1. Gawk program *ctan_chk.gawk*.
Basic Gawk program that uses Ctan's published guidelines for authors to help eliminate slopiness in uploaded files/project. It is completely open for users to program additional guidelines and Ctan's future adjustments. The program came about when I first attempted to upload "Yacco2" for Ctan consideration. As Yacco2 was quite large, it took me 3 attempts to clean up my project. Depending on how u develop your project, in my case the Yacco2 project for Ctan upload is a subset of the complete system and so each uplift required the guideline assessment/purging process.

This is my attempt to help both the author in getting the new system in shape for Ctan uplift and to lower the Ctan volanteers interaction to aid for publishing on their servers. It is my thank you to these volanteers / Ctan.

This *gawk* program verifies whether an upload project for CTAN follows its published guidelines on their website:

http://ctan.org/upload/

Here is the link to the awk/gawk reference manual for people needing a refresher to its programming statements:

https://www.gnu.org/software/gawk/manual/gawk.html#Reading-Files

2. License.
ctan_chk's distributed source code is subject to the terms of the GNU General Public License (version 3). If a copy of the MPL was not distributed with this file, you can obtain one at https://gnu.org/licenses/gpl.html.

Project: ctan guidelines verifier and corrector program for uploading projects
Distributed under license: GNU General Public License (version 3).
Distribution Date: February 15, 2015
Distribution version: 1.0
Comments: Currently for the Unix flavoured Platforms running Gawk
Author: Dave Bone

Contributors list:
Dave Bone

3. *Ctan_chk* **reference.**
Unzip the *ctan_chk.zip* file. Please read *ctan_chk.pdf* document. It instructs one in "how to" use the gawk *ctan_chk.gawk* program.

4. Literate Programming genre.
ctan_chk.pdf user manual and *ctan_chk.gawk* program are generated by the Cweb system's *cweave* and *ctangle* programs along with the *pdftex*. The *ctan_chk_bash* script does it all for u; have a read it doesn't bite. Please consider using Cweb and joining "www.tug.org" where u'll find the Cweb system.

5. Comments on program's functions.
Normally u edit the *ctan_chk.gawk* file (using gawk comments) as to what "verification function" is called to check out your project's files. Here is a list of the verification functions to call:
Helper functions:

is_file_a_directory — is file a directory type rather than a data type?
is_file_an_executable — Maybe too platform dependent: is it an executable

Verification functions:

chk_auxiliary_files — files that should be removed / deleted
chk_file_permissions — file execute permissions check
chk_extended_file_attributes — does file have extended attribute like @
chk_empty_files — zero byte size file?
chk_empty_directory — empty folder
chk_file_to_bypass_in_zip — file to bypass by zip using its -x option

Correction functions:

remove_file_s_execute_attribute — Read "Remove file's execute attribute" section for details
delete_file — interacts with user. Used by itself or from other functions like *chk_auxiliary_files*
remove_file_s_extended_attributes — Remove extended attribute from file (platform dependent)

6. Running Gawk program has 2 run/pass attitude.
U run this program twice with appropriate editing sessions inbetween the run passes:

run Pass 1 function where your edited functions to call and capturing output with *tee* utility
⇒ pass 1's input file is an absolute pathed files list. See *How to use* **this** *gawk program* section
⇒ assess the messages outputted as to what guidelines need to be corrected
⇒ outputted messages have 2 parts: part 1 the filename and part 2 the quoted message

run Pass 2 function where u comment out pass 1 call and uncomment pass 2 function call
⇒ inside the *pass2_correct* function, uncomment the appropriate correction function to call
⇒ the captured messages from *tee* of pass 1 is its inputted file
⇒ in awk/gawk terms this *gawk*'s record is composed of fields where the field separator is space
⇒ so the 2nd pass receives the 2 fields filename, and message

Output example of guideline violations from Pass 1:

⇒ "/yacco2/.DS_Store" 'Extended attributes -rwxr-xr-x@'
⇒ "/yacco2/.DS_Store" 'Write permissions -rwxr-xr-x@ to possibly delete file type: data'
⇒ "/yacco2/.DS_Store" 'Bypass file in zip'
⇒ "/yacco2/.gitignore" 'Auxilary file to be deleted'
⇒ "/yacco2/.nbattrs" 'Extended attributes -rwxr-xr-x+'
⇒ "/yacco2/.nbattrs" 'Write permissions -rwxr-xr-x+ to possibly delete file type: XML'
⇒ "/yacco2/.nbattrs" 'Bypass file in zip'
⇒ "/yacco2/bin/.DS_Store" 'Extended attributes -rwxr-xr-x@'
⇒ "/yacco2/bin/.DS_Store" 'Write permissions -rwxr-xr-x@ to possibly delete file type: data'
⇒ "/yacco2/bin/.DS_Store" 'Bypass file in zip'
⇒ "/yacco2/bin/man/man1/1lrerrors.log" 'Auxilary file to be deleted'
⇒ "/yacco2/bin/man/man1/1lrerrors.log" 'Empty file to be deleted'
⇒ "/yacco2/bin/man/man1/1lrtracings.log" 'Auxilary file to be deleted'
⇒ "/yacco2/bin/man/man1/1lrtracings.log" 'Empty file to be deleted'
⇒ "/yacco2/bin/man/man1/o2.man" 'Extended attributes -rw-r–r–@'
⇒ "/yacco2/bin/man/man1/o2linker.man" 'Extended attributes -rw-r–r–@'
⇒ "/yacco2/bin/man/man1/yacco2.man" 'Extended attributes -rw-r–r–@'
⇒ "/yacco2/bin/man/man1/yacco2cmd.tmp" 'Auxilary file to be deleted'
⇒ "/yacco2/bin/man/man1/yacco2cmd.tmp" 'Empty file to be deleted'
⇒ "/yacco2/bld_bash_APPLE" 'Extended attributes -rwxr-xr-x@'
⇒ "/yacco2/bld_bash_APPLE" 'Write permissions -rwxr-xr-x@ to possibly delete file type: ASCII'
⇒ "/yacco2/book/appendix_a.aux" 'Auxilary file to be deleted'
⇒ "/yacco2/book/ch_fsm.tex" 'Write permissions -rwxr-xr-x@ to possibly delete file type: LaTeX'

Running the initial program could potentially output the above sampling. If there are just 1 message type outputted then u would just focus on "pass 2" to correct it. Having a mix of messages means "pass 1" should be edited to just deal with the 1 message type at a time. This allows the homogenous output to be captured and edit the removal of files to bypass. Now the program can be edited to support the message type action correcting the violation and rerun. Each listed guideline type would go thru this edit behaviour: "pass 1" to see the violations, analyse its outcome, re-edit program, and then run the correction.

7. Anatomy of this *gawk* program.
It contains 3 *gawk* actions: BEGIN, Body, and END. This program's BEGIN and END actions are just read record stats. U can add your own code as this program is open to your creativity. The action Body contains the 2 passes: *pass1_guidelines_verify* and *pass2_correct*. To comment out a function call, add a `#` at the beginning of its line. To activate a commented out function call, just remove its *gawk*'s comment character: `#` .

8. How to generate the pdf document and this gawk program.

The *Cweb* "literate programming" framework is used to generate its pdf document and its gawk program from *ctan_chk.w*. Though Cweb emits a "c" type code, gawk code is very similar to "c".

0) Run from a terminal using bash.

. ctan_chk_bash

⇒ The various utilities used are outputed on the terminal

U are open to add your own code to the *ctan_chk.w* program and rerun the above script. If u do not have the *Cweb* framework installed, u can add the code to the *ctan_chk.gawk* file.

9. How to use this gawk program.

0) Run from a terminal using bash.

1) Generate a file containing your project's files whose path is absolute

find /absolute path of your project > xxx

⇒ xxx is a holding file containing your project's files

⇒ example of an absolute path: **/usr/local/yacco2**

2) Edit *ctan_chk.gawk* to what function calls u want to use inside *pass1_guidelines_verify*

3) Run gawk ctan_chk program verifying your project

gawk -f ctan_chk.gawk xxx # xxx is the file read containing the files to assess

⇒ it outputs files to correct on the terminal with appropriate guideline message

⇒ no outputted messages means your project passed with flying colours :}

⇒ To capture the verification output:

gawk -f ctan_chk.gawk xxx|tee yyy

⇒ Use of *tee* utility to capture the output into yyy file

10. Example: Checking auxilary files to delete.
Here are the steps to see whether there are files to remove/delete. File xxx contains the files to verify as exampled earlier in this document.
1) **gawk -f ctan_chk.gawk xxx**

2) Its output should have messages similar to this:
⇒ "/yacco2/bin/man/man1/1lrerrors.log" 'Auxilary file to be deleted'

3) Rerun **gawk -f ctan_chk.gawk xxx|tee yyy** where the output messages are put in **yyy** file.

4) Edit: comment out pass1_guidelines_verify and uncomment #pass2_correct in Body action code:
The gened *ctan_chk.gawk* program could have different commented "section no:" used in the below edit session. What is important are function names referenced: for example, *pass1_guidelines_verify* which you'll find in your *ctan_chk.gawk* program. Same comments apply against the other pieces of code being edited.

```
# section no:22*/
# section no28:*/
{
 pass1_guidelines_verify($1);
 #pass2_correct($1,$2);
}
```

to

```
# section no:22*/
# section no28:*/
{
 #pass1_guidelines_verify($1);
 pass2_correct($1,$2);
}
```

5) Edit: uncomment out #delete_file in pass2_correct:

```
# section no:28*/
# section no29:*/
function pass2_correct(filename,message)
{
 #remove_file_s_execute_attribute(filename,message);
 #remove_file_s_extended_attributes(filename,message);
 #delete_file(filename);
}
```

to

```
# section no:28*/
# section no29:*/
function pass2_correct(filename,message)
{
 #remove_file_s_execute_attribute(filename,message);
 #remove_file_s_extended_attributes(filename,message);
 delete_file(filename);
}
```

6) Save edited *ctan_chk.gawk* program and rerun with **yyy**:

⇒ **gawk -f ctan_chk.gawk yyy**

Each file being deleted is questioned and allows u to bypass it.
This is due to the **-i** option used in *rm -i file-to-delete*

Other guideline contraints would follow the same run / edit / rerun template above adjusting the program accordingly.

⇒ **Just remember to reset the program back to its initial setting.**

As an aside comment, u could have uncommented the *delete_file* call inside the *chk_auxiliary_files* function and do the correction while in the "assess guideline" pass. So why did u not express this before :{? Well the normal correction pattern is see what guidelines are violated by "pass 1', edit the program, and then rerun against "pass 2" correction. Once u are familiar with the program throw out my patterns and jig your own :}.

11. Function code sections.

12. ctan gawk's comments — author, license etc.

⟨ Emit gawk comments 12 ⟩ ≡

```
#
# Program: ctan_chk.gawk
#
# Author: Dave Bone
#
# License:
# This Source Code Form is subject to the terms of the GNU General Public License (version 3) .
# If a copy of the MPL was ¬distributed with this file ,
# You can obtain one at: "https://gnu.org/licenses/gpl.html" .
#
# Purpose: Implementation of some suggested CTAN guidelines that an upload project should respect .
# Correction functions help u cleanup the droppings .
# See www.ctan.org website for "upload␣guideline" document
# Read ctan_chk.pdf document describing the program with various run scenarios .
#
#
```

This code is used in section 34.

13. Helper functions.

14. Is file a directory.

Directory found returns 1.

```
⟨ is_file_a_directory 14 ⟩ ≡
  function is_file_a_directory(filename, filetype)
  {
    x = "file␣\"%s\"";
    y = sprintf(x, filename);
    y | getline a;
    close(y);
    split(a, parts);
    filetype[1] = parts[2];
    xx = parts[2];
    str = "^directory$";
    if (xx~str) {
      return 1;
    }
    return 0;
  }
```

This code is used in section 34.

15. Is file an executable.

This function **might be too platform dependent**.
If so then don't call it or find an alternative.
A return 1 means it's an executable.

```
⟨ is_file_an_executable 15 ⟩ ≡
  function is_file_an_executable(filename, filetype)
  {
    x = "file␣%s";
    y = sprintf(x, filename);
    y | getline a;
    close(y);
    split(a, parts);
    filetype[1] = parts[2];
    xx = parts[2];
    str = "(executable|POSIX|Mach-O|ELF)$";
    if (xx~str) {
      return 1;
    }
    return 0;
  }
```

This code is used in section 34.

16. Guideline assessment functions.

17. Check for auxiliary type files.
"str" contains a regular expression of file extensions to search for starting with a "." followed by the round bracket expression of extensions to search on ending by the end-of-string regular expression indicator: $. Inside the rounded bracket expression is the choice of extensions separated by | which is the "or" operator of a regular expression.

The double \ in str is due to how gawk parses the program code: it does a double pass:
1) on the string: str
2) on the regular expression called
Just having a "." at the start of regular expression means it is a single "wild character" to accept. To accept a period "." that starts the file extension, u have to escape it. Pass 1 for the literal string is the first escape sequence on \ the 2nd backslash. In pass 2 for the regular expession called, the \ escapes the "." to not interpret as the regular expression wild character.

str can be added to. Make sure u include the added extension started with the | when appended to before the closing off rounded bracked ")".

chk_auxiliary_files outputs a 2 part message: file name and message it considers as an auxiliary file to be deleted. Calling *delete_file* interacts with the user whether to delete it or not. It is commented out so that the messages can saved and reviewed before the correction pass takes place: See *Pass*2−− − *correct violated guidelines* section providing more information. U can uncomment the *delete_file* statement below if u prefer to delete the file in the assessment pass rather than the correction pass. Your call and temperment.

```
⟨ chk_auxiliary_files 17 ⟩ ≡
  function chk_auxiliary_files(filename)
  {
    if (is_file_a_directory(filename) ≡ 1) return 0;
    str = "\\.(ps|gitignore|git|aux|log|bbl|bcf|blg|brf|ilg|ind|idx|glo|loa|lof|lot|nav|ou\
        t|snm|vrb|toc|dvi|glg|gls|tmp|o|bak|mpx|scn|toc)$";
    if (filename~str) {
      a = "\"%s\" 'Auxilary file to be deleted'";
      b = sprintf(a, filename);
      print b;
      #delete_file(filename);
      return 1;
    }
    return 0;
  }
```
This code is used in section 34.

18. Check for extended attributes.
Basicly looks at the output from the "ls -al" of a file. If there are extended attributes it will be displayed at the end of the first field as "@" or "+". It outputs the filename and message to the terminal. Return 1 if found.

```
⟨ chk_extended_file_attributes 18 ⟩ ≡
  function chk_extended_file_attributes(filename)
  {
    x = "ls␣-al␣\"%s\"";
    y = sprintf(x, filename);
    y | getline a;
    close(y);
    number_of_fields = split(a, parts);
    if (number_of_fields < 9) return 0;
    xx = parts[1];
    str = "(@|+)$";
    if (xx ¬∼ str) {
      return 0;
    }
    a = "\"%s\"␣'Extended␣attributes␣%s'";
    b = sprintf(a, filename, parts[1]);
    print b;
    #remove_file_s_extended_attributes(filename, b);
    return 1;
  }
```

This code is used in section 34.

19. Check for empty files.
Uncomment out the *delete_file* statement below if u want to remove it in the assessment phase. Asks whether to delete the zero sized file and returns 1.

```
⟨ chk_empty_files 19 ⟩ ≡
  function chk_empty_files(filename)
  {
    if (is_file_a_directory(filename) ≡ 1) return 0;
    x = "ls␣-al␣\"%s\"";
    y = sprintf(x, filename);
    y | getline a;
    close(y);
    number_of_fields = split(a, parts);
    if (number_of_fields < 9) {
      print "ERROR␣===>␣ls␣-al␣should␣be␣9␣fields␣an\
          d␣it␣isn't:␣"a"␣no␣fields:␣" number_of_fields;
      return 0;
    }
    i = strtonum(parts[5]);
    if (i > 0) {
      return 0;
    }
    a = "\"%s\"␣'Empty␣file␣to␣be␣deleted'";
    b = sprintf(a, filename);
    print b;
    #delete_file(filename);
    return 1;
  }
```

This code is used in section 34.

20. Check for empty directory.
Empty directory found returns a 1.
This check requires u to assess what u want to do: delete it, or add an "info.txt" file inside it. For example in the "Yacco2" project, it is a full development framework of a compiler/compile, and its API library. It has empty Debug folders. Its Release folders contained content. Depending on what the user wanted to debug and link against, the empty Debug folder provided consistency to the framework and hence it was needed. So an "info.txt" file was created inside each Debug folder expressing their intent/use to the developer.

```
⟨ chk_empty_directory 20 ⟩ ≡
  function chk_empty_directory(filename)
  {
    if (is_file_a_directory(filename) ≡ 0) return 0;
    x = "du␣-sk␣\"%s\"";
    y = sprintf(x, filename);
    y | getline a;
    close(y);
    number_of_fields = split(a, parts);
    i = strtonum(parts[1]);
    if (i > 0) return 0;
    a = "\"%s\"␣'Empty␣directory␣to␣be␣deleted␣or␣needs␣to␣add␣info.txt␣file␣inside␣it'";
    b = sprintf(a, filename);
    print b;
    return 1;
  }
```

This code is used in section 34.

21. Check file permissions.
Bypass a directory. If the file does not have an executable attribute then exit stage gracefully.

```
⟨ chk_file_permissions 21 ⟩ ≡
  function chk_file_permissions(filename)
  {
    filetype[1] = "";
    if (is_file_a_directory(filename, filetype) ≡ 1) return 0;
    if (is_file_an_executable(filename, filetype) ≡ 1) return 0;
    x = "ls␣-al␣\"%s\"";
    y = sprintf(x, filename);
    y | getline a;
    close(y);
    number_fields = split(a, parts);
    str = "x";
    if (parts[1]~str) {
      a = "\"%s\"␣'Write␣permissions␣%s␣to␣possibly␣delete␣file␣type:␣%s'";
      b = sprintf(a, filename, parts[1], filetype[1]);
      print b;
      return 1;
    }
    return 0;
  }
```

This code is used in section 34.

22. Check file to bypass in zip.
This is just a check function that lists files to bypass in the zip file creation process.
An example of creating a zip file:

zip -r -ll yacco2.zip yacco2 -x '*DS_Store' -x '*.nbattrs'
⇒ -r recurse through the folder getting all its subfolders and their contents
⇒ -ll parameter reduces carriage return/line feed combo to line feed
⇒ -x parameter lists by regular expression the file(s) to bypass
⇒ The bypassed exampled files are Apple OSX specific

```
⟨ chk_file_to_bypass_in_zip 22 ⟩ ≡
  function chk_file_to_bypass_in_zip(filename)
  {
    str = ".(DS_Store|.nbattrs)$";
    if (filename~str) {
      a = "\"%s\" 'Bypass file in zip'";
      b = sprintf(a, filename);
      printb;
      return 1;
    }
    return 0;
  }
```

This code is used in section 34.

23. Correct guideline violations.

24. Remove file's execute attribute.
Due to security reasons, u cannot have an indirect bash script run against a list of files to change their ownership attributes. It will run but nothing will be changed.
So what to do?

⇒ gawk -f *ctan_chk.gawk* xxx |tee yyy
⇒ u can use your own temporary file name instead of yyy
⇒ just remember to substitute your file name in place of yyy
⇒ where the *chk_file_permissions* function is the only function run
⇒ and the other functions have been commented out

Edit yyy file containing the list of potential files, removing files keeping their execute attribute.

Run this below gawk script interactively from your bash terminal
⇒ This means copy the below script line and paste it in the bash terminal
gawk '{x="chmod a-x %s;";y=sprint(x,$1);system(y);}' yyy

Some comments on the above interactive *gawk* program:
1) It reads the yyy file where each line read contains 2 fields: filename, and message separated by a space
2) It runs the *chmod* utility by the **system** statement
3) Not sure about **chmod** utility, do a "man chmod" on your bash terminal for more information
4) If u boobooed on a file and removed the execute attribute, here is a correction:
⇒ **chmod a+x file-to-reimplement**

⟨ *remove_file_s_execute_attribute* 24 ⟩ ≡
```
function remove_file_s_execute_attribute(filename, message)
{
  print "Please␣read␣|ctan_chk.pdf|␣to␣find␣instructions␣on␣how␣remove␣execute␣attribute\
      ␣from␣a␣file";
}
```

25. Remove file's with extended attributes.
This function is platform dependent. The *xattr* is the utility to run on Apple's OSx platform to remove the extension.

⟨ *remove_file_s_extended_attributes* 25 ⟩ ≡
```
function remove_file_s_extended_attributes(filename, message)
{
  x = "echo␣%s\";xattr␣-c␣%s";
  y = sprintf(x, message, filename);
  print y;
  #y | getline a;
  #close(y);
}
```
This code is used in section 34.

26. Delete file.
The file name passed to it is displayed and asks whether it should be deleted.

```
⟨ delete_file 26 ⟩ ≡
  function delete_file(filename)
  {
    x = "rm␣-i␣\"%s\"";
    y = sprintf(x, filename);
    y | getline a;
    close(y);
  }
```

This code is used in section 34.

27. Pass 1 / 2 guidelines assessment and correction.

28. Pass 1 — guidelines assessment.

Using a text editor (un)comment out the appropriate function calls in *ctan_chk.gawk* that u'd like to verify. For example checking just the files to be deleted, comment out all the below functions except *chk_auxiliary_files*.

U can run pass 1 as is to see whether any of your files for upload do not pass the guidelines. From there u can adjust what function to deal with by commenting out the others fuctions and teeing out its found infidelities for pass 2 correction.

⟨ pass 1 guidelines verify 28 ⟩ ≡

```
function pass1_guidelines_verify(filename)
{
  chk_auxiliary_files(filename);
  chk_extended_file_attributes(filename);
  chk_empty_files(filename);
  chk_empty_directory(filename);
  chk_file_permissions(filename);
  chk_file_to_bypass_in_zip(filename);
}
```

This code is used in section 34.

29. Pass 2 — correct violated guidelines.

There are 3 functions whereby 2 of them do usefull things.

delete_file function asks whether its passed file should be deleted
⇒ Remove the -i in **rm -i** statement if u don't want the interactive question

remove_file_s_extended_attributes function is tailored to the Apple platform
⇒ Other platforms should have an equivalent *xattr -c* utility
⇒ **Caveat:** know what utility to use and adjust accordingly to remove the extension attribute
⇒ Have a read at https://en.wikipedia.org/wiki/Extended_file_attributes#cite_note-14

remove_file_s_execute_attribute tells u to read the **ctan_chk.pdf** document
⇒ on "how to remove" the execute attribute from a file

Uncomment the appropriate below function to call. And comment out *pass1_guidelines_verify* statement, and uncomment *pass2_correct* statement in *Body* action before reruning this program.

⟨ pass 2 guidelines correction 29 ⟩ ≡

```
function pass2_correct(filename, message)
{
  #remove_file_s_execute_attribute(filename, message);
  #remove_file_s_extended_attributes(filename, message);
  #delete_file(filename);
}
```

This code is used in section 34.

30. Gawk Begin, Body, End actions.

31. Begin.

```
⟨ BEGIN 31 ⟩ ≡
  BEGIN
  {
    rec_cnt = 0;
  }
```

This code is used in section 34.

32. Body.

```
⟨ Body 32 ⟩ ≡
  {
    pass1_guidelines_verify($1);
    #pass2_correct($1,$2);
  }
```

This code is used in section 34.

33. END.

```
⟨ END 33 ⟩ ≡
  END{
# print"no␣records␣read:␣"NR;
      }
```

This code is used in section 34.

34. Write out gawk program.

```
⟨ ctan_chk.gawk 34 ⟩ ≡
  ⟨ Emit gawk comments 12 ⟩;
  ⟨ is_file_a_directory 14 ⟩;
  ⟨ is_file_an_executable 15 ⟩;
  ⟨ chk_auxiliary_files 17 ⟩;
  ⟨ remove_file_s_extended_attributes 25 ⟩;
  ⟨ chk_file_permissions 21 ⟩;
  ⟨ chk_extended_file_attributes 18 ⟩;
  ⟨ chk_empty_files 19 ⟩;
  ⟨ chk_empty_directory 20 ⟩;
  ⟨ delete_file 26 ⟩;
  ⟨ chk_file_to_bypass_in_zip 22 ⟩;
  ⟨ pass 1 guidelines verify 28 ⟩;
  ⟨ pass 2 guidelines correction 29 ⟩;
  ⟨ BEGIN 31 ⟩;
  ⟨ Body 32 ⟩;
  ⟨ END 33 ⟩;
```

35. Index.

⟨ Emit gawk comments 12 ⟩ Used in section 34.
⟨ `ctan_chk.gawk` 34 ⟩
⟨ pass 1 guidelines verify 28 ⟩ Used in section 34.
⟨ pass 2 guidelines correction 29 ⟩ Used in section 34.
⟨ `BEGIN` 31 ⟩ Used in section 34.
⟨ *Body* 32 ⟩ Used in section 34.
⟨ `END` 33 ⟩ Used in section 34.
⟨ *chk_auxiliary_files* 17 ⟩ Used in section 34.
⟨ *chk_empty_directory* 20 ⟩ Used in section 34.
⟨ *chk_empty_files* 19 ⟩ Used in section 34.
⟨ *chk_extended_file_attributes* 18 ⟩ Used in section 34.
⟨ *chk_file_permissions* 21 ⟩ Used in section 34.
⟨ *chk_file_to_bypass_in_zip* 22 ⟩ Used in section 34.
⟨ *delete_file* 26 ⟩ Used in section 34.
⟨ *is_file_a_directory* 14 ⟩ Used in section 34.
⟨ *is_file_an_executable* 15 ⟩ Used in section 34.
⟨ *remove_file_s_execute_attribute* 24 ⟩
⟨ *remove_file_s_extended_attributes* 25 ⟩ Used in section 34.

CTAN˙CHK