

総説・解説

原稿中の漢字カタカナ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 to ease proofreading of katanaka and Chinese character strings in Japanese texts

Hiroyuki TAKASAKI[1]

Abstract: This article presents use of shell scripts to ease proofreading of katakana and Chinese character strings in Japanese texts. Modern Japanese texts in print are made of kanji (Chinese characters), katakana, hiragana, and romaji (Roman alphabets) in addition to numerics and symbols. Particularly in scholastic writings, key terms are mostly in katakana and kanji, whereas hiragana strings may be skipped when the text is skimmed through for its rough outline. In biological articles, for example, technical terms and names of organisms, in addition to proper nouns, are mostly written in katakana and kanji if not in romaji. However, many are uncheckable with commercially available spellcheckers. This situation gives a headache to the proofreader. Therefore, the proofreader's burden will be greatly reduced if katakana and kanji typos can be checked easily by any means. The present article is an attempt toward a computer-aided routine simplifying part of the proofreading process of texts. First, all katakana and kanji strings are extracted without redundancy from the electronic text and sorted. In this arrangement, some or all strings placed in neighbors with one another, differing in one or a few characters, are easily identified as typos. This makes katakana and kanji strings be checked easier than by the traditional practice of repeated sequential reading of texts. The checklist of unique and sorted katakana and kanji strings can be easily prepared using shell scripts presented here, which are run in terminals of POSIX-compliant systems (e.g. Linux, Mac OS X, and Windows 10). Furthermore, the article presents scripts for preparing general and custom spelling dictionaries for katakana and kanji strings by text-mining private and open source documents. It also presents methods of tuning them for use in specific disciplines. Use of the dictionaries greatly cuts the size of the checklist of dubious strings, and further eases the proofreading procedures. Use of the scripts presented here in combination with those previously provided by the same author for checking alphabetical strings would greatly reduce burdens of proofreaders and editors when handling scholastic manuscripts and proofs in Japanese.

目次

I．はじめに

誤植も楽しむための文芸作品(e.g. 高橋 2013)ならともかく，学術文書では誤植は困る．「アオタテハモドキ(*Junonia orithya*)は近似種のタテハモドキ(*Junonia almana*)・・・」が誤植で「アオタチハモドキ(*Junoria erithya*)・・・近侍種のタテハモドキドキ(*Juronia ulmana*)・・・」となったり，「コキクガシラコウモリ(*Rhinolophus cornutus*)はキクガシラコウモリ科キクガシラコウモリ属・・・」が「コキクガラシコウモリ(*Rhinolephus cornateus*)・・・キクガシラコモリ・・・」，「ノスリ(*Buteo japonicus*)はヨーロッパノスリ(*Buteo buteo*)・・・」が「・・・(*Butejo japonico*)・・・ヨーロッパノリス(*Butejo butejo*)・・・」，「カダヤシ(*Gambusia affinis*)は，姿の似るメダカ(*Oryzias latipes*)・・・」が「カヤダシ(*Ganbusia affinos*)・・・メカダ(*Olyzias ratipes*)・・・」，「コイワザクラ(*Primula reinii*)は別名オオミネコザクラとも呼ばれ，サクラソウ科(*Primulaceae*)・・・」が「ロイワザクラ(*Primulla reini*)・・・オオミミネコザクラ・・・サフラソウ科(*Primuraceae*)・・・」と誤植されて公刊されてしまったのでは困るのである．

市販のつづり辞書のない分野の欧文スペルチェッカでは対処不能な誤植のうち，学名など難易度の高いアルファベットつづりの簡易校正については，

1. 〒700−0005 岡山県岡山市北区理大町1−1 岡山理科大学理学部動物学科 Shell scripts to ease proofreading of katanaka and Chinese character strings in Japanese texts. Hiroyuki TAKASAKI. 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

スクリプト集で対処可能となるように，すでに検出のための工夫ができている(高崎　2018)．それらを使うことで“*Junoria*，*erithya*，*Juronia*，*ulmana*，*Rhinolephus*，*cornateus*，*Butejo*，*japonico*，*butejo*，*Ganbusia*，*affinos*，*Olyzias*，*ratipes*，*Primulla*，*reini*，*Primuraceae*”は，いずれもミススペリングであることが検出できる．

その一方で，いわば日本語における隠蔽擬態的な誤植である「近侍種，アオタチハモドキ，タテハモドキドキ，コキクガラシコウモリ，キクガシラコモリ，ヨーロッパノリス，カヤダシ，メカダ，ロイワザクラ，オオミミネコザクラ，サフラソウ」などは，注意深い校正者にも検出は簡単ではない．このような漢字やカタカナの文字列については，従来の非効率な繰り返し読みに頼るしか，これまで誤植として検出する術はなかった．しかし眼精疲労や老眼，思い込みによる読み飛ばしが，容赦なく校正作業の邪魔をする．発刊後に気付いた途端，著者も編集者も博物学系の学術専門家としては面目も権威も丸潰れで，あとの祭り，時すでに遅し，顔面蒼白，佇立したまま，心拍数が急上昇したり，茫然自失，「効く辛子」のショックを受けて，「不十分だった校正を悔やむ」という心的外傷の後遺症が末長く続く．そこで高崎(2018)の続編として，２文字以上からなる漢字カタカナ文字列について，誤字・脱字・余字を簡易に点検するための手順とスクリプト集を作成したので，ここに発表する．

本稿では「文系までも含む広範囲の学術分野だけでなく，文学やジャーナリズムなどの文書全般でも需要が高い」と考えられる漢字・カタカナ文字列の簡易つづり校正法を公開する．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，生物学系の日本語文書では，すでに高崎(2018)が公開したアルファベット文字列の簡易点検法と併用することで，校正作業の苦労を大きく軽減するとともに，校正の精度をさらに上げることができる．

現代日本語の文書では，漢字・カタカナ・ひらがな・欧文の４種の文字が混在する．それらのうち，難解な専門用語を含む漢字やカタカナの文字列は，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に劣らず，熟練の校正者にとっても難物である．本稿で紹介する試みは，これら漢字カタカナ文字列の校正作業を簡易化する．21世紀には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になった．公開と同時に個々の論文全体が世界中から閲覧可能な状態では，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，公開前の入念な校正作業が従来以上に求められる．

新聞・雑誌などの大規模な商業出版では，このような電子テキストの処理には専用の校正プログラムを組んで対処しているのだろう．一方，零細な出版社や，ボランティア作業に頼る弱小学会等の編集・校正業務では，慢性的な人手不足を抱え，旧態依然の校正作業を続け，校正不足のままの出版物をやむなく出さざるをえないところもまだ多い．他方，現在では法令的に義務化されている博士論文のウェブ公開で，回避しえたはずの誤植を含む不備が残っていれば，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る時代である．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校正作業に求められる．高崎(2018)の試み同様，本稿も「車輪の再発見にすぎない」かもしれない．しかし，いまだに「車輪」が認知されず，普及していない文化圏や研究者コミュニティにあっては，無意味ではないだろう．

本稿で紹介するスクリプトを動かすには，POSIX準拠のUnix互換ターミナルが使えるコンピュータを使う．ここまで読んで，本稿を投げ出したくなった読者は，もう少し先まで読んで欲しい．たとえば点検すべきpdf電子テキストがあったとしよう．それを入力として得られる主な出力の種類の概説を一覧にしておくので，投げ出すのは，それらにざっと目を通してからにしてもよいだろう．POSIX-Unix互換のシステムでコマンドモードやスクリプト類を使える人に，本稿を見せて，自分の「虎の子」原稿の電子テキスト解析を代行してもらうという使い方もあるのだから．

(1)`text.txt`: pdfプリントアウトの各行頭に行番号を添えて出力したプレーンテキスト．

(2)`freq_data.txt`: アルファベットとひらがな以外の文字列について，出現頻度を添えた一覧．文字列順にソートしてあり，似た文字列が並ぶので相互点検に便利．

(3)`segmented.txt`: 日本語を読むときに適当な長さ(長すぎず，短すぎず)に分節化し，ソートして文字列の重複を除去した一覧．アルファベット文字列は除去してある．重複が除かれて，ソートによって似たひらがな付きの漢字カタカナ文字列が並ぶので，単純なテキストの流し読みよりも，誤字・脱字・余字などの検出に便利．

(4)`strings2check.txt`: 参照辞書(作成法解説は本稿のIII・IV)にない２文字長以上の漢字カタカナ文字列の一覧．ソートによって似た文字列が並ぶので，誤字・脱字・余字などの点検に便利．１回だけ出現の文字列には「#」印が付けてある．

(5)`lines2check.txt`: 参照辞書にない漢字カタカナ文字列のうち，１回出現の文字列((4)にある#付きの文字列)を含む点検すべき行の一覧．行番

号は(１)のtext.txtの行番号.

(６)`check_strings2add.txt`: あらたに辞書に追加してよいかもしれない文字列の候補一覧(個別の点検が必要). 参照辞書になくテキストに２回以上出現している２文字以上の漢字カタカナ文字列(すなわち(４)で#印が付かなかった文字列). ２回以上同じ並びで出現しているので, 正しい文字列である可能性が高い.

(７)上記の(３)以外のカタカナ版. とくに生物和名の点検に有用.

(８)欧文つづり点検用に作られた高崎(2018)の改良版のスクリプトによる出力.

本スクリプト類は, Microsoft Windows 10なら,「Bash on Ubuntu on Windows」などで, Linuxを導入すれば使える.(たとえば, 日経Linux2018年11月号特別付録「Windows版Linuxの全てがわかる本」を参照されたい.)また, OS X以降のMacintoshユーザーは, そのターミナル(BSD派生のPOSIX準拠Unix互換)が使える. ただし, 本稿に紹介するスクリプトが完全に動くことが期待されうるのは, UbuntuやLinuxmintなどDebian GNU/Linux系のものである.「ハードウェアとソフトウェアの準備」の詳細やコマンドの表記法については, 高崎(2018)のIIにある解説を参照されたい. 参考書としては, Linuxの操作やターミナルの使い方の入門には奈佐原(2016)と中島(2018)を, シェルスクリプトの上級参考書には上田(2016)をあげておく.

本稿のIIでは, 解析に先立って行う文字コード・改行コードの変換や, 分かち書きされていない日本語文書から, 点検する漢字カタカナ文字列を抽出する手法を述べる. IIIでは, 英文の場合とは異なり, 日本語の学術文書に現実的に使える既存のスペルチェッカに, 無償公開されたものがない状態で,「漢字カタカナつづりチェッカ」を実現する手法を解説する. さらに応用と発展をはかるIVでは, テキストマイニングによる「一般用語のつづり辞書」さらに「専門用語等のカスタムつづり辞書」の作成法と, スクリプトとつづり辞書の組み合わせを分野ごとにチューニングする手法を解説する. まとめとしてV「おわりに」で,「現在の日本語の印刷物等の校正もれの現状を憂い, 本稿のようなシェルスクリプトの利用が事態を打開しうる」ことを, ごく短く述べる.

以下, 高崎(2018)と同様, Unixの素人の素人による素人のための手引である. Unixの玄人にはくどいばかりだろうが,「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, 玄人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, II(1.03)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, 本文は読まずにスクリプトを直接解読されたい. 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, スクリプトの解説部分には通し番号を付けてある.

紹介するシェルスクリプト類(**`boldface`**表記)や応用例は, 本稿のテキストpdfと同じURL(http://www1.ous.ac.jp/garden/kenkyuhou・・・)から, zipファイル(foolproof2_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. 本稿および高崎(2018), さらにそれら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.

II. 漢字カタカナ文字列の抽出処理

日本語のつづりチェックが, おそらくマスコミ・大手出版業界以外では, なかなか行われていない原因は, 使い勝手のよいつづりチェッカが存在しないことにあるだろう. その背景には,「PC用ワープロのスペルチェッカは, 文章の正書法が分かち書きを基本とする英語用に作られたものの焼き直し」ということがある. 現在では, MeCabなどの形態素解析ソフトを使えば, 分かち書きは自動的に出力されることにはなっているが, 校正用途に使うには完成度がもうひとつである. したがって, 使い勝手のよいつづりチェッカが, 日本語では実現できていない.

高崎(2018)は, 英文であっても専門分野によっては, 参照するつづり辞書を準備しなければスペルチェッカが容易には使えない状況を憂い, とくに専門用語や学名を多用する広義の生物学分野では障害が著しいことから, 代替法を工夫した. 本稿では日本語を扱うつづりチェッカの代替法を紹介する. まずは, 分かち書きの代わりとなる漢字カタカナ文字列の抽出法を解説する. アルファベット文字列だけを抽出する方法(高崎 2018)の基本は「アルファベット以外の文字を空白文字に置き換える」手法だった. 漢字カタカナ文字列の抽出も, 大雑把には「漢字やカタカナ以外の文字を空白文字に置き換える」ことで実現できる.

(1.01)点検したいプレーンテキスト形式の電子テキストが, `TEST.txt`であるとする. これから使うディレクトリとしてkatchproofをhomeあるいはDesktopディレクトリ内に作って, その中に置く.(ディレクトリの扱いについての簡単な解説は高崎(2018)にあるほか, 奈佐原(2016)などを参照されたい.)(pdfや, docx, doc形式等からプレーンテキスト[.txt]に変換する方法については, ウェブ検索で調べるか, 高崎(2018)を参照されたい.)

まずは，文字化け等を未然に防ぐ念のための準備作業として，文字コードを「UTF-8」，改行コードを「LF」にそろえる．このコード形式が，今ではDebian GNU/Linux系のPOSIX準拠のUnix互換ターミナルではデファクト・スタンダードとなっているからである．このためにはnkfを使う．(nkfが入っていない場合，次のコマンドを実行すると，nkfが入っていない旨のメッセージが出て，インストール法の指示も出るので，指示にしたがってインストールした後，再度，同じことを試みる．)これらの作業は，ターミナルでkatchproofディレクトリを開いて行う．

```
$ nkf -wLu TEST.txt [Enter]
```

ターミナル画面上にテキストが流れる．このままでは，せっかくの変換結果がターミナル画面に見えるだけで保存されないので，「>」(リダイレクト)を使った次のコマンドで，TEST0.txtに保存しておく．

```
$ nkf -wLu TEST.txt > TEST0.txt [Enter]
```

あるいはパイプ「|」と「tee」コマンドを使って，画面上で目的とするテキスト変換ができていることを確認しながら結果を保存するには，次を実行する．(「|」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」のような働きをし，分岐の片方が画面出力になる．)

```
$ nkf -wLu TEST.txt | tee TEST0.txt
[Enter]
```

さらに「塵も積もれば山となる」時間節約のために(上田[2016]を参照)，次のようにするとよい．

```
$ cat TEST.txt | tr \\r \\n |
nkf -wLu | tee TEST0.txt [Enter]
```

いずれのやり方であっても，文字コードUTF-8，改行コードLFとなった変換結果が，TEST0.txtに保存される．なお，TEST.txtが最初から「UTF-8+LF」コードで書かれていても，たいした害はない(半角の仮名文字類は全角に変換される)．(念のため，Macの文書から処理を出発すると，異なる改行コードが残って不具合を生じることがあるので，その変換処理(tr \\r \\n)もここには挿入してある．)

(1.02)「分かち書きされていない日本語文書から，点検する漢字カタカナ文字列を抽出する」には，どうすればよいか．テキストから平仮名を消すと，「語順は日本語のままのカタカナ混じりのチンプンカンプン漢文モドキ」になる．実際にやってみよう．この段落(1.02)をTEST.txtとして，１行のコマンドスクリプトで実現するには，次を実行すればよい．

```
$ cat TEST.txt | tr \\r \\n |
nkf -wLu | sed 's/[ぁ-ん]//g' [Enter]
```

ターミナル画面に出力される結果は次のようになる．

「分書日本語文書,点検漢字カタカナ文字列抽出」,.テキスト平仮名消,「語順日本語カタカナ混チンプンカンプン漢文モドキ」.実際.段落(1.02)TEST.txt,１行コマンドスクリプト実現,次実行.

これでは，別々に抽出したかった漢字カタカナ文字列が連結して，不都合だ．スクリプトを解読すれば目的を達するように対処できる．「sed 's/[ぁ-ん]//g'」は，ひらがな(日本語UTF-8コード表で「ぁ」から「ん」までを一括する正規表現は[ぁ-ん])をストリームエディタ(sed)で，全て(g，global)「空」で置換(s，substitute)したのだが，「空」の代わりに「＝」で置き換えるには，次を実行すればよい．

```
$ cat TEST.txt | tr \\r \\n |
nkf -wLu | sed 's/[ぁ-ん]/＝/g' [Enter]
```

結果は次のようになる．

「分＝＝書＝＝＝＝＝＝＝日本語文書＝＝,点検＝＝漢字カタカナ文字列＝抽出＝＝」＝＝,＝＝＝＝＝＝＝＝.テキスト＝＝平仮名＝消＝＝,「語順＝日本語＝＝＝＝カタカナ混＝＝＝チンプンカンプン漢文モドキ」＝＝＝.実際＝＝＝＝＝＝＝＝.＝＝段落(1.02)＝TEST.txt＝＝＝＝,1行＝コマンドスクリプト＝実現＝＝＝＝,次＝実行＝＝＝＝＝.

同時に任意の算用数字やアルファベットおよび記号類(まとめて[!-~])，句読点，括弧などの全角記号類([]内には，全角記号類が個別に列記してあり，今後も不都合が見つかった記号類については，同様に個別に追加可能)にも同様の置換を施すとすれば，次を実行する．

```
$ cat TEST.txt | tr \\r \\n |
nkf -wLu | sed 's/[ぁ-ん]/＝/g' | sed 's/
[!-~]/＝/g' | sed 's/[,..:;,「」『』()[]｜]/
＝/g' [Enter]
```

結果は次のようになる．

＝分＝＝書＝＝＝＝＝＝日本語文書＝＝＝点検＝＝漢字カタカナ文字列＝抽出＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝テキスト＝＝平仮名＝消＝＝＝＝語順＝日本語＝＝＝＝カタカナ混＝＝＝漢文モドキ＝＝＝＝＝実際＝＝＝＝＝＝＝＝＝＝段落＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝行＝コマンドスクリプト＝実現＝＝＝＝＝次＝実行＝＝＝＝＝＝

(1.03)ここまでで漢字カタカナ文字列の抽出・一覧表化も実現間近となった．次は「＝」を改行「LF(\n)」に変換するのだが，sedは改行の扱いが苦手である．代わりに一文字置換のtrを，「tr ＝ \\n」という形で使う．これをパイプ「|」で繋いで，次を実行すればよい．

```
$ cat TEST.txt | tr \\r \\n |
nkf -wLu | sed 's/[ぁ-ん]/＝/g' | sed 's/
[!-~]/＝/g' | sed 's/[,..:;,「」『』()［］]/
＝/g' | tr ＝ \\n [Enter]
```

さらに重複を取り除くには，sortとuniqを「|」で繋いで追加する．

```
$ cat TEST.txt | tr \\r \\n |
nkf -wLu | sed 's/[ぁ-ん]/＝/g' | sed 's/
[!-~]/＝/g' | sed 's/[,..:;,「」『』()［］]/
＝/g' | tr ＝ \\n | sort | uniq [Enter]
```

改行を「,」で書き換えた出力結果は，次のようになる．

��,**��**漢文,**��**混,**��**文字列,,漢字,語順,行,次,実現,実行,実際,書,消,段落,抽出,点検,日本語,日本語文書,分,平仮名

「**��**」という文字化けした部分が見られる．(システムによってはターミナルでは**�**が見えないこともある．)さらに細かくチェックするとカタカナ文字列が消えたところなどで不具合を生じているようだ．UTF-8文字コードでは３バイト長になる「＝」の代わりに１バイト長ですむタブ「\t」で置換してみよう．同時に，タブに置換する記号に「＝」も追加しておく．

```
$ cat TEST.txt | tr \\r \\n |
nkf -wLu | sed 's/[ぁ-ん]/\t/g' | sed
's/[!-~]/\t/g' | sed 's/[,..:;,「」『』()[]
＝]/\t/g' | tr \\t \\n | sort | uniq
[Enter]
```

これで次のように，期待した通りの結果が得られた．

カタカナ混,コマンドスクリプト,チンプンカンプン漢文モドキ,テキスト,漢字カタカナ文字列,語順,行,次,実現,実行,実際,書,消,段落,抽出,点検,日本語,日本語文書,分,平仮名

そこで，最後のuniqコマンドにオプション「-c」を付けて，出現頻度情報を付加した出力を得て，freq_katch1_TEST.txtに保存する．次を実行すればよい．

```
$ cat TEST.txt |
tr \\r \\n | nkf -wLu | sed 's/[ぁ-
ん]/\t/g' | sed 's/[!-~]/\t/g' | sed 's/
[,..:;,「」『』()[]＝]/\t/g' | tr \\t \\n |
sort | uniq -c | tee freq_katch1_TEST.
txt [Enter]
```

出力の一行目は，空行の頻度になっている．この行は校正作業には不要なので，空行を削除するコマンド「sed '/^$/d'」をsortの前に挿入する．また頻出しそうな記号類も余分目に正規表現[]内に追加しておく．

```
$ cat TEST.txt | tr \\r \\n |
nkf -wLu | sed 's/[ぁ-ん]/\t/g' | sed
's/[!-~]/\t/g' | sed 's/[,..:;,「」『』()[]
·■◇◆●★《》〜〔〕【】×＝〈〉ヽ〃ゞ-]/\t/g' | tr \\t
\\n | sed '/^$/d' | sort | uniq -c | tee
freq_katch1_TEST.txt [Enter]
```

この形で，これから後，別のTEST.txtの点検に使える．

(1.04)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．そこで(1.02)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行をエディタで書いて，**katch1.sh**としてkatchproofディレクトリに保存する．

```
  #!/bin/bash
  ##katch1.sh:
  ##katakana kanji character string checker
  cat $1.txt | tr \\r \\n | nkf -wLu | sed 's/[ぁ-ん]/\t/g' |
sed 's/[!-~]/\t/g' | sed 's/[,..:;,「」『』()[]・■◇◆●★《》〜〔〕【】×＝〈〉〝〟ゞ-]/\t/g' | tr \\t \\n | sed '/^$/d' | sort | uniq -c | tee freq_katch1_$1.txt
```

スクリプトの本体である左端が#で始まらない行は，(1.02)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．

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 katch1.sh TEST [Enter]
```

(1.05)短い文章であれば，直上の出力結果freq_katch1_TEST.txtを目視で点検するのは，たいした作業ではない．しかし，本稿のIIくらいの長さの文章になってくると，これが苦痛となり始める．直上の段落までの本稿草稿(2018年4月上旬)を新たにTEST.txtとして，(1.03)の最後のコマンドを実行してみると，500強の文字列が抽出され，スクリーン上でスクロールしての点検ではミスを誘う．詰めた印字のプリントアウトを作っても，わずらわしいばかりだ．

ひたすらテキストを読む従来型の校正で，見逃しが起きやすいのは，活用語の送り仮名(ワープロの種類よっては不具合を指摘してくれる)などのひらがなとの組み合わせで読まれる単漢字1文字ではないだろう．また，動物名の「ウ」とか「カ」などカタカナ1文字も目立つので，校正もれは起こりにくい．したがって，2文字以上からなる文字列の頻度付きのチェックリストを追加で作り，単漢字や1文字のカタカナは，ワープロや読み合わせでの従来型の校正にまかせて，2文字以上からなる文字列だけに焦点を絞る．そうすることで校正作業を効率化する．これには，スクリプト言語AWKで書かれた短いスクリプトを使う．(AWKについての解説は，Aho et al. (1989)や中島ら(2015)などの教科書・参考書類にゆずる．)次のコマンドを実行する．ターミナル画面で出力が確認できて，freq_katch2_TEST.txtに結果が保存される．

```
  $ cat freq_katch1_TEST.txt | awk 'length($2)>1 {print $0}' | tee freq_katch2_TEST.txt [Enter]
```

しかし，これでも400を超える文字列を点検しなければならない．さらに出現頻度1回のものは，頻度2回以上のものよりも誤っている可能性が高い．また頻度2回以上のものは，問題ないことが多い．そこで，頻度が1回の文字列のチェックリストも同時に作る．

```
  $ cat freq_katch2_TEST.txt | awk '$1<2 {print $2}' | tee katch21_TEST.txt [Enter]
```

これで2文字以上からなり，しかも出現頻度は1回だけの文字列300程度を，今回は点検するだけですむようになった．要点検の文字列数を，6割未満に減らすことに成功したのである．

これまでのスクリプトをkatch1.shを含んだ形で，今後も使える汎用スクリプトとしてファイルに残そう．次を，**katch2.sh**としてkatchproofディレクトリに保存する．これはkatch1.shも保存されている同じディレクトリの中で使う．

```
  #!/bin/bash
  ##katch2.sh:
  ##better katakana kanji character string checker
  bash katch1.sh $1
  cat freq_katch1_$1.txt | awk 'length($2)>1 {print $0}' | tee freq_katch2_$1.txt
  cat freq_katch2_$1.txt | awk '{print $2}' | tee katch20_$1.txt
  cat freq_katch2_$1.txt | awk '$1>=2 {print $2}' | tee katch22_$1.txt
  cat freq_katch2_$1.txt | awk '$1<2 {print $2}' | tee katch21_$1.txt
```

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 katch2.sh TEST [Enter]
```

III．つづりチェッカの作成

(2.01)本稿のIIまで程度の原稿であれば，1回の点検でkatch21_TEST.txtにリストアップされた文字列がTEST.txt中の「誤植ではないか」丹念に点検し，それにしたがってTEST.txtを修正するだけ

で，漢字カタカナ文字列の校正は済むだろう．しかし本稿には，まだIII～Vや謝辞，引用文献のリストもある．それらもいずれ点検しなければならない．修正版のTEST.txtに再度katch2.shを実行して得られたkatch21_TEST.txtに並ぶ文字列のうち，大半の文字列は今後も出現する可能性が高い．一般性が低く，おそらくもう稀にしか出てこないと判断される文字列をkatch21_TEST.txtから削除して，okstring.txtと名付けて保存しよう．これをプロトタイプのつづり辞書として参照し，つづりチェッカとして使えるスクリプトを作成してみようというわけだ．少なくとも，本稿の残りの校正・点検には使えるだろう．

このような「つづりチェッカとして使えるスクリプト」として，高崎(2018)はpp. 24–27で別法を紹介した．しかし，最終校正のときにdiffコマンドを使う方法を思いつき，p. 24に「(校正時補記：diffを使う別法も可能)」と記しておいた．diffは，本来の用途としてはプログラムやマニュアル等，情報処理のテキストファイルの複数版での改版履歴を点検するために，バージョン間の差異を検出するユーティリティだ．textversion1，textversion2がプレーンテキストであるとすれば，「diff textversion1 textversion2」を実行することで，行単位で両者の差異が検出・出力される．この仕組みを使ったスペルチェッカのプロトタイプは，高崎(2018)のfoolproof_scripts.zipに，**spellbydiff.sh**として入れられて，最終版のスクリプト集では部品的に使うスクリプトの１つとなった．今回，本文中に再録しておく．

```
  #!/bin/bash
  ##spellbydiff.sh
  ##spellchecker using command "diff"
  ##usage: bash spellbydiff.sh CORRECT_
SPELL.txt WORDLIST2CHECK.txt
  ##where CORRECT_SPELL and
WORDLIST2CHECK correspond to $1 and $2
below
  diff -u $1.txt $2.txt | grep "+" | sed
's/\+//g' | tail -n +4 | grep -v @ | tee
$2_words2check.txt
```

参照するつづり辞書CORRECT_SPELL.txtにはokstring.txtを使うとして，WORDLIST2CHECK.txtには何を準備すればよいか．スクリプトkatch2.shを注意深く解読した読者は気づいていたかもしれないが，katch2.sh TESTの出力の１つに，そのテストに使えそうなソート済みの単純な文字列リストkatch20_TEST.txtが，何の説明もなしに作られていたのである．freq_katch2_TEST.txtから頻度情報を省いたリストだ．とりあえず，これを使ってみよう．

```
  $ bash spellbydiff.sh okstring katch20_
TEST [Enter]
```

この試行で得られる出力katch20_TEST_words2check.txtにリストアップされる文字列は，これまた何の説明もなしに作られていたkatch22_TEST.txtにリストアップされた文字列(頻度２回以上出現)にくわえて，最初のkatch21_TEST.txtからokstring.txtを作るときに削除された文字列が合わさって構成されている．すなわち，今回のTEST.txtから抽出された文字列からつづり辞書として使い回しのできる文字列を，さらにokstring.txtに補充するには，katch22_TEST.txtから候補を選べばよい．不要な文字列を削除したものをstrings2add.txtと名付けて保存しておく．

(2.02)次にstrings2add.txtにリストアップされた文字列を追加して，okstring.txtを更新するスクリプトを作る．これは，高崎(2018，p. 16)にあるupdateokspell.shを改変すれば，簡単に作れる．

```
  #!/bin/bash
  ##updateokstring.sh
  ##update okstring.txt with new
strings2add.txt for string checker
  cat okstring.txt > okstring_bak.txt
  cat strings2add.txt >> okstring.txt
  sort okstring.txt | uniq |
sed '/^$/d' | tee okstring.txt
```

「bash updateokstring.sh」を実行して，実際に更新する．その後「bash spellbydiff.sh okstring katch20_TEST」を再度実行して，katch20_TEST_words2check.txtを確認すると，30文字列程度を点検するだけで済むことが分かる．すなわち，(1.04)の要点検500文字列から，その６%程度を点検するだけでよくなった．省力化の効果は著しい．本稿草稿のような場合，このような作業を繰り返すことで，草稿が完成する頃には，漢字カタカナの文字列については，ほぼ誤植のない原稿を準備できる．

(2.03)文字列抽出とつづりチェッカを連結したスクリプトがあれば，ときどき参照つづり辞書を更新するだけで，TEST.txtを入力して一気に点検すべ

き文字列候補の出力を得ることができるだろう．ついでに「文字列抽出」のkatch1.shに相当する部分を読みやすく書き直そう．さらに，文字列末の接尾辞的な「等」などを無視できる細工を組み込む．（この部分は，今後もフィルター的に「regexをreplaceで全置換」する“sed 's/regex/replace/g'”などのパターンをパイプ「|」で繋ぐ拡張ができる．）加えて，「つづりチェッカ」spellbydiff.shも組み込んで，出力ファイル名が分かりやすくなるように細工しておく．

```
  #!/bin/bash
  ##katch3.sh:
  ##superb katakana kanji character string checker
  #bash katch1.sh $1
  cat $1.txt | tr \\r \\n | nkf -wLu | sed 's/[ぁ-ん]/\t/g' |
sed 's/[!-~]/\t/g' |
sed 's/[ 　,..:;,「」『』()［］·■◇◆●★《》～〔〕【】×＝〈〉〝〞ゞ-]/\t/g' |
sed 's/等\t/\t/g' | tr \\t \\n | sed '/^$/d' |
sort | uniq -c | tee freq_katch1_$1.txt
  cat freq_katch1_$1.txt | gawk 'length($2)>1 {print $1, $2}' | tee freq_katch2_$1.txt
  cat freq_katch2_$1.txt | gawk '{print $2}' | tee katch20_$1.txt
  cat freq_katch2_$1.txt | gawk '$1>=2 {print $2}' | tee katch22_$1.txt
  bash spellbydiff.sh okstring katch22_$1
  cat katch22_$1_words2check.txt > katch22strings2check.tmp
  cat freq_katch2_$1.txt | gawk '$1<2 {print $2}' | tee katch21_$1.txt
  bash spellbydiff.sh okstring katch21_$1
  cat katch21_$1_words2check.txt | gawk '{print $0 "#"}' >> katch22strings2check.tmp
  sort katch22strings2check.tmp | tee $1_strings2check.txt
  rm *.tmp
  rm *words2check.txt
```

次のコマンドを実行すれば，目的のTEST_strings2check.txtが得られる．（これまでと同様，入力ファイル名のTEST.txtのうち，拡張子.txtは省略する仕様になっている．）

```
$ bash katch3.sh TEST [Enter]
```

IV．一般用語および専門用語等のカスタムつづり辞書の作成

直上に作り上げた**katch3check.sh**を用いれば，okstring.txtの代わりに，あるいは併用して汎用のつづり辞書を使ったり，複数の専門つづり辞書を追加・併用することで，さらに便利になるだろう．それらの辞書をどのようにして作ればよいか，次に述べる．

(3. 01)まずは汎用性の高い一般用語つづり辞書を確保しよう．『GENE95辞書』という電子英和辞書があって，インターネット経由でダウンロードできる(http://www.namazu.org/~tsuchiya/sdic/data/gene.html；本段落以降の他URL同様，2018年３月31日アクセス)．この辞書を入力としてkatch2.shを実行してみよう．これで容易に得られる出力katch20_gene.txtは，４万語を超える漢字カタカナ文字列つづり辞書となる．作者が「現代事典的な」辞書としているので，多くの一般的な漢字カタカナ文字列がカバーされていると期待できる．『GENE95辞書』由来の２文字長以上の漢字カタカナ文字列辞書という意味で，仮にGENE95_katch2dict.txtと名前を付け替えておく．また『EDICT』(http://www.edrdg.org/jmdict/edict.html)という電子英和辞書もある．それも同様に処理して，８万語を超える漢字カタカナ文字列辞書，仮称EDICT_katch2dict.txtが得られる．これらの一般用語つづり辞書の和集合をとって，**EGENE_katch2dict.txt**とした．重複がとれて，11万語超の辞書となっている．

さらに翻訳用辞書等が公開されているサイトから(e.g. http://www.vector.co.jp/vpack/filearea/data/writing/dic/translat)自分の分野に使えそうなものを選べば，同様に専門つづり辞書を作ることができる．また例えば，高崎(2018)のpp. 22–23に紹介したインターネット上の電子辞書類を渉猟すればよい．主だったものを再度リストしておく．DBCLSメタ用語集(http://lifesciencedb.jp/lsdb.cgi?gg=dic)にある「メタ用語集 学名編」や「メタ用語集 学術用語編生物学」，さらに医学・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/)などがある．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)もある．元にする辞書テキストは，**katch2.sh**を用いて漢字カタカナ文字列だけを抽出するので，和英でも英和でもかまわない．すなわち元辞書が和文の誤植のなさに関して信頼できるものでありさえすればよい．

そのほか，日本の地名の多くを網羅しているは

ずの郵便番号簿をもとにしたx-ken-all(https://github.com/sanemat/x-ken-all)から地名の漢字カタカナ文字列を抽出して，**JPLACES_katch2dict.txt**(７万７千超の地名文字列)を作成した．今回の漢字カタカナ文字列チェッカを，真っ先に試用するのは，日本蝶類学会誌*Butterflies*や同学会*Newsletter*，さらに本誌*Naturalistae*の校正用pdfである．*Butterflies*での用途には，私家版の和文英訳翻訳補助ソフトTrex(高崎[2017]参照)用に蓄積してきた和英対訳表(元になったのは，五十嵐・福田[2000]や松香[2001]，五十嵐・原田[2015]や多くの同僚や教え子ほかの和文原稿電子テキスト)に加えて，本田・加藤(2005)の索引をOCRで取り込み，目視校正のあと，katch2.shで処理した出力を得て，**TREX_katch2dict.txt**(９万９千文字列超)と名付けたカスタム辞書も作った．IVのここまでの解説のついでに作ったこれらの辞書類*_katch2dict.txtは，foolproof2_scripts.zipの中に，漢字カタカナ文字列つづり辞書類例として入れてある．

(3.02)手持ちの辞書類は使わなければ，もったいない．統合した参照辞書がなければ，利用に不都合なので，それをrefdict.txtとしてokstring.txtや手持ちの辞書類から作成する．そのスクリプト**poolrefdict.sh**を，次のように書いて，先に動かす．

```
  #!/bin/bash
  ##poolrefdict.sh:
  ##to make refdict.txt from okstring.
txt and *_katch2dict.txt
  #bash poolrefdict.sh
  cat okstring.txt > refdict.tmp
  #below delete or add # as necessary
  cat TREX_katch2dict.txt >> refdict.tmp
  ##(katakana/Chinese character strings
found in Trex translation tables)
  cat JPLACES_katch2dict.txt >> refdict.
tmp #(Japanese place names)
  cat  TAXA_katch2dict.txt  >>  refdict.
tmp #(biological taxon names)
  cat EGENE_katch2dict.txt >> refdict.
tmp #(general terms)
  #cat ANATOMY_katch2dict.txt >>
refdict.tmp #(anatomical terms)
  #cat  MED_katch2dict.txt  >>  refdict.
tmp #(medical terms)
  #cat *_katch2dict.txt >> refdict.tmp
#(to add any custom dictionaries)
  sort refdict.tmp | uniq |
sed '/^$/d' | tee refdict.txt
  rm *.tmp
```

上記の例では，参照辞書refdict.txtに取り込むつづり辞書はokstring.txtとTREX_katch2dict.txtなどとしてある．これが本稿執筆者の普段の使用では，十分と考えられる設定である．使用者は必要に応じて，#を付けたり，外したり，さらに各自の専門辞書を補足したりすればよい．末尾行から２行目では重複を削除している．この設定で，20万種以上の固有の正しい漢字カタカナ文字列と比較して，そこにないものを列挙できるようになる．(2018年９月上旬現在の試用段階では，最大設定の汎用の用途にはokstring.txtと全*_katch2dict.txtの和集合となるrefdict.txtに，26万種以上の漢字カタカナ文字列が入った状態で文書の点検が行われている．)

(3.03)次にkatch3.shのスクリプト中にあるokstringを参照辞書refdict.txtの略記であるrefdictに書き換えた，**katch4.sh**を作る．ついでにkatch3.shの文字列末の接尾辞的な「等」などを無視できる細工に，文字列末での同様パターンとして「氏」「産」「寄」「著」「以後」「以降」なども無視できる細工を追加したり，実際に(3.04)の**katch5.sh**を動かして，不具合を修正しておく．(#でコメントアウトした行で不具合を生じた．)

```
  #!/bin/bash
  ##katch4.sh:
  ##superb katakana kanji character
string checker II
  #bash katch4.sh $1
  cat $1.txt | tr \\r \\n | nkf -wLu |
   sed 's/及び/および/g' | sed 's/並び/な
らび/g' |
   sed 's/且つ/かつ/g' | sed 's/[ぁ-ん]/\
t/g' |
  #sed 's/[①-⑳]/\t/g' |
sed 's/[!-~]/\t/g' |
   sed 's/[ ,..:;,「」『』()[]･■◇◆●★《》～〔〕【
】×〓〈〉♂♀→←〝〟〻-]/\t/g' |
   sed 's/等\t/\t/g' |
sed 's/氏\t/\t/g' |
   sed 's/産\t/\t/g' |
sed 's/寄\t/\t/g' |
   sed 's/著\t/\t/g' |
  ######################################
  ######細工の大半はこのリストからは省略######
  ######foolproof2_scripts.zipの#######
  ##########katch4.shには残す##########
  ######################################
   sed '/\t故..*/ s/故/\t/g' |
  # sed '/..*市/ s/市/\t/g' |
   sed '/..*本島/ s/本島/\t本島/g' |
   tr \\t \\n | sed '/^$/d' | sort |
   uniq -c | tee freq_katch1_$1.txt
  cat freq_katch1_$1.txt | gawk
'length($2)>1 {print $1, $2}' |
    tee freq_katch2_$1.txt
  cat freq_katch2_$1.txt | gawk '{print
$2}' |
    sort | uniq | tee katch20_$1.txt
  cat freq_katch2_$1.txt | gawk '$1>=2
{print $2}' |
   sort | uniq | tee katch22_$1.txt
  bash spellbydiff.sh refdict katch22_$1
  cat katch22_$1_words2check.txt >
katch22strings2check.tmp
  cat freq_katch2_$1.txt | gawk '$1<2
{print $2}' |
   sort | uniq | tee katch21_$1.txt
  bash spellbydiff.sh refdict katch21_$1
  cat katch21_$1_words2check.txt |
   gawk '{print $0 "#"}' >>
katch22strings2check.tmp
  sort katch22strings2check.tmp | uniq |
   tee $1_strings2check.txt
  rm *.tmp
  rm *words2check.txt
```

(3.04)直上の(3.02)，(3.03)にある２つのスクリプトは連続して実行するので，それらをまとめて操作する**katch5.sh**を作る．

```
  #!/bin/bash
  ##katch5.sh:
  ##superb katakana kanji character
string checker III
  bash poolrefdict.sh
  bash katch4.sh $1
```

実際に動かしてみよう．

```
  $ bash katch5.sh TEST [Enter]
```

TEST_strings2check.txtに，「点検したほうがよい」文字列がリストアップされる．すなわち，katch5.shを使うことで，UTF-8のプレーンテキストになっている文書は，２文字以上の漢字カタカナ文字列については，本稿の点検はもちろんのこと，少なくとも日本蝶類学会誌*Butterflies*の１号分程度の校正作業には，十分に使えることが分かった．

(3.05)印刷用の版下pdf文書での校正作業で，いちいち文字コードUTF-8のプレーンテキストに変換するのは煩わしい．また多くの場合，原稿はMS Wordなどのワープロで作成されて，docやdocxのファイルになっている．そこで，高崎(2018)では付録のfoolproof_scripts.zipの中に，pdf，docx，doc文書を直接入力として，「アルファベット文字列」用の簡易つづり解析結果を出力するスクリプト類pdfprfrd.sh，docxprfrd.sh，docprfrd.shを収録した．

2018年９月上旬現在，試用段階から１年以上が経過したが，圧倒的に使用頻度が高いのは，印刷用pdf文書を入力とするpdfprfrd.shであった．プリントアウトの形態を模した，改行コードの入ったプレーンテキストを作成し，行番号付きの出力も生成して，固有文字列の出現頻度などに加えて「どこに『点検したほうがよい』アルファベット文字列が現れるか」を行番号とともに示す出力(本稿のIの(５)lines2check.txtに相当)も生成する．

現在では，doc文書もdocx文書も，まずエクスポートして生成したpdf文書あるいはプレーンテキストを抽出したtxt文書を入力に使っている．そのほうが圧倒的に使い勝手がよいのだ．今回は，docやdocx文書を入力とするスクリプト類は割愛して，pdf文書を入力として「２文字以上の漢字カタカナ文字列」

用の簡易つづり解析結果を出力する**katchprfrd.sh**等を作成し，foolproof2_scripts.zipの中に収録してある．

本稿中に詳しい説明のないものについては，foolproof2_scripts.zipのreadme1st.txtに簡単な解説がある．とくに「２文字以上の漢字カタカナ文字列」用の簡易つづり結果（本稿のIの（３）参照）を出力するkatchprfrd.shや，生物名などカタカナ表記の文字列だけを点検する**katakanaprfrd.sh**は，和文論文の校正作業に有用であろう．

さらに欧文つづりの混じるものについては，**alphaprfrd.sh**（pdfprfrd.sh［高崎 2018］の改良版としてfoolproof2_scripts.zipに収録）を併用するとよい．alphaprfrd.shでは，pdfprfrd.shでは考慮されていない「欧文のハイフン“-”で終わる行末についてハイフネーションの位置が不適切ではないことを目視点検する」ための出力をlines2check.txtの末尾に追加しているほか，ドイツ語やフランス語など，特殊なアクセント記号付きの文字が入った人名や単語だけのリストも出力するようになっている．

ついでに３文字以上の「ひらがな文字列」用のスクリプトも作ってはみた．「ひらがな以外の文字を空白文字に置き換える」ことから出発した試作である．しかし，MS Wordなどのワープロでは，活用語の送り仮名などの不具合は指摘できるようになっているので，通常はalphaprfrd.sh（高難易度「欧文つづり」用）とkatchprfrd.sh（「漢字カタカナ文字列」用），katakanaprfrd.sh（「カタカナ文字列」用）の併用で十分だろう．

V．おわりに

本稿の準備中，引用した文献以外にも多数の文書・文字処理関係の参考書や雑誌記事に執筆者は目を通した．多くは弱小でもない出版社から21世紀以降に出た商業出版物である．驚いたことに，高崎(2018)に述べた方法に本稿の方法を併用すれば，容易に検出できたはずの欧文・和文文字列の誤植が，それらにも少なからず見られた．「文書・文字処理の専門家でさえ，自分たちの文書の校正・点検に無頓着なまま，本来，自在に使えるはずの専門知識と技術を十分には使っていない」とも判断せざるをえない現状は，残念な限りである．

現代生物学を大雑把に鳥瞰すれば，20世紀後半から現在まで，DNA関連の情報が爆発的に増大し，コンピュータ・情報処理技術の利用と組み合わさってバイオインフォマティクスを生むに至ったことが見てとれる．単純化して考えれば，A，G，C，Tの４文字のアルファベットからなる文字列テキスト=「生物の設計を記した暗号文書」を，さまざまにこねくりまわして解読する学際領域である．そこに駆使されているテキスト解析処理の技法の数々（e.g. Mount［2004］参照）に比べれば，高崎（2018）や本稿に述べられた手法は，児戯にすぎない．しかし，それでも本稿原稿や，本誌*Naturalistae*や日本蝶類学会誌*Butterflies*の校正に用いて，作業を高速に精確に簡易化することができた．

かつて，シンチンゲルほか編（1972）『現代独和辞典』にある収録語Aurora（p. 93，右欄）の最後の日本語訳には，「クモマツキチョウ」という誤植の珍品があった．今回準備した専門つづり辞書には，動植物の和名を多数含むものもあって，「クモマツマキチョウ」も収録されている．したがって，本稿に紹介したスクリプトを使えば，「クモマツキチョウ」のような誤植はたちどころに検出できる．

学術情報が写本や印刷物の流通というゆるやかな流れでしか拡散しなかった時代には，明らかな誤植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）における弱有害突然変異の出現・拡散を彷彿とさせるような現象を生じたのである．しかし進化上の意味をもつにいたることもある遺伝子の突然変異とは違って，文字列の誤植は学術の発展・進化の原動力となることは期待できない．（ただし，高橋（2013）などの文芸や本稿を含む校正技法の発展への誤植例としての寄与を除く．）「クモマツキチョウ」のような誤植は，出現してもらっては困るのである．「そのような誤植を少しでも減らす方向に向かう学術文書の準備作法の改善に，本稿がささやかにでも貢献することができれば」と願う．

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：小林秀司，小林優大，黒木 出，名取真人，西村直樹，長峰 隆，太田雄太郎，斎藤基樹，高崎和美，高崎弥生，目加田和之．また，本誌*Naturalistae*および日本蝶類学会誌*Butterflies*の編集事務局には，実際の校正作業に，本稿で紹介したスクリプトを試用する機会をいただきました．以上，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．）

本田計一・加藤義臣（編）（2005）．『チョウの生物

学』東京大学出版会．ISBN 4-13-060216-0．xv+626pp.
五十嵐邁・福田晴夫(2000)．『アジア産蝶類生活史図鑑Ⅱ』東海大学出版会．712pp.
五十嵐邁・原田基弘(2015)．『続アジア産蝶類生活史図鑑』自費出版(五十嵐昌子)．355pp.
松香宏隆(2001)．『トリバネチョウ生態図鑑』松香出版．367pp.
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.
中島雅弘(2018)．『UNIX独習への近道』技術評論社，ISBN: 978-4-7741-9596-4，247pp.
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.
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.
シンチンゲル，R・山本明・南原実(編)(1972)．『現代独和辞典』三修社 0584-170111-2767，1327pp.
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．）
高橋輝次(編著)(2013)．『増補版 誤植読本』筑摩書房(ちくま文庫)，ISBN: 978-4-43067-0，314pp.
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.
高崎浩幸(2017)．日本蝶類学会大会2016要旨英訳：舞台裏のからくり．*BSJ Newsletter* 71: 5－8．
高崎浩幸(2018)．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．*Naturalistae* 22: 9－29.
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

本稿は，日本語テキストでの漢字カタカナ文字列の校正の労を軽減するシェルスクリプトを提示する．現代日本語のテキストは，漢字，カタカナ，ひらがな，ローマ字，および数字や記号で構成されている．学術文書における主要な用語の大半は，漢字とカタカナ，ローマ字の文字列で表記され，ひらがな文字列は読み飛ばしても，ほとんど概略の把握はできる．生物学の文書でも，固有名詞に加えて専門用語や生物名は，ローマ字でなければ，大部分は漢字とカタカナで表記される．それらの多くは，市販のスペルチェッカでは点検不可能で，校正者の頭痛のタネとなっている．したがって，何らかの方法で漢字・カタカナだけからなる文字列の誤植が容易に点検できるなら，校正者の負担は大いに軽減される．本稿は，電子テキストでの校正作業の一部をコンピュータ利用によって単純化する．まず，すべての漢字カタカナ文字列を，テキストから重複なく抽出してソートする．この配置で，互いに少数文字だけで異なる類似した文字列は，隣接して近傍に並ぶことが多く，誤植を容易に検出できる．すなわち，何度も繰り返してテキストを読む従来の校正手順によるよりも，文字列を簡単に点検できる．漢字カタカナ文字列のチェックリストの準備には，ここに公開するシェルスクリプト類を使用すればよい．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．さらに，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類から，漢字カタカナ文字列の一般およびカスタムつづり辞書を作成するスクリプトや，特定の専門分野での使用にスクリプトを調整する方法を公開する．これで参照辞書にない漢字カタカナ文字列だけを検出できるようになって，目視点検の手間が大幅に節約できる．これらをアルファベット文字列の簡易点検法(すでに同著者により公開済)と併用することで，日本語で書かれた学術文書の校正作業が，大きく省力化される．

(2018年11月26日受理)