

gitinfo.sty

A package for accessing metadata
from the *git* DVCS

Brent Longborough

12th May, 2014

Release: 2.0-beta (461e7db)

CONTENTS

Introduction	1
How <i>gitinfo</i> works	1
Setup	2
Migrating from version 1	2
Tailoring release tags	3
Using the package	4
Package options	4
The metadata	6
For <i>memoir</i> users	8
Etc	10
Acknowledgements & dependencies	10
Copyright & licence	10
From the author	11

INTRODUCTION

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.

gitinfo 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 *gitinfo* works

1. Whenever you commit work or check out a branch in *git*, *git* executes a *post-commit* or *post-checkout* hook.
2. The *gitinfo* package includes a sample hook (placed in your *git* hooks directory), which extracts metadata from *git* and writes it to a T_EX file, named `gitHeadInfo.gin` ('gin' for **git info**).
3. When you format your document, *gitinfo* reads `gitHeadInfo.gin` and stores the metadata in a series of L^AT_EX 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 *gitinfo* reads the metadata with the equivalent of `\input{.git/gitHeadInfo.gin}` in the repository (module or submodule) root.

If you actually want to use *gitinfo*, 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 4. If you like what you see, you can come back to read (Setup) later.

SETUP

gitinfo 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` as `post-checkout` 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 `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 *gitinfo* 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 *gitinfo* 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.

Install the new *git* hooks. Replace the hooks (`post-checkout`, `post-commit`, and `post-merge`) with the contents of the file `post-xxx-sample.txt`. Since *gitinfo* 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.

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

Recreate the new `gitHeadInfo.gin` file. After you've replaced the hooks, the simplest way is to re-check-out the current branch. A simple `git checkout` command should do.

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 *gitinfo* version 2, we no longer override the standard *memoir* page styles. Instead, *gitinfo* provides three new page styles: *giplain*, *giruled*, and *giheadings*, which you should now use to provide *gitinfo*-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 *gitinfo*, since whatever tag is found is used as-is.

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

USING THE PACKAGE

Once you've set up your *githooks*, and done your first commit, merge, or checkout to drive them, you can start incorporating the metadata into your document.

The *gitinfo* package is loaded in the usual way:

```
\usepackage[< options >]{gitinfo}
```

Package options

The following options are available:

grumpy By default, If *gitinfo* can't find `gitHeadInfo.gin` (the metadata file), 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*

If *gitinfo* can't find `gitHeadInfo.gin`(the metadata file), it will normally set all the metadata to a default common value, "(None)". If you wish, you can change this value to something else with, say `missing=Help!` or `missing={What a mess}`.

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

dirty=*text*

If *gitinfo* detects that your working copy is dirty (in the git sense — it has uncommitted changes), it makes this information available as a command. For a clean repository, the command is empty, while for a dirty repository it provides the default text "(*)". If you wish, you can change this value to something else with, say `dirty=Unclean!` or `dirty={This repo is grotty}`.

Again, for 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, *gitinfo* starts in the directory containing the document being processed, and searches up the directory tree until it finds it. This search is limited to *n* levels — 4 by default. If you are sufficiently unfortunate to have to deal with documents deeper in your repository tree, you can extend this limit with, say, `maxdepth=8`.

footinfo This parameter is no longer used, and if present is silently ignored. For *memoir* users, *gitinfo* now creates, automatically, three new page styles. More detail is given below (For *memoir* users).

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. More detail is given below (For *memoir* users).

The metadata

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

gitAbbrevHash

The seven-hex-char abbreviated commit hash

Example: *461e7db*

gitHash The full 40-hex-character commit hash

Example: *461e7db2f60041c8ce81234a7a38e006c7758a65*

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: *2014-05-12*

gitAuthorIsoDate

The date and time this change was committed by the author, in ISO format

Example: *2014-05-12 18:16:04 +0100*

gitAuthorUnixDate

The date and time this change was committed by the author, as a Unix timestamp

Example: *1399914964*

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: *2014-05-12*

gitCommitterIsoDate

The date and time this change was committed by the committer,

in ISO format

Example: *2014-05-12 18:16:04 +0100*

gitCommitterUnixDate

The date and time this change was committed by the committer, as a Unix timestamp

Example: *1399914964*

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, tag: 2.0-beta, tag: 1.995, origin/master, localtex/master, master)*

Additional metadata (Version 1)

Three more commands are available, but their use should be considered experimental. *gitinfo* 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 any 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: *1.995*

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. *gitinfo* 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’.² 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-beta*

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 tag holding the release number and the current HEAD. If the tag refers to the current HEAD, zero.

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.

Example: *master*

For *memoir* users

If you use *memoir*, *gitinfo* 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:

```
Revision\gitVtags: \gitAbbrevHash{} (\gitAuthorDate)
```

²*gitinfo* doesn’t check for numerics, so you can use a tag like ‘1.5-beta’ if you wish.

³For example, checking out unnamed branches

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

Note that, in contrast to version 1 of *gitinfo*, 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.

Acknowledgements & dependencies

The T_EX.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's also like to register my thanks to the owners of the packages on which *gitinfo* depends: *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 version 2.

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 version 2 a whole lot better than version 1.

I think I owe a special mention to Kai Mindermann, Clea Rees, and Jörg Weber for code and ideas which have helped greatly to improve the handling of *git* references; and to Ross Vandegrift for enabling the deduplication of `gitHeadInfo.gin`.

My sincere thanks, too, to Adrian Burd, Felix Wenger, Johannes Hoetzer, Martin W Leidig, Michael Rans, Nik (of `gwdg.nokta.de`), Omid (of `gmail.nokta.com`), Robbie Morrison, Ryan Matlock, and Sasaki Suguru.

The failings, of course, I claim for my self.

Copyright & licence

Copyright © 2011, Brent Longborough.

This work — *gitinfo* — may be distributed and/or modified under the conditions of the LaTeX Project Public License: either version 1.3 of this license, or (at your option) any later version.

The latest version of this license is at <http://www.latex-project.org/lppl.txt>, and version 1.3 or later is part of all distributions of L^AT_EX version 2005/12/01 or later.

This work has the LPPL maintenance status ‘maintained’; the Current Maintainer of this work is Brent Longborough.

This work consists of the files `gitinfo.sty`, `gitsetinfo.sty`, `gitinfo.tex`, `gitinfo.pdf`, `post-git-sample.txt`, and `gitHeadInfo.gin`.

From the author

Although my limitations as a T_EXnician mean that I've implemented *gitinfo* 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 (at) longborough (dot) org
and at T_EX.SE