gitinfo2.sty

A package for accessing metadata from the *git* DVCS

Brent Longborough

November 2015

Release: 2.0.7 (10541bd)

CONTENTS

More and more, writers are using version control systems to manage the progress of their works. One popular distributed version control system commonly used today is *git*.

Among other blessings, *git*provides some useful metadata concerning the history of the developers' work, and, in particular, about the current state of that work.

gitinfo2 allows writers to incorporate some of this metadata into their documents, to show from which point in their development a given formatted copy was produced.

How *gitinfo2* works

1. Whenever you commit work or check out a branch in *git*, *git* executes a *post-commit* or *post-checkout hook*.
2. The *gitinfo2* package includes a sample hook (placed in your *git* hooks directory), which extracts metadata from *git* and writes it to a TEX file, named `gitHeadInfo.gin` ('gin' for **g**it **in**fo).
3. When you format your document, *gitinfo2* reads `gitHeadInfo.gin` and stores the metadada in a series of LATEX commands.
4. You may use these commands to insert the metadata you need at any point in the document.

It is important to note that *gitinfo2* reads the metadata with the equivalent of `\input{.git/gitHeadInfo.gin}` in the repository (module or submodule) root.

Under some circumstances (for example, when shipping something to CTAN), a document will not be able to access the standard metadata. In such a case, an option is available to read the metadata from a *manual* copy in the current directory, called `gitHeadLocal.gin`.

If you actually want to use *gitinfo2*, then please read on. But you may just be reading this to see whether it will be useful; in this case, please skip the next chapter and go on to 'Using the package' on page 5. Then, if you like what you see, come back to read 'Setup and tailoring' later.

Why a new package?

The improvements to *gitinfo* version 2 mean that existing repositories would not work; so if I just dumped a new version of *gitinfo* on to CTAN, anyone who updated *gitinfo* would need to migrate all their active repositories before they could resume work.

So in order to avoid the rebukes that would follow, I have changed the package name to *gitinfo2*, and the inner package name from *gitsetinfo* to *gitexinfo*.

gitinfo is now informally deprecated, and the first reply to any bug reports or feature requests will be 'try *gitinfo2*'. Of course, I will support *gitinfo2* itself to the best of my somewhat limited ability.

For obvious reasons, *do not try* to use both packages in the same repository; for a new repository, use *gitinfo2*, and for an old one, either leave it alone with *gitinfo*, or migrate it to *gitinfo2*.

To the administrators of CTAN, I'm sorry I caused you this clutter by not getting it right the first time.

gitinfo2 will be installed by your favourite package or distribution manager, but before you can start to use it, you need to configure each of your *git* working copies by setting up hooks to capture the metadata.

If you're familiar with tweaking *git*, you can probably work it out for yourself. If not, I suggest you follow these steps:

1. First, you need a *git* repository and working tree. For this example, let's suppose that the root of the working tree is in `~/compsci`
2. Copy the file `post-xxx-sample.txt` (which is in the same folder of your TEX distribution as this pdf) into the *git* hooks directory in your working copy. In our example case, you should end up with a file called `~/compsci/.git/hooks/post-checkout`
3. If you're using a unix-like system, don't forget to make the file executable. Just how you do this is outside the scope of this manual, but one possible way is with commands such as this:

   ```
   chmod g+x post-checkout.
   ```

4. Test your setup with "git checkout master" (or another suitable branch name). This should generate copies of `gitHeadInfo.gin` in the directories you intended.
5. Now make two more copies of this file in the same directory (`hooks`), calling them `post-commit` and `post-merge`, and you're done. As before, users of unix-like systems should ensure these files are marked as executable.

If you don't want to install *gitinfo2* using a package manager, you can instead just copy the two *.sty files into your document directory. However, it may be simpler, for more complex project trees, to install the package as part of your TEX distribution.

Migrating from version 1

Version 2 of *gitinfo2* simplifies setup and brings additional convenience. But this comes at a cost: existing repositories need to be upgraded to be compatible. This decision wasn't taken lightly; I really do think the incompatible changes will give us a much firmer base for the future. The changes needed, which should be applied to every repository, follow.

Update *git* hooks. Replace the hooks (`post-checkout`, `post-commit`, and `post-merge`) with the contents of the file `post-xxx-sample.txt`. Since *gitinfo2* now uses only a single `gitHeadInfo.gin` file for the whole repository, you do *not* need to tailor these hooks to map your document folders.

If you are fortunate enough to be using a unix-like system, don't forget to ensure the hooks are executable.

Recreate the new `gitHeadInfo.gin` file. After you've replaced the hooks, the simplest way is to re-check-out the current branch, using a command such as `git checkout`.

Delete the old `gitHeadInfo.gin` files, when you're ready.

If you're a *memoir* user, and you use the `footinfo` package option, you will need to adjust your page styles.

From *gitinfo2* version 2, we no longer override the standard *memoir* page styles. Instead, *gitinfo2* provides three new page styles: *giplain*, *giruled*, and *giheadings*, which you should now use to provide *gitinfo2*-tailored pages.

Tailoring release tags

As shipped, the *git* hook code uses a certain convention for identifying 'release' tags: the tag must begin with a numeric digit, and contain at least one decimal point.[1]

Here is the line in the hook where this convention is established:

```
RELTAG=$(git describe … --match '[0-9]*.*' …)
```

By changing the `--match` parameter, you can decide exactly which tags qualify as 'release' tags for your needs. Thus

```
RELTAG=$(git describe … --match 'R.*' …)
```

would allow you to find tags like 'R.2.0.1' and 'R.L.Stevenson', or

```
RELTAG=$(git describe … --match '*' …)
```

would allow you to find any tag whatsoever.

You can change the hooks without needing to alter *gitinfo2*, since any tag found is used 'as-is'.

Important note: The `--match` parameter is a unix glob, *not* a regular expression. This has tripped up a number of users.

Once you've set up your *git*hooks, and done your first commit, merge, or checkout to drive them, you can start incorporating the metadata into your document, by loading the *gitinfo2* package in the usual way:

`\usepackage[`$< options >$`]{gitinfo2}`

Package options

The following options are available:

General options

`local` By default, *gitinfo2* normally reads `.git/gitHeadInfo.gin` (the metadata file). If, however, the document needs to be shipped outside of its own repository, this file will not be found. Under these circumstances, the `local` option may be used. This option causes *gitinfo2* to search instead for `gitHeadLocal.gin` in the current directory *only.* Creating this file (normally by copying from `.git/gitHeadInfo.gin`) is your responsibility.

`grumpy` In those cases where *gitinfo2* is unable to find the metadata file (`.git/gitHeadInfo.gin`, or `gitHeadLocal.gin` if the **`local`** option is used), it will set all the metadata to a common value, "(None)", issue a package warning, and carry on. If the `grumpy` option is used, this warning becomes an error, and processing stops.

`missing=`*`text`*`, notags=`*`text`*`, dirty=`*`text`

These three options allow you to tailor the default text used by *gitinfo2* when metadata is missing '(None)', the branch head has no tags '(None)', and the working copy has uncommitted changes '(*)'. For example:

```
[missing=Help!,notags={No tags?},dirty=Eww!]
```

If you have complex needs, as in the second example, don't forget to enclose your text in {}s.

`maxdepth=`*`n`*

In order to locate `.git/gitHeadInfo.gin` in the repository root, *gitinfo2* starts in the directory containing the master document, and searches up the directory tree until it finds it. This search is limited to *n* levels — 4 by default. If have to deal with documents deeper in your repository tree, you can extend this limit with, say, `maxdepth=8`.

Options for watermarking

These options allow you to place a watermark of *git* metadata, at the bottom of the paper, conditionally or unconditionally

mark This option causes *gitinfo2* to generate a watermark, centred at the bottom of each sheet, containing 'useful' *git* metadata. The watermark is always added.

markifdraft
This option is like **mark,** but only activates the watermark if the document is being processed with the **draft** option.

markifdirty
This option is like **mark,** but only activates the watermark if the last commit or checkout left uncommitted changes in the repository.

marknotags
If *gitinfo2* can't find any tags in the *git* references, it suppresses the second line of the watermark. If, however, you would like the second line always to appear, add this option to the package options.

raisemark=*vertical space*
By default, *gitinfo2* sets the bottom line of the watermark at 1.5 baseline skips[2] above the bottom of the paper. If you prefer something different, you can specify it here. For example:

```
[raisemark=0.95\paperheight]
```

draft This option *should not be used;* it only exists to 'capture' the **draft** option from the document class definition.

Options for *memoir* users

For more about these options, please read 'For *memoir* users' on page 11.

footinfo This option is no longer used, and if present is silently ignored. For *memoir* users, *gitinfo2* now creates, automatically, three new page styles.

pcount For *memoir* users, this option will replace the folio in the new page styles with one of the form *x/y*, where *x* is the folio and *y* is the page count.

No warning is given, and no action taken, if this parameter is used with another document class.

The metadata

The *git* metadata, for the current HEAD commit, is made available in the document as a series of parameter-less LaTeXcommands. Here they are:

gitReferences
: A list of any *git* references (tags, branches) associated with this commit. This string is not for the faint-hearted; its format and order may vary between versions of *git*.
Example:
(HEAD -> CTAN, develop)

gitBranch
: The name of the current branch. Depending on how you use *git*,[3] this information may not be available, and will then be shown as the default or specified value of the `missing` package option. For *git* versions before 2.0.0, where the current HEAD commit refers to more than one branch head, this value may be different from the currectly checked-out branch.
Example: *CTAN*

gitDirty
: If the last commit or checkout left uncommitted changes in the working tree, the default or specified value of the `dirty` package option; otherwise empty.

gitAbbrevHash
: The seven-hex-char abbreviated commit hash
Example: *10541bd*

gitHash
: The full 40-hex-character commit hash
Example: *10541bd1db148ba538c521d4168a9bdca89f37cd*

gitAuthorName
: The name of the author of this commit
Example: *Brent Longborough*

gitAuthorEmail
: The email address of the author of this commit
Example: *myemail@evilspam.net*

gitAuthorDate
: The date this change was committed by the author, in the format *yyyy-mm-dd*
Example: *2015-11-20*

gitAuthorIsoDate
: The date and time this change was committed by the author, in ISO format
Example: *2015-11-20 21:17:46 +0000*

gitAuthorUnixDate
: The date and time this change was committed by the author, as a Unix timestamp
Example: *1448054266*

gitCommitterName
: The name of the committer of this commit
Example: *Brent Longborough*

gitCommitterEmail
: The email address of the committer of this commit
Example: *watcher@gchq.gov.uk*

gitCommitterDate
: The date this change was committed by the committer, in the format *yyyy-mm-dd*
Example: *2015-11-20*

gitCommitterIsoDate
: The date and time this change was committed by the committer, in ISO format
Example: *2015-11-20 21:17:46 +0000*

gitCommitterUnixDate
: The date and time this change was committed by the committer, as a Unix timestamp
Example: *1448054266*

Additional metadata (Version 1)

Three more commands are available, but their use should be considered experimental. *gitinfo2* searches the *git* references metadata for anything (probably a *git* tag) that looks like a number with a decimal point. The first such number it finds is taken as a "Version Number" and made available in three different formats, explained here:

gitVtag The version number, without decorations. If no version number is found, empty (i.e. zero width).

gitVtags The version number, with a leading space. If no version number is found, empty.

gitVtagn The version number, with a leading space. If no version number is found, a space, followed by the default or specified value of the `missing` package option.
Example: *(None)*

These versioning tags have been superseded by release tags in Version 2, although they should continue to work as before.

Additional metadata (Version 2)

From Version 2 onwards, additional *git* metadata is available, in general improving or extending the facilities available in Version 1. The Version 1 metadata is retained for backward compatibility.

Included is a new set of `gitRel` commands, designed to replace `gitVtag` and its cousins. *gitinfo2* searches the *git* metadata for tags on the current HEAD commit or its ancestors, and makes the first tag found available. It also looks for the latest such tag whose name begins with a digit, and which contains a full stop (period), and makes the tag, and the number of commits following it, available as a 'release number'.[4] Here are the new commands in Version 2:

`gitFirstTagDescribe`
: The last tag reachable from the current HEAD. Please see git-describe for more information. If the working copy is *dirty* (has uncommitted changes), the string has '-*' appended.
Example: *2.0.7-1-g10541bd*

`gitRel`
: The release number, without any decorations. If no release number is found, empty (i.e. zero width).

`gitRels`
: The release number, with a leading space. If no release number is found, empty.

`gitReln`
: The release number, with a leading space. If no release number is found, a space, followed by the default or specified value of the `missing` package option.

`gitRoff`
: The number of commits between the current HEAD and the tag holding the release number. If the tag refers to the current HEAD, zero.

`gitTags`
: A comma-separated list of tags associated with the current HEAD.
Example: *(None)*

`gitDescribe`
: The raw output from the git-describe command for the last release tag reachable from the current HEAD, including tag name, commit offset, short hash, and a dirty flag. Please see git-describe for more information. Example: *2.0.7-1-g10541bd*

Watermark tailoring commands

If you use the Version 2 options to place a watermark, you can tailor the format of the watermark to suit your needs, by redefining one or more of the following commands.

gitMark Contains the text of the watermark. Output from the default definition can be seen below the footer at the bottom of this page. The definition (here split onto four lines to fit) is:

```
Branch: \gitBranch\,@\,\gitAbbrevHash{}
\textbullet{}
Release:\gitReln{} (\gitAuthorDate)\\
Head tags: \gitTags
```

You can tailor this by redefining `\gitMark`. For example:

```
\renewcommand{\gitMark}{\gitHash\hfill\gitRel}
```

gitMarkFormat
Defines typesetting parameters for the whole watermark. The default definition is:

```
\color{gray}\small\sffamily
```

if the *xcolor* package is loaded; if not, is simply omitted. You can tailor this by redefining `\gitMarkFormat`. For example:

```
\renewcommand{\gitMarkFormat}{\color(red)\ttfamily}
```

gitMarkPref
Contains the text of the watermark prefix, which depends on the reason the document is being watermarked. The default values are *[Dirty]*, *[Draft]*, or *[git]*. You can tailor this by redefining `\gitMarkPref`. For example:

```
\renewcommand{\gitMarkPref}{[Pending review]}
```

Handling TEX-hostile e-mail addresses

Occasionally, a repository will contain 'TEX-hostile' e-mail addresses such as `my_name@somewhere.net`. As a result, using the gitAuthorEmail or gitCommitterEmail commands can cause errors.

gitinfo2 provides an e-mail wrapping command, `\gitWrapEmail`, to allow you to tailor your use of email addresses. Its default definition does nothing:

```
\newcommand{\gitWrapEmail}[1]{#1}
```

You can tailor this by redefining `\gitWrapEmail`. For example, a number of packages (*hyperref* is one) provide a `\url` command which provides the necessary protection. Using such a package (independently of *gitinfo2*), you can redefine the *gitinfo2* wrapper in this way:

```
\renewcommand{\gitWrapEmail}[1]{\url{#1}}
```

Please note that from Release 2.0.6 this precaution is optional, since *gitinfo2* now detokenises Author and Committer names and email addresses, and all *git* reference names.

For *memoir* users

If you use *memoir*, *gitinfo2* provides you with three new pagestyles, based on *plain*, *ruled*, and *headings*. The new pagestyles are called *giplain*, *giruled*, and *giheadings*.

For the *giplain* and *giruled* pagestyles, the folio is moved from the centre to the outer margin of the footer, and a revision stamp is placed in the inner margin.

For the *giheadings* pagestyle, the folio is moved from the outer margin of the header to the outer margin of the footer, and a revision stamp is placed in the inner margin of the footer.

If you use the `pcount` option, a solidus, and the page count, are appended to the folio.

The revision stamp is generated by this fragment:

```
Release\gitRels: \gitAbbrevHash{} (\gitAuthorDate)
```

which is set at tiny in the sans-serif font.

Note that, in contrast to version 1 of *gitinfo2*, version 2 no longer modifies the existing page styles. If you wish to use this facility, you must now select the appropriate *gi…* page style explicitly.

You can see an example in the footer of this page, above the *gitinfo2* watermark.

For *datetime2* users

If you use Nicola Talbot's excellent *datetime2* package, *gitinfo2* copies the value of the tag **`gitAuthorDate`** as a new date object named *gitdate*, which you can then refer to with other *datetime2* functions.

Please refer to the *datetime2* manual for further details, especially in the section entitled 'Saving Dates'.

If you use it, you must load *datetime2* before you load *gitinfo2*.

The *datetime* package is also supported, though deprecated.

Notes on the sequence of events

Users of *gitinfo2* (including me) are frequently surprised by what appear to be incorrect results in their output,[5] but are determined in fact by the precise sequence of operations on a working copy.

As it's impossible to foresee every possible workflow, what follows is a sequence of steps of a simple workflow which will serve as an example to help users to understand what is happening, and to avoid it.

The example repository manages two main files: *abc.tex* and its output file, *abc.pdf*.[6] Intuitively, the overall sequence of one update & release cycle might be like this:

1. Edit *abc.tex*, (perhaps) commit intermediate changes, format the output into *abc.pdf*, and check and repeat until ready for release;
2. Commit the release version of *abc.tex*;
3. Tag the release;
4. (Missing step);
5. Format the release version of *abc.pdf*; and
6. Commit the release version of *abc.pdf*.

For reasons that I hope will become obvious, that doesn't work. The *git* metadata is extracted and stored at step (2). At that point, the release tag doesn't yet exist, and the working copy is dirty, since *abc.pdf* has been regenerated, and has changed since the last recorded version.

It's simple to fix; all we have to do is ensure the working copy is clean and regenerate the saved metadata (now with correct tag information). Here's the missing step, kept, perhaps, somewhere in a *makefile:*

4. `git checkout abc.pdf`

This will revert the .pdf file to its repository version, and then regenerate the stored metadata, including the release tag. Step (5) will now produce output showing the release tag and a clean working copy.

Release notes

R2.0.7: 2015-11-22 – Allow metadata to be shipped

- Allow metadata to be read from `gitHeadLocal.gin` in the current directory, rather than from `.git/gitHeadInfo.gin`, via the `local` package option.

R2.0.6: 2015-11-14 – Detokenise the metadata

- Detokenise names, emails, branches, and tags. This means you should be able to use branch names like `yes@top_dol$lar` without TEX chewing you out. I'd welcome feedback on this, as I have an uneasy feeling there may be unintended consequences.

R2.0.5: 2015-11-09 – Bug fixes and general improvements

- Support for the *datetime2* package
- Provide correct committer metadata in hook sample
- Change Warning to Info when *gitHeadinfo.gin* is found
- Move all package dependencies from *gitexinfo.sty* to *gitinfo2.sty*
- Support for *git* Version 2 log output, with more accurate branch name analysis
- Only use colours if the *xcolor* package is loaded

R2.0.4: 2014-10-03 – Fixes and documentation improvements

- More robust *git* hooks, for improved detection of dirty working copies
- A new section, *Notes on the sequence of events*, to help with doing things in the right order
- Other minor improvements to the manual

R2.0.3: 2014-09-05 – Handle hostile e-mail addresses

- Provide an e-mail address wrapper command, to allow users to tailor protection against '_' and other characters in email addresses.
- This release was not shipped to CTAN

R2.0.2: 2014-09-04 – Mostly cosmetic

- Fix packaging problems for CTAN and TEX Live
- Improve appearance of watermark
- Improve documentation: correct file references. remove gibberish, extend acknowledgments

Acknowledgements

The TEX.SE community has been a constant source of help, inspiration, and amazement. In particular, I'd like to thank Joseph Wright, who rescued me from the jaws of the TeX parser by explaining \expandafter.

I'd also like to register my thanks to the owners of the packages on which *gitinfo2* depends: datetime2, eso-pic, etoolbox, kvoptions, and xstring.

Many people have written to me kindly to point out some of the defects in *gitinfo*, and to offer code. I owe you all an apology for the amount of time that elapsed from your suggestions to the making of *gitinfo2*.

In some cases, I have not taken up suggestions other than as food for thought, in others used the code or suggestions directly, and, in yet others, adapted. I thank you all, especially for stimulating my thought processes, and thus, hopefully, helping to make *gitinfo2* a whole lot better than *gitinfo*.

I think I owe a special mention, both for ideas and code, to Clea Rees, Jörg Weber, and Kai Mindermann for improving the handling of *git* references; to Jörg Weber for watermarking; to Michael Rans and Ross Vandegrift for the deduplication of `gitHeadInfo.gin`; and to *ivokabadshow* on GitHub for a welcome example of how to detokenise the metadata.

My sincere thanks, too, to Adrian Burd, cedb12 (GitHub), Maximilian Held, Johannes Hoetzer, Mikko Korpela, Martin W Leidig, Enrico Malizia, Ken Mankoff, Ryan Matlock, Robbie Morrison, Nik (gwdg nokta de), Omid (gmail nokta com), Sasaki Suguru, Torbjørn T (GitHub and TeX.SE), and Felix Wenger.

Special thanks to Karl Berry for helping me to reduce my incompetence with `ctanify`. And, of course, for TEX Live and everything else.

Finally, but by no means least, my thanks to the CTAN elves, and their dæmons, particularly, in my case, Ina Dau, Manfred Lotz, Petra Rübe-Pugliese, and Robin Fairbairns, for their infinite patience and unstinting dedication to the TEX community.

The failings, of course, I claim for myself.

Copyright & licence

From the author

Although my limitations as a TeXnician mean that I've implemented *gitinfo2* in a rather simplistic way that needs some setup that is more complicated than I wanted, I hope you find the package useful. I'll be very happy to receive your comments by email.

Brent Longborough

brent+ctancontrib (bei) longborough (punkt) org
and at TeX.SE

Chapter: Setup and tailoring

1. (p.4) I.e. full stop or period, depending on which variant of English you use.

Chapter: Using the package

2. (p.6) `1.5\baselineskip`, an admittedly arbitrary value, chosen for my Canon printer.
3. (p.7) For example, checking out unnamed branches
4. (p.9) *gitinfo2* doesn't check for numerics, so you can use a tag like '1.5-beta' if you wish.
5. (p.12) Most frequent are 'wrong tag' and 'falsely flagged as dirty'
6. (p.12) Although derived files are often excluded from version control, the .pdf file in this case might need to be archived — think of versions of a proposal, for example.

Chapter: Etc

7. (p.15) (`http://www.latex-project.org/lppl.txt`)