.1
40
41 bob
42
43 curl https://raw.githubusercontent.com/github/gitignore/main/TeX.gitignore > .gitignore
44 git add .gitignore
45 git commit -m "Add gitignore"
46
47 Get the TeX.gitignore from the gitignore repository and
48 use it for this project.
49
50 From github" --date="2017-08-06 12:03"
51
52 charlie
53

```



```
54 export GIT_COMMITTER_DATE="2017-08-06 08:41"  
55 git tag -a 0.1.0 -m "Version 0.1.0"
```