Gitfile-Info

LaTeX-Paket zum Auslesen von git Versionsinformationen für eine Datei

LaTeX-package for reading git commit info for a specific file

André Hilbig
mail@andrehilbig.de – andrehilbig.de/u/gfi

Zusammenfassung

If you are using git to control versions of LaTeX-files, you may want to show yourself or other users or devs the current version of the file, information about the author and last edited date. All packages for git known make that kind of information available for the whole repository. But sometimes you have a lot of files within the same repository in different versions, from different authors etc. Perhaps you also split up a big project in small files and want to show within the document who had edited what. This package gives you the opportunity to do so.

Wenn Versionen von LaTeX-Dateien mit git kontrolliert werden, dann kommt es vor, dass für einen selbst, andere Nutzern oder Entwickler der aktuelle Entwicklungsstand der Datei, Hinweise zum Autor und dem Datum der letzten Bearbeitung im PDF gezeigt werden sollen. Mir bekannte Pakete können zwar den Stand des Repositories auslesen, jedoch nicht für eine bestimmte Datei unterscheiden. Allerdings wird manchmal mit vielen Dateien in einem Repository gearbeitet, die jeweils in verschiedenen Versionen vorliegen. Möglicherweise soll auch ein großes Projekt in mehrere TeX-Dateien aufgeteilt werden. Dann soll dennoch für jede einzelne Datei die Versionsinfo angezeigt werden können. Dieses Paket soll diese Funktionalität liefern.

Inhaltsverzeichnis

1 Änderungen

v0.1 Intialisierung und erste Veröffentlichung

v0.2 Dokumentation auf CTAN inkludiert

v0.3 Pfad der Scripte in texlive in der Dokumentation aktualisiert – Pythonscripte erzwingen Py3 und geben Fehler bei falscher Lokalisierung aus.

2 Installation

2.1 Systemanforderungen

Um die Informationen über einzelne Dateien aus dem git auszulesen, müssen entsprechende Scripte bzw. Hooks innerhalb des Repositories platziert werden. Damit eine möglichst breite Nutzbarkeit möglich ist, habe ich mich dazu entschieden mit Python und der gitpython-Bibliothek zu arbeiten.

- Python>=3
- gitpython: GITPYTHON-DEVELOPERS, Hrsg. (Juni 2016). *GitPython. GitPython is a python library used to interact with Git repositories.* URL: https://github.com/gitpython-developers/GitPython (besucht am 23. 06. 2016)

  (kann über `pip` installiert werden)

Hinweis: Die Scripte werden in erster Linie für Unix-basierte Betriebssysteme geschrieben. Support für andere Systeme kann und möchte ich nicht leisten.

2.2 Automatische Installation

Das Paket ist über `CTAN` verfügbar und kann so mit dem `tlmgr` bzw. der Paketverwaltung des Betriebssystems[1] abgerufen werden.

2.3 Manuelle Installation

Falls eine automatisierte Installation nicht möglich ist, können die Pakete auch manuell installiert werden. Es wird jedoch empfohlen, eine aktuelle Distribution zu verwenden, etwa `texlive2016`. Für Versionen davor kann keine Kompatibilität gewährleistet werden. Zur Installation werden die Dateien `gitfile-info.ins` und `gitfile-info.dtx` benötigt.

- Erzeugung der Paketdateien und Dokumentation

```
latex gitfile-info.ins
latexmk -pdf gitfile-info.dtx
```

[1] Leider halten viele Distributionen ihre LaTeX-Installationen nicht aktuell. Daher wird empfohlen die direkten Quellen, etwa von `texlive`, zu verwenden.

- Die erzeugte Paketdatei (*.sty) muss in einem für TEX lesbarem Verzeichnis platziert werden. Für eine lokale Installation bietet sich dafür

  `~/texmf/tex/latex/gitfile-info/`

  an.

- Außerdem werden die drei Python-Scripte

  `gfi-run.py`, `post-commit.py` und `post-merge.py`

  erstellt. Diese Dateien müssen im Repository platziert werden (vgl. Abschnitt 2.4, S. 3).

2.4 Einrichtung des Repositories

Die beiden Scripte `post-commit.py` und `post-merge.py` müssen innerhalb des Repositories im Verzeichnis `.git/hooks` als ausführbare Dateien unter den Namen `post-commit` bzw. `post-merge` platziert werden.

`post-commit` wird bei jedem Commit ausgeführt (nachdem der Commit vollständig beendet ist) und schreibt für die veränderten TEX-Dateien Änderungen in eine Hilfsdatei.

`post-merge` wird nach jedem merge (erfolgreich und nicht erfolgreich – explizit auch nach einem pull) ausgeführt, um Veränderungen in die Hilfsdatei einzutragen.

Außerdem sollte das Script `gfi-run.py` möglichst für jeden Nutzer im Repository ausführbar platziert werden. Wird das Script ohne Parameter ausgeführt, liest es sämtliche unter Versionsverwaltung stehende Dateien aus und erstellt die passenden Hilfsdateien. Wird dem Script eine TEX-Datei (inkl. Endung) übergeben, wird die Hilfsdatei für diese spezielle Datei neu erstellt.

```
# alle *.tex-Dateien aktualisieren
python gfi-run.py
# eine spezielle *.tex-Datei aktualisieren
python gfi-run.py datei.tex
```

Die Scripte sollten in einer Standard `texlive`-Installation unter dem Pfad `.../texmf-dist/doc/support/gitfile-info/` gefunden werden können. Bei einer manuellen Installation sollten die Scripte mit erstellt worden sein.

Hinweis: Die drei Dateien sollten Nutzern zur Verfügung gestellt werden. Typischerweise sind sie nach einem Clone nicht im Baum enthalten. Jeder Nutzer muss sich die Hooks selbstständig einrichten – außer es werden entsprechende Konfigurationen festgelegt.

Außerdem muss in der `*.gitignore` der Filter `*.gfi` festgelegt werden, da die Hilfsdateien in **keinem** Fall unter Versionsverwaltung stehen dürfen. Daher muss ein Nutzer nach einem *frischen* Clone das Script `gfi-run.py` aufrufen, um alle Hilfsdateien lokal zu erstellen.

3 Funktionsweise

3.1 Vorüberlegungen

Zunächst stand die Überlegung im Raum, Meta-Daten ähnlich wie beim Paket `svninfo` (vgl. Brucker 2010) direkt in die betreffenden TEX-Dateien einzutragen. Dadurch wird jedoch der Arbeitsstand verändert und der eingetragene Commit ist nicht mehr aktuell. Es müsste ein erneuter Commit erfolgen usw. Hier gäbe es die Möglichkeit, automatisierte Commits zu erstellen. Diese würden jedoch das Repository aufblähen.

Daher entschied ich mich dafür, die passenden Meta-Daten in eine Hilfsdatei (`*.gfi`) einzutragen. Hier können per simplem LATEX-Befehl Metadaten eingegeben werden. Das bereits bestehende Paket gitinfo2 (vgl. Longborough 2015) konnte nicht benutzt werden, da es das gesamte Repository ausliest und nicht zwischen einzelnen Dateien unterscheidet.

3.2 Umsetzung

Bei jedem Commit oder Pull gehen die Scripte alle geänderten TEX-Dateien durch und aktualisieren die entsprechenden Hilfsdateien. Hier werden **nur** Dateien mit der Endung `*.tex` berücksichtigt! Bei Problemen kann das Script `gfi-run.py` per Hand aufgerufen werden, um eine Aktualisierung zu erzwingen (vgl. Abschnitt 2.4, S. 3).

4 Nutzung des Pakets

Alle Makros geben immer die Versionsinformationen für die *aktuelle* Datei zurück, sofern diese geeignet geladen wurde (vgl. Abschnitt 4.3, S. 6). Das Paket wird dazu in der Präambel des Dokumentes geladen.

```
\usepackage{gitfile-info}
```

Weitere Optionen müssen nicht angegeben werden. Sofern notwendige Hilfsdateien mit der Endung `*.gfi` noch nicht vorhanden sind, werden alle Makros mit Standardwerten belegt und eine Warnung ausgegeben.

4.1 Auslesen der Metadaten

`\gfiGet` Über die `\gfiGet*`-Makros können die Metadaten aus dem Repository ausgelesen werden.

`\gfiGetDay` gibt den Tag der letzten Änderung als zweistellige Ziffer zurück.

`\gfiGetMonth` gibt den Monat der letzten Änderung als zweistellige Ziffer zurück.

`\gfiGetYear` gibt das Jahr der letzten Änderung als zweistellige Ziffer zurück.

`\gfiGetHour` gibt die Stunde der letzten Änderung als zweistellige Ziffer zurück.

`\gfiGetMin` gibt die Minute der letzten Änderung als zweistellige Ziffer zurück.

`\gfiGetDate` gibt das volle Datum der letzten Änderung mit Uhrzeit im Format `dd. Monat yyyy HH:MM` zurück[2]

`\gfiGetAuthorName` gibt den Namen des Autors der letzten Änderung zurück.

`\gfiGetAuthorMail` gibt die E-Mailadresse des Autors der letzten Änderung zurück.

`\gfiGetCommit` gibt den Hash des letzten Commits zurück.

`\gfiGetCommitAbr` gibt die Kurzform des letzten Commits zurück.

Ergänzend sei auf das Paket datenumber (vgl. Schröder 2001) verwiesen. Damit ist es möglich obige Makros für Tag, Monat und Jahr zwischen lokalisierten Monatsbezeichnungen usw. zu konvertieren.

4.2 Versionsinfo

`\gfiInfo` Sofern eine kleine Zusammenfassung der aktuellen Datei gezeigt werden soll, kann dazu das Makro

`\gfiInfo[`⟨*Hashlänge*⟩`][`⟨*Datumsformat*⟩`][`⟨*Autorformat*⟩`][`⟨*tcolorbox*⟩`]`

benutzt werden. Sofern alle optionalen Argumente leer gelassen werden, wird der lange Hash, das Standard `\gfiGetDate` und der Name des Autors als Hyperlink auf die E-Mailadresse in einer `tcolorbox` mit dem Namen `gfiInfoBox` ausgegeben.

gitfile-info.dtx

git

Version: ac5f4546b3f9533f287c2ce27843a92e2dfa4a68
Stand: 09. April 2019 19:40
Zuletzt bearbeitet von: André Hilbig

Hashlänge kann durch Angabe von `abr` als verkürzter Hash ausgegeben werden. Standard: lang.

Datumsformat, Autorformat können jeweils beliebige TEX-Befehle enthalten. Standard: langes Datum und Name als Hyperlink.

tcolorbox kann einer beliebigen über `\newtcolorbox{}` eingeführten tcolorbox entsprechen. Standard: `gfiInfoBox`.

`gfiInfoBox` Durch Verwendung der definierten Box `gfiInfoBox` können auch beliebige andere Zusammenstellungen erstellt werden.

```
\begin{gfiInfoBox}
  \vspace{1mm}
  Die letzte Änderung wurde durch den Autor \gfiGetAuthorName\
  (\href{mailto:\gfiGetAuthorMail}{\gfiGetAuthorMail}) am
  \gfiGetDay.\gfiGetMonth.\gfiGetYear\ um
  \gfiGetHour:\gfiGetMin\,Uhr commited. Die letzte Änderung
  hat den Commit \gfiGetCommitAbr.
  \vspace{1mm}
\end{gfiInfoBox}
```

[2] Das Format wird durch die Scripte vorgegeben und muss in diesen ggfs. angepasst werden, sofern eine Lokalisierung gewünscht ist (vgl. Abschnitt 5.2, S. 6)

gitfile-info.dtx

git

Die letzte Änderung wurde durch den Autor André Hilbig (mail@andrehilbig.de) am 09.04.2019 um 19:40 Uhr commited. Die letzte Änderung hat den Commit ac5f45.

4.3 Laden weiterer TEX-Dateien

Ähnlich wie im Paket `svninfo` (vgl. Brucker 2010) soll auch die Aufsplittung eines größeren Projekts in mehrere Teildateien mit den entsprechenden Versionen der einzelnen Dateien auslesbar sein. Dafür müssen diese Dateien ebenfalls die Endung `*.tex` haben, um von den Scripten erkannt zu werden.

`gfiInclude` `gfiInput` Im Hauptdokument werden die Metadaten automatisch beim Beginn geladen. Sofern eine weitere Datei per `\include` oder `\input` geladen werden soll, müssen dafür die Befehle

`\gfiInclude{`⟨*Datei*⟩`}` bzw. `\gfiInput{`⟨*Datei*⟩`}`

genutzt werden. Die Endung der Datei sollte dabei **nicht** mit angegeben werden. Intern werden die jeweiligen Befehle zum Laden einer weiteren Datei entsprechend genutzt. Außerdem wird die zugehörige Hilfsdatei eingebunden, um die notwendigen Metadaten zu erhalten. Nachdem die inkludierte Datei vollständig bearbeitet wurde, werden die Metadaten der vorherigen bzw. dann aktuellen Datei geladen. Es ist auch möglich, beliebige Verschachtelungen vorzunehmen.

5 Implementierung

5.1 Paket

Das Paket lädt automatisch die zugehörige Hilfsdatei eines Hauptdokuments über den entsprechenden `\jobname`. Der Nutzer musst hierfür keine Anpassung vornehmen. Sollten weitere Dokumente eingebunden werden, müssen die bereitgestellten Befehle genutzt werden, sofern die zugehörigen Versioninformationen geladen werden sollen (vgl. Abschnitt 4.3, S. 6). Die Hilfsdatei trägt den selben Namen, wie die zugehörige TEX-Datei, und enthält passende Aufrufe der `\gfiSet*`-Makros.

`\gfiSetDate` `\gfiSetDate{`⟨*Tag*⟩`}{`⟨*Monat*⟩`}{`⟨*Jahr*⟩`}`
`{`⟨*Stunde*⟩`}{`⟨*Minute*⟩`}{`⟨*Lokalisierte Langfassung*⟩`}`

Tag, Monat, Stunde und Minute sind jeweils als zweistellige Ziffern einzulesen. Das Jahr wird als vierstellige Ziffer eingelesen und in der Langassung kann beliebiger Text stehen, der einer durch die Scripte lokalisierten Version entspricht. Dadurch werden die `\gfiGet*`-Makros definiert. `\gfiGetDate` entspricht der Langfassung.

`\gfiSetAuthor` `\gfiSetAuthor{`⟨*Name*⟩`}{`⟨*E-Mail*⟩`}`

Name und E-Mail sollten die zugehörigen Daten enthalten und werden mit den `\gfiGet*`-Makros verknüpft.

`\gfiSetCommit` `\gfiSetCommit{`⟨*Hash*⟩`}{`⟨*Hash-Abr*⟩`}`

Die volle Fassung des Commits wird im Hash, die kurze Version im Hash-Abr eingegeben und entsprechend mit `\gfiGet*` verknüpft.

5.2 Scripte

Die Scripte sind auf deutsche Monatsbezeichnungen eingestellt. Sofern hier eine andere Lokalisierung gewünscht wird, muss in *allen* Scripten der entsprechende

Eintrag in der Präambel geändert werden! Prinzipiell reicht es aus im ersten Block zu ändern, da bei einer individuellen Anpassungen Fehler nicht auftreten sollten.

```
try:
    locale.setlocale(locale.LC_ALL, ↷
        ⇝ 'de_DE.utf8')
except:
    try:
        locale.setlocale(locale.LC_ALL, ↷
            ⇝ 'de_DE')
```

5.2.1 gfi-run.py

Das `gfi-run.py` kann sowohl zur Initialisierung als auch zur erzwungenen Aktualisierung aller TEX-Dateien benutzt werden.

```
# alle *.tex-Dateien aktualisieren
python gfi-run.py
# eine spezielle *.tex-Datei aktualisieren
python gfi-run.py datei.tex
```

Das Script sucht dabei mithilfe von `git ls-files` nach allen unter Versionsverwaltung stehenden Dateien und erstellt (überschreibend) aufgrund von `git log` für jede Datei einzeln die passende Hilfsdatei mit den Metadaten. Wird eine Datei übergeben, so wird nicht geprüft, ob diese unter Verwaltung steht und eine Hilfsdatei (im Zweifel leer) erstellt.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
# -*- mode:python -*-
import os
import sys
import time
import codecs
from git import Repo
import locale
# Locales for date set up to de_DE
# Please edit to you needs
try:
    locale.setlocale(locale.LC_ALL, 'de_DE.utf8')
except:
    try:
        locale.setlocale(locale.LC_ALL, 'de_DE')
    except:
        print ("Fehler: bitte prüfen Sie das Ergebnis von ↷
            ⇝ 'locale -d' und fügen es in das Script ein.")

# set up the git repo
# path is the current working directory
pathrepo = os.getcwd()
repo = Repo(pathrepo)
assert not repo.bare
git = repo.git
```

```
26 headcommit = repo.head.commit
27 index = repo.index
28
29 # check for specific file (else) or
30 # every tex file within the git
31 if len(sys.argv) <= 1:
32     # get all the files within the git
33     commFiles = git.ls_files(full_name=True).split("\n")
34
35     # iterate through all files and read date/author/commit and
36     # write in the help file
37     for fl in commFiles:
38         flname, flext = os.path.splitext(fl)
39         if flext == '.tex':
40             rawdate = int(git.log('-1', fl,
41                           pretty='format:"%at"').split('"')[1])
42             date = [time.strftime("%d. %B %Y %H:%M", ↷
                    ⇝ time.localtime(rawdate)),
43                     time.strftime("%d", ↷
                        ⇝ time.localtime(rawdate)),
44                     time.strftime("%m", ↷
                        ⇝ time.localtime(rawdate)),
45                     time.strftime("%Y", ↷
                        ⇝ time.localtime(rawdate)),
46                     time.strftime("%H", ↷
                        ⇝ time.localtime(rawdate)),
47                     time.strftime("%M", ↷
                        ⇝ time.localtime(rawdate))]
48             author = [git.log('-1', fl, ↷
                    ⇝ pretty='format:"%an"').split('"')[1],
49                       git.log('-1', fl, ↷
                          ⇝ pretty='format:"%ae"').split('"')[1]]
50             commit = [git.log('-1', fl, ↷
                    ⇝ pretty='format:"%H"').split('"')[1],
51                       git.log('-1', fl, ↷
                          ⇝ pretty='format:"%h"').split('"')[1]]
52             f = codecs.open(flname+".gfi", "w", ↷
                    ⇝ encoding="utf-8")
53             f.write("% gitfile-info control file\n")
54             f.write("\\gfiSetDate{" + date[1] + "}{" + ↷
                    ⇝ date[2] + "}{" + date[3]
55                     + "}{" + date[4] + "}{" + date[5] + "}{" ↷
                        ⇝ + date[0] + "}\n")
56             f.write("\\gfiSetAuthor{" + author[0] + "}{" + ↷
                    ⇝ author[1] + "}\n")
57             f.write("\\gfiSetCommit{" + commit[0] + "}{" + ↷
                    ⇝ commit[1] + "}")
58             f.close
59 else:
60     # get the specific file, read date/author/commit and
61     # write the help file
62     fl = sys.argv[1]
63     flname, flext = os.path.splitext(fl)
64     rawdate = int(git.log('-1', fl, ↷
```

```
        ⇝ pretty='format:"%at"').split('"')[1])
65      date = [time.strftime("%d. %B %Y %H:%M", ↷
            ⇝ time.localtime(rawdate)),
66              time.strftime("%d", time.localtime(rawdate)),
67              time.strftime("%m", time.localtime(rawdate)),
68              time.strftime("%Y", time.localtime(rawdate)),
69              time.strftime("%H", time.localtime(rawdate)),
70              time.strftime("%M", time.localtime(rawdate))]
71      author = [git.log('-1', fl, ↷
            ⇝ pretty='format:"%an"').split('"')[1],
72                git.log('-1', fl, ↷
                    ⇝ pretty='format:"%ae"').split('"')[1]]
73      commit = [git.log('-1', fl, ↷
            ⇝ pretty='format:"%H"').split('"')[1],
74                git.log('-1', fl, ↷
                    ⇝ pretty='format:"%h"').split('"')[1]]
75      f = codecs.open(flname+".gfi", "w", encoding="utf-8")
76      f.write("% gitfile-info control file\n")
77      f.write("\\gfiSetDate{" + date[1] + "}{" + date[2] + ↷
            ⇝ "}{" + date[3]
78              + "}{" + date[4] + "}{" + date[5] + "}{" + ↷
                  ⇝ date[0] + "}\n")
79      f.write("\\gfiSetAuthor{" + author[0] + "}{" + author[1] ↷
            ⇝ + "}\n")
80      f.write("\\gfiSetCommit{" + commit[0] + "}{" + commit[1] ↷
            ⇝ + "}")
81      f.close
```

Quelltext von gfi-run.py

5.2.2 post-commit.py

Der `post-commit`-Hook von `git` wird nach jedem Commit, der erfolgreich ausgeführt wurde, automatisch ausgeführt. Der Hook liest aus, welche Dateien sich verändert haben und ändert für die passenden TEX-Dateien die Hilfsdateien mit den neuen Metadaten. Das Script sollte im Verzeichnis `.git/hooks` ausführbar platziert werden.

```
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3  # -*- mode:python -*-
4  import os
5  import time
6  import codecs
7  from git import Repo
8  import locale
9  # Locales for date set up to de_DE
10 # Please edit to you needs
11 try:
12     locale.setlocale(locale.LC_ALL, 'de_DE.utf8')
13 except:
14     try:
15         locale.setlocale(locale.LC_ALL, 'de_DE')
16     except:
```

```
17          print ("Fehler: bitte prüfen Sie das Ergebnis von 'locale -d' und fügen es in das Script ein.")
18
19  # set up the git repo
20  # path is the current working directory
21  pathrepo = os.getcwd()
22  repo = Repo(pathrepo)
23  assert not repo.bare
24  git = repo.git
25  headcommit = repo.head.commit
26  index = repo.index
27
28  # get the commited/changed files and date/author/commit
29  commFiles = git.diff_tree('-r', 'HEAD', no_commit_id=True,
30                            name_only=True).split("\n")
31  date = [time.strftime("%d. %B %Y %H:%M",
32                        time.localtime(headcommit.authored_date)),
33          time.strftime("%d", time.localtime(headcommit.authored_date)),
34          time.strftime("%m", time.localtime(headcommit.authored_date)),
35          time.strftime("%Y", time.localtime(headcommit.authored_date)),
36          time.strftime("%H", time.localtime(headcommit.authored_date)),
37          time.strftime("%M", time.localtime(headcommit.authored_date))]
38  author = [headcommit.author.name, headcommit.author.email]
39  commit = [headcommit.hexsha, headcommit.hexsha[:6]]
40
41  # iterate through all files and write the gfi help-files
42  for fl in commFiles:
43      flname, flext = os.path.splitext(fl)
44      if flext == '.tex':
45          f = codecs.open(flname+".gfi", "w", encoding="utf-8")
46          f.write("% gitfile-info control file\n")
47          f.write("\\gfiSetDate{" + date[1] + "}{" + date[2] + "}{" + date[3]
48                  + "}{" + date[4] + "}{" + date[5] + "}{" + date[0] + "}\n")
49          f.write("\\gfiSetAuthor{" + author[0] + "}{" + author[1] + "}\n")
50          f.write("\\gfiSetCommit{" + commit[0] + "}{" + commit[1] + "}")
51          f.close
```

Quelltext von post-commit.py

5.2.3 post-merge.py

Der `post-merge`-Hook von `git` wird nach jedem Merge ausgeführt. Er wird auch bei einem nicht erfolgreichen Merge aufgerufen. Explizit bedeutet dies auch, dass nach jedem Pull das Script ausgeführt wird. Der Hook liest aus, welche Dateien

sich verändert haben und ändert für die passenden TEX-Dateien die Hilfsdateien mit den neuen Metadaten. Das Script sollte im Verzeichnis `.git/hooks` ausführbar platziert werden.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
# -*- mode:python -*-
import os
import time
import codecs
from git import Repo
import locale
# Locales for date set up to de_DE
# Please edit to you needs
try:
    locale.setlocale(locale.LC_ALL, 'de_DE.utf8')
except:
    try:
        locale.setlocale(locale.LC_ALL, 'de_DE')
    except:
        print ("Fehler: bitte prüfen Sie das Ergebnis von 'locale -d' und fügen es in das Script ein.")

# set up the git repo
# path is the current working directory
pathrepo = os.getcwd()
repo = Repo(pathrepo)
assert not repo.bare
git = repo.git
headcommit = repo.head.commit
index = repo.index

# get the commited/changed files and date/author/commit
commFiles = git.diff_tree('-r', 'ORIG_HEAD', 'HEAD',
                          no_commit_id=True, name_only=True).split("\n")

# iterate through all files and write the gfi help-files
for fl in commFiles:
    flname, flext = os.path.splitext(fl)
    if flext == '.tex':
        rawdate = int(git.log('-1', fl, pretty='format:"%at"').split('"')[1])
        date = [time.strftime("%d. %B %Y %H:%M", time.localtime(rawdate)),
                time.strftime("%d", time.localtime(rawdate)),
                time.strftime("%m", time.localtime(rawdate)),
                time.strftime("%Y", time.localtime(rawdate)),
                time.strftime("%H", time.localtime(rawdate)),
                time.strftime("%M", time.localtime(rawdate))]
        author = [git.log('-1', fl, pretty='format:"%an"').split('"')[1],
                  git.log('-1', fl, pretty='format:"%ae"').split('"')[1]]
```

```
45          commit = [git.log('-1', fl, ↷
                ⇝ pretty='format:"%H"').split('"')[1],
46                  git.log('-1', fl, ↷
                        ⇝ pretty='format:"%h"').split('"')[1]]
47          f = codecs.open(flname+".gfi", "w", encoding="utf-8")
48          f.write("% gitfile-info control file\n")
49          f.write("\\gfiSetDate{" + date[1] + "}{" + date[2] + ↷
                ⇝ "}{" + date[3]
50                  + "}{" + date[4] + "}{" + date[5] + "}{" + ↷
                        ⇝ date[0] + "}\n")
51          f.write("\\gfiSetAuthor{" + author[0] + "}{" + ↷
                ⇝ author[1] + "}\n")
52          f.write("\\gfiSetCommit{" + commit[0] + "}{" + ↷
                ⇝ commit[1] + "}")
53          f.close
```

Quelltext von post-merge.py

Weiterführende Quellen

Brucker, Achim (März 2010). *The svninfo package*. URL: http://www.ctan.org/pkg/svninfo (besucht am 27. 06. 2016).

gitpython-developers, Hrsg. (Juni 2016). *GitPython. GitPython is a python library used to interact with Git repositories*. URL: https://github.com/gitpython-developers/GitPython (besucht am 23. 06. 2016).

Longborough, Brent (Nov. 2015). *gitinfo2.sty. A package for accessing metadata from the git dvcs*. URL: http://www.ctan.org/pkg/gitinfo2 (besucht am 27. 06. 2016).

Schröder, Jörg-Michael (Aug. 2001). *The datenumber.sty package v0.02*. URL: http://www.ctan.org/pkg/datenumber (besucht am 27. 06. 2016).

6 Code

```
1  \NeedsTeXFormat{LaTeX2e}[1999/12/01]
2  \ProvidesPackage{gitfile-info}
3      [2016/06/30 v0.2 read file infos of one specific tex-file]
4  \RequirePackage{ifthen}
5  \RequirePackage{currfile}
6  \RequirePackage{xparse}
7  \RequirePackage{tcolorbox}
8  \tcbuselibrary{fitting, skins, breakable}
9  \tcbset{center title,fonttitle=\bfseries\sffamily}
10 \RequirePackage{hyperref}
```

\gfiGetDay

```
11 \providecommand{\gfiGetDay}{gfi day}
```

\gfiGetMonth

```
12 \providecommand{\gfiGetMonth}{gfi month}
```

\gfiGetYear

```
13 \providecommand{\gfiGetYear}{gfi year}
```

\gfiGetHour
```
\providecommand{\gfiGetHour}{gfi hour}
```
\gfiGetMin
```
\providecommand{\gfiGetMin}{gfi min}
```
\gfiGetAuthorName
```
\providecommand{\gfiGetAuthorName}{gfi author name}
```
\gfiGetAuthorMail
```
\providecommand{\gfiGetAuthorMail}{gfi author email}
```
\gfiGetDate
```
\providecommand{\gfiGetDate}{gfi date}
```
\gfiGetCommit
```
\providecommand{\gfiGetCommit}{gfi commit}
```
\gfiGetCommitAbr
```
\providecommand{\gfiGetCommitAbr}{gfi commit short}
```
\gfiSetDate
```
\providecommand{\gfiSetDate}[6]{%
\renewcommand{\gfiGetDay}{#1}
\renewcommand{\gfiGetMonth}{#2}
\renewcommand{\gfiGetYear}{#3}
\renewcommand{\gfiGetHour}{#4}
\renewcommand{\gfiGetMin}{#5}
\renewcommand{\gfiGetDate}{#6}
}
```
\gfiSetAuthor
```
\providecommand{\gfiSetAuthor}[2]{%
\renewcommand{\gfiGetAuthorName}{#1}
\renewcommand{\gfiGetAuthorMail}{#2}
}
```
\gfiSetCommit
```
\providecommand{\gfiSetCommit}[2]{%
\renewcommand{\gfiGetCommit}{#1}
\renewcommand{\gfiGetCommitAbr}{#2}
}

\colorlet{colback}{yellow!20!black}
\newtcolorbox{gfiInfoBox}{%
every box on layer 2/.style={reset},%
breakable,%
title=\currfilename,%
enhanced,%
attach boxed title to top right={%
yshift=-2mm,%
xshift=-3mm,%
},%
```

```
boxed title style={%
drop fuzzy shadow,%
},%
fontupper=\small,%
before skip=2mm,after skip=3mm,%
boxrule=0.4pt,left=5mm,right=2mm,top=1mm,bottom=1mm,%
colback=yellow!50,%
colframe=yellow!20!black,%
sharp corners,rounded corners=southeast,arc is angular,arc=3mm,%
underlay={%
\path[fill=colback!80!black] ([yshift=3mm]interior.south east)--++(-0.4,-0.1)--++(0.1,-0.2);
\path[draw=black!75!white,shorten <=-0.05mm,shorten >=-0.05mm] ([yshift=3mm]interior.south eas
\path[fill=yellow!50!black,draw=none] (interior.south west) rectangle node[white,rotate=90]{\f
},%
drop fuzzy shadow,%
}
\ProvideDocumentCommand{\gfiInfo}{O{none} O{none} O{none} O{gfiInfoBox}}{%
\begin{#4}
\vspace{1mm}
\textbf{Version:} \ifthenelse{\equal{#1}{none}}{\gfiGetCommit}{\gfiGetCommitAbr}\newline
\textbf{Stand:} \ifthenelse{\equal{#2}{none}}{\gfiGetDate}{#2}\newline
\textbf{Zuletzt bearbeitet von:}
\ifthenelse{\equal{#3}{none}}{\href{mailto:\gfiGetAuthorMail}{\gfiGetAuthorName}}{#3}
\vspace{1mm}
\end{#4}
}
```

\gfiCurrentConfig

```
\providecommand{\gfiCurrentConfig}{none}
```

\gfiInitJob

```
\providecommand{\gfiInitJob}{%
\IfFileExists{\jobname.gfi}{%
\input{\jobname.gfi}
}{%
\@latex@warning@no@line{gitfile-info: No \jobname.gfi information-file given}
\@latex@warning@no@line{gitfile-info: All hooks and initialization run?}
}
}
```

\gfiInitInc

```
\providecommand{\gfiInitInc}[1]{%
\IfFileExists{#1.gfi}{%
\input{#1.gfi}
}{%
\@latex@warning@no@line{gitfile-info: No #1.gfi information-file given}
\@latex@warning@no@line{gitfile-info: All hooks and initialization run?}
}
}

\AtBeginDocument{%
\gfiInitJob
}
```

\gfiInput

```
93 \providecommand{\gfiInput}[1]{%
94 \gfiInitInc{#1}
95 \input{#1}
96 \gfiInitInc{\currfiledir\currfilebase}
97 }
```

\gfiInclude

```
98 \providecommand{\gfiInclude}[1]{%
99 \gfiInitInc{#1}
100 \include{#1}
101 \gfiInitInc{\currfiledir\currfilebase}
102 }
```