ispell.info

COLLABORATORS

	TITLE : ispell.info	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		September 19, 2022	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

1 ispell.info **1**

1.1 ispell.info . . . 1
1.2 ispell.info/License . . . 2
1.3 ispell.info/Emacs . . . 2
1.4 ispell.info/Word . . . 3
1.5 ispell.info/Buffer . . . 4
1.6 ispell.info/Region . . . 4
1.7 ispell.info/Old Emacs . . . 5
1.8 ispell.info/Standalone . . . 5
1.9 ispell.info/Ask . . . 6
1.10 ispell.info/Private . . . 6
1.11 ispell.info/Compatibility . . . 7
1.12 ispell.info/Command summary . . . 7
1.13 ispell.info/Near misses . . . 8
1.14 ispell.info/Arguments . . . 8
1.15 ispell.info/Interface . . . 9
1.16 ispell.info/New interface . . . 9
1.17 ispell.info/Old interface . . . 11
1.18 ispell.info/Dictionary flags . . . 11
1.19 ispell.info/History . . . 13

Chapter 1

ispell.info

1.1 ispell.info

Ispell is a program that helps you to correct typos in a file, and
to find the correct spelling of words. When presented with a word that
is not in the dictionary, ispell attempts to find near misses that
might include the word you meant.

This manual describes how to use ispell, as well as a little about
its implementation.

License
Licensing information.

Emacs
Using ispell from emacs

Standalone
Using ispell by itself

Ask
Using ispell to look up individual words

Private
Your private dictionary

Compatibility
Compatibility with the traditional spell program

Command summary
All commands in emacs and standalone modes

Implementation details:

Near misses
Definition of a near miss

Arguments

Flags to the ispell command

Interface
How other programs can use ispell

Dictionary flags
How the suffix stripper works

History
Where it came from; authors

1.2 ispell.info/License

Licensing Information
****

Ispell is free; this means that everyone is free to use it and free to redistribute it on a free basis. Ispell is not in the public domain; it is copyrighted and there are restrictions on its distribution, but these restrictions are designed to permit everything that a good cooperating citizen would want to do. What is not allowed is to try to prevent others from further sharing any version of ispell that they might get from you.

Specifically, we want to make sure that you have the right to give away copies of ispell, that you receive source code or else can get it if you want it, that you can change ispell or use pieces of it in new free programs, and that you know you can do these things.

To make sure that everyone has such rights, we have to forbid you to deprive anyone else of these rights. For example, if you distribute copies of ispell, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must tell them their rights.

Also, for our own protection, we must make certain that everyone finds out that there is no warranty for ispell. If these programs are modified by someone else and passed on, we want their recipients to know that what they have is not what we distributed, so that any problems introduced by others will not reflect on our reputation.

The precise conditions of the licenses for ispell are found in the General Public Licenses. The programs that are part of GNU Emacs are covered by the GNU Emacs copying terms (see License), and other programs are covered by licenses that are contained in their source files.

1.3 ispell.info/Emacs

```
                Using ispell from emacs
=======================


                Word
                 Checking a single word

                Buffer
                 Checking a whole buffer

                Region
                 Checking a region

                Old Emacs
                 Using ispell with older versions of emacs
    Look here if M-$ calls the old spell program in your emacs.
```

1.4 ispell.info/Word

```
Checking a single word
----------------------
```

The simplest emacs command for calling ispell is 'M-$' (meta-dollar. On some terminals, you must type ESC-$.) This checks the spelling of the word under the cursor. If the word is found in the dictionary, then a message is printed in the echo area. Otherwise, ISPELL attempts to generate near misses.

If any near misses are found, they are displayed in a separate window, each preceded by a digit. If one of these is the word you wanted, just type its digit, and it will replace the original word in your buffer.

If no near miss is right, or if none are displayed, you have four choices:

I
: Insert the word in your private dictionary. Use this if you know that the word is spelled correctly.

A
: Accept the word for the duration of this editing session, but do not put it in your private dictionary. Use this if you are not sure about the spelling of the word, but you do not want to look it up immediately. The next time you start ispell, it will have forgotten any accepted words. You can make it forget accepted words at any time by typing M-x reload-ispell.

SPC
: Leave the word alone, and consider it misspelled if it is checked again.

R
Replace the word. This command prompts you for a string in the minibuffer. You may type more than one word, and each word you type is checked again, possibly finding other near misses. This command provides a handy way to close in on a word that you have no idea how to spell. You can keep trying different spellings until you find one that is close enough to get a near miss.

L
Lookup. Display words from the dictionary that contain a specified substring. The substring is a regular expression, which means it can contain special characters to be more selective about which words get displayed. See Regexps.

If the only special character in the regular express is a leading ^, then a very fast binary search will be used, instead of scanning the whole file.

Only a few matching words can be displayed in the ISPELL window. If you want to see more, use the look program directly from the shell.

Of course, you can also type?G to stop the command without changing anything.

If you make a change that you don't like, just use emacs' normal undo feature See undo.

1.5 ispell.info/Buffer

Checking a whole buffer

If you want to check the spelling of all the words in a buffer, type the command M-x ispell. This command scans the file, and makes a list of all the misspelled words. When it is done, it moves the cursor to the first word on the list, and acts like you just typed M-$ See
Word
.

When you finish with one word, the cursor is automatically moved to the next. If you want to stop in the middle of the list type Q or . Later, you can pick up where you left off by typing C-X $.

1.6 ispell.info/Region

Checking a region

You may check the words in the region with the command M-x ispell-region. See See mark.

The commands available are the same as for checking a whole buffer.

1.7 ispell.info/Old Emacs

Old Emacs
=========

Until ispell becomes part of the standard emacs distribution, you will have to explicitly request that it be loaded. Put the following lines in your emacs init file See init file.

```
(autoload 'ispell "ispell" "Run ispell over buffer" t)
(autoload 'ispell-region "ispell" "Run ispell over region" t)
(autoload 'ispell-word "ispell" "Check word under cursor" t)
(define-key esc-map "$" 'ispell-word)
```

(It will do no harm to have these lines in your init file even after ispell is installed by default.)

1.8 ispell.info/Standalone

Using ispell by itself
======================

To check the words in a file, give the command ispell FILE. This will present a screen of information, and accept commands for every word that is not found in the dictionary.

The screen shows the offending word at the top, as well as two lines of context at the bottom. If any near misses are found, they are shown in the middle of the screen, each preceded by a digit.

You may use the same commands as inside of emacs to accept the word, place it in your private dictionary, select a near miss, or type a replacement See
Word
. You may also choose from the following commands:

?
Print a help message.

Q
Quit. Accept the rest of the words in the file and exit.

X
Exit. Abandon any changes made to this file and exit immediately.

```
     You are asked if you are sure you want to do this.

!
     Shell escape.  The shell command that you type is executed as a
     subprocess.

^Z
     Suspend.  On systems that support job control, this suspends
     ISPELL.  On other systems it executes a subshell.

^L
     Redraw the screen.
```

If you type your interrupt character (usually?C or DEL), then ispell will immediately enter its command loop. If ispell was generating near misses at the time, then all that it had found so far will be displayed, along with a message stating that there might be more, and that you can type RET to generate them. If it was scanning the file, it will display (INTERRUPT) where it would normally display a bad word, and the commands that change the file will be disabled.

The feature is handy if you have left out a space between words, and ispell is futilely looking up the 1000 potential near misses for a string that has twenty letters.

1.9 ispell.info/Ask

```
Using ispell to look up individual words
========================================
```

When ispell is run with no arguments, it reads words from the standard input. For each one, it prints a message telling whether it is in the dictionary. For any words not in the dictionary, near misses are computed, and any that are found are printed.

```
     % ispell
     word: independant
     how about: independent
     word: xyzzy
     not found
     word:?D
```

1.10 ispell.info/Private

```
Your private dictionary
=======================
```

Whenever ispell is started the file ispell.words is read from your home directory (if it exists). This file contains a list of words, one

per line, and neither the order nor the case of the words is important. Ispell will consider all of the words good, and will use them as possible near misses.

The I command adds words to ispell.words, so normally you don't have to worry about the file. You may want to check it from time to time to make sure you have not accidentally inserted a misspelled word.

1.11 ispell.info/Compatibility

Compatibility with the traditional spell program
==========

The -u flag tells ispell to be compatible with the traditional spell program. This flag is automatically turned on if the program is invoked by the name spell.

This flag causes the following behavior:

All of the files listed as arguments (or the standard input if none) are checked, and misspellings are printed on the standard output. The output is sorted, only one instance of each word appears (however, a word may appear more than once with different capitalizations.)

You may specify a file containing good words with +filename.

The troff commands .so and .nx (to include a file, or switch to a file, respectively) are obeyed, unless you give the flag -i.

The other spell flags -v, -b, -x and -l are ignored.

By the way, ispell seems to be about three times faster than traditional spell.

1.12 ispell.info/Command summary

All commands in emacs and standalone modes
==========

Commands valid in both modes:

DIGIT Select a near miss I Insert into private dictionary A Accept for this session SPACE Skip this time R Replace with one or more words L Lookup: search the dictionary using a regular expression

Standalone only:

Q Accept rest of file X Abandon changes ! Shell escape ? Help ^Z Suspend ^L Redraw screen ^C Give up generating near misses, or

```
show position in file

   Emacs only:

   M-$      Check word M-x ispell   Check buffer M-x ispell-region  Check
region M-x reload-ispell  Reread private dictionary M-x
kill-ispell   Kill current subprocess, and start a new one      next
time ^G     When in M-x ispell, stop working on current       bad word list ^X
$     Resume working on bad word list.
```

1.13 ispell.info/Near misses

```
Definition of a near miss
=========================

   Two words are near each other if they can be made identical with one
of the following changes to one of the words:

     Interchange two adjacent letters.
     Change one letter.
     Delete one letter.
     Add one letter.

   Someday, perhaps ispell will be extended so that words that sound
alike would also be considered near misses.  If you would like to
implement this, see Knuth, Volume 3, page 392 for a description of the
Soundex algorithm which might apply.
```

1.14 ispell.info/Arguments

```
                Flags to the ispell command
============================

   Ispell's arguments are parsed by getopt(3).  Therefore, there is
considerable flexibility about where to put spaces between arguments.
The way to be safe is to give only one flag per dash, and put a space
between a flag and its argument.

   If ispell is run with no arguments, it enters ask mode See
                Ask
                .
With one or more file name arguments, it interactively checks each one.

-p privname
     Use privname as the private dictionary.

-d dictname
     Use dictname as the system dictionary.  You may also specify a
     system dictionary with the environment variable ISPELL_DICTIONARY.
```

```
-l
     List mode.  Scan the file, and print any misspellings on the
     standard output.  This mode is compatible with the traditional
     spell program, except that the output is not sorted.  See

                Compatibility
                .

-u
     Compatibility mode. See
                Compatibility
                .

-a
     Old style program interface, See
                Interface
                .

-S
     New program interface, See
                Interface
                .

-D
     Print the dictionary on the standard output with flags.

-E
     Print the dictionary on the standard output with all flags
     expanded.
```

1.15 ispell.info/Interface

```
                How other programs can use ispell
================================

   Ispell can be used as a subprocess communicating through a pipe.  Two
interfaces are available:



                 New interface
                  New style, for EMACS

                 Old interface
                  Old style, like ITS
```

1.16 ispell.info/New interface

New style, for EMACS
====================

To use this interface, start ispell with the '-S' flag. Ispell will print a version number and greeting message that looks like:

```
(1 "ISPELL V4.0")=
```

The number is the version number of the protocol to be spoken over the pipe. The string is a message possibly of interest to the user.

All messages from ispell end in an equal sign, and ispell guarantees not to print an equal sign except to end a message. Therefore, if you do not want to deal with the greeting, just throw away characters until you get to an equals.

Ispell then reads one line commands from the standard input, and writes responses on the standard output.

If a command does not start with a colon, then it is considered a single word. The word is looked up in the dictionary, and if it is found, the response is t. If the word is not in the dictionary, and no near misses can be found, then the response is nil. If there are near misses, the response is a line containing a list of strings in lisp form. For example:

```
INPUT   OUTPUT  the   t   xxx   nil   teh   ("tea" "ten"
"the")
```

The near miss response is suitable for passing directly to the lisp read function, but it can also be parsed simply in C. In particular, ispell promises that the list will appear all on one line, and that the structure will not change. A parser that reads the whole line, then treats the parentheses and quotes as whitespace will work fine.

The list will contain a maximum of ten strings, and each string will be no longer than 40 characters. Also, the capitalization of the near misses is the same as the input word.

Colon commands

If the input line starts with a colon, then it is one of the following commands:

:file filename Run the word checker over the named filename. The response is zero or more lines each containing a number. The numbers are file offsets of words that do not appear in the dictionary. Since the near miss checker is not run, this is fairly fast.

After the last number, there will be a line containing either t if the checker got to the end of the file, or nil if it received an interrupt. If ispell ignores any interrupts received except while scanning a file.

:insert word Place word in the private dictionary.

:accept word Do not complain about word for the rest of the session.

:dump Write the private dictionary.

:reload Reread the private dictionary.

:tex Enable the tex parser for future :file commands.

:troff Enable the tex parser for future :file commands.

:generic Disable any text formatter parsers for future :file commands.

1.17 ispell.info/Old interface

Old style, like ITS
==========

To use this interface, start ispell with the '-a' flag. Ispell will read words from its standard input (one per line), and write a one line of output for each one.

If the first character of the line is *, then word was found in the dictionary. (Other versions of ispell made a distinction between words that were found directly, and words that were found after suffix removal. These lines began with +, followed by a space, then followed by the root word. To remain compatible with these version, treat + and * the same.)

If the line starts with &, then the input word was not found, but some near misses were found. They are listed on the output line separated by spaces. Also, the output words will have the same capitalization as the input.

Finally, if the line starts with #, then the word was not in the dictionaries, and no near misses were found.

INPUT OUTPUT the * xxx # teh & tea ten the

1.18 ispell.info/Dictionary flags

How the suffix stripper works
==========

This section is excerpted from the ITS spell.info file.

Words in SPELL's main dictionary (but not the private dictionary) may have flags associated with them to indicate the legality of suffixes without the need to keep the full suffixed words in the

```
dictionary.  The flags have "names" consisting of single  letters.
Their  meaning  is  as follows:

   Let  #  and  @  be  "variables"  that can stand for any letter.
Upper case letters are constants.  "..."  stands  for  any  string  of
zero  or  more letters,  but note that no word may exist in the
dictionary which is not at least 2 letters long, so, for example, FLY
may not be produced  by  placing the  "Y"  flag  on "F".  Also, no flag
is effective unless the word that it creates is at least 4 letters
long,  so,  for  example,  WED  may  not  be produced by placing the
"D" flag on "WE".

   "V" flag:          ...E -> ...IVE  as in CREATE -> CREATIVE
if # .ne. E, ...# -> ...#IVE  as in PREVENT -> PREVENTIVE

   "N" flag:          ...E -> ...ION  as in CREATE -> CREATION
...Y -> ...ICATION  as in MULTIPLY -> MULTIPLICATION         if # .ne.
E or Y, ...# -> ...#EN  as in FALL -> FALLEN

   "X" flag:          ...E -> ...IONS  as in CREATE -> CREATIONS
...Y -> ...ICATIONS  as in MULTIPLY -> MULTIPLICATIONS         if #
.ne. E or Y, ...# -> ...#ENS  as in WEAK -> WEAKENS

   "H" flag:          ...Y -> ...IETH  as in TWENTY -> TWENTIETH
if # .ne. Y, ...# -> ...#TH  as in HUNDRED -> HUNDREDTH

   "Y" FLAG:          ... -> ...LY  as in QUICK -> QUICKLY

   "G" FLAG:          ...E -> ...ING  as in FILE -> FILING         if #
.ne. E, ...# -> ...#ING  as in CROSS -> CROSSING

   "J" FLAG"          ...E -> ...INGS  as in FILE -> FILINGS         if
# .ne. E, ...# -> ...#INGS  as in CROSS -> CROSSINGS

   "D" FLAG:          ...E -> ...ED  as in CREATE -> CREATED         if
@ .ne. A, E, I, O, or U,                 ...@Y -> ...@IED  as in IMPLY
-> IMPLIED         if # .ne. E or Y, or (# = Y and @ = A, E, I, O, or U)
               ...@# -> ...@#ED  as in CROSS -> CROSSED
                or CONVEY -> CONVEYED

   "T" FLAG:          ...E -> ...EST  as in LATE -> LATEST         if @
.ne. A, E, I, O, or U,                 ...@Y -> ...@IEST  as in DIRTY
-> DIRTIEST         if # .ne. E or Y, or (# = Y and @ = A, E, I, O, or
U)                 ...@# -> ...@#EST  as in SMALL -> SMALLEST
                    or GRAY -> GRAYEST

   "R" FLAG:          ...E -> ...ER  as in SKATE -> SKATER         if @
.ne. A, E, I, O, or U,                 ...@Y -> ...@IER  as in MULTIPLY
-> MULTIPLIER         if # .ne. E or Y, or (# = Y and @ = A, E, I, O,
or U)                  ...@# -> ...@#ER  as in BUILD -> BUILDER
                     or CONVEY -> CONVEYER

   "Z FLAG:          ...E -> ...ERS  as in SKATE -> SKATERS         if @
.ne. A, E, I, O, or U,                 ...@Y -> ...@IERS  as in
MULTIPLY -> MULTIPLIERS         if # .ne. E or Y, or (# = Y and @ = A,
E, I, O, or U)                 ...@# -> ...@#ERS  as in BUILD ->
BUILDERS                              or SLAY -> SLAYERS
```

```
   "S" FLAG:         if @ .ne. A, E, I, O, or U,                  ...@Y
-> ...@IES  as in IMPLY -> IMPLIES         if # .eq. S, X, Z, or H,
           ...# -> ...#ES  as in FIX -> FIXES         if # .ne. S, X,
Z, H, or Y, or (# = Y and @ = A, E, I, O, or U)                 ...# ->
...#S  as in BAT -> BATS                                  or CONVEY ->
CONVEYS

   "P" FLAG:         if @ .ne. A, E, I, O, or U,                  ...@Y
-> ...@INESS  as in CLOUDY -> CLOUDINESS         if # .ne. Y, or @ = A,
E, I, O, or U,                  ...@# -> ...@#NESS  as in LATE ->
LATENESS                                 or GRAY -> GRAYNESS

   "M" FLAG:         ... -> ...'S  as in DOG -> DOG'S
```

Note: The existence of a flag on a root word in the directory is not by itself sufficient to cause SPELL to recognize the indicated word ending. If there is more than one root for which a flag will indicate a given word, only one of the roots is the correct one for which the flag is effective; generally it is the longest root. For example, the "D" rule implies that either PASS or PASSE, with a "D" flag, will yield PASSED. The flag must be on PASSE; it will be ineffective on PASS. This is because, when SPELL encounters the word PASSED and fails to find it in its dictionary, it strips off the "D" and looks up PASSE. Upon finding PASSE, it then accepts PASSED if and only if PASSE has the "D" flag. Only if the word PASSE is not in the main dictionary at all does the program strip off the "E" and search for PASS.

Therefore, never install a flag by hand. Instead, just add complete new words to the dictionary file, then use the build program with the options '-a -r' to replace as many roots with flags as possible.

1.19 ispell.info/History

```
Where it came from
==================
```

I first came across ispell on TOPS-20 systems at MIT. I tracked it down to ITS where I found the PDP-10 assembly program. It appeared that it had been in use at the MIT-AI lab since at least the late 1970's. I think it case from California before then.

I wrote the first C implementation in the spring of 1983, mostly working from the ITS INFO file.

The present version was created in early 1988, and was motivated by the desire to make it run on 80286's, and to provide a better interface for GNU EMACS.

There is another widely distributed version of ispell, which was forked from my 1983 version and has a different set of features and clever extensions. It is available from the directory /u/public/ispell at celray.cs.yale.edu.

People who have contributed to various versions of ispell include:
Walt Buehring, Mark Davies, Geoff Kuenning, Rober McQueer, Ashwin Ram,
Greg Schaffer, Perry Smith, Ken Stevens, and Andrew Vignaux.

Pace Willisson
pace@ai.mit.edu pace@hx.lcs.mit.edu
(617) 625-3452