

# Spell Correcting Documents under Linux

Peter Hiscocks

Department of Electrical and Computer Engineering  
Ryerson Polytechnic University

*phiscock@ee.ryerson.ca*

July 31, 2004

## Contents

|          |  |          |
|----------|--|----------|
| <b>1</b> | <b>Context</b>   | <b>2</b> |
| <b>2</b> | <b>Introduction</b>  | <b>2</b> |
| <b>3</b> | <b>Interactive Mode</b>  | <b>3</b> |
| <b>4</b> | <b>Useful Flags</b>  | <b>3</b> |
| <b>5</b> | <b>Spell-Checking Multiple Files</b>                             | <b>5</b> |
| <b>6</b> | <b>Spell-Correcting Multiple Files</b>                           | <b>6</b> |
| <b>7</b> | <b>Appendix: Spell-Checking Multiple Files, Alternate Method</b> | <b>7</b> |
| <b>8</b> | <b>The Last Word</b>   | <b>8</b> |
|          | <b>References</b>  | <b>9</b> |

# 1 Context

To make the context of this document perfectly clear, the approach described here is useful when the document typesetter program  $\LaTeX$  is being used to typeset the document. It will not be useful if you are using a word processor such as `Abiword`, `Kword` or `Star Office`. Those programs contain a built-in spell checker.

Under  $\LaTeX$ , the basic *source code* is created using an editor program. The document source code is spell-checked. It is then processed by  $\LaTeX$ , to produce a `.dvi` file which is further converted to postscript and then `.pdf` format.

My view is that a short document such as a letter can most easily be produced using a word processor. A large document containing mathematics, diagrams and graphs is best produced using  $\LaTeX$  and related tools.

# 2 Introduction

When I was a kid in public school, I took part in a spelling contest. At the end of the contest another student and I were the last ones left. The test word was 'solder'. As a budding electrical engineer, I definitely knew how to spell that word - and the other contestant didn't. She complained that it was an unfair word - and she was probably right. But I won the contest and ever since then, I've thought of myself as a great speller. So I'm rather cavalier about checking my documents for spelling mistakes.

Well. Fast forward a few gazilion years. I'm in the middle of a large book writing project. At this point, the book is contained in 335 files, all unchecked for spelling. Furthermore, I've discovered that the word I've been spelling as *hysterysis* is actually *hysteresis*<sup>1</sup>. As it turns out, there are probably hundreds of spelling mistakes. Now what?

Fortunately, the Linux operating system contains the utilities for avoiding or fixing this situation. The program `ispell` (*Interactive Spell*) is available to detect and possibly correct spelling mistakes in documents.

`ispell` can be used in *interactive* mode, in which each query is made on the fly. This is suitable for people who check each file as it is written.

`ispell` can also be used in *batch* mode, in which queries are written to another file or piped to another program for further processing. This is suitable for people like me, who prefer (or, more accurately, *neglect*) to do all the spell-checking until the end of the writing.

`ispell` works well with  $\LaTeX$ . It recognizes that files ending in the suffix `.tex` should be treated specially, ignoring many of the  $\LaTeX$  keywords.

This document provides the information to perform spell-checking and, if necessary, to make wholesale changes in spelling, across one file or many.

## Dictionaries

`ispell` uses two dictionaries: a dictionary that comes with the program, and a *user dictionary*.

The main dictionary is in `/usr/lib/ispell`. It is in hashed format for fast lookup, so it cannot be edited directly by hand.

The user dictionary is in your home directory as `.ispell_english` (a hidden file). The user dictionary can be edited by hand to add or remove words. This can be useful if you incorrectly add a word to your user directory or you want to add a set of words that `ispell` should but does not recognize.

---

<sup>1</sup>I like my spelling of better, but what can you do?

### 3 Interactive Mode

In interactive mode, `ispell` scans through the specified document and queries possible spelling mistakes. A typical invocation to spell a  $\LaTeX$  document would be:

---

```
ispell zener-amplifier.tex
```

---

At each questionable word, `ispell` shows context line with the questionable word highlighted. A menu of possible actions appears at the bottom of the screen. At that point, the operator can take a number of possible actions:

| Key      | Operation  |
|----------|--|
| <space>  | Accept the word as it is, this time only, ie, skip this one.   |
| <number> | Replace the word with one of the suggested words, indicated by the number.   |
| R        | Replace the word. <code>ispell</code> prompts for input. The user dictionary is not updated, but the corrected document is written out at the end. So you might use this for a <i>one-time</i> correction. |
| I        | Replace the word and <b>insert</b> it into the user directory. This dictionary update takes effect immediately, so that the word is recognized thereafter.   |
| L        | Use the dictionary <b>lookup</b> facility. The program prompts for a lookup string. When carriage return is pressed, the program gives a list of words similar to that word.                               |
| U        | Accept the word and add an all-lowercase version to the dictionary. Use this instead of <b>I</b> when the word to be replaced is capitalized.  |
| X        | Skip any further corrections and write out the modified file. This is useful if you want to exit part way through a correction session.  |
| Q        | Quit without changing the file. This is useful if you have screwed up in some fashion and want to escape without actually changing the original file.  |

When `ispell` has completed scanning through the document, it copies the original file with `.bak` appended and then replaces the original file with the corrected version.

### 4 Useful Flags

- The `-p` flag causes `ispell` to use a different personal directory than the default, which is `.ispell_english` in your home directory. This is useful if you are working on a particular project that has a large number of words that must be recognized as legal but are not likely to recur in another project. So the command

```
ispell -p foobar zener-amplifier.tex
```

would cause `ispell` to use a personal directory, located in the current directory, named `foobar`.

- The `-d` flag selects a different system directory. This allows spell checking under different languages. For example:

```
ispell -d deutsch zener-amplifier.tex
```

chooses a German directory.

- The `-L` flag specifies the number of context lines. For example,

```
ispell -L30 zener-amplifier.tex
```

displays 30 lines of text preceding the word of interest. This may help in determining the location of an error.

- The `-x` flag prevents the creation of a backup file.
- The `-l` (*list*, that's the letter *el*, not the numeral *one*) option directs `ispell` to produce direct mis-spelled words to the standard output. So, for example:

---

```
cat zener-amplifier.tex | ispell -l
```

---

causes the file `zener-amplifier.tex` to be piped into `ispell`. Then `ispell` generates on the screen a list of words, for example:

```
clearpage  
bsection  
sec  
wrapfigure  
ef  
pictex  
ef  
wrapfigure  
ref  
ef  
mV
```

all of which it regards as mis-spellings.

There are various ways we could use this list. For example, we could redirect it into a file named `boo-boos`.

---

```
cat zener-amplifier.tex | ispell -l > boo-boos
```

---

Then we could cut and paste from the file `boo-boos` into the user directory to update it.

This option is useful in checking multiple files, as described in detail below.

## 5 Spell-Checking Multiple Files

The `ispell -l` command can be wrapped in a shell script to scan through a collection of files and generate *all* the mis-spelled words in all the files. Some of them will be duplicates. You can then use the `sort` and `uniq` utility programs to eliminate duplicates in that file.

Putting all these ideas together, we have the following shell script:

---

```
# This script spell checks every .tex file in the current directory.
# Type this file into an editor and call the file 'spell-checker.sh'
# or some name of your choice.
# Use the 'chmod' command to make sure this script is executable.
# When run, the output goes into a file called {\tt booboos.txt}.
# It is easiest to run in the directory of the files being checked.
# You'll have to give the complete path name to the script to run it.

rm booboos.txt
rm temp.txt
for file in *.tex
do
    echo $file
    cat $file | ispell -l >> temp.txt
done
echo "Sorting mistakes..."
cat temp.txt | sort | uniq > booboos.txt
echo "Done"
```

---

This shell-script works in three phases.

1. Remove the files `booboos.txt` and `temp.txt`.
2. Show the name of each file as it is being checked and route the spelling errors into the temporary file `temp.txt`. Notice the use of the concatenation operator `>>`, which appends the result of each file to the existing contents of `temp.txt`.
3. Sort the contents of the temporary file and remove redundant entries, copying the result into `booboos.txt`.

When I did this on my 335 files, the script generated a list of about 1300 unique 'errors' in `booboos.txt`. It was evident that many of these were spurious bits of labels in the  $\text{\LaTeX}$  text and should be ignored. So I cut and pasted all the irrelevant material into a file called `vv-dictionary.txt`. Then I reran the shell script with `ispell` configured by means of the `-p` flag to ignore words in that dictionary:

```
cat $file | ispell -l -p vv-dictionary.txt >> temp.tx
```

This time, the shell script generated about 110 spelling errors. Most of these were genuine errors on my part. Now I could find the location of those errors and correct them. For example, if `amlifier` is an indicated error, we can find all the occurrences of that error with the command:

---

```
grep amlifier *.tex
```

---

This is fine for the single occurrence of an error. However, it turned out that I misspelled `similarly` as `similarily` some 30 times. Hand correcting all 30 files is rather tedious, but there is an easier way, as described in section 6.

### On the User Dictionary

In the process of creating a user directory I discovered that

- the user dictionary should be in the same directory as the spell-checking is taking place.
- the entries in the dictionary, one per line, must contain no spaces.

## 6 Spell-Correcting Multiple Files

If the spelling problem is in one file, the file may be loaded into an editor and changed. If the spelling problem is in one hundred files, then some sort of 'batch correction' is required. The *stream editor* program `sed` is the appropriate tool for this. An example script using `sed` is shown below.

---

```
# This script substitutes one text string for another.
# To use it:
#     modify the type of file in the first line.
#     modify the original and substitute string in the 4th (sed) line
#     eg: sed 's/original/new/' changes 'original' to 'new'.

for file in *.tex
do
    echo $file
    sed 's/amlifier/amplifier/' $file > sedtemp
    mv sedtemp $file
done
```

---

This changes all occurrences of `amlifier` to `amplifier`. As shown the script acts on files with a `.tex` subscript. You would change this as required.

### Working from a List of Spelling Changes

If there are a large number of corrections in a large number of files, the `sed` utility can be provide with a list of changes in a separate file, as shown in the following script.

---

```
# This script substitutes text strings from a list.
# It is useful in correcting common spelling mistakes.
# It assumes that latex files are being corrected. If not, change the *.tex
# as required.
```

```

# To use it:
# make sure your spelling corrections are listed in the file
# 'spellcorrections.txt' in the form:
#         s/oldstring/newstring/
# Of course, this script can be used to make any substitutions,
# not just spelling corrections.

for file in *.tex
do
    echo $file
    sed -f spellcorrections.txt $file > sedtemp
    mv sedtemp $file
done

```

---

This script lists each file as it scans through all of the `.tex` files in the directory. If and when it finds a pattern in the substitution list in `spellcorrections.txt`, it makes the specified substitution. As it completes each file, it overwrites it with the new version.

A three-entry example of `spellcorrections.txt` is:

```

s/amplifer/amplifier/
s/analagous/analogous/
s/disappate/dissipate/

```

Obviously, you should

- run test versions and make sure they are doing what is required.
- back up all your files (by copying them to another directory, for example)

before running this script.

## 7 Appendix: Spell-Checking Multiple Files, Alternate Method

I'm not sure that the this method will be useful, but in the interest of completeness, here it is.

If you wish to pipe some text through the `ispell` program, you can also use the `-a` option as in the following:

---

```

cat zener-amplifier.tex | ispell -a | more

```

---

This causes the `tex` file `zener-amplifier.tex` to be processed and then paged by `more`. That is, one page of processing is displayed at a time. Pressing the space bar moves to the next page.

Alternatively, you could pipe the output into a file called `junk` which could then be examined:

```

cat zener-amplifier.tex | ispell -a > junk

```

For each word in the file, `ispell` prints out a line. The line consists of a preface character, with the following subsequent optional text:

| Prefix | Situation  | Optional Text   |
|--------|--|---|
| *      | The word is found in the main or personal dictionary, ie, it is correct    | None  |
| +      | The word is a legal root word plus a prefix or suffix.                     | The root word.  |
| -      | The word was found by concatenation of two words                           | None  |
| &      | The word is not in the main or personal dictionary, ie, it is not correct. | The original word plus (possibly) some guesses for the correct word |
| #      | The word is not in the main or personal dictionary                         | The program has no ideas for the correct word.                      |

Summing up, the `&` and `#` symbols mark the lines of interest – the ones that contain an incorrect spelling. For example, here is a typical output and its interpretation:

```
*
& sec 9 7: Dec, sea, sect, see, set, sew, sex, sic, spec
& zener 1 11: zoner
+ AMPLIFY
& wrapfigure 2 7: wrap figure, wrap-figure
```

1. A correct spelling.
2. `sec` is an incorrect spelling. This is a red herring, but nonetheless, `ispell` looks for possible alternatives. Specifically, it consists of:

the misspelled word, a space, the number of near misses, the number of characters between the beginning of the line and the beginning of the misspelled word, a colon, another space, and a list of the near misses separated by commas and spaces.

Most of this can be ignored. The key item is the misspelled word.

3. A new word: `zener`. This is a legal word, but it's not in the `ispell` dictionary or the user dictionary. `ispell` suggests that we might mean `zoner`, but that's not it. Probably the best approach is to manually add `zener` to the user dictionary.
4. `AMPLIFY` is a legal root word. Ignore this entry.
5. `wrapfigure` is a legal word used by the `wrapfigure.sty` package. Ignore this entry or add it to the user dictionary.

## 8 The Last Word

Of course, a spell checker is no substitute for some basic competence in spelling, as evidenced by this perfectly-spelled poem.

Eye have a spelling chequer  
It came with my pea sea  
It plane lea marques for my revue  
Miss steaks eye kin knot sea.

Eye strike a key and type a word  
And weight for it two say  
Weather eye am wrong or rite  
It tells me rite a weigh.

As soon as a mist ache is maid  
It nose bee fore too long  
And eye can put the air roar rite  
Its rare lea ever wrong.

Eye halve run this poem threw it  
Eye am shore your pleased two no  
Its letter purr fact awl the weigh  
My chequer tolled me sew.

Author unknown, provided by Paul King <pk123@sympatico.ca>.

## References

- [1] *Visual QuickStart Guide: Unix*,  
Deborah S. Ray, Eric J. Ray  
Peachpit Press, 1998
- [2] *sed & awk*  
Dale Dougherty  
O'Reilly & Associates, 1990
- [3] *Advanced UNIX – A rogrammer's Guide*  
Stephen Prata  
Howard Sams, 1985