ParEdit チュートリアル

Daregada

2012-03-07

概 要

ParEdit(`paredit.el`) は、Lisp コードで頻出する括弧類のバランスを維持することを目的とした GNU Emacs のマイナーモードだ。この文書は、ParEdit をはじめて使う (あるいは一度利用を断念した) 人へのチュートリアルとなることを目指して書かれたもので、読者層として GNU Emacs の操作や Emacs Lisp に関する基礎的な知識を持ったレベルを想定している。

目 次

1 はじめに **2**
- 1.1 ParEdit の本質は括弧類のバランスを維持すること 2
- 1.2 ParEdit を使う利点と欠点 2
- 1.3 お急ぎの方へ 3

2 ParEdit の導入とメジャーモードへのフック **3**

3 ParEdit を利用した基本操作 **4**
- 3.1 空リストを挿入してリストの内容を書く 4
- 3.2 「飲み込み」と「吐き出し」で既存のリストを伸縮させる 5
- 3.3 リストの不要な部分を削除してから上位リストに接合する 6

4 ParEdit を利用した削除とキル **7**
- 4.1 括弧類のバランスを維持した 1 文字削除 7
- 4.2 括弧類のバランスを維持したキル 8
- 4.3 括弧類のバランスを崩す行単位のカット&ペーストに注意 9

5 括弧類のバランスが崩れたときの対処法 **10**
- 5.1 閉じ括弧入力時の通常動作 10
- 5.2 閉じ括弧が不足・過剰しているときの修正方法 11
- 5.3 括弧類のバランスを無視した挿入と削除 11

6 おわりに **12**

7 著作権情報 **12**

1 はじめに

ParEdit[1]は、あなたの GNU Emacs を擬似的な **構造化 Lisp コードエディタ** に変身させるマイナーモードだ。通常は、Emacs Lisp や Scheme、Common Lisp などの Lisp コードを書くときの各メジャーモードにフックして利用する。また、`M-x paredit-mode` とキー入力することで、いつでも有効・無効を切り替えられる。

1.1 ParEdit の本質は括弧類のバランスを維持すること

最初のうちは、「右丸括弧) を自動補完する」といった入力支援機能に目が行くかもしれないが、これは ParEdit にとってさほど重要な要素ではない。

ParEdit の本質は、 **リスト表記の丸括弧 () のバランスを維持する** 点にある。たとえば、

- リスト内部で `M-s` を押すと、そのリストの丸括弧 () を左右とも削除し、内容を上位リストに接合する。
- リスト内部で `C-k` を押すと、ポイントから (同じ行内にある) そのリストの右丸括弧) の直前までキルする。

といった具合に、さまざまな編集操作において丸括弧のバランスが崩れない[2]ように設計されている。このほか、文字列表記の二重引用符ペア `""` や、ベクター表記などに使われる角括弧[3] `[]` についても、丸括弧と同様にバランスを維持してくれる。

1.2 ParEdit を使う利点と欠点

ParEdit の機能をうまく使えれば、「右丸括弧) の数が不足していないか (あるいは過剰ではないか)」を意識せずに Lisp コードを編集できる。つまり、 **括弧類の対応は ParEdit に任せてコードの内容に集中** できるのだ。

その反面、括弧類を始めとする様々なキーの動作を大胆に変更していることから、何も知らない状態で使うと、

- 「好きな位置に右丸括弧) を挿入できないじゃないか」
- 「BackSpace キーを押したのに括弧が消えてくれない」
- 「コードをコピペして括弧類のバランスが崩れると悲惨だな」

といったネガティブな感想を抱きがちである。

実は、これらの問題にはそれぞれちゃんと解決策が用意されている。たとえば、右丸括弧) を単独で挿入するとバランスが崩れてしまうので、「飲み込み」や「吐き出し」と呼

[1] ParEdit の表記は、ParEdit/Paredit/paredit のように文書によって揺れがある。この文書では、EmacsWiki の ParEdit 解説ページで使われている「ParEdit」で統一している。

[2] とはいえ、ユーザーが括弧類のバランスを無視してコピー (カット)&ペーストを強行することは防げない。この場合、ユーザーが自分でバランスを回復させる必要がある。

[3] Clojure や Emacs Lisp ではベクター表記に使う。Scheme 系実装では丸括弧と同じ意味で使えるものが多い (R6RS では正式採用)。Arc だと無名関数の糖衣構文だったりする。

ばれる操作で既存の右丸括弧) を移動させる。また、リストの内容をすべて削除すれば丸括弧 () は消せるし、内容を残したまま丸括弧 () だけを消す「接合」や「上昇」と呼ばれる操作も用意されている。さらには、括弧類のバランスが崩れてしまった場合に備えて、ユーザーが括弧類を単独で挿入・削除する方法が用意されている、といった具合だ。

しかし、ParEdit に関する日本語の解説が (特に Web 上には) ないに等しいため、こうした解決策に気がつかないまま、ParEdit の利用を断念する人が多いのではないかと思われる。この文書がそうした人への助けとなれば幸いだ。

1.3 お急ぎの方へ

長い説明を読むのは面倒だから実例だけでいいよという人は、EmacsWiki の Paredit Cheet Sheet (http://www.emacswiki.org/emacs-jp/PareditCheatsheet) をご覧あれ。これは PNG 形式の画像で、PareEdit の機能/関数 (要は function) ごとに、標準のキーバインドと操作例の一覧がまとめられている。

この文書では、ParEdit の機能/関数のうち、**これは覚えていないと損だよ** と言えるものに絞って、実例を交えつつ説明する。チートシートに紹介された 30 個の機能/関数[4]のうち、キーバインドを覚えておくべきなのはほんの数個で、残りは左丸括弧 (のように覚えなくても自然に使えるものか、逆にほとんど使う機会がないものだ。

2 ParEdit の導入とメジャーモードへのフック

まずは、`paredit.el`(あるいは `paredit.elc`) を load-path の通った場所に置いてロード可能な状態にしておこう。2012 年 3 月現在、ParEdit(`paredit.el`) の最新バージョンは 22(ベータは 23) で、作者である Taylor R. Campbell 氏の Web サイト (http://mumble.net/~campbell/emacs/) や、EmacWiki の ParEdit 解説ページ (http://www.emacswiki.org/emacs/ParEdit) から入手できる。なお、`package.el` か `auto-install.el` を利用して、ダウンロードからバイトコンパイルまで行なうことも可能だ。

ParEdit のロード方法は、 require でも load-library でも autoload でもお好きなものをどうぞ。遅延ロードにこだわりがなければ、require でロードするように初期化ファイル (いまどきは `~/.emacs.d/init.el`) に書いておけばいいだろう。設定が必須な変数などはないので、試用するにはこれだけで OK だ。

```
(require 'paredit)
```

試用段階では、Lisp 用のメジャーモードに設定されたバッファで、`M-x paredit-mode` とキー入力して ParEdit モードを有効にするといい。なお、`paredit-mode` にはチェック機能が組み込まれており、括弧類のバランスが崩れている場合には有効にできない[5]。括弧類のバランスが崩れた状態での利用は慣れないと危険なので、試用中は括弧類のバランスを直してから ParEdit モードを有効にすることをお奨めする。

[4] この文書では、キー操作の説明に関数名が伴わない場合が多いので、キーバインドの変更などで関数名を知りたいならチートシートを参照してほしい。もちろん、自分で `M-x help k` して調べてもよい。

[5] `C-u M-x paredit-mode` と引数を付ければ、強制的に ParEdit モードを有効にできる。

関連する設定として、括弧類の対応の目視を容易にする show-paren-mode を有効にしておこう。なお、show-paren-style や show-paren-match-face の設定により、対応する括弧の (間も含めた) 外見をカスタマイズできる。

```
(show-paren-mode 1)
```

試用してみた結果、ParEdit を常用することに決めたら、各種 Lisp 用のメジャーモードに対するモードフックに enable-paredit-mode を登録しておこう[6]。

```
(add-hook 'emacs-lisp-mode-hook 'enable-paredit-mode)
(add-hook 'lisp-interacton-mode-hook 'enable-paredit-mode)
```

3 ParEdit を利用した基本操作

それでは、 *scratch* バッファなど、Lisp コードの編集に失敗しても問題ないバッファで ParEdit モードを有効にして、基本的な操作を試してみることにしよう。

3.1 空リストを挿入してリストの内容を書く

左丸括弧 (を入力すると、現在のポイント (以下の例では | で示す) に空リスト () が挿入される。ポイントはリスト内に移動するので、そのままリストの内容を入力すればいい。

```
(|)           ; ( を入力して空リスト () を挿入
(setq dbdir|) ; そのままリストの内容を入力
```

なお、文字列やコメントの内部のように、**左丸括弧 (を入力してもリストにはならない**場所では、ParEdit を使っていないときと同じように左丸括弧 (だけが挿入される。

左角括弧 [や二重引用符 " を入力した場合も、ベクター表記や文字列表記のバランスを維持するために、同様の挙動になる。つまり、現在のポイントに空ベクター [] や空文字列 "" が挿入され[7]、ポイントがそれらの内部に移動する。

```
(setq dbdir "|")          ; " を入力して空文字列 "" を挿入
(setq dbdir "~/Dropbox|") ; そのまま文字列の内容を入力
```

こうした入力時の補完機能は ParEdit の本質ではなく、括弧類だけにしか対応していない。let のように複雑な構造のスペシャルフォームのコード片 (スニペット) を挿入したいなら、Yet Another Snippet(yasnippet.el) (https://github.com/capitaomorte/yasnippet) の利用をお奨めする。もちろん、ParEdit と併用することも可能だ。

[6] EmacsWiki の ParEdit 解説ページ では、メジャーモードのフックに (lambda () (paredit-mode +1)) を追加しろと記述されている。enable-paredit-mode は、このラムダ式と同じ内容に名前を付けた関数だ。

[7] コメントの内部では左角括弧 [や二重引用符 " だけが挿入される。文字列の内部では、左角括弧 [はそのまま挿入されるが、二重引用符 " の場合は (文字列に二重引用符を埋め込むための) バックスラッシュを伴った\" が挿入される。

3.2 「飲み込み」と「吐き出し」で既存のリストを伸縮させる

すでに入力済みの S 式 (複数可) を丸括弧 () で囲んでリストにするには 2 つの方法がある。S 式をひとつずつリストに取り込んでいくか、 C-M-SPC などでまとめて範囲指定してから左丸括弧 (を入力するかだ。ここでは前者の方法を説明する。

リスト内部で C-<right> (<right> は→キー) を押すと、そのリストの右丸括弧) が右に移動して、直後にある S 式をリストに **飲み込む (slurp)**。S 式を飲み込み過ぎてしまった場合には、 C-<left> (<left> は←キー) を押せば、同じ右丸括弧) が左に移動して、リスト末尾の S 式を **吐き出す (barf)**。これらは説明を聞くよりも、例を見てもらうほうが、さらには実際に自分で操作してみるほうが理解が早い。なお、「オレのキーボードには軟弱な矢印キーなぞないのだ」という場合 (HHK Pro かな?) は、 それぞれ C-) と C-} で同じ結果が得られる[8]。

```
(setq dbdir |"~/Dropbox")      ; 文字列の直前をポイント
(setq dbdir (|) "~/Dropbox")   ; ( を入力して空リスト () を挿入
(setq dbdir (| "~/Dropbox"))   ; C-<right> で文字列をリストに飲み込む
(setq dbdir (expand-file-name| "~/Dropbox")) ; リストの内容を追加
```

もう少しニュアンスを込めて訳すと、slurp は **ズルズル音を立てて (飲食物を) すすり込む** 、barf は **ゲロゲロと吐き出す** となる。イメージにはピッタリなんだけどね…。

もし、空リストを挿入して飲み込む手順がまだるっこしいなら、S 式の直前をポイントして M-(を押せば、その S 式を即座に丸括弧 () で囲むことができる。ただし、Meta(Alt) キーと Shift キーを同時に押すことになるので、かえって面倒かもしれない。あるいは、C-u 個数 (のようにして、飲み込む個数を引数で指定することもできる。

```
(setq dbdir |"~/Dropbox")    ; 文字列の直前をポイント
(setq dbdir (|"~/Dropbox")) ; M-( を入力して文字列を () で囲む
```

ところで、移動すべき右丸括弧) が上位リストの右丸括弧) の直前にあると、飲み込む S 式がないのでそれ以上右には移動できない。この状態で C-<right> を押すと、 代わりに **上位リストの右丸括弧) が右に移動** して、直後の S 式を飲み込む (このとき下位の右丸括弧) は動かない)。さらに C-<right> を押すと、こんどは下位の右丸括弧) が右に移動して、直前に上位リストが飲み込んだ S 式を飲み込む。

```
(a ((b| c)) d e) ; 最下位の ) はこれ以上右に移動できない
(a ((b| c) d) e) ; C-<right> を入力すると、ひとつ上位の ) が右に移動
(a ((b| c d)) e) ; 再度 C-<right> を入力すると、下位の ) が右に移動
(a ((b| c d) e)) ; さらに C-<right> を入力すると、上位の ) が右に移動
```

このほか、リストの左丸括弧 (を左右に移動させて飲み込みと吐き出しを行なうキー操作も用意されているが[9]、必要になる場面がほとんどないので説明は省略する。

[8] Ctrl キーと Shift キーを同時押しするのが面倒なら、あなたが押しやすいキーにバインドしなおせばいい。飲み込む関数は paredit-slurp-forward-sexp 、吐き出す関数は paredit-forward-barf-sexp だ。

[9] 直前の S 式を飲み込むには C-M-<left> か C-(、先頭の S 式を吐き出すには C-M-<right> か C-{。

3.3 リストの不要な部分を削除してから上位リストに接合する

こんどは、不要になった () を削除するときの操作だ。リスト内部で M-s を押すと、そのリストの丸括弧 () を左右とも削除し、リストの内容を上位リストに **接合 (splice)** する。splice とは、ロープやフィルムなどの **端と端を繋ぎあわせる** こと (撚り継ぎ/重ね継ぎ) を意味する言葉で、これもイメージによく合っている。

```
(list (list| "Okabe" "Shiina") "Hashida") ; 下位リスト内を適当にポイント
(list list| "Okabe" "Shiina" "Hashida")   ; M-s で丸括弧 () を削除
(list |"Okabe" "Shiina" "Hashida")        ; 不要な「list 」を削除
```

実際には、接合後にリストの内容の一部を削除することが多いので、接合時にリストの不要な部分も取り除いて **ポイント以降の内容だけを残す** 操作や、 **S 式ひとつだけを残す** 操作を覚えておくと、Lisp コードの編集作業がとても捗る。

残したい S 式の直前をポイントして、 M-<up> (<up> は↑キー) を押してみよう[10]。そのリストの丸括弧 () を削除し、リスト先頭からポイント直前までをキルして、ポイント以降のリストの内容だけを上位リストに接合してくれる。

たとえば、when の条件判定が不要になったときや、let のローカル変数が不要になったときに M-<up> を使えば、内部フォームだけを残してさっくりと削除できるのだ。丸括弧 () のバランスは ParEdit が維持してくれるので、「前半だけ消して右丸括弧) を消し忘れた」とか、「誤って別の右丸括弧) を消してしまった」といったミスを防げる。

```
(when tty-p |(menu-bar-mode -1)) ; 内部フォームの直前をポイント
|(menu-bar-mode -1)              ; M-<up> で内部フォーム以外を削除
```

いっぽう、リスト内部の S 式をひとつだけ[11]残す操作は **上昇 (raise)** と呼ばれる。操作の対象とする S 式の直前をポイントして M-r を押すと、その S 式を含む最下位リストの丸括弧 () や残りの内容がすべて削除される。説明より、以下の例を見た方が早い。

```
(when (or |nt-p linux-p) (tool-bar-mode -1)) ; nt-p の直前をポイント
(when |nt-p (tool-bar-mode -1))              ; M-r で nt-p のみが残る
```

リストが複数行にわたっていても、ちゃんと処理される。

```
(if expert-p
   |(menu-bar-mode -1)
  (menu-bar-mode 1))   ; if スペシャルフォームのリスト 3 行から

(menu-bar-mode -1)     ; M-r で then フォーム部分のみを取り出す
```

[10] 矢印キーを持たないキーボード向けのキーバインドが用意されていないので、「オレのキーボードには軟弱な (以下略)」という人は、関数 paredit-splice-sexp-killing-backward を好みのキーにバインドしておこう。この機能を使わないのでは ParEdit を使う意味が **大幅に** 減少する。

[11] C-u 2 M-r のように数値を前置することで、ポイントから末尾方向に並ぶ指定数の S 式が上昇の対象になる (負数の場合はポイントから先頭方向へ)。M-<up> してから不要な部分を削除する方が直感的だが。

4 ParEdit を利用した削除とキル

ParEdit の標準キーバインドでは、削除やキルに関するキー操作 (DEL, C-d, C-k など) に対して、Emacs の初期設定と同じような動作で、なおかつ括弧類のバランスを維持する関数が割り当てられている。

4.1 括弧類のバランスを維持した 1 文字削除

ポイント直前の 1 文字を削除する DEL (BackSpace キーのこと[12]) や C-h (設定が必要[13]) には、通常の delete-backward-char の代わりに、バランス維持機能を持たせた paredit-backward-delete が割り当てられている。関数名から char が取れているのは、以下に示すように文字単位ではない処理を含むからだろう。

- 通常は、ポイント直前の 1 文字を削除する。
- ただし、ポイント直前がリストの丸括弧 () なら、ポイントのみ左に移動。
- ただし、ポイント直前が空リストの左丸括弧 (なら、空リストをまとめて削除。

```
(a (b))| ; リストの直後をポイント。ここで BackSpace を連続して押すと…
(a (b)|) ; ポイントが最初の ) を飛び越える
(a (b|)) ; ポイントが 2 番目の ) を飛び越える
(a (|))  ; b が削除される
(a |)    ; () がまとめて削除される
```

この例でもわかるように、左丸括弧 (や右丸括弧) を単独で削除できないようにしてリストのバランスを維持しているわけだ。こうしたバランス維持機能は、文字列表記の二重引用符ペア "" や、ベクター表記などに使われる角括弧 [] に対しても働く。ただし、コメントや文字列の内部ではバランスを維持する必要はないので、 DEL を使って、左丸括弧 (などを単独で削除できるようになっている。

同様に、ポイント直後の 1 文字を削除する C-d や <delete> (Delete キーのこと) に割り当てられた paredit-forward-delete もバランス維持機能を持っている。

```
|(a ) ; リストの直前をポイント。ここで C-d を連続して押すと…
(|a ) ; ポイントが ( を飛び越える
(| ) ; a が削除される
(|)  ; 空白が削除される
|    ; () がまとめて削除される
```

[12] 歴史的経緯により、GNU Emacs で BackSpace キーを入力すると、 BS ではなく DEL (文字コード 128) に変換される。もっとも、 DEL には delete-backward-char や、(今回のように) それに類似した関数が割り当てられるため、ユーザーが期待する **BackSpace キーとしての動作** に変わりはない。

[13] GNU Emacs の初期設定では C-h に help を割り当てているが、多くの人は **BackSpace キーと同じ動作** を望む。ミニバッファでの動作も考慮すると、正しい解決策は global-set-key を使うことではなく、keyboard-translate で C-h を DEL に変換することだ。

4.2 括弧類のバランスを維持したキル

ポイントから行末までをキルする C-k には、通常の kill-line の代わりに、バランス維持機能を持たせた paredit-kill が割り当てられている。関数名から line が取れているのは、行単位ではない処理が含まれるからだろう。というのも、**ポイントを含むリストが行の途中で終わっている場合** や、**キル範囲のリストが次行以降に継続している場合** にも対応しなければいけないからだ。いずれの場合も、単純に行末までをキルすると丸括弧 () のバランスが崩れてしまう。

- 通常は、ポイントから行末までキルする (ポイントが行末なら改行をキル)。

```
(if (or |nt-p
        linux-p)
    (tool-bar-mode -1)) ; kill-line と同様に動作する例

(if (or |
        linux-p)
    (tool-bar-mode -1)) ; ポイントから行末までキル
```

- ただし、ポイントを含むリストの右丸括弧) が行内に存在するなら、右丸括弧) の直前まで (リストの内容のみ) をキルする。

```
(if (or |nt-p linux-p)
    (tool-bar-mode -1)) ; リスト内部での動作の例

(if (or |)
    (tool-bar-mode -1)) ; C-k で ) の直前までキル
```

- また、キル範囲に含まれるリストが次行以降に継続しているなら、そのリスト全体を複数行にわたってキルする。

```
(if |(or nt-p
         linux-p)
     (tool-bar-mode -1)) ; キル範囲のリストが次行に継続している例

(if |
     (tool-bar-mode -1)) ; C-k でリスト全体をキル
```

1 文字削除の場合と同様、こうしたバランス維持機能は、二重引用符ペア "" や角括弧 [] に対しても働く。たとえば、行内で完結する文字列をポイントして C-k すると、ポイントから文字列末尾の二重引用符 " の直前までをキルする。いっぽう、キル範囲に複数行にわたる文字列が含まれていれば、その文字列全体をキルする。

4.3 括弧類のバランスを崩す行単位のカット&ペーストに注意

Lisp コードの編集において、これまで紹介した ParEdit の機能を使っていれば、括弧類のバランスが崩れることはない。バランスが崩れる最大の要因は、ユーザーが S 式の構造を無視して (たいていは行単位で)、Lisp コードをカット&ペースト (Emacs 風に言うと、キル&ヤンク) することにある。

ParEdit は、**リージョンのキルやヤンクでは括弧類のバランスを維持しない** [14]。このため、ユーザーが安易に Lisp コードを行単位で切り貼りすると、切り取られた部分や貼り付けられた部分でリスト表記の右丸括弧) が不足し (あるいは過剰になり)、容易にバランスが崩れてしまうのだ。二重引用符ペア "" や角括弧 [] についても同じことが言える。

```
(when (require 'server nil t)
| (unless (equal (server-running-p) t)
    (message "Starting server...")
    (server-start)))                    ; ポイントから 3 行カットすると

(when (require 'server nil t)
|                                       ; ) が 1 個不足してしまう
```

バランスの崩れを修正するのは慣れていないと大変なので、リージョンのキルやヤンクにおいても括弧類のバランスを維持することが求められる。

お薦めの方法は、行単位ではなく **S 式の構造に基づいたリージョンを選択する** ことだ。とは言っても、自分で S 式の終わりを見つける必要はない。S 式の先頭 (リストの場合は左丸括弧 (の直前) をポイントして `C-M-SPC` [15] を押せば、その S 式の終わりまで GNU Emacs が自動的にリージョンにしてくれる。あとはキルするなりコピーするなり、左丸括弧 (を入力して全体を丸括弧 () で囲むなりすればいい。`C-M-k` [15] で S 式のリージョン選択とキルを併せて行なうことも可能だ。

```
(when (require 'server nil t)
| (unless (equal (server-running-p) t)
    (message "Starting server...")
    (server-start)))                    ; C-M-k で S 式をカット

(when (require 'server nil t)
|)                                      ; 末尾の ) がちゃんと残る
```

なお、S 式の先頭や末尾へのポイント移動は `C-M-b` [15] と `C-M-f` [15]、上位や下位の S 式へのポイント移動は `C-M-u` [15] と `C-M-d` [15] で行なえることも押さえておこう。

---

[14] リージョン内部で括弧類のバランスが取れているときだけコピー (あるいはカット) を行なう関数も用意されているが、キー操作には割り当てられていない。

[15] こうした Ctrl キーと Meta(Alt) キーの同時押しがイヤな人は、それぞれの関数 (`mark-sexp` など。残りは `M-x help k` で調べてくれ) を適当なキーにバインドすればいい。Ctrl キーや Meta キーを修飾キーとする割り当てが限界なら、Menu キーや Command キーを Hyper/Super キーとして利用するのがお薦めだ。

5　括弧類のバランスが崩れたときの対処法

行単位のカット&ペーストなどによって括弧類のバランスが崩れた状態では、ParEditのさまざまな機能が正常に働かない。以下では、バランスが維持されている状態での閉じ括弧の動作を説明してから、バランスが崩れた場合の対処法を説明することにしよう。

5.1　閉じ括弧入力時の通常動作

すでに説明したように、左丸括弧 (の入力によって空リスト () が挿入されるのだから、右丸括弧) を単独で挿入する必要はない。左角括弧 [と 右角括弧] についても同様だ。

このため、括弧類のバランスが維持されている状態では、閉じ括弧の入力に対して、文字の挿入ではなくポイント移動 (および空白文字の整形) を行なう。具体的には、リストやベクターの内部で右丸括弧) や右角括弧] を入力すると、どちらも以下の挙動をする。

- ポイントを含む最下位リスト (またはベクター) の閉じ括弧の直後までポイントを移動する (リスト・ベクターからの脱出)。

```
(a (b| c)  d e)  ; 内側のリスト内で ) か ] を入力すると
(a (b  c)| d e)  ; 内側のリストの ) の直後にポイントが移動
(a (b  c)| d e)  ; この状態で再度 ) か ] を入力すると
(a (b  c)  d e)| ; 外側のリストの ) の直後にポイントが移動
```

- そのさい、閉じ括弧直前の S 式と閉じ括弧との間に空白文字 (改行を含む) があるなら、それらをすべて削除する (空白文字の整形)。

```
(a| b
  )         ; このリスト内で ) か ] を入力すると

(a  b)|     ; 末尾の b と ) の間の空白文字 (改行含む) をすべて削除
```

いっぽう、文字列やコメントの内部では **リストやベクターにはならない** ため、右丸括弧) や右角括弧] を単独で挿入できる。

```
(setq hoge "|")   ; 文字列の内部で ) を入力すると
(setq hoge ")|")  ; ) が挿入される
(setq hoge ")|")  ; 同様に、] を入力すると
(setq hoge ")]|") ; ] が挿入される
```

ちなみに、いずれの内部でもない場所で閉じ括弧を入力しても、エコーエリア (最下行) に「Not inside a list.」(リストの内部ではない) と表示されて挿入できない。

5.2　閉じ括弧が不足・過剰しているときの修正方法

行単位のカット&ペーストなどによって不足する (あるいは過剰になる) のは、リストの右丸括弧) であることが圧倒的に多い。このため、右丸括弧) (ついでに角右括弧]) については、不足・過剰かどうかを ParEdit が自動的に検知し、ユーザーによるバランス修正を手助けしてくれる。

たとえば、右丸括弧) が不足しているリストでは、**右丸括弧) を任意のポイントに単独で挿入できる**。右丸括弧) を入力すると、通常のポイント移動と整形ではなく、そのままポイントに右丸括弧) が挿入されるのだ。このとき、エコーエリア (最下行) には、「Missing closing delimiter:)」(不足している閉じ区切り文字) と表示される。

```
(when (require 'server nil t)|  ; ) が不足したリスト末尾をポイント
(when (require 'server nil t))| ; ) を押せばそのまま単独で挿入される
```

逆に、右丸括弧) が過剰なリストでは、 DEL (BackSpace キーのこと) により、 **リスト末尾の右丸括弧) を単独で削除できる** 。エコーエリアには、「Deleting spurious closing delimiter.」(不要な閉じ区切り文字を削除) と表示される。削除できるのはリスト末尾の右丸括弧) に限定されることに注意されたい (任意の位置で削除する方法は次小節を参照)。

なお、閉じ括弧や DEL のこうした挙動は、リストのバランスが崩れている間だけ有効で、バランスが修正されると通常の動作に戻る。

5.3　括弧類のバランスを無視した挿入と削除

開き括弧 (左丸括弧 (や左角括弧 [) が過剰な場合は、C-d を使って **リスト先頭の開き括弧を単独で削除できる**。いっぽう、開き括弧が不足している場合は、閉じ括弧のように ParEdit が手助けしてくれないため、困ったことになる。

たとえば、不足している左丸括弧 (を単独で挿入しようしても、いつもの空リスト () が挿入される。それなら、不要な右括弧) だけ削除しようと DEL (BackSpace キーのこと) を押すと、空リスト () がまとめて削除されてしまうのだ。角括弧 [] や二重引用符ペア "" でも似たような事態になる。

こうした場合に備えて、ParEdit が有効な状態でも、

- C-q を前置すれば **任意の括弧類を単独で挿入できる** こと
- C-u を前置すれば **括弧類のバランスを無視して削除/キルできる** こと

を覚えておこう。

たとえば、C-q (や C-q) によって、左丸括弧 (や右丸括弧) を任意のポイントに単独で挿入できる。また、C-u DEL や C-u C-d では任意の位置の括弧類を 1 文字削除できるし、C-u C-k でポイントから行末までを (括弧類のバランスを無視して) キルすることも可能だ。

なお、これらの操作は GNU Emacs 標準の関数 (quoted-insert や delete-backward-char など) を呼び出すため、常用しては ParEdit を使う意味がない。崩れてしまった括弧類のバランスを回復させる目的に限定すべきだ。

6 おわりに

この文書では、ParEdit の機能/関数のうち、**これは覚えていないと損だよ** と言えるものに絞って紹介してみた。簡単にまとめてみよう。

- (で空リスト () を挿入
- C-<right> と C-<left> でリストを伸縮 (飲み込みと吐き出し)
- M-s でリストの内容を残して () だけを削除 (接合)
- M-<up>, M-r でリストの内容の一部を残して残りを削除 (接合と上昇)
- DEL, C-d, C-k は括弧類のバランスを維持しつつ削除・キル
- 行単位の選択より C-M-SPC による S 式単位の選択
- 閉じ括弧の不足・過剰は) や DEL で修正可能
- バランスが崩れたら C-q や C-u を前置した操作で修正可能

このほか、紹介できなかった機能/関数の中にも、

- 適切なコメント記号 (; や ;;;) の挿入とポイント移動を行なう M-;
- リストや文字列の分割を行なう M-S と統合を行なう M-J

など、作業が捗りそうなものがある。操作例などは EmacsWiki の Paredit Cheet Sheet (http://www.emacswiki.org/emacs-jp/PareditCheatsheet) をご覧あれ。

なお、ParEdit を利用して Emacs Lisp 以外の Lisp コードを書いているときの困った挙動 (たとえば、「Scheme のユニフォームベクター表記で左括弧 (の前に不要な空白が挿入される」など) については、誰かがすでに解決していることも多いので、「paredit」と言語名や実装名を組み合わせて Web 検索してみるとよい。

7 著作権情報

同一の Org テキストからエクスポートされた Web ページ版 (http://www.daregada.sakuraweb.com/paredit_tutorial_ja.html) と PDF 版 (http://www.daregada.sakuraweb.com/paredit_tutorial_ja.pdf) が用意されている[16]。

この文書に関する意見・ツッコミ・問い合わせ・その他もろもろの連絡は、Daregada (daichi14657@gmail.com) まで。

[16] Web ページ版では節見出し付近にナビゲーションを追加、PDF 版には印刷を考慮したリンク URL の明示などを行なっているため内容は多少異なる。