

総説・解説

原稿中の専門用語・学名等アルファベット文字列のシェルスクリプトによる簡易つづり校正法

高崎浩幸[1]

Shell scripts for spellcheck of texts rich with technical terms and scientific names

Hiroyuki TAKASAKI[1]

Abstract: Modern Japanese texts in print are made of kanji (Chinese characters), katakana, hiragana, and romaji (Roman alphabets) in addition to numerics and symbols, comprising a total of several tens of thousand character codes. As biology is full of scientific names of organisms and technical terms, biological articles in Japanese are disastrous if many alphabetical misspellings remain uncorrected. This situation often gives a headache to the proofreader in Japan. When preparing papers in English too, biologists face the same problem that many of their technical terms are uncheckable with commercially available spellcheckers. The present article is an attempt toward a computer-aided alternative routine making this proofreading process simpler. First, all alphabetical words and tokens (excluding those readily checked with conventional spellcheckers) are extracted without redundancy from the text and sorted alphabetically. In this arrangement, some or all words and tokens placed in neighbors with one another, differing in one or a few letters, are easily identified as misspellings. This makes words and tokens spelled in alphabets be checked easier than by the traditional practice of repeated sequential proofreading of texts. The checklist of unique words and tokens, sorted alphabetically, can be easily prepared using several lines of shell scripts presented here, which are run in terminals of POSIX-Unix compatible systems (e.g. Linux, Mac OS X, and Windows 10). The article first explains the setup of hardware, OS, and softwares required, and introduces an open-source vintage spellchecker. Then, it introduces scripts used for checking alphabetical strings in Japanese texts. Furthermore, it presents scripts for preparing custom spelling dictionaries by text-mining private and open source documents, and custom spellchecker scripts tuned for use in specific disciplines. As the ASCII alphanumerics constitute a subset of UTF-8 Japanese character codes, these scripts are also applicable to English texts without any modification. Note that the same approach is applicable to writings in other languages made of tens of thousand character codes as in Japanese (e.g. Chinese).

目次

I. はじめに

21世紀になって学術論文がさまざまな形態でインターネット上に公開されるのが普通の時代に入った．学術論文は，公開と同時に人類共通の文化遺産として，世界中で共有されるようになったのである．個々の論文全体が世界中から閲覧可能な状態になると，些細なミスタイピングがコピー・アンド・ペーストや直接引用によって拡散し，のちのち多大な無用の混乱が生じかねない．したがって，論文には公開前の入念な校正作業が従来以上に求められる．動物学や植物学など博物学関連，あるいは医学，薬学，農学分野など，生物学系の文書は，学名や特殊な欧文専門用語を多数含み，横文字の苦手な校正者を苦しめる．Microsoft Wordや他のソフトの「スペルチェック＋単語登録」だけでは，対処に苦慮する場面があまりに多い．本稿は，それらアルファベット文字列部分の点検・校正作業の労を少しでも

〒700−0005 岡山市北区理大町1−1 岡山理科大学理学部動物学科 Department of Zoology, Faculty of Science, Okayama University of Science, 1-1 Ridai-cho, Kita-ku, Okayama-shi, Okayama-ken 700-0005, Japan. E-mail: takasaki@zool.ous.ac.jp

軽減する試みとして公開するものである．

日本語の文章では，歴史的な事情で文字種が多岐にわたる．理系の論文を日本語で書く場合，漢字・カタカナ・ひらがな・欧文の４種の文字が混在するのは普通のことだ．とくに日本語に入ってからの歴史が短いアルファベット文字列の校正に関していえば，語学に堪能な校正者であっても，幾度も読み返したところで，目視ではミスタイピングの根絶に成功することは稀である．英文であれば，アルファベット大文字・小文字の52種類や句読点，さまざまな記号類を入れても文字種の識別に，せいぜい７ビット(128種類が識別可能)もあれば十分で，しかも分かち書きされている．そのような言語による論文での校正と，日本語で書かれた論文でのアルファベット文字列校正は，難易度が格段に異なるのだ．本稿で紹介する試みは，この作業を簡易化する．

学術情報が写本や印刷物の流通によってゆるやかにしか拡散しなかった時代には，明らかなミスタイピングは自然に修正された．他方，正誤の判定が容易にはつけがたいミスタイピングは，ゆるやかに拡散し，さらに変異型を加えた．『分子進化のほぼ中立説』(太田　2009)の弱有害突然変異の出現・拡散を彷彿とさせるような現象である．

このような「文字列の誤り」累積の典型例は，命名者たちにもチンプンカンプンなラテン語モドキの生物学名に，とくに顕著に見られる．そのため，各種の『国際〇〇命名規約』(「〇〇」は「動物」や「植物」ほか)には，ミスタイピングに由来する学名の取り扱いについてのルールが設けられているほどである(e.g. ICZN Code, http://www.iczn.org/iczn/index.jsp；2017年１月25日ダウンロード)．いまだに，そのルールによる学名改訂(revision)が絶えない(e.g. Cotton 2016)．すなわち，学名の校正失敗に起因するトラブルは，洋の東西を問わないのである．情報拡散・コピー増殖のスピードが格段に速まった現代にあって，そのようなルールの適用例を，不十分な校正によって増やしてはならない．

現在，日本国では「博士の学位を授与された者は当該博士の学位の授与に係る論文をインターネットの利用により公表するものとする」と法令的に規定されている(「学位規則の一部を改正する省令の施行等について(通知)」24文科高第937号，平成25年３月11日，http://www.mext.go.jp/a_menu/koutou/daigakuin/detail/1331796.htm；2017年１月16日ダウンロード)．すなわち，回避しえたはずのミスタイピングを含む不備のある博士論文の公開は，学位取得者本人だけでなく，学位授与機関にも責任が厳しく問われて当然である．また，世界中の学術雑誌の大半が電子ジャーナル化されて久しい．したがって，学生の論文指導であれ，論文執筆であれ，雑誌編集であれ，これまで以上の注意が論文の校正作業に求められる．

このような時代的・社会的な背景と切迫した必要のもとに，この試みは生まれ，公開するにいたった．「車輪の再発見にすぎない」かもしれない．そうであったとしても，いまだに「車輪」が認知されず，普及していない文化圏やローカルな研究者コミュニティにあっては，まったくの無意味ではないだろう．

最初の導入部となるIIでは，「ハードウェアとソフトウェアの準備」を解説し，IIIではつづり簡易点検スクリプトの実例として既存のスペルチェッカを使用する「英語文書への対処」を導入解説する．続くIVでは，つづり簡易点検スクリプトの発展的な実例として，本稿オリジナルとなる「日本語文書への対処」を解説する．さらに応用と発展をはかるV・VIでは，テキストマイニングによる「専門用語や学名のカスタムつづり辞書」の作成法と，分野ごとに特殊化させた「カスタム・スペルチェッカ」の作成法を解説する．まとめとしてVII「おわりに」で，現代日本において「生物学系の文書処理では，シェルスクリプトの使用が日常的な実用性をもたらす」ことの背景を，歴史的な鳥瞰とともに述べる．

以下，素人の素人による素人のための手引である．いわゆる「Unix」に慣れ親しんでいる読者には，「釈迦に説法」の部分が多々あるだろうが，「初心者にも順を追ってなんとか読めるように噛み砕いて書かれている」と寛容に読み飛ばしていただきたい．なお，「Unix」のターミナルでのコマンド操作に慣れている読者が，本稿にあるスクリプトだけを早急に使いたい場合には，III(1. 12)以降にある1行目左端が[タブ空白字下げ#]で始まるスクリプトと，その近辺前後の左端に[タブ空白字下げ$]がついたコマンド行，さらにその解説を読むだけでよいだろう．あるいは，本文は読まずに，それらのスクリプトを直接解読・試用する．OSに最初からインストールされているユーティリティも多いので，インストールされていないものについてはエラーが出た段階で対処すればよい．

紹介するシェルスクリプト類(**boldface**表記)や応用例(本稿では直接解説してない)は，本稿のテキストpdfと同じURL(http://www1.ous.ac.jp/garden/kenkyuhou3.html)から，zipファイル(foolproof_scripts.zip)としてダウンロードできる．ただし，使い方に関する質問は，本誌編集事務局も執筆者も受けかねる．本稿およびその参考文献を熟読して対処いただきたい．また，スクリプト利用者の行為およびその行為の結果については，本誌も執筆者も責任は負わない．

なお，本稿では，全体を通じてスクリプトを相互参照するため，スクリプトを解説する段落の頭に付

けた番号は，共通の通し番号としてある．

II．ハードウェアとソフトウェアの準備

POSIX準拠のUnix互換ターミナル（極力，Debian GNU/Linux系のLinux OS）が使えるコンピュータを使う．Microsoft Windows 10なら，「Bash on Ubuntu on Windows」を導入すれば使える．また，OS X以降のMacintosh（ターミナルはBSD派生のPOSIX準拠Unix互換）のほか，旧来のUnix機やLinux，Windowsで動くCygwinなども使えるが，Debian GNU/Linux系のLinux OS以外では完全互換ではない．

Desktopに適当なテスト用のプレーンテキストTEST.txt（文字コードはまずは不問として，和英文どちらでも）を用意して，ターミナルに次のコマンドやシェルスクリプト（「コマンドの数珠つなぎ」のようなもの）を打ち込んで[Enter]し，何が起こるか試してみよう．

以下，コマンド入力の後ろに，[Enter]と表記することで，「Enter（リターン）キーを押してコマンドを実行させる」ことを表す．左端の「$ 」は，コマンドプロンプトすなわち「一般ユーザーとして実行可能な」コマンド入力待ち状態を示すターミナルの表示部分なので，打ち込む必要はない．また，本稿でTESTなど大文字表記の部分は，任意のプレーンテキストTEST.txtあるいはそれに由来・関連する名称を意味するので，適宜，置き換えて読めばよい．なお，下記の印刷２−４行目などは，本誌の２段組レイアウトの都合で複数行になっているにすぎず，実際にはタブ字下げで新しい行の入力が示されるところまで，ターミナルで打ち込むときは１行である．

```
  $ cd Desktop [Enter]
  $ LANG=C sed 's/[^a-zA-Z]/ /g'
TEST.txt | tr ' ' \\n | grep -v '^s*$' |
sort | uniq | tee TEST_out.txt [Enter]
```

テキストTEST.txt中のアルファベット文字列がすべて抽出されて，重複なくアルファベット順に並んだ処理結果がターミナル画面に流れ，DesktopにTEST_out.txtとして保存される．「処理結果のリスト中に，１文字あるいは２−３文字違いで前後や近傍に並ぶよく似た文字列は，いずれか，あるいはいずれも誤ったつづりでは？」と校正の焦点をしぼることで，作業を簡略化しようというのである．（このようなスクリプトを自在に書き換えて応用するために，ここに到達する過程をIII–IVで追い，適用性の高いスクリプトの汎用化や，発展的な応用についてIV〜VIで述べる．）

POSIX準拠Unix互換のターミナルが使えるコンピュータを新たに準備するには，すでに減価償却済みになったPC/AT互換機（あるいはIntel系MPU搭載のMac）にフリーOSのLinuxをインストールすることで，安価に簡単に調達できる．本稿の解説との完全互換をはかるには，Debian GNU/Linux系のディストリビューションが望ましい．（Linuxに関する詳しい情報は，図書館や書店に並ぶ各種単行本やムック，雑誌『日経リナックス』等のほか，ウェブ検索にゆずる．やさしい入門書には奈佐原（2016）が入手も容易だろう．）

なお，Windows 10には2016年８月に「Windows Subsystem for Linux」がβ版ながら搭載できるようになって（麻生ら　2017，大橋　2017），2017年４月にはアップデート版の「Bash on Ubuntu on Windows」が出た．「Windows Subsystem for Linux」や「Bash on Ubuntu on Windows」をキーワードに，ウェブ検索でも日本語で情報が得られるほか，2017年５月現在，すでに印刷出版された解説もある（e.g.　東京シェル研究会　2017，高橋・加藤　2017）．2017年秋には，「Fall Creators Update」で一般ユーザーにも公開された（『日経Linux』2017年11月号）．

とにかく，POSIX準拠Unix互換のターミナルを開いて，本稿のスクリプトは使う．Debian GNU/Linuxのコマンドやユーティリティコマンドの組み合わせなので，それらが動くターミナルであれば何でもよい．

本稿の草稿は，Linux Mint（以下，Mintと略記；長期サポートLTS版Rosa，Serena；Ubuntuの同バージョンLTS版を元にしている2017年末段階での最新版はSonya）のターミナルで動作を確認しながら，テキストエディタgeditを使って書いた．また，本稿草稿の脱稿が近くなった2017年７月下旬までに，一部のスクリプトについてはMac OS XやWindows 10「Bash on Ubuntu on Windows」での動作を確認できている．

MacintoshなどBSD系システムでDebian GNU/Linux系のコマンドやユーティリティ類を使う方法については，上田・後藤（2014）のpp. 42–60や大橋（2017）のp. 75「10」が参考になるほか，「Mac command GNU」「Mac GNU coreutils」等でウェブ検索すれば，情報が得られる．

Linux OSのインストールで，日本語入力を含む日本語化が自動的に設定されない場合には，Linuxディストリビューション名に「日本語化」を加えてウェブ検索すれば，情報が得られる．ただし，ターミナル中のディレクトリ表示まで日本語化されてしまうこともある．ターミナルを開いて次のコマンドを実行してみよう．

```
  $ ls [Enter]
```

「デスクトップ」とか「ダウンロード」など，最初

から用意されている標準のディレクトリ名が日本語で表示される場合，このままでは，ディレクトリ(日本語表記)の移動やコマンド(英数表記)の入力のたびに，頻繁に日本語と英数の入力モードを切り替えなければならず，コマンド操作での打鍵に時間を食う．そこで，ディレクトリの名称は英数表示とする．すなわち，ホームディレクトリの下に最初からあるディレクトリの「デスクトップ」等の表示もアルファベット表示に変えてしまう．ファイル名も英数文字を使用する．こうすればターミナルを使うとき，英数入力モードだけでキーボード操作を済ますことができる．MintなどUbuntuベースのディストリビューションの場合，ターミナルで次のコマンドを入力・実行することで，ディレクトリ名が英語表記に変わる(ターミナルの起動方法やコマンドの実行法の詳しい解説については，他の情報源にゆずる)．

```
$ LANG=C xdg-user-dirs-gtk-update [Enter]
```

「Update standard folders to current language」と題したダイアログ窓が表示される．この時に左下に出る「Don't ask me again」の左にある「□」のボックスにマウスのクリックでマークし，右下の「Update Names」ボタンをクリックすれば，このマシンのターミナルではディレクトリ名の表記が英語に置き換わる．なお，ディレクトリ(directory)はフォルダー(folder)と呼ばれることもあるが，ターミナルで使うコマンドでは，旧来のディレクトリという呼称が使われている．

上記のコマンドが使えないディストリビューションでは，mvコマンドを使う．

```
$ mv デスクトップ Desktop [Enter]
```

ターミナルでexitコマンドを入力・実行することで，ターミナル操作を終了して閉じることができる．いつまでたっても「$ 」の後ろに実行中のカーソルの点滅が消えず，暴走が疑われるときは，[Ctrl]キーと[C]キーを同時に押すことで，入力待ち状態のコマンドプロンプト「$ 」に復帰する．あと必要なアプリケーションは，使い勝手のよいテキストエディタだけである．Mintのほか多くのLinuxディストリビューションには，最初からDebian GNU/Linux標準のテキストエディタgeditや他の多くのアプリケーションが一緒に入っている．

III．英語文書への対処：既存スペルチェッカの導入

難易度が低く，使用可能な出来合いのスペルチェッカもある英語文書の場合から始めよう．和文文書への対処を急ぐ場合には，「IV．日本語文書への対処」に進む前に，この「III．英語文書への対処」は読み飛ばすだけで，まずはかまわない．

事前にワープロソフトやテキストエディタのスペルチェッカを使って，学名や特殊専門用語には修正は加えずに，普通の単語のつづりだけの修正を済ませる．修正済み原稿のプレーンテキスト版を作る．(元がMS Wordのファイル，すなわちち拡張子「.doc」や「.docx」の場合，「名前を付けて保存」で拡張子「.txt」を選んで保存すれば作れる．)それらのコピーの中からテスト用の原稿テキストを選んでTEST.txtと名付けて準備する．学名あるいは複雑なつづりの専門用語が多数使われているものなら理想的である．

(1.01)ターミナルを開いてターミナルの枠内にカーソルを移動し，次のコマンドを実行する．

```
$ ls [Enter]
```

lsコマンドの意味は“list”である．ターミナルモードで使うホームディレクトリの中に配置された(サブ)ディレクトリの一覧が出てくるはずである．

(1.02)次にcdコマンドを試す．

```
$ cd Desktop [Enter]
```

Desktopディレクトリに配置された(サブ)ディレクトリやファイルの一覧が表示される．普段はその範囲だけで仕事をしている．cdコマンドの意味は“change directory”である．次のように「../」で親ディレクトリを指示することで，元のディレクトリ(親ディレクトリ)に戻る．

```
$ cd ../ [Enter]
```

(1.03)ターミナルが最初に見せている(カレント)ディレクトリ(current directory)は，Desktopディレクトリではなく，ホームディレクトリである．そこに，練習か今後とも論文の校正に使うためのfoolproofディレクトリを作る．

```
$ mkdir foolproof [Enter]
```

これでfoolproofディレクトリがホームディレクトリにできている．mkdirコマンドの意味は“make directory”である．foolproofディレクトリができていることを確認しよう．

```
$ ls [Enter]
```

次にfoolproofをカレントディレクトリにして，中のファイルを確認しよう．

```
$ cd foolproof [Enter]
$ ls [Enter]
```

いまはカレントディレクトリであるfoolproofディレクトリが空であることが分かる．(1.02)の最後のコマンドと同じ操作で，元の(親)ディレクトリに戻る．

```
$ cd ../ [Enter]
```

(1.04)ここでfoolproofディレクトリの中にTEST.txtを入れる．TEST.txtを入れたusbメモリをLinuxマシンに装着すれば，その中を示す窓が現れるので，マウス操作で目的を達成できる．Desktop上のGUIモードで，usbメモリのディレクトリ窓とfoolproofディレクトリ窓を開いた状態で，コピー・アンド・ペーストあるいはドラッグ・アンド・ドロップするだけだ．コンピュータの中に最初から適当なTEST.txtがある場合は，そのコピー・アンド・ペーストでもかまわない．(ターミナルでのコマンド操作に習熟してくれば，“copy”を意味するcpコマンドや“move”を意味するmvコマンド，さらに(1.09)の最後に実例を示すcatコマンドを使っても，同じ目的を達成できる．)

(1.05)ターミナルにカーソルを戻して，ディレクトリやファイルの一覧を見て，ホームディレクトリにいることを確認し，そこにfoolproofディレクトリもあることを確認する．さらにfoolproofディレクトリの中に移動して，ファイルの存在を確認する．

```
$ ls [Enter]
・・・ [中略]
$ cd foolproof [Enter]
$ ls foolproof [Enter]
```

TEST.txtが入っている．次のcatコマンドを実行すれば，TEST.txtの中身がターミナルの中にザーッと流れる．

```
$ cat TEST.txt [Enter]
```

catコマンドの元の意味はネコではなく“concatenate”に由来するが，とりあえずは「ファイルの中身を連続的に標準出力先となっている画面に出す」コマンドと捉えてよい．

(1.06)「車輪の再利用」には意味がある．少なくとも構成部品は，GUIはまだなかった頃のUnix時代からの遺産であり，ターミナルで使える使い込まれたスペルチェッカspellを試してみよう．

```
$ LANG=C spell TEST.txt [Enter]
```

最初に置かれた「LANG=C」は，「言語の初期設定がシステムによっては邪魔をするのを回避する」ための呪文である．Macで動かないことがあるなどの現象が確認できたので，ASCII文字だけで処理する場合，各システムでのデフォルトのLANG指定の影響を回避する目的のものである．(無くても動くことがあるが，無いと動かないことがある．また，いつも付けてあっても不具合は生じないので，付けておいた方が無難である．)

MintなどDebian GNU/Linux(詳細はウェブ検索で)でコマンド・プログラム類が整備されているディストリビューションでは，spellがまだ使えるようになっていない場合，「プログラム‘spell’はまだインストールされていません．次のように入力することでインストールできます:[改行]sudo apt-get install spell」というメッセージが出る．その指示にしたがう．(他のシステムなどで，別のメッセージが出た場合は，ウェブ検索等で情報収集して対処する．Macの場合は，上田・後藤(2014)に他のコマンド・ユーティリティを例にした解説がある．)

```
$ sudo apt-get install spell [Enter]
```

管理者のパスワードを尋ねられるので指示にしたがう．許可を求められる質問には[Y]あるいは[Enter]を返せば，spellがインストールされて使えるようになる．再度，(1.06)の最初のコマンドを試そう．まったく同じ文字列を続けて再入力するのは煩わしいので，このような場合にはキーボード上のカーソルキー[↑PgUp]を使って打鍵を減らす．

```
$ LANG=C spell TEST.txt [Enter]
```

デフォルトでspellが参照しているつづり辞書に，載っていなかった文字列が，ターミナルの中にザーッと流れる．これでスペルチェッカspellが使えることが分かる．

どうしてもspellが使えるようにならない場合には，解決の日を待つ間，「III．英語文書への対処」はスキップして，「IV．日本語文書への対処」に進んでよい．普通の英単語は事前にワープロやテキストエディタでスペルチェック済みであるし，汎用性の高い「IV．日本語文書への対処」の簡単な文字種の少ないケースとして，英語文書にも対処できるからだ．また「VI．カスタム・スペルチェッカ」の仕組みを充実させることで，代替処理も可能である．

(1.07) さてspellの処理結果の記録が，(1.06)のままではファイルに残らない．なんとか結果をファイルTEST_out.txtに残したい．この場合，「> TEST_out.txt」という指示を付け加える．

```
$ LANG=C spell TEST.txt >
TEST_out.txt [Enter]
```

「> TEST_out.txt」を付け加えたことで，「処理結果を標準出力の画面ではなく，変更してTEST_out.txtに書き出す」という呪文が付け加えられた．(「>」は「リダイレクト」と呼ばれる．)画面には何の変化も見られず，コマンドプロンプト「$ 」が更新されるだけである．TEST_out.txtができているか確認する．

```
$ ls [Enter]
```

少なくともTEST_out.txtというファイルができていることが分かる．その中身を確認する．

```
$ cat TEST_out.txt [Enter]
```

ファイルの中身である「spellが参照しているつづり辞書に載っていなかった文字列」が，(1.06)の最後と同じように，ターミナルの中にザーッと流れる．TEST_out.txtは，テキストエディタで開けば分かるように，読み書きできる状態である．

(1.08) 出力ファイルの中身を，いちいちcatコマンドで確認するのは面倒だ．そこでパイプ「|」という小道具とteeコマンドを組み合わせて加える．「|」は「直前の処理結果を直後のコマンドの入力として，あたかも情報の流れをパイプで繋ぐように流して，受け渡す」指示となる．teeコマンドは「|」と組み合わせて「パイプのT字分岐継ぎ手」のような働きをするコマンドで，分岐の片方が画面出力になる．

```
$ LANG=C spell TEST.txt | tee
TEST_out.txt [Enter]
```

これでTEST.txtのspellによる処理結果が画面に出力されると同時に，TEST_out.txtに記録される．

(1.09) このままでは「疑わしい」と判定された同じつづりがTEST.txtに出現する回数だけ現れるので煩わしい．sortコマンドとuniqコマンドを途中に加えてパイプで繋ぐ．sortコマンドは分節された文字列(改行で区切られている)をアルファベット順に並べ直し，uniqコマンドは直前直後に連続する入力に重複があれば１つだけ残して重複を取り除く．

```
$ LANG=C spell TEST.txt | sort | uniq
| tee TEST_out.txt [Enter]
```

こうすれば出力のTEST_out.txtファイルには，spellが参照したつづり辞書に載っていない文字列が，重複なく全てリストアップされる．あとはリストを目視で総点検すればよい．簡単には画面上で，長大な場合にはプリントアウトを作って点検する．１文字あるいは２−３文字違いで前後や近傍に並ぶよく似た文字列は，どれか，あるいはいずれも誤ったつづりである．

テキストエディタなどで，TEST.txtにある誤ったつづりを正しいつづりにシラミ潰し的に一括変換する．直上のコマンド実行から直前の文までの処理を，結果に変化が見られなくなるまで十分に繰り返す．(ターミナルとテキストエディタを開いたまま，テキストエディタで修正するたびに保存し，コマンドの再実行にはターミナルでカーソルキー[↑PgUp]，続けて[Enter]を押せばよい．)変化がなくなったことは，テキストエディタを行番号付きに設定して「最終行番号が同じである」と確認することで判断できる．

このようにして，TEST.txtに混じっていた，つづり辞書には登録されていない学名や専門用語などの校正作業が，何回もの読み返しによる校正よりもはるかに簡易化される．

入力に使ったTEST.txtに入っていて，spellの参照辞書に載っていない学名や専門用語が，いずれ再度使われる可能性が高いなら，直上の最終出力となった正しいつづりのみで構成されたTEST_out.txtを，okspell.txtというファイルに書き出しておく．

```
$ cat TEST_out.txt > okspell.txt
[Enter]
```

(1.10)spellが参照するつづり辞書を，初期設定で使われているものよりも，登録単語数の多いものにすることもできる．コマンドの使用法を簡略に調べるmanコマンド(ヒトではなく“manual”に由来)を使って，spellの使い方を確認しよう．

```
$ man spell [Enter]
```

スペースバーを押してページをめくるか，カーソルキー[↓PgDn][↑PgUp]で流し読みすると，さまざまなオプションがあることがわかる．-Dオプションで参照する出来合いの外部つづり辞書を指定できるのだ．たとえばアメリカ英語の巨大つづり辞書パッケージiamerican-hugeがインストールしてあるなら，オプション指定を「-Damerican-huge」とすれ

ばよい．必要ならibritish-hugeというイギリス英語の巨大つづり辞書パッケージもあるので，オプション指定を「-Dbritish-huge」とすることで使い分けも可能である．

つづり辞書パッケージのインストールは，パッケージマネージャーを使うか，「ispell つづり辞書 パッケージ インストール」，「ispell dictionary package install」等でウェブ検索して情報を集める．分からなければ，分かる日が来るまで待つことにして，(1.10)はスキップしてよい．(自前のつづり辞書さえ準備できれば「VI．カスタム・スペルチェッカ」の仕組みを充実させるか，(1.11)の仕掛けでの代替処理も可能である．)

キー[q]を押してmanコマンドを終了し，コマンドプロンプトに戻って，試してみる．

```
  $ LANG=C spell -Damerican-huge
TEST.txt | sort | uniq | tee
TEST_out.txt [Enter]
```

これを(1.09)に置き換えて得られたokspell.txtは(1.09)のokspell.txtよりもコンパクトで利用価値が高いことがある．

(1.11)okspell.txtは，spellで-dオプションを指定することで，再利用可能である．オプション指定を「-dokspell.txt」とすればよい．(-Dオプションが使えない場合など，okspell.txtを「V．専門用語や学名のカスタムつづり辞書の作成」と合わせて整備することで，代替処理が可能になる．)

```
  $ LANG=C spell -dokspell.txt
TEST_out.txt [Enter]
```

こうすればspellが参照する個人つづり辞書はokspell.txtに指定され，処理する入力はTEST_out.txtとなる．ここまではokspell.txt中の登録「つづり」だけしか，最後に出力されたTEST_out.txtの中には残っていないので，画面ではほとんど変化が見られず，入力待ちのコマンドプロンプト「$ 」に更新されるだけである．ちゃんと-dオプションがはたらいているか，(1.08)のコマンドの途中にパイプとteeを組み合わせて挿入してテストする．

```
  $ LANG=C spell TEST.txt | tee | spell
-dokspell.txt | tee TEST_out.txt [Enter]
```

前半のteeで(1.08)と同じ結果が画面に流れ，その中の文字列はすべて参照つづり辞書に指定したokspell.txtにすでに登録してあったので，後半のteeでは出力が空となってコマンドプロンプト「$ 」が更新される．テストがうまくいけば，前半のteeはなくても最終出力には影響しないから，省く．

```
  $ LANG=C spell TEST.txt | spell
-dokspell.txt | tee TEST_out.txt [Enter]
```

この形で，これから後，別のTEST.txtの点検に使える．

(1.12)よく使うのに，毎回毎回，こんな長い呪文を書くのは煩わしい．次に使う頃までには忘れてしまう．そこで(1.11)の最終パターンを，今後くり返し使える汎用スクリプトとしてファイルに残そう．次の４行を(面倒なら最初と最後の１行ずつだけでも)エディタで書いて，foolproofディレクトリに**myspellcheck.sh**として保存する．

```
  #!/bin/bash
  ##myspellcheck.sh:
  ##custom spellchecker
  LANG=C spell -dokspell.txt $1.txt |
sort | uniq | spell | tee $1_out.txt
```

コマンド・スクリプトの本体である最終行は，(1.11)の最終スクリプトにあるTEST.txtを$1.txtに置換したものにすぎない．(すなわち，その行だけでも動く．)１行目は，なくてもおそらく動くが，書くのが慣例のシェバン行と呼ばれ，「これはbashというコマンドshellで実行する」と宣言する呪文である．(すなわち，ターミナルで通常使うシェルは，本稿ではbashを想定している．MintほかDebian GNU/Linuxシステムの多くでは，デフォルトでそうなっている．)２，３行目は，このスクリプトのタイトルと，処理内容の要約コメントである．(なくても動くが，このスクリプトを使うことになった他人や，あるいは作った本人でさえ，のちのち「何をするスクリプトか分からない」という事態を，防ぐ目的で付ける．)

TEST.txtでテストしてみよう．拡張子「.txt」は入力の記入から省略する仕様になっている．

```
  $ bash myspellcheck.sh TEST [Enter]
```

あるいは後の処理にiamerican-huge辞書を使う(1.10)を用いる**myspellcheck2.sh**なら，次のように書けばよい．オプション-Damerican-hugeを付けると，直前のパイプからの出力を入力として受け取ることがうまくいかなくなるので，リダイレクト「>」を使って一時的な中間ファイルとしてtmpfile.txtに一度出力し，それを次の入力としている．目的とす

るすべての処理が終わった最後に，不要となった一時的なファイルは，rmコマンド(“remove”)で除去する．tmp*.*は，tmpで始まり拡張子が任意のファイルに相当し，最後までつづると長いtmpfile.txtも含まれる．

```
  #!/bin/bash
  ##myspellcheck2.sh
  ##custom spellchecker for English
texts using iamerican-huge
  LANG=C spell -dokspell.txt $1.txt |
sort | uniq > tmpfile.txt
  spell -Damerican-huge tmpfile.txt |
tee $1_out.txt
  rm tmp*.*
```

TEST.txtでテストすれば結果は(1.11)と同様に，TEST_out.txtにファイルとして残される．

「bash」と打つのが面倒な場合には，chmodコマンドを使う．(+xオプションでシェルスクリプトを実行可能ファイルにする．詳しい解説は他にゆずる．)

```
  $ chmod +x myspellcheck.sh [Enter]
```

これでbash部分は簡略化できる．再確認する．

```
  $ ./myspellcheck.sh TEST [Enter]
```

「./myspellcheck.sh」は，「カレントディレクトリ中の実行可能設定済みの**myspellcheck.sh**を実行する」という意味で「./」を付けなければ動かない．「chmod +x」で「bash 」部分(スペースも勘定して５キーストローク)を「./」(２キーストローク)と略記できるようになったとも言える．３キーストロークの打鍵節約になる．**myspellcheck2.sh**も同様にchmodコマンド処置しておく(以下の汎用スクリプトでも同様に)．

(1.13)英語論文原稿について，(1.09)のように(1.12)の**myspellcheck.sh**を使った校正を完了するたびに，okspell.txtに正しいつづりだけが残ったTEST_out.txtを追加して更新しておく．(リダイレクト「>」は上書きするのに対し，ダブルの「>>」は追記する．)

```
  $ cat TEST_out.txt >> okspell.txt
[Enter]
```

そうはいっても何回か校正完了時の更新をおこたったり，あるいは何らかの理由でokspell.txtに重複して同じつづりが登録されるようなことが起こる可能性も残る．それを回避するように，いつも次の３行のスクリプトを連続して用いるとよい．(１行目では，使うはずでなかった校正前のTEST_out.txtを，早まってokspell.txtに追記した場合に復旧可能なように，古いokspell.txtのバックアップokspell_bak.txtを作る．)

```
  $ cat okspell.txt > okspell_bak.txt
[Enter]
  $ cat TEST_out.txt >> okspell.txt
[Enter]
  $ sort okspell.txt | uniq | tee
okspell.txt  [Enter]
```

あるいは３つをまとめたokspell.txtを更新するスクリプト**updateokspell.sh**を，次のように書いてファイルに残す．

```
  #!/bin/bash
  ##updateokspell.sh
  ##update okspell.txt with new
TEST_out.txt for myspellcheck.sh
  cat okspell.txt > okspell_bak.txt
  cat TEST_out.txt >> okspell.txt
  sort okspell.txt | uniq | tee
okspell.txt
```

これは，次のように実行するだけでよい．ただし，実行の前にTEST_out.txtは正しいつづりだけで構成されていることを再確認する．(なお，バックアップが残してあるので，復旧可能ではある．)

```
  $ bash updateokspell.sh [Enter]
```

(1.14)ここに紹介したレトロなスペルチェッカspellは，1980年代にその進化版がWordStarなど８ビットPC用のワープロに搭載された．現在の32/64ビットPC用のGUIワープロには，誤つづりに対して正つづりを自動で示してくれる一見「はるかに便利な」スペルチェッカが搭載されているのが普通である．

しかし，英語で書かれた論文であっても，出来合いのつづり辞書に載っていない学名や専門用語が多数混じる場合には，ワープロやテキストエディタに付属するスペルチェッカでは，お手上げとなる．自分で準備すべき特殊つづりのリストが，最初は準備できない場合がほとんどなので，ただただ煩わしいばかりだ．

たとえばN種の学名が含まれる英語論文の場合，属名や種小名にそれぞれ重複がなければ，まず本来の英単語(つづり辞書に載っている専門英単語を含む)のミスタイピングが「疑わしいつづり」として指摘される．それに加えて，つづり辞書に載っていな

い専門用語と，２N種類の正しい学名のつづりが「疑わしいつづり」としてスペルチェッカに指摘される．さらにそれらのミスタイピングも「疑わしいつづり」に加わって，「疑わしい」と幾度も指摘される．すなわち，スペルチェッカを上手に使えても，この上なく煩わしい．

Nが大きい上にタイピングが下手な場合など，眼精疲労からか，あるいはついうっかりとワープロのスペルチェッカの学習辞書に誤ったつづりを登録してしまい，それに気づいて学習辞書を空っぽにリセットし，また振り出しに戻る．この繰り返しに陥りかねない．つづりの点検だけで途方にくれ，本文の推敲段階にまでなかなか到達しないうちに，月日が経過する．

旧「Unix」の時代には完成していたスペルチェッカspellは，このような事態を解消する上で，まだまだ使い道があるだろう．これは，IVで紹介する「日本語文書中のアルファベット文字列への対処法」と組み合わせることで，さらに威力を発揮する．

IV．日本語文書への対処：アルファベット文字列の抽出処理

「III．英語文書への対処」と同じく，事前にワープロソフトやテキストエディタに付属している英文スペルチェッカを使って，学名や特殊専門用語には修正を加えずに，一般英単語つづりの修正だけを済ませておく．修正済み原稿のプレーンテキスト版を作る．テスト用の原稿テキストをTEST.txtと名付けて準備する．学名あるいは複雑なつづりの欧文専門用語が多数使われている文書ならテストには理想的である．

(2.01)入力文書テキストの文字コードは，POSIX準拠Unix互換ターミナルでの標準となっているUTF-8の日本語コードにそろえる方が望ましい．ただし，英文アルファベット文字列の部分だけを点検するのであれば，とりあえず文字コードの変換作業は略してもかまわない．理由は，英数字部分は現行すべての文字コードのセットが，ASCII英数字コード由来のコードをサブセットとして取り込んでいるからだ．改行コードは，単語に切り分けるときに処理されるので，これも考慮しなくてよい．

プレーンテキストになっているTEST.txtは，欧文であれ，和文であれ，次のスクリプトで処理すれば，ターミナルの画面にアルファベット文字だけがザーッと流れる．

```
  $ LANG=C sed 's/[^a-zA-Z]/ /g'
TEST.txt [Enter]
```

「sed 's/[^a-zA-Z]/ /g'」は「a-z，A-Z以外の文字をスペースに全て置換せよ」というスクリプトである．[^a-zA-Z]は正規表現の一種である．詳しくは，「sed 使い方」でウェブ検索すれば解説が見つかる．なお，sedは“stream editor”の略に由来するスクリプト言語の一種である．正規表現とともにスクリプトに習熟するまでは，深く考えずにただ真似ればよいだろう．

(2.02)このままでは画面に流れてしまって処理結果が残らないので，結果を一時的なファイルに残してみよう．(1.06)-(1.07)でほどこした細工と同じパターンである．

```
  $ LANG=C sed 's/[^a-zA-Z]/ /g'
TEST.txt > alph_out.txt1 [Enter]
```

次のコマンドで中身の確認をする．

```
  $ cat alph_out.txt1 [Enter]
```

先ほど画面で流れたものと同じものが画面に流れる．alph_out.txt1は，適当なテキストエディタで開いて読み書きすることもできる．

(2.03)このまま，つづりをチェックしていたのでは「日暮れて道遠し」なので，次のコマンドで単語を改行で切り分ける．

```
  $ cat alph_out.txt1 | tr ' ' \\n
[Enter]
```

スペースが改行に変換されて，単語が改行を挟みながら切り分けられ，画面に流れる．前半の出力をパイプ「|」がつないで受け渡した先は，「tr ' ' \\n」である．「スペースを改行\nに全置換せよ」というスクリプトである．(ここでsedを使わないのは，sedは改行の置換が苦手なことと，１文字相当の置換の場合には「tr」の方が表記が手軽だからである．)この結果を，alph_out.txt2に残し，中身を確認する．

```
  $ cat alph_out.txt1 | tr ' ' \\n >
alph_out.txt2 [Enter]
  $ cat alph_out.txt2 [Enter]
```

先ほど画面で流れたものと同じ結果が画面に流れる．これも，適当なテキストエディタで開いて読み書きすることができる．

パイプ「|」とteeコマンドを使うことで，(2.01)-(2.03)は，一気に済ますことができる．

```
  $ LANG=C sed 's/[^a-zA-Z]/ /g'
TEST.txt | tr ' ' \\n | tee
alph_out.txt2 [Enter]
```

(2.04)単語風のアルファベット文字列だけになっているalph_out.txt2中のつづり点検の準備として，アルファベット順に並べ直す．

```
  $ sort alph_out.txt2 >
alph_out.txt3 [Enter]
```

パイプ「|」とteeコマンドを使う次のスクリプトで，処理結果を画面出力し，さらに結果をalph_out.txt3に残すことができる．

```
  $ sort alph_out.txt2 | tee
alph_out.txt3 [Enter]
```

(2.05)アルファベット文字列が少なければ，alph_out.txt3を目視でじっくりと点検するだけでもなんとかなるが，多くなると根気が続かなくなるので，重複する文字列(空行を含む)を除去する．

```
  $ uniq alph_out.txt3 [Enter]
```

次のスクリプトで，処理結果を画面に出力させ，それをalph_out.txt4に残すことができる．

```
  $ uniq alph_out.txt3 | tee
alph_out.txt4 [Enter]
```

(2.06)(2.01)-(2.05)をパイプで連結して，次のスクリプトを得る．

```
  $ LANG=C sed 's/[^a-zA-Z]/ /g'
TEST.txt | tr ' ' \\n | sort | uniq |
tee alph_out.txt4 [Enter]
```

画面出力も得て，結果がalph_out.txt4に残る．途中のalph_out.txt1，alph_out.txt2とalph_out.txt3は，もう姿を消しているので，alph_out.txt4の最後についた添え字4は不要となる．

```
  $ LANG=C sed 's/[^a-zA-Z]/ /g'
TEST.txt | tr ' ' \\n | sort | uniq |
tee alph_out.txt [Enter]
```

結果はalph_out.txtに残る．alph_out.txtをざっと眺めると，先頭に改行で挟まれた空行が1行入っている．流れる画面で見るときには気にもならないが，静止状態やプリントアウトでは違和感を覚える．そこで，空行を削除するコマンド「grep -v '^s*$'」を「sort」の直前に「|」を付けて挿入する．

```
  $ LANG=C sed 's/[^a-zA-Z]/ /g'
  TEST.txt | tr ' ' \n | grep -v '^s*$'
| sort | uniq | tee alph_out.txt [Enter]
```

これで，IIの最初に出たスクリプトと同じものが得られる．

(2.07)(1.12)や(1.13)と同様に，コマンドとして実行できるファイルにして残そう．次をエディタで書いて，**alphsort.sh**という名前を付けてfoolproofディレクトリに保存しておく．万が一の文字コード由来による不具合を回避するために，処理の最初にUTF-8の日本語文字コードへの変換を追加してある．それには，nkf(https://ja.osdn.net/projects/nkf/; Mintでは，(1.06)のspell導入と同じ手順でダウンロード・インストール)を使う．最初からUTF-8の文字コードが使われている場合にも，とりたてて悪影響はない．

```
  #!/bin/bash
  ##alphsort.sh
  ##alphabet string checker for
multilingual texts
  LANG=C nkf -wLu $1.txt | sed 's/
[^a-zA-Z]/ /g' | tr ' ' \\n | grep -v
'^s*$' | sort | uniq | tee
$1_alph_out.txt
```

bashと組み合わせて実行する方法は，(1.12)，(1.13)と同様である．入力のTEST.txtの拡張子「.txt」は省略する仕様になっている．

```
  $ bash alphsort.sh TEST [Enter]
```

なお[]内の正規表現部分に「.」と「-」を付け加えて次のように変えれば，「a-z，A-Z，.，-以外の文字をスペースに全て置換せよ」というスクリプトになる．(正規表現内で特別な意味をもつ予約文字「.」や「-」を，それらの文字自身の意味で使う場合には「\.」や「\-」と表記する．)学名など命名者名の略に付いているピリオドやハイフンとの組み合わせ文字列まで点検する場合には，こちらを使うとよい．

```
  #!/bin/bash
  ##alphsort2.sh
  ##alphabet (including dot and hyphen)
string checker for multilingual texts
  LANG=C nkf -wLu $1.txt | sed
's/[^a-zA-Z\.\-]/ /g' | tr ' ' \\n | grep
-v '^s*$' | sort | uniq | tee
$1_alph2_out.txt
```

これもカレントディレクトリでは「./**alphsort2.sh**」として使えるようにしておく．

（2．08）もし，（1．06）で問題なくspellが使え，(1.10)のアメリカ英語の巨大つづり辞書iamerican-hugeも使える状態の場合，(2.07)の最終出力TEST_alph_out.txtを(1.12)の最後のスクリプト・コマンドの入力とすれば，出力TEST_alph_out_out.txtを得ることができるだろう．

これで，少なくとも相当な語彙の英単語の正しいつづりは，もうTEST_alph_out_out.txtには残っていない．原稿の中からつづり辞書的には「疑わしい」と抽出されたアルファベット文字列のリストがTEST_alph_out_out.txtである．これを，画面あるいはプリントアウトで落ち着いて目視で点検すればよい．

出力TEST_alph_out_out.txtの目視点検の要点は，「１文字あるいは２−３文字違いで前後や近傍に並ぶよく似た文字列は，どれか，あるいはいずれとも誤ったつづりであることが多い」という事実にある．それを最大の手がかりにして校正を進める．最後に残った文字列は，慣れ親しんだ分野の専門用語や学名であるはずだから，同分野の専門家であるならシラミつぶしの目視で十分に対処できるだろう．

入力のTEST.txtは，最初にワープロあるいはテキストエディタに付属のスペルチェッカで「一般」英単語については修正済みである．すなわち，たとえspellが使えなくても，それほど大きなハンディキャップではない．したがって，TEST_alph_out.txtだけの点検でも相当に意味がある．

(2.09)ここで，TEST.txtの構成文字列を，それらの出現頻度の情報も考慮しながらスペルチェックすることも考えてみよう．１回しか現れないアルファベット文字列は，それによく似ていながらも複数回現れる文字列よりも，誤ったつづりである可能性が高いだろう．まず，１回しか現れない文字列と，複数回現れる文字列に分けて，それぞれを巨大つづり辞書のスペルチェッカで処理して，それぞれの疑わしい文字列を得る．さらに，両グループの文字列が末尾で区別できるように，片方のグループの文字列に識別子を付けた上で，和集合を作り，ソートする．こうすれば，点検をさらに簡易にできるだろう．「１回しか現れないつづりは，リスト中で識別されて，すべて点検するように指示されている」と見ることで，「つづり辞書にはなくても，その正しいつづり候補が複数回現れた文字列として，隣接して並んで見つかる」可能性があるからだ．

まず和文のTEST.txtから出発しても支障がないように，次のように本稿の最初に出たスクリプトの前半部を流用する．これで，TEST.txtが和文であっても欧文であっても，アルファベットだけからなる文字列が，改行で切り分けられたテキストtmpTEST.txtの中に並ぶ．

```
  $ LANG=C sed 's/[^a-zA-Z]/ /g'
TEST.txt | tr ' ' \\n | tee
tmpTEST.txt [Enter]
```

テキスト中の文字列を出現頻度統計と一緒に頻度順にリストアップするスクリプトは，どこかに誰かが作っていそうだ．なるべく「車輪の再発明」は避けたい．テキスト処理のスクリプト言語AWKの開発者らによる教科書(Aho et al. 1989)に類例が見つかった(足立[訳 2004]p. 164)．それを写して修正を加えたものが次のスクリプトである．（スクリプト言語AWKについての解説は，Aho et al.(1989) や中島ほか(2015) などの教科書・参考書類にゆずる．）なお，このAWKによるスクリプトはここだけで使うのであれば，もっと簡略化できるが，のちにVの(3.04)でも使うので，汎用性を保つためにこのようにしてある．

```
  # wordfreq.awk: print count of each
word
  #   input: text
  #   output: count-word pairs sorted
by count
      {gsub(/[.,:;!(){}*_0-9\-\\]/,
" ") #treat punctuation marks and numbers
      for (i = 1; i <= NF; i++)
      count[$i]++
      }
  END {for (w in count)
      print count[w], w | "sort -rn"
      }
```

これを**wordfreq.awk**と名前を付けたスクリプトファイルとして，foolproofディレクトリ内に保存する．これでtmpTEST.txtを処理するには，次のように実行する．

```
  $ gawk -f wordfreq.awk tmpTEST.txt |
tee wordfreq_out.txt [Enter]
```

出力wordfreq_out.txtの中には，出現頻度順に頻度と単語が左右の対になったリストが得られる．（なお，「gawk」の代わりに「awk」と書いても，エイリアス扱いとなって，そのままで動くシステムもあるが，本稿で動作を確認しているDebian GNU/Linuxのawkコマンドであることを明示するために「gawk」としておく．BSD系などへのDebian GNU/Linuxのgawkの導入ほかの注意については，IIにあげた情報源で見つかる．）

(2.10)つづいて，wordfreq_out.txt中に出現頻度１回とリストされたものについて，各行の右側にある単語だけを出力しなければならない．tmpwords1.txtに書き出すことにする．これには次の１行awkスクリプトを使う．

```
  $ gawk '$1 == 1 {print $2}' <
wordfreq_out.txt | tee tmpwords1.txt
[Enter]
```

次のスクリプトで，巨大つづり辞書を併用したスペルチェックの結果が，tmpwords1_out.txtに出力される．

```
  $ bash myspellcheck2.sh tmpwords1
[Enter]
```

同様に，出現頻度が２回以上のものについては，次の２行のスクリプトで，tmpwords2_out.txtにスペルチェックの結果が出力される．

```
  $ gawk '$1 >= 2 {print $2}' <
wordfreq_out.txt | tee tmpwords2.txt
[Enter]
  $ bash myspellcheck2.sh tmpwords2
[Enter]
```

(2.11)巨大つづり辞書で点検しても「疑わしく」しかも「１回しか現れない」から「かなり疑わしい」文字列の集合テキストtmpwords1_out.txtに，注意喚起の「#」を末尾に付けてtmpQwords.txtに出力するには，次の１行awkスクリプトのコマンドを使う．

```
  $ gawk '{print $0 "#"}' <
tmpwords1_out.txt | tee tmpQwords.txt
[Enter]
```

これと複数回現れる単に「やや疑わしい」ので末尾には何も付いていない文字列リストtmpwords2_out.txtを合わせた和集合は，次のコマンドでtmpwords2_out.txtをtmpQwords.txtに追記することで得られる．

```
  $ cat tmpwords2_out.txt >>
tmpQwords.txt [Enter]
```

さらに，よく似たつづりが前後に並びやすいようにソートすることで，目的のTEST_out.txtが得られる．

```
  $ sort tmpQwords.txt | tee
TEST_out.txt [Enter]
```

(2.12)(2.09)-(2.11)のコマンドラインを連続させて整理しよう．

```
  $ LANG=C sed 's/[^a-zA-Z]/ /g'
TEST.txt | tr ' ' \\n | tee tmpTEST.txt
[Enter]
  $ gawk -f wordfreq.awk tmpTEST.txt |
tee wordfreq_out.txt [Enter]
  $ gawk '$1 == 1 {print $2}' <
wordfreq_out.txt | tee tmpwords1.txt
[Enter]
  $ bash myspellcheck2.sh tmpwords1
[Enter]
  $ gawk '$1 >= 2 {print $2}' <
wordfreq_out.txt | tee tmpwords2.txt
[Enter]
  $ bash myspellcheck2.sh tmpwords2
[Enter]
  $ gawk '{print $0 "#"}' <
tmpwords1_out.txt | tee tmpQwords.txt
[Enter]

  $ cat tmpwords2.txt >> tmpQwords.txt
[Enter]
  $ sort tmpQwords.txt | tee
TEST_out.txt [Enter]
```

ちゃんと動いていることを出力TEST_out.txtで確認できたら，rm tmp*.*で不要になった一時的な中間ファイルを消す．今後も任意のテキストファイルに対して使える汎用スクリプト**myspellcheck3.sh**として残そう．念のために開始行に文字コード由来の不具合を回避するnkfコマンドも入れて，**myspellcheck2.sh**に相当する部分も書き直す．１回しか出ない単語と複数回出る単語の要つづり点検単語リストは，場合によっては参照したくなることもあるだろうから，それぞれ残すように($1_words1_out.txt, $1_words2_out.txt)，細工しておく．wordfreq_out.txtは，残してもよいし，消してもよい(最終行の左端「#」をとる)．

```
  #!/bin/bash
  ##myspellcheck3.sh
  ##superb custom spellchecker
  ##to run, wordfreq.awk is needed
  LANG=C nkf -wLu $1.txt | sed
's/[^a-zA-Z\.\-]/ /g' | tr ' ' \\n | tee
tmpTEST.txt
  gawk -f wordfreq.awk tmpTEST.txt | tee
wordfreq_out.txt
  gawk '$1 == 1 {print $2}' <
wordfreq_out.txt | tee tmpwords1.txt
  LANG=C spell -dokspell.txt
tmpwords1.txt | sort | uniq > tmpfile1.
txt
  spell -Damerican-huge tmpfile1.txt |
tee $1_words1_out.txt
  gawk '$1 >= 2 {print $2}' <
wordfreq_out.txt | tee tmpwords2.txt
  LANG=C spell -dokspell.txt
tmpwords2.txt | sort | uniq > tmpfile2.
txt
  spell -Damerican-huge tmpfile2.txt |
tee $1_words2_out.txt
  gawk '{print $0 "#"}' <
$1_words1_out.txt | tee tmpQwords.txt
  cat $1_words2_out.txt >> tmpQwords.txt
  sort tmpQwords.txt | tee $1_out.txt
  rm tmp*.*
  #rm wordfreq_out.txt
```

(2. 13)プレーンテキストは，元がMS Wordのファイルで拡張子が「.doc」や「.docx」等の場合，「名前を付けて保存」で拡張子「.txt」を選んで保存すれば作れる．しかし，スペルチェッカの辞書に載っていない学名や専門用語を含む，少なくとも今回のような校正作業をほどこす草稿は，テキストエディタで書いた方が，プレーンテキストへの変換の手間が省ける．

そうはいっても，研究者の世界でも電子テキスト原稿はワープロMS Wordで作成された文書で溢れている．拡張子「.doc」のついた文書については，ターミナルで使えるプレーンテキストへの変換ツールがある．このツールがantiwordだ(http://www.winfield.demon.nl/)．ダウンロード・インストールして(Mintでは，(1. 06)のspell導入と同じ手順でダウンロード・インストール)，入力がTEST.docであるとすれば，次のコマンドで標準出力先であるターミナル画面に出力が出る．

```
  $ antiword TEST.doc [Enter]
```

リダイレクト「>」を使って出力をTEST.txtに保存するか，あるいは次のようにパイプ「|」とteeを使って出力を画面で確認すると同時にTEST.txtに保存する．

```
  $ antiword TEST.doc | tee TEST.txt
[Enter]
```

日本語部分は，元文書でUTF-8の文字コードを使っていない場合には文字化けするが，半角英数文字は文字化けしないので，半角英文字つづりを点検するためだけに使うのであれば，TEST.txtはこのまま使える．UTF-8の日本語文字コードへの変換を追加するには，(2. 07)で紹介したnkfも同時に使う．

```
  $ antiword TEST.doc | nkf -wLu | tee
TEST.txt [Enter]
```

最初からTEST.docにUTF-8の文字コードが使われている場合には，このスクリプトで出力には半角日本語文字が全角に変換される以外には，とりたてて影響もないので，ここまでの例にならって「TEST」を「$1」に換えて，これをいつも使えるように**doc2txt.sh**などとしておくとよいだろう．

(2. 14) 拡張子「.docx」のファイルをプレーンテキストに変換するには，次のスクリプトを使う．

```
  $ unzip -p TEST.docx
word/document.xml | sed -e
's/<\/w:p>/\n/g; s/<[^>]\{1,\}>//g; s/
[^[:print:]\n]{1,\}//g'| nkf -wLu | tee
TEST.txt
[Enter]
```

これも**docx2txt.sh**などに変換しておくとよい．また，「.pdf」文書のテキスト部分を「.txt」に変換するpdftotextというユーティリティなども，ウェブ検索で見つかる．それらの利用は読者の必要と工夫にまかせる．

(2. 15)Linuxの手引書類(e.g. Stutz 2001)には，テキスト処理の工夫例が数々ある．また，数値計算よりも文字列操作に長けたプログラミング言語AWKの誕生は，1977年であった(Aho et al. 1989)．文献による詳細な裏付けはしないが，これらのことから，「同様のsortやuniqを使った工夫は，米欧では言語表記法の性質上，おそらくは32ビット標準のUnixマシンが普及し始めた1970年代後半，遅くとも８ビットのパソコンが普及した1980年代前半から，長年，より簡略な形で行われてきた」と推察される．

改めて紹介するほどのことでもない当然の手技として，いまも研究者の間でひっそりと伝承されてい

るかもしれない．この推察が正しいとすれば，ここに紹介した手法は，異なる言語文化圏での「車輪の再発明」のようなものかもしれない．すなわち同時に，欧文で書かれた論文は当然のこと，また日本語に限らず，８ビットでは文字コードが足りない他言語での学術論文にも，このアルゴリズムは応用可能だ．ASCII英文アルファベットは，UTF-8の日本語文字コードにサブセットとして含まれるので，英語の文章にはそのまま適用可能である．

現在では，パソコンをGUIで使うのが普通で，わざわざターミナルを開いて，CUIでコマンド操作をする人は圧倒的な少数派だろう．しかし，普段はGUIを使っていても，出来合いのソフトでは手に負えない細かい作業が必要な場合には，CUIでスクリプトが便利に使えるとありがたい．現代の木工職人や大工が，普段は出来合いのパーツを組み合わせるだけで仕事を済ませていても，それでは足りないところは，仕事に合わせて電動工具を使い，ジグを工夫し，さらに手引き鋸，手鉋，手のみを使い分けるようなものである．

８ビットの時代からパソコンを使い始めた世代は当時のCP/Mや，16ビット時代のMS-DOSなどコマンドモードが主流だった時代のことを覚えているから，スクリプトの利用にそれほど違和感はないかもしれない．ただ，当時は使えるツールが少なかった．今は，32ビット旧「Unix」の膨大なソフト資産の多くが，POSIX準拠Unix互換ターミナルが動きさえすれば，無償でパソコンで使える．それが，Mac OS XやWindows 10でも，POSIX準拠Unix互換ターミナルが動くようになってきた背景であろう．

(2. 16)IIIでは旧「Unix」の時代に完成したレトロなスペルチェッカspellを紹介した．それとの併用で，学名や専門用語が原稿中に多数ちらばる生物学分野の和文原稿の準備校正では，今回紹介した**alphsort.sh**や**alphsort2.sh**，**myspellcheck3.sh**が役に立つ．アルファベット文字列の点検が終わった段階で，さらに日本語文字列にも類似の点検ができると便利であろう．その実験的なスクリプト類(**jpnsort0.sh**，**rmshort.awk**)も本稿執筆者は書いて，本稿ほかの草稿校正に使っている．**alphsort.sh**の構文を応用すれば，作ることができる．

具体的には，「sed」で文章中から「拾う」あるいは「拾わない」文字を指定する「正規表現」と変換のパターンを，日本語の文字列を校正が行ないやすいように，文章から抽出・分割するように変えたものを，多重的にパイプ「|」でつなぐだけのことだ．しかし，スクリプトがやたらに長くなるので，作成過程は読者の練習問題として残しておく．本稿をスクリプトの入門参考として，シェルスクリプトの解説書(e.g. 上田　2016)や正規表現に関する書籍，スクリプト言語AWKについての解説(Aho et al.(1989)や中島ら(2015)などの教科書・参考書類)，ウェブ上の公開情報を参照すればよい．

V．専門用語や学名のカスタムつづり辞書の作成

読者が経験年数を重ねた専門分野については，すでに自分で丹念な校正作業をほどこして出版した論文原稿の電子テキストが残してあるだろう．それをもとにして，(1. 13)，(2. 08)の手順でokspell.txtを作れば，それがそのまま自家用のカスタムつづり辞書になる．しかし，自分では論文を書いたことのない，新たな研究分野に参入するような場合，あるいは指導している学生が指導者の専門分野外まで含む学際領域の研究を選んだり，指導教員が詳しくない分類群の研究を選んだ場合，手元に蓄積がないので，その方法ではカスタム辞書は作れない．研究を始めたばかりの初学者の場合も同様だ．

少ないながらも先行研究者がいて研究者コミュニティが存在していても，博物学など研究者の少ない分野のつづり辞書の需要は小さく，商業ベースにのるはずもない．すなわち，篤志家がいなければ，そんなものは誰も作ってはくれない．ところが，このような場合でもアルファベット文字列のカスタム辞書は，ここに説明するいくつかの方法で，わずかな労力で作ることができる．このような課題は，もっとも単純なタイプのテキストマイニング(text mining)にすぎない．

(3. 01) すでに懇意にしている先行研究者がいて，電子テキストを提供してもらえれば，その電子テキストが容易にプレーンテキストに変換できるならありがたい．特定の分類群全体をカバーするような学名リストである場合には，つづりが信頼できるとすれば，カスタム辞書に加工しておくとよい．

本稿執筆者は，ある植物分類学者に頼まれて，これまでに知られる日本産全種・全亜種の命名者名を含む，その分類群の学名カスタムつづり辞書を作成した．元になった電子テキストファイルは，MS Excelのファイル(拡張子「.xlsx」)であった．「名前を付けて保存」のところで，「テキスト(.csv)」を選んで保存したのち，拡張子「.csv」を「.txt」に書き替えるだけで，(2. 07)の**alphsort2.sh**が使える．(1. 13)，(2. 08)を適用してokspell.txtを作り，たとえばCUSTOM_DICT.txtとでもファイル名を付け替えておけばよい．

ただし，次のことには，改めて注意しておく．「このようなCUSTOM_DICT.txtが誤ったつづりを含んでいる場合には，そのつづりについては間違いを検出することができない」すなわち，事前に十分な目視による校正が必要である．

(3. 02)学名であれば，特定地域あるいは日本

全国，地球全体をカバーする特定分類群の学名リストが，学会や研究機関などが運営するインターネット上のサイトに見つかることがある．たとえば，Mammals' Planetというサイト中のページ，Classification of Mammals: Notions of taxonomy（http://www.planet-mammiferes.org/drupal/en/node/37?taxon=1）には，地球上の哺乳動物の学名がシノニムといっしょにリストされている．2017年１月現在，20, 501種の独立種および亜種が識別されていることが記され，過去に使われた学名もシノニムとして整理されている．すなわち，哺乳類関係の学名に出てくる属名と種小名（動物学では慣例的には「種名」とされるが，植物学で使われる「種小名」の方が混乱をまねかないので，本稿では種小名を用いる）は，ここに多くは含まれている．これをつづり辞書に加工しておけば，本稿執筆者を含む同分野関係者は重宝するだろう．

本稿の準備を契機に，そのページをブラウザー上で右クリックして，「すべて選択」を選び，コピーし，テキストエディタにペーストして，不要部分を削除するなどの編集をしたのち「名前を付けて保存」を選び，mammal_names.txtとして保存した（2017年１月13日ダウンロード）．これから，14, 920語からなる哺乳類学名つづり辞書mammal_dict.txtを得た．このようなつづり辞書は，spellの「-d」オプションと組み合わせて使うことができる．

他にも，DBCLSメタ用語集（http://lifesciencedb.jp/lsdb.cgi?gg=dic；2017年１月25日ダウンロード）にある「メタ用語集　学名編」などを，分類群や出典ごとにまとめたりして（表計算ソフトを使うか，grepとAWKを使えばできる），便利な学名つづり辞書を作れるだろう．

（3. 03）特定分野の専門用語辞書を，ウェブに掲載している大学や学会，無償サイトもある．たとえば，DBCLSメタ用語集（http://lifesciencedb.jp/lsdb.cgi?gg=dic；2017年１月25日ダウンロード）にある「メタ用語集 学術用語編生物学」や医学・薬学・農学分野の辞書類多数にもリンクがはられた「翻訳と辞書」関連の無償サイト（http://www.kotoba.ne.jp/）から適当に移動して，自分の分野をカバーする専門用語辞書をもとに，（3. 02）と同じ手順で，CUSTOM_DICT.txtを作ることができる．

動物学とも共通する専門用語を多数含む人体解剖学や寄生虫学も入る医学関係であれば，「医歯薬英語辞書（MEDO）」のサイト（http://www.medo.jp/0.htm；2017年１月13日ダウンロード）もある．元にする辞書テキストは，alphsort.sh（あるいは必要に応じてalphsort2.sh）を用いるので，和英・英和でも，英英でもかまわない．元辞書のつづりが信頼できるものでありさえすれば，alphsort.shで処理したものについては，spellに「-Damerican-huge」あるいは「-Dbritish-huge」オプションを指定して，のちのちiamerican-huge/ibritish-hugeと一緒に使う，よりコンパクトなCUSTOM_DICT.txtを得ることができる．

この手順で，寄生虫学の専門用語集をparasitol.txtとして，それを元に専門用語つづり辞書parasitol_dict.txtを作ったところ，813語の辞書が得られた．これは「`spell -Damerican-huge`」を適用する前のparasitol_alphsort_out.txtが2, 659語だったので，共通にカバーされた分が，1, 849語もあったことを示す．すなわち，この1, 849語については，日本語で寄生虫学関係の文書を書くような場合，spellを使わなくても（3. 07）や（3. 08）の方法で，parasitol_alphsort_out.txt自体を参照つづり辞書代わりに使えることを示している．「この辞書には載っていない」と判定された文字列だけを別途点検するという使い方ができるのだ．

また，iamerican-huge/ibritish-hugeと一緒に使う場合でも，メモリに十分な余裕のあるHDDやSSDを使って，MPUの処理速度もそこそこある場合には，出発点で小さな辞書であれば，コンパクトなCUSTOM_DICT.txtに削る手間は省いてもよいだろう．

（3. 04）私家版の論文コーパスを使う方法もある．本稿執筆者は，日頃，pdfなど電子テキストになっている論文を読むときは，ついでにpdftotext（（2. 14）末参照）を使ってプレーンテキスト化し，分野ごとに自家用の論文コーパスを作っている．無償で使えるコンコーダンサ（concordancer），AntConc（http://www.laurenceanthony.net/software/antconc/）と組み合わせて，分野特有の用語の使い方などを参照する「活用辞典」として，これまで使ってきた（高崎 2012, Takasaki 2013）．

AntConcにはword listを得る機能があって，以前試作した蝶類学用のコーパス（高崎　2012）の拡張版からは，総数1, 227, 120 word tokensが得られ，35, 241 word types に分けられた．出現頻度順にソートして，頻度１回のものは誤植の可能性もあるだろうから，頻度２回以上の単語を拾ったつづり辞書を作ることを考えた．（ただし，AntConcのword listは，大文字・小文字を区別せずに作られるので，ここで構想しているような学名（科名や属名などは必ず大文字で始まる）を含むつづり辞書には使えない．おおまかな数値の予備情報が得られただけである．）

次のようにして，これは実現できる．（2. 07）のalphsort.shとは異なる手法を使って，CUSTOM_DICT.txtを作る元となるTEST.txt相当のファイルを得るのである．まず，コーパスを構成する複数のテキストファイル全部を，１つのテキストファイルUNITEDTEXTS.txtにまとめる．

1つのディレクトリに入っているすべてのテキストファイルを一気にUNITEDTEXTS.txtにまとめる

には，ターミナルでそのディレクトリ内に移動して，ワイルドカード「*」を使った次のコマンドを実行すればよい．

```
$ cat *.txt > UNITEDTEXTS.txt [Enter]
```

得られたUNITEDTEXTS.txtの構成単語を，出現頻度統計と一緒に頻度順にリストアップするスクリプトは，すでにII(2. 09)に紹介したwordfreq.awkである．このスクリプトでUNITEDTEXTS.txtを処理するには，次のように実行する．

```
$ gawk -f wordfreq.awk
UNITEDTEXTS.txt | tee
wordfreq_results.txt [Enter]
```

出力wordfreq_results.txtの中には，出現頻度順に頻度と単語が対になったリストが得られる．つづいて，出現頻度が２回以上のものについて，各行の右側にある単語だけを出力しなければならない．これには次のawkスクリプトを使う．

```
$ gawk '$1 >= 2 {print $2}' <
wordfreq_results.txt | tee
candidate_words.txt [Enter]
```

これら２つのコマンド・ラインの実行最終結果は，パイプ「|」を使って途中をつなぎ，次の１行でも得られる．

```
$ gawk -f wordfreq.awk
UNITEDTEXTS.txt | gawk '$1 >= 2 {print
$2}' | tee candidate_words.txt [Enter]
```

さらに不要文字・記号等を一気に取り去るために，alphsort.shで処理する．

```
$ bash alphsort.sh candidate_words |
tee result_words.txt [Enter]
```

このようにして蝶類学コーパスから得たresult_words.txtをもとに，iamerican-hugeならびにibritish-hugeつづり辞書との重複を除いて，さらにエディタで不要なものを取り除いたりして，目視校正中の2017年８月下旬現在，日本産チョウ類全種の属名，種小名を含む約4, 300語弱の蝶類学つづり辞書butterfly_dict.txtが得られている．

(3. 05)alphsort.shを用いて作ったCUSTOM_DICT.txtに含まれる単語のつづりには，ドット「.」やハイフン「-」は含まれていない．そのため(1. 11)で説明したように，spellの「-d」オプションを使う「spell -dCUSTOM_DICT.txt」で，問題なく利用できる．あるいは，ワープロのスペルチェッカに，CUSTOM_DICT.txtを参照辞書として指定することで，再利用可能になる．

たとえば，butterfly_dict.txtを使って，TEST.txtをチェックするには，次のようにすればよい．

```
$ spell -dbutterfly_dict.txt TEST.txt
[Enter]
```

応用的には，**myspellcheck3.sh**で「-dokspell.txt」の部分を「-dbutterfly_dict.txt」に書き換えるなどして組み込む，あるいはMS Wordなどの「ユーザー辞書」として使うという使い方もできるだろう．

VI．カスタム・スペルチェッカの作成

植物分類学や昆虫分類学のように，いまなお新種の記載と過去の命名種の整理統合が普通に続く分類群では，指し示す学名とタイプ標本を特定する上で，命名者の略名が重要情報として識別に使われる．このような場合には，(3. 01)の例のように**alphsort2.sh**を使って作った，「.」と「-」を含む文字列つづり辞書を使わなければならない．何とかこれを使ったスペルチェッカを工夫する必要がある．(spellのオプションの工夫で実現可能かもしれないが，manコマンドやinfoコマンドでは判明しなかった．)既成のスペルチェッカを使わない別の方法を１つだけ紹介しておく(校正時補記：diffを使う別法も可能)．

(4. 01)まず，準備作業を行なう．点検するテキストTEST.txtに前処理を加える．(前処理の理由は次の(4. 02)で明らかになる．)処理には，**alphsort2.sh**の先頭部分を参照して書きなおした次の４行を連続して使う．

```
$ LANG=C echo "" > tmpworkfile.txt
[Enter]
$ nkf -wLu TEST.txt >> tmpworkfile.
txt [Enter]
$ echo "" >> tmpworkfile.txt [Enter]
$ cat tmpworkfile.txt | tr \\n ' ' |
sed 's/[^a-zA-Z\.\-]/@ @/g' | tr ' ' \\n
| sort | uniq | tee mid_text.txt [Enter]
```

これでTEST.txtの前後に空行が付け加えられ，文字・改行コードはPOSIX標準に揃えられて，改行が一度すべてスペースに置換される．さらに，「a-z」や「A-Z」，「.」，「-」「ではない」(正規表現先頭の「^」が「ではない」を示す)文字は，すべて(sedの''内末尾の「g」は“globally”の意)スペースを挟んだ@２つ

「@ @」に置換(sedの''内先頭の「s」は“substitute”)される．次にスペースが改行に置換され，アルファベット順に並べ替えられて，重複を除去されたものが画面に流れ，前処理済みの中間テキストファイルmid_text.txtとして保存される．

(4.02)紹介するスペルチェッカは，sedのスクリプトで実現する(4.03)−(4.05)の方法である．CUSTOM_DICT.txtを加工して作る．まず準備段階の背景説明をしておこう．

参照辞書の中の文字列を最初から最後まで１つずつ選んでは，点検テキスト中のアルファベット文字列の中の同一の文字列をすべて消す，すなわち空白文字に置換することを考えてみよう．

思考実験のためのつづり辞書CUSTOM_DICT.txtとしてはpapp_dict.txtと名づけた，７単語{i, have, a, pencil, an, apple, applepie}だけからなるものを想定する．点検テキストTEST.txtは「i have a pencil i have an apple apple pencil i have an apple pencil i have a pencil i have an applepie applepie pencil」とする．このTEST.txtの構成単語は，papp_dict.txtにすべて載っているので，ミスタイピングがなければ，すべて空白文字に置換されるはずである．

テキストで，まず辞書中の最初の文字列「i」と同じ文字列を，「@」に全置換すると次を得る．(「@」は空「」でもよいのだが，ここでは何が起こったのか，分かりやすくするために「@」を使っている．)

```
@ have a penc@l @ have an apple apple penc@l @ have an apple penc@l @ have a penc@l @ have an applep@e applep@e penc@l
```

困ったことに，「i」を部分文字列に含む「pencil」と「applepie」が化けてしまった．続いて辞書中２番目の文字列「have」と同じ文字列を「@」に全置換する処理をほどこす．

```
@ @ a penc@l @ @ an apple apple penc@l @ @ an apple penc@l @ @ a penc@l @ @ an applep@e applep@e penc@l
```

今度は困ったことは起きていない．続いて辞書中３番目の文字列「a」でも同様に全置換する．

```
@ @ @ penc@l @ @ @n @pple @pple penc@l @ @ @n @pple penc@l @ @ @ penc@l @ @ @n @pplep@e @pplep@e penc@l
```

「a」を部分に含む文字列まで化けた．同様に，「pencil」でも全置換する．

```
@ @ @ @ @ @ @n @pple @pple @ @ @ @n @pple @ @ @ @ @ @ @ @ p@ne@pple p@ne@pple @
```

当然のことながら変化はない．さらに文中の「an」，「apple」，「applepie」を次々と「@」で全置換する処理をしても，もう何も変わらない．「@」とスペースだけが残った結果を得るには，単語の長さ順，すなわち，「applepie, pencil, apple, have, an, a, i」の順に，全置換をほどこせばよかったのだろう．

(4.03)単語数７語程度で構成されているのなら，目視で簡単に長い順に登録単語の並べ替えができる．しかし，少なくとも数百語，普通でも数千語((3.01)であげた例「ある全日本産植物分類群の学名つづり辞書」で3,000語弱)，さらに(3.02)の例に見るように，10,000語超から構成されるつづり辞書も珍しくはない．単語を長さ順に並べ替えるのは，AWK((2.09)−(2.12)，(3.04)でも使った)によるスクリプトでも作れば実現できるが，とりあえずは読者への説明を難解にしたくない．限られた知識と普通のテキストエディタを使って，ここまでに説明したsedのスクリプトの応用で実現できる方法をとろう．

テキストTEST.txt「i have a pencil i have an apple apple pencil i have an apple pencil i have a pencil i have an applepie applepie pencil」を(4.01)のスクリプト４行で処理すると，出力mid_text.txtは「@applepie@, @pencil@, @apple@, @have@, @an@, @a@, @i@」となるだろう(実際には「, 」の代わりに改行が入る)．これを７単語{@i@, @have@, @a@, @pencil@, @an@, @apple@, @applepie@}からなるつづり辞書にもとづいて，先ほどと同じ順番で(すなわち文字列の長い順にすることはしない)無文字(スペースも含まない)の空「」で全置換をする．

結果は７つの空「, , , , , , 」になる(実際には「, 」の代わりに改行が入る)．「@」を空「」で全置換し(この場合，無変化だが)，さらに空行を削除すれば，最終出力結果は空っぽとなる．すなわち，TEST.txtにはこの参照つづり辞書から判断する限り，「間違いと指摘されるつづりはない」，つまり「つづり間違いがない」ことが示される．

別の入力テキストTEST2.txtに辞書にない文字列が入っていて，それを同様に処理すれば，最初に「@」と「@」に挟まれ，辞書にあったつづりが消えたあと，処理の終わり近くで辞書にない文字列を挟む両端の「@」と「@」だけが消える．すなわち，辞書にない文字列が最終出力に残る．このことを使って，スペルチェッカとして使える仕掛けができる．次のように，sedを使ったスクリプトで書ける．

```
  $ sed 's/@i@//g' mid_text.txt | sed
's/@have@//g' | sed 's/@a@//g' | sed
's/@pencil@//g' | sed 's/@an@//g' | sed
's/@apple@//g' | sed 's/@applepie@//g'
| sed 's/@//g' | grep -v '^s*$' | tee
papp_dict_checked.txt [Enter]
```

ここの中間出力mid_text.txtは，(4. 01)の最終行を組み合わせることで，途中の「| tee」とともに消去・削除できる．

```
  $ sed 's/[^a-zA-Z.-]/@ @/g'
tmpworkfile.txt | tr ' ' \n | sort | uniq
| sed 's/@i@//g' | sed 's/@have@//g' |
sed 's/@a@//g' | sed 's/@pencil@//g' |
sed 's/@an@//g' | sed 's/@apple@//g' |
sed 's/@applepie@//g' | sed 's/@//g' |
grep -v '^s*$' | tee
papp_dict_checked.txt [Enter]
```

シェルスクリプト中のパイプ「|」は直後に改行があっても有効に働くので，読みやすく書き直す．

```
  $ sed 's/[^a-zA-Z\.\-]/@ @/g'
tmpworkfile.txt | tr ' ' \\n | sort |
uniq |
  sed 's/@i@//g' |
  sed 's/@have@//g' |
  sed 's/@a@//g' |
  sed 's/@pencil@//g' |
  sed 's/@an@//g' |
  sed 's/@apple@//g' |
  sed 's/@applepie@//g' |
  sed 's/@//g' |
  grep -v '^s*$' | tee
papp_dict_checked.txt
```

左端が「 sed」で始まり，右端がパイプ「|」で終わる２～９行目部分はより簡潔に書き直すことができる．次の９行を**papp_dict.sed**と名付けたスクリプトファイルとして保存する．すると，上記の２～９行目部分については「sed -f papp_dict.sed」というコマンドで同等の結果を得ることができる．

```
  ##papp_dict.sed
  s/@i@//g
  s/@have@//g
  s/@a@//g
  s/@pencil@//g
  s/@an@//g
  s/@apple@//g
  s/@applepie@//g
  s/@//g
```

すなわち，全体が次のように書き直せる．

```
  $ sed 's/[^a-zA-Z.-]/@ @/g'
tmpworkfile.txt | tr ' ' \\n | sort |
uniq | sed -f papp_dict.sed | grep -v
'^s*$' | tee papp_dict_checked.txt
[Enter]
```

スクリプトファイル**papp_dict.sed**は，(3. 06)のCUSTOM_DICT.txtとしたpapp_dict.txtで，改行を挟んで並ぶ各構成単語の前後に「s/@」と「@//g」を付け，さらにそれらに最終行「s/@//g」を加えたものである．これは，「cat papp_dict.txt > papp_dict.sed」を実行するか，あるいはテキストエディタで，papp_dict.txtを新たに「名前を付けて保存」することで，編集前の**papp_dict.sed**を作って，高機能なテキストエディタで一括置換等をほどこして作ればよい．

(4. 04)このようにして，CUSTOM_DICT.txtから，次のようなそのつづり辞書に特化したスペルチェッカCUSTOM_dict.shを作ることができる．ここの例では**papp_dict.sh**と名付けておこう．なお，最終結果としては不要なtmpworkfile.txtは処理の最後に除去(rm，”remove”の意味)する．

```
  #!/bin/bash
  ##papp_dict.sh
  ##CUSTOM_DICT spellchecker with
papp_dict.txt as CUSTOM_DICT.txt
  #to run, papp_dict.sed is needed
  LANG=C echo "" > tmpworkfile.txt
  nkf -wLu $1.txt >> tmpworkfile.txt
  echo "" >> tmpworkfile.txt
  cat tmpworkfile.txt | tr \\n ' ' | sed
's/[^a-zA-Z\.\-]/@ @/g' | tr ' ' \\n |
sort | uniq |
  sed -f papp_dict.sed |
  grep -v '^s*$' | tee
$1_papp_dict_checked.txt
  rm tmp*.*
```

ここまでの汎用スクリプトと同様に，入力の拡張子「.txt」は省略して入力する仕様になっている．

```
  $ bash papp_dict.sh TEST [Enter]
```

処理結果が画面に流れて，TEST_papp_dict_

checked.txtに書き出される．取り扱う研究分野の数が多くなければ，その数だけCUSTOM_dict_check.shを作ることで対処できる．すなわち，対応するCUSTOM_dict.sedを作ればよい．個人でカバーできる専門分野はせいぜい数分野であろうから，手間はかからない．（CUSTOMを適当に専門分野に合わせた短い文字列に書き換えると，使い勝手がよい．）

(4. 05)なお，パイプ「|」を使った連結やsedスクリプトがあまりにも長くなると，喩えるなら「長すぎる用水路は集水域が広すぎて使えない」のに似て，“stream editor”を意味するsedの処理が正常に進まなくなる．画面には，そのあたりの処理行番号とともに不具合のエラーが示される．

その場合には，エラーが出る行よりも前の適当な行末にある「|」を「> tmpresult.txt」に換えたり，問題の出る行で使っているsedスクリプトを適当に分割して，そこまでの結果を一時的に「遊水桝」的なtmpresult.txtに書き出す．さらに，その次の行の入力をtmpresult.txtに換えて処理を続けるように，スクリプトを書き直す．処理を分節化するのだ．これを同種のエラーが出なくなるまで繰り返すことで，最終目的のスクリプトが作れる．

papp_dict.shの最終行がワイルドカード「*」を使った「rm tmp*.*」としてあるのは，不要になるtmpresult.txtのような中間出力ができる場合には，それもtmpworkfile.txtと一緒に除去するためである．

(4. 06)「III．英語文書への対処：既存スペルチェッカの導入」のspellが使えるようにならない場合や-Dオプションが使えない場合などには，この「VI．カスタム・スペルチェッカ」の仕組みを充実させることで，代替処理も可能になる．ここまで(III–V)のスクリプト類のほか，それらを組み合わせた応用例(.doc，.docx，.pdf文書を直接解析する**docprfrd.sh**，**docxprfrd.sh**，**pdfprfrd.sh**など)も本稿のpdfと同じURL(http://www1.ous.ac.jp/garden/kenkyuhou3.html)から，zipファイル(foolproof_scripts.zip)としてダウンロードできる．

VII．おわりに

「1970年01月01日00時00分00秒から，Unixのシステム時計は時を刻み始めた」という想定で，POSIX準拠Unix互換機は動いている．1970年当初から使われてきたASCIIコードは，大文字・小文字の全アルファベットを含む128種類の文字・記号類を識別する．すなわち，７ビット(2^7=128)の文字・記号コードだ．したがって，1980年代前半までに８ビットCPUのパソコンが普及するのと同時進行で，米欧では文字情報のコンピュータ処理が社会一般で速やかに進み，ワープロソフトの利用も浸透した．

その間に，旧大英帝国領を含む英語圏の国々や，いわゆるローマ字アルファベットを表記文字の基本とするヨーロッパ，さらにそれらの旧植民地などでも，８ビット以内で十分な文字情報のコンピュータ処理，すなわちワードプロセッシングが普通になった．

20世紀後半は，コンピュータ誕生・発展の時代であると当時に，生物学分野ではDNA情報解明の時代でもあった．世紀末には，この両分野の融合がバイオインフォマティクスとなって，現在も加速的に発展している(e.g. Mount 2004参照)．DNAのコンピュータによる解析とワードプロセッシングは，一見，無関係に思えるかもしれない．

ここで，特定の塩基配列パターンで切断する制限酵素を，連続した塩基配列SEQUENCE.txtをもつDNA鎖に反応させて，得られる断片の塩基配列SEGMENTS.txtを求めることを想定してみよう．シェルスクリプトが使えれば，解は簡単に求めることができる．たとえば，反応させる制限酵素が*Alu*Iであれば，5′AGCTという配列部分をAGとCTの間で切断する．そのスクリプトは「`sed 's/AGCT/AG CT/g' SEQUENCE.txt | tr ' ' \\n | tee SEGMENTS.txt`」でよい．

コンピュータによるDNA塩基配列解析の基本は，アミノ酸配列への翻訳を含め，文字列処理にほかならない．８ビットCPUのパソコン時代に文字列処理が普通になったアルファベット使用言語文化圏では，この文字列処理一般の裾野の広がりが，やがて分子生物学におけるコンピュータによる解析法を理解し，応用研究を発展させる素養をもつ人々を育てるといったこともあっただろう．

ハードウェアが16ビットCPU機になって，PC/AT互換機が普及し，1991年にはLinuxの原型が生まれた．やがて無償で使えるOSであるDebian GNU/LinuxやFreeBSDが，リサイクル品を含む廉価なPC/AT互換機というハードウェアにのって，当時，発展途上にあった国々の多くに，情報革命を含む社会変革の波(Toffler 1980，1990)を確実に伝播させ，世界の産業地図を塗り替えた．その典型例がインドからの情報処理専門家の輩出である．この波は，21世紀に入って学術分野にまで波及する兆しを見せている．

たとえば，蝶類学は欧米や日本など，食うに困らない国々の博物学者や趣味人が研究する学問だった．しかし，いまやインド人研究者が中心的な役割を果たした刮目すべき研究も現れている．Kunte et al.(2014)は，シロオビアゲハ(*Papilio polytes*)に見られる擬態多形の研究を再統合した．遺伝学や分子生物学の研究結果を，コンピュータによる解析も駆使して，それまで仮説(Charlesworth and Charlesworth 1975)にすぎなかった同種内のベーツ型擬態多形に関与する超遺伝子(supergenes)が存在し，しかも単

一の超遺伝子の場合もあるという実例の抽出に導いたのである．

他方，日本語は８ビットでは通常使う「漢字・カタカナ・ひらがな・alphanumerics」をカバーすることができず，ワープロが一般化するのは，16ビットCPUを搭載したパソコンの登場と20世紀終盤になってのその普及まで，待たなければならなかった．Unixには1970年誕生以来の膨大なソフトウェア資産があって，現在ではその多くが無償で使える．しかし，英語で書かれたUnix関連の名著テキストの初版が，早くても数年遅れでやっと和訳で読める頃には，英語で普通にコミュニケーションのとれる国々では，何版か後の最新版が読まれている．

このような具合で，Unixのソフトウェア利用は，日本では広く一般に浸透することはなく，海外に大きく立ち遅れた．日本のガラパゴス化は携帯電話に限らないのだ．（８ビットCPU時代までに日本で広く普及したのは，電卓と自動販売機，デジタル時計だった．）

21世紀に入って出た自然言語処理の入門書，Bird et al.（2009；http://www.nltk.org/book/；2017年５月10日ダウンロード）による*Natural Language Processing with Python*の日本語版（2010）には，わざわざ日本語自然言語処理の章が，翻訳者による書き下ろしの最終章（http://www.nltk.org/book-jp/ch12.html；2017年５月10日ダウンロード）として追加された．それほど，異なる言語文化圏を行き来したい者の前に立ちはだかる壁は厚く，高い．

日本語文化圏では，本稿執筆者を含む，まったく「蚊帳の外」分野の理系研究者もいまだに多く，なんとか絶滅をまぬがれている．Unix誕生から半世紀が過ぎる日も遠くない現在，情報処理やロボティクス，バイオインフォマティクスの分野等を除けば，日本でのシェルスクリプトの利用は，さほど浸透していない．また，それら広義の情報処理分野の専門家ですら，本稿で紹介したようなつづり簡易校正法を自らの原稿作成に使っている，という話は聞かない．

シェルスクリプトは，Unixターミナルでの基本的な道具である．POSIX準拠Unix互換ターミナルの使えるコンピュータを，研究の手足に使うようになった学術職人の「シェル芸」（上田・後藤 2014，上田 2016）は，木工職人の手鋸や手鉋，あるいは電動鋸や電動ドリル程度の基本ツール，それらを使って作ったジグの使用術に相当する．GUIで使う便利なアプリケーションだけでは，仕事上の細工が不足する場合，職人技の現場合わせ（bricolage）をもって対処しなければならない．そのような場合には不可欠の道具であろう．

生物学系の読者が関与する文書原稿の準備作業に，また本誌のように和文と欧文が混じる，とくに博物学系の雑誌の編集作業に，本稿で紹介したシェルスクリプトによる簡易校正法は，広く実用的に役立つはずである．ハードウェアは現在では64ビットMPU機が普通となっている．処理スピードは，Unixが誕生した1970年代から格段に速くなり，価格も安くなった．文字コードもUnicodeとなって，その日本語UTF-8もパソコンで普通に使われ，POSIX準拠Unix互換のターミナルが使えるパソコンも増えた．減価償却済みのPC/AT互換機（あるいはIntel系MPUを積んだMacintosh）をLinuxで再生させるだけで，POSIX準拠Unix互換機が手に入る．言語文化の壁を乗り越えて，よく噛み砕かれたシェルスクリプト関連の手引書が，日本語による書き下ろしでも現れ始めた（e.g. 中島ほか 2015，上田 2016）．

海外に遅れること数十年で，ようやく情報処理の素人（本稿執筆者を含む）でも，日本語の文字列処理に独学で立ち向かうことができる条件が整ったのである．すくなくとも理系，とくに医学や農学も含む生物学・博物学系の文書処理では，シェルスクリプトの使用が日常的な実用性をもたらす．本稿は，その例証にもなっているだろう．

謝辞

本稿の準備には，推敲やさまざまな段階でのテスト等で多くの方々の協力をいただきました．とくに次の方々（敬称略，順不同）には大いに助けていただきました：長谷川 大，櫃本泰雄，池之上 公，栗山 定，増井暁夫，目加田和之，長峯 隆，名取真人，西村直樹，太田雄太郎，斎藤基樹，篠原明男，高崎弥生，矢後勝也，山本順司，山崎真理恵，横地 隆．また，本誌*Naturalistae*および日本蝶類学会誌*Butterflies*の編集事務局には，最終校正段階での校正漏れの点検に，本稿で紹介したスクリプトの試用機会をいただきました．入念に丹念に行ったとしても，許容時間内で目視だけに頼る校正には限界があることと，本稿で紹介したスクリプトの高い有効性が確認できました．記して感謝します．

引用文献

Aho, A.V., Kernighan, B.W., and Weinberger, P.J. (1989). The AWK Programming Language.（足立高徳［訳］2004『プログラミング言語AWK』新紀元社，ISBN：4-7753-0249-3．287pp.）

麻生二郎・大橋源一郎・斎藤幾郎・高橋正和・水野 源（2017）．特集１．今すぐできる！Windows版Linux超活用．『日経リナックス』2017. 01：18-53.

Bird, A., Klein, E., Loper., E.（2009）．Natural Language Processing with Python.（萩原正人，中山敬広，水野貴明［訳］2010『入門 自然言語処理』 オライリー・ジャパン，ISBN：978-4-87311-470-5.

556pp．）
Charlesworth, D. and Charlesworth, B. (1975). Theoretical genetics of Batesian mimicry I. Single-locus models. J. Theoret. Biol. 55: 283-303; II. Evolution of supergenes. J. Theoret. Biol. 55: 305-324; III. Evolution of dominance. J. Theoret. Biol. 55: 325-337.
Cotton, A.M. (2016). The correct spelling of the name for the subspecies of *Graphium evemon* (Boisduval, 1836) (Lepidoptera: Papilionidae) in mainland SE Asia, confirmation of the female phenotype and distribution of the taxon. Butterflies 73: 48-52.
Kunte, K., Zhang, W., Tenger-Trolander, A., Palmer, D.H., Martin, A., Reed, R.D., Mullen, S.P., and Kronforst, M.R. (2014). *doublesex* is a mimicry supergene. Nature 507: 229-232.
Mount, D. W. (2004). Bioinformatics: Sequence and Genome Analysis, 2nd ed. Cold Spring Harbor Laboratory Press, Cold Spring Harbor, New York.（岡崎康司・坊農秀雅[監訳]『バイオインフォマティクス第2版：ゲノム配列から機能解析へ』メディカル・サイエンス・インターナショナル，ISBN：978-4895924269．644pp.）
中島雅弘・富永浩之・國信真吾・花川直己(2015)．『AWK実践入門』技術評論社，ISBN： 978-4-7741-7369-6，399pp.
奈佐原顕郎(2016)．『入門者のLinux―素朴な疑問を解消しながら学ぶ』講談社(ブルーバックス)，ISBN：978-4-06-257989-6，315pp.
大橋源一郎(2017)．特集２．自分にぴったりの学習法が見つかるLinuxの始め方30．『日経リナックス』2017. 04：67-87.
太田朋子(2009)．『分子進化のほぼ中立説―偶然と淘汰の進化モデル』講談社(ブルーバックス)，ISBN：978-4062576376，170pp.
Stutz, M. (2001). The Linux Cookbook: Tips and Techniques for Everyday Use. http://dsl.org/cookbook/cookbook_toc.html (retrieved 16 September 2016; pdf [528pp.] also retrieved).
高橋正和・加藤慶信(2017)．特集２．Windows乗っ取り活用大作戦：Windows版Linuxだからできる．『日経リナックス』2017. 06号：63-83.
高崎浩幸(2012)．一日で作る私家版「蝶類学英語活用電子辞典」．Butterflies（*Teinopalpus*）61：48-51.
Takasaki, H. (2013). An instant electronic dictionary of English collocations in natural history realized using a concordancer and open resources. Naturalistae 17: 59-62.
Toffler, A. (1980). The Third Wave. William Collins and Sons, London. (Pan Books paperback [1981], 544pp.)
Toffler, A. (1990). Power Shift. Bantam Books, New York. (Bantam paperback [1991], 611pp.)
東京シェル研究会(2017)．『シェルスクリプト「レシビ」ブック』工学社，ISBN：978-4-7775-1994-1.
上田隆一・後藤大地(2014)．『シェルスクリプト高速開発手法入門』KADOKAWA，ISBN：978-4-04-866068-6，279pp.
上田隆一(2016)．『シェルプログラミング実用テクニック』技術評論社，ISBN：978-4-7741-7344-3，393pp.

要旨

現代日本語の印刷テキストは，漢字や片仮名，平仮名，ローマ字にくわえて数字や記号で構成され，総計数万もの文字種コードで成り立っている．生物学は，アルファベットでつづられる学名や専門用語にあふれ，日本語で書かれた生物学の論文で，アルファベット文字列のつづり間違いが未訂正のまま多数残ると，混乱の極みとなる．この状況は，日本国内の校正者の頭痛のたねである．また英語で論文を作成するときも，生物学者は「学名や専門用語の多くが市販のスペルチェッカでは点検不可能である」という同じ問題に直面する．本稿は，このような校正プロセスを，コンピュータを利用して簡易化する代替手法の試みである．まず，アルファベットの単語や文字列(従来のスペルチェッカで直ちに検出・修正されるものを除外)すべてを，テキストから重複なく抽出し，アルファベット順に並べる．このように配置することで，互いに少数文字だけで異なる類似した単語や文字列は，隣接して近傍に並ぶことが多く，つづり間違いが容易に検出できる．このようにすれば，テキストを何度も繰り返して読む従来の校正手順によるよりも，アルファベット文字列は簡単に点検できる．アルファベット順に重複なく並ぶ単語と文字列のチェックリストは，ここに公開するシェルスクリプト類を使用して準備することができる．これらのスクリプトは，POSIX-Unix互換のシステム(例えばLinuxやMac OS X，Windows 10)のターミナルで動く．本稿は，最初に必要となるハードウェアやOS，ソフトウェアの準備を説明して，オープンソースの枯れたスペルチェッカを導入する．さらに，日本語テキストでアルファベットの文字列を点検するスクリプトを紹介する．また，テキストマイニングによって個人やオープンソースの文書から，カスタムつづり辞書を作成するスクリプトや，特定の専門分野で使用するように調整したカスタムスペルチェッカとなるスクリプトの作成法を公開する．ASCII英数字は，UTF-8日本語文字コードのサブセットを構成しているので，これらのスクリプトは，修正なしで英文テキストにも適用できる．同じアプローチは，日本語のように何万もの文字コードで構成されている日本語以外の言語で書かれた文書にも適用できる．

(2017年11月22日受理)