The LaTeXgit package

Camil Staps[*]

Version e4cc7b9
Monday 20th April, 2020, 10:33 (+0200)

this does not work currently due to a known bug

Abstract

This is the documentation of the LaTeXgit package. Several macros are defined to fetch `git` information and typeset it. The macros defined by LaTeXgit can be helpful to documentation authors and others to whom clear document versioning is important.

Contents

1 Overview

1.1 Related packages

Brent Longborough's gitinfo and gitinfo2 packages have similar features, with a different interface. Instead of running `git` commands, it relies on special `git` files.

---

[*] Email: info@camilstaps.nl

This has the advantage that it works on Windows, however, `git` hooks have to be put into place.

André Hilbig's gitfile-info does something similar as gitinfo and LaTeXgit, but the package manual is in German so who knows. It seems that Python and `git` hooks are required.

1.2 Dependencies

LaTeXgit gets its information by running `git` commands. It uses `\write18` for this, so documents using LaTeXgit will have to be compiled with the `-shell-escape` flag.

Most of the information is retrieved using `git` but then parsed using `grep`, `cut` and similar tools. Unfortunately, this is necessary: `git log` accepts arbitrary formats, but uses the percent sign in these formats, and running commands with percent signs does not seem possible using `\write18`.

An unfortunate side effect of this is that this package will not work on Windows.

1.3 Getting information

LaTeXgit runs a shell command using `\git@command`. This macro reads the result into `\git@rawresult.` This result contains a newline character at the end that needs to be removed to avoid unwanted spacing. Therefore, `\git@result` is defined as a wrapper for `\git@rawresult` that removes this spacing with `\unskip`.

1.4 Interface

For each information-fetching command, two versions are defined: one, which only executes the command (leaving the result available in `\git@result`); and one, which executes the command and writes the result. An example is `\git@commithash` with its counterpart `\gitcommithash`. Usually, the latter is most useful.

2 Options

A number of options is available to all macros that fetch information through a pgfkeys interface. All keys are recognised by all macros, but not all are considered by each macro. Check the documentation for specific macros to see which options are relevant.

`directory=DIR`

Use the git repository in directory `DIR` (or its first parent directory that is a git repository). LaTeXgit will `cd` to this directory before executing the actual `git` command.

`file=FILE`

Get the information for the last commit modifying `FILE`.

`formatDate`
Format dates using `datetime`’s `\formatdate`.

`formatTime`
Format times using `datetime`’s `\formattime`.

`formatInterDateTime`
If both `formatDate` and `formatTime` are set, this is put in between (e.g. ‘`\space{}at\space{}`’ in English — this is also the default).

`revision=REV`
Get the hash of revision `REV` (e.g. `master`, `HEAD^`, etc.). Note that if multiple circumflexes are desired, the catcode of `^` has to be changed. For example:

```
\catcode`^=11
\gitcommithash[revision=HEAD^^^]
```

2658eaf

`showTimeZone`
When `formatTime` is set: show the time zone between parentheses after the time.

`shortHash`
Get only the first seven characters of the hash, as in `git log --oneline`.

3 Examples

`\gitcommithash`
`\gitcommitmsg`
`\gitcommitauthor`
`\gitcommitauthorname`
`\gitcommitauthoremail`

These macros are used to get and print basic commit information.

```
\catcode`^=11
This is commit \gitcommithash[shortHash=false], or in short \gitcommithash.
Three commits ago was \gitcommithash[revision=HEAD^^^].
The latest commit was by \gitcommitauthorname{} (\gitcommitauthoremail).
In raw format, the author is \texttt{\gitcommitauthor}.
The last commit modifying .gitignore was \gitcommithash[file=.gitignore].

The last three commits were:

{\tt\small
        \gitcommithash{}                \gitcommitmsg                \\
        \gitcommithash[revision=HEAD^]  \gitcommitmsg[revision=HEAD^]\\
        \gitcommithash[revision=HEAD^^] \gitcommitmsg[revision=HEAD^^]}
```

This is commit e4cc7b9888c0dc7a05a6f89ec6b1436454f59571, or in short e4cc7b9. Three commits ago was 2658eaf. The latest commit was by Camil Staps (info@camilstaps.nl). In raw format, the author is `Camil Staps <info@camilstaps.nl>`. The last commit modifying .gitignore was d175d01.
The last three commits were:

```
e4cc7b9 Add gitcommand macro
e58cb43 Copy editing
b9f2c12 Update copyright
```

`\gitcommitdate` This macro displays the `git` commit date. The following example shows the effect of the options `formatDate`, `formatTime` and `showTimeZone`.

```
\footnotesize
\begin{tabular}{c c c c}
  \texttt{formatDate} & \texttt{formatTime} & \texttt{showTimeZone} &
    Result
  \\[2pt]\hline\rule{0pt}{3ex}
             &            & (any)      &
    \gitcommitdate[showTimeZone] \\
             & \checkmark &            &
    \gitcommitdate[formatTime] \\
             & \checkmark & \checkmark &
    \gitcommitdate[formatTime,showTimeZone] \\
  \checkmark &            & (any)      &
    \gitcommitdate[formatDate,showTimeZone] \\
  \checkmark & \checkmark &            &
    \gitcommitdate[formatDate,formatTime] \\
  \checkmark & \checkmark & \checkmark &
    \gitcommitdate[formatDate,formatTime,showTimeZone] \\
\end{tabular}
```

`formatDate`	`formatTime`	`showTimeZone`	Result
		(any)	2020-04-20 10:33:56 +0200
	✓		10:33
	✓	✓	10:33 (+0200)
✓		(any)	Monday 20th April, 2020
✓	✓		Monday 20th April, 2020 at 10:33
✓	✓	✓	Monday 20th April, 2020 at 10:33 (+0200)

`\gitcommand` This macro executes an arbitrary `git` command and directly typesets the result. It only accepts the option `directory`.

```
\gitcommand{describe --always}
```

```
e4cc7b9
```

4 Implementation

Define the package and load required packages.

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{latexgit}[2020/04/20]

\RequirePackage{pgfkeys}
\RequirePackage{datetime}
```

`\latexgit` Prints the name of the package.

```
\newcommand{\latexgit}[0]{\LaTeX{}git}
```

`\git@result` When a `git` command is run, the result is stored in `\git@rawresult`. However, using `\read` results in spacing at the end of the macro. Hence, we need to use `\unskip` to remove this spacing. This is a wrapper for `\git@rawresult` that adds this `\unskip`.

```
\newcommand{\git@result}[0]{\git@rawresult\unskip}
```

We now define the keys accepted by git macros. The following options are recognised. Check the documentation of the macro to see which options are considered.

```
\newif\ifgit@opt@FormatDate
\newif\ifgit@opt@FormatTime
\newif\ifgit@opt@ShortHash
\newif\ifgit@opt@ShowTimeZone
\pgfkeys{
  /git/.is family, /git,
  default/.style={
    directory=.,
    file=,
    formatDate=false,
    formatInterDateTime=\space{}at\space{},
    formatTime=false,
    revision=HEAD,
    showTimeZone=false,
    shortHash=true,
  },
  directory/.estore in=\git@opt@Directory,
  file/.estore in=\git@opt@File,
  formatDate/.is if=git@opt@FormatDate,
  formatInterDateTime/.estore in=\git@opt@FormatInterDateTime,
  formatTime/.is if=git@opt@FormatTime,
  revision/.estore in=\git@opt@Revision,
  showTimeZone/.is if=git@opt@ShowTimeZone,
  shortHash/.is if=git@opt@ShortHash,
}
```

`\git@command` Run a `git` command and read the output into `\git@rawresult`. Before running the command, the directory will change to `\git@opt@Directory`.

```
\newread\git@stream%
\newcommand{\git@command}[1]{%
  \def\git@rawresult{}%
  \openin\git@stream|"cd \git@opt@Directory; #1"
  \ifeof\git@stream%
    \PackageError{latexgit}%
      {invoke LaTeX with the -shell-escape flag}%
      {invoke LaTeX with the -shell-escape flag}%
  \else%
    \begingroup%
    \catcode`\^^M9%
    \endlinechar=-1%
    \readline\git@stream to \git@streamoutput%
    \global\let\git@rawresult\git@streamoutput%
    \endgroup%
  \fi%
}
```

`\gitcommand` A wrapper around `\git@command` and `\git@result`, to directly typeset the result of arbitrary commands.

```
\newcommand{\gitcommand}[2][]{%
  \pgfkeys{/git,default,#1}%
  \git@command{git #2}%
  \git@result}
```

In what follows, `%` may be used in calls to `git`. Therefore we use `&` as comment character.

```
\catcode`\&=14\catcode`\%=11
```

`\git@space` For internal use, when a space is required after a `csname` in a call to `\git@command`.

```
\def\git@space{ }
```

`\gitcommithash` Get a commit hash. Recognised options: `directory`, `revision`, `shortHash`.

```
\newcommand{\gitcommithash}[1][]{&
  \git@commithash[#1]\git@result}
```

`\git@commithash` Like `\gitcommithash`, but does not return the output.

```
\newcommand{\git@commithash}[1][]{&
  \pgfkeys{/git,default,#1}&
  \ifgit@opt@ShortHash&
    \git@command{git log -n 1 --format=%h
      \git@opt@Revision\git@space -- \git@opt@File}&
  \else&
    \git@command{git log -n 1 --format=%H
      \git@opt@Revision\git@space -- \git@opt@File}&
  \fi&
}
```

`\gitcommitmsg` Get a commit message. Recognised options: `directory`, `revision`.

```
\newcommand{\gitcommitmsg}[1][]{&
  \git@commitmsg[#1]\git@result}
```

`\git@commitmsg` Like `\gitcommitmsg`, but does not return the output.

```
\newcommand{\git@commitmsg}[1][]{&
  \pgfkeys{/git,default,#1}&
  \git@command{git log -n 1 --format=%B
    \git@opt@Revision\git@space -- \git@opt@File}&
}
```

`\git@formatCommitDate` Format a commit date in ISO format using datetime’s `\formatdate`.

```
\def\git@formatCommitDate#1-#2-#3 #4:#5:#6 +#7\relax{&
  \formatdate{#3}{#2}{#1}&
}
```

`\git@formatCommitTime` Format a commit time using datetime’s `\formattime`. Recognised options: `showTimeZone`.

```
\def\git@formatCommitTime#1-#2-#3 #4:#5:#6 +#7\relax{&
  \formattime{#4}{#5}{#6}&
  \ifgit@opt@ShowTimeZone&
    \space(+#7\unskip)&
  \fi&
}
```

`\gitcommitdate` Get a commit date. If `formatDate` is set, `\git@formatCommitDate` will be used. If `formatTime` is set, and `formatDate` is unset, `\git@formatCommitTime` will be used. Recognised options: `directory`, `formatDate`, `formatTime`, `revision`, `showTimeZone`.

```
\newcommand{\gitcommitdate}[1][]{&
  \git@commitdate[#1]&
  \ifgit@opt@FormatDate&
    \expandafter\git@formatCommitDate\git@rawresult\relax&
    \ifgit@opt@FormatTime&
      \git@opt@FormatInterDateTime&
      \expandafter\git@formatCommitTime\git@rawresult\relax&
    \fi
  \else\ifgit@opt@FormatTime&
    \expandafter\git@formatCommitTime\git@rawresult\relax&
  \else
    \git@result&
  \fi\fi&
}
```

`\git@commitdate` Like `\gitcommitdate`, but does not return the output.

```
\newcommand{\git@commitdate}[1][]{&
  \pgfkeys{/git,default,#1}&
  \git@command{git log -n 1 --format=%ai
    \git@opt@Revision\git@space -- \git@opt@File}&
}
```

`\gitcommitauthor` Get a commit author. Recognised options: `directory`, `revision`.

```
\newcommand{\gitcommitauthor}[1][]{&
  \git@commitauthor[#1]\git@result}
```

`\git@commitauthor` Like `\gitcommitauthor`, but does not return the output.

```
\newcommand{\git@commitauthor}[1][]{&
  \pgfkeys{/git,default,#1}&
  \git@command{git log -n 1 --format=’%an <%ae>’
    \git@opt@Revision\git@space -- \git@opt@File}&
}
```

`\gitcommitauthorname` Get a commit author’s name. Recognised options: `directory`, `revision`.

```
\newcommand{\gitcommitauthorname}[1][]{&
  \git@commitauthorname[#1]\git@result}
```

`\git@commitauthorname` Like `\gitcommitauthorname`, but does not return the output.

```
\newcommand{\git@commitauthorname}[1][]{&
  \pgfkeys{/git,default,#1}&
  \git@command{git log -n 1 --format=%an
    \git@opt@Revision\git@space -- \git@opt@File}&
}
```

`\gitcommitauthoremail` Get a commit author’s email address. Recognised options: `directory`, `revision`.

```
\newcommand{\gitcommitauthoremail}[1][]{&
  \git@commitauthoremail[#1]\git@result}
```

`\git@commitauthoremail` Like `\gitcommitauthoremail`, but does not return the output.

```
\newcommand{\git@commitauthoremail}[1][]{&
  \pgfkeys{/git,default,#1}&
  \git@command{git log -n 1 --format=%ae
    \git@opt@Revision\git@space -- \git@opt@File}&
}
```

`\git@commitparent` Get the hash of the first parent.

```
\newcommand{\git@commitparent}[1][]{&
  \pgfkeys{/git,default,#1}&
  \git@command{git log -n 1 --format=%p
    \git@opt@Revision\git@space -- \git@opt@File
    | cut -d’ ’ -f2}&
}
```

`\gitchanges` Record the full `git` commit history (following first parents using `\git@commitparent`) using `\changes`.

```
\newcommand{\gitchanges}[1][]{&
  \git@changes[#1]{HEAD}
}
```

`\git@changes` Like `\gitchanges`, but accepts an extra argument for the revision.

```
\newcommand{\git@changes}[2][]{&
  \edef\git@@revision{#2}&
  \git@commithash[revision=\git@@revision]&
  \edef\git@@thishash{\git@rawresult}&
  \git@command{git log -n 1 --format=%ad --date=short \git@opt@Revision}&
  \edef\git@@thisdate{\git@rawresult}&
  \git@commitmsg[revision=\git@@revision]&
  & TODO: this removes ’=’ characters because they break \changes, but the real
  & solution would be to put something back that restores these characters.
  \StrSubstitute[0]{\git@rawresult}{=}{}[\git@@thismsg]&
  \changes{\git@@thisdate\unskip: \git@@thishash}{\git@@thisdate}{\git@@thismsg}&
  \git@commitparent[revision=\git@@revision]&
  \let\git@@parent\git@rawresult&
  \setbox0=\hbox{\git@@parent\unskip}\ifdim\wd0=0pt
  \else&
    \git@changes{\git@@parent}&
  \fi&
}
```

Reset the catcodes for `%` and `&`:

```
\catcode‘\&=4\catcode‘\%=14
```

5 Change History

2016-07-23: 721067b
  General: Initial commit 1

2016-07-24: 0548c53
  General: documentation: added overview section 1

2016-07-24: 2ed89f0
  General: Removed readme from makefile 1

2016-07-24: 3abab51
  General: documentation fixes; protecting GitCommitDate and the like with @ 1

2016-07-24: 6ce1a94
  General: first implementation: commit hashes, messages, dates and authors 1

2016-07-24: 8cbd796
  General: gitignore, makefile 1

2016-07-25: 14c6347
  General: added new intermediary files to Makefile 1

2016-07-25: ea7048e
  General: documentation: related packages 1

2016-07-25: f070e93
  General: documentation: added options, examples, index, history 1

2016-08-01: 05c430b
  General: moved table about date formatting to examples section 1

2016-08-02: 50a7eee
  General: Added gitchanges 1

2016-08-02: 8ba4321
  General: readline is less prone to errors with special characters than read 1

2016-08-24: d175d01
  General: Preparing CTAN release . 1

2020-03-18: 13c7faf
  General: Use same catcodes throughout 1

2020-03-18: 1b278df
  General: Rely less on external tools like grep and cut 1

2020-03-18: 2658eaf
  General: Fix git@formatCommitDate; add macros to DoNotIndex 1

2020-03-18: 4fddb8a
  General: Add file option 1

2020-03-18: 62142fd
  General: Fix gitchanges when a commit message contains the equals character 1

2020-03-19: b9f2c12
  General: Update copyright 1

2020-04-20: e4cc7b9
  General: Add gitcommand macro . 1

2020-04-20: e58cb43
  General: Copy editing 1

6 Index

Numbers written in italic refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in roman refer to the code lines where the entry is used.