● GNU Common Lisp 用ヒストリ・エディタ使用説明書

GNU Common Lisp 用ヒストリ・エディタ「line-edit」は大きくは、ふたつの要素から構成されています。ひとつは１行１列の編集用ウインドウに対して各種編集機能を提供する line-edit-pkg.lsp。もうひとつは、この編集用ウインドウをリング状に連結し履歴検索・編集を可能にする history-pkg.lsp です。これらは連動して使用することも単独で使用することも可能です。

添付テキストの INSTALL.txt にしたがって、オリジナルの top.lsp を同梱の top.lsp と入れ替えて make した GNU Common Lisp は、上記２つのパッケージとトップ・レベルで連動する状態に変更されています（トップ・レベル・リードに line-edit 関数が使用される状態）。

以下では、編集用ウインドウに対して各種編集用コマンドを提供する line-edit 関数の機能を説明します。この説明は help-edit 関数を実行すると表示されます。また、history-pkg.lsp が提供する履歴編集用機能

●編集用コマンド

line-edit の編集用コマンドに対するキー・シークエンスはユーザが自由に定義できます。さらに編集用コマンド自体を追加・変更することも可能です。このようなカスタマイズ方法については HowToCustomize.txt で説明しています。ここでは添付してある Emacs 互換編集コマンドと VI 互換編集コマンドのうちデフォルトとしている Emacs 互換の emacs-mode について説明します。

コマンドの書式

コマンドのキー・シーケンスを表記する際には以下の記法を用います。

C-x	コントロール・キーを押しながら x キーを押す。
M-x	メタ・キーを押しながら x キーを押す。 メタ・キーがない場合は Alt キーで代用する。 さらに ESC キーをタイプしてから x キーを押しても良い。

●前置引数

前置引数は「次」のコマンドを複数回実行するように実行回数を指定するコマンドです。

C-u [num]　Do next command 'num' times (optional, default=4 times).

C-u のあとに数値を指定することもできます。指定しなかった場合は４が指定されたものと見なします。数値の前に '+' または '-' を付加することもできます。'+'だけが指定された場合は +1、'-'だけが指定された場合は '-1' が指定されたと見なされます。「数字」を指定回数回入力できるようにするために '/'を「数値」の入力終了マークとしています。つまり 3 という「数字」を 3 回入力したい場合は

C-u 3/3

と指定します。'/'自体を複数個入力したい場合は '/'を２個指定します。

C-u 3//

●基本的ポイント移動コマンド

ポイントというのは文字と文字の間を指し示す仮想的な位置指示子です。いわゆる「カーソル」はポイントの直後の文字上に表示されます。すべての操作はポイントを基準に示されます。

C-f　Move forward one character.
M-f　Move forward one word.
M-C-d　Move forward down one level of parenthesis.
M-C-n　Move forward across one balanced group of parenthesis.
M-C-f　Move forward across one balanced symbolic-expression.

順に、ポイントを１文字分進める、１単語分進める、リスト構造を１レベル下がって前向きに移動する、リストを横断して前向きに移動する、そして、Ｓ式を横断して前向きに移動するコマンドです。

C-b Move backward one character.
M-b Move backward one word.
M-C-u Move backward out of one level of parenthesis.
M-C-p Move backward across one balanced group of parenthesis.
M-C-b Move backward across one balanced symbolic-expression.

上記と逆の意味のコマンドです。順に、１文字分戻る、１単語分戻る、リスト構造を１レベル上がって後ろ向きに移動する、リストを横断して後ろ向きに移動する、そして、Ｓ式を横断して後ろ向きに移動するコマンドです。それぞれのコマンドは「対称的」な関係にあり、

C-f の逆操作は C-b
M-f の逆操作は M-f
M-C-d の逆操作は M-C-u
M-C-n の逆操作は M-C-p
M-C-f の逆操作は M-C-b

となっています。また

C-u -1 C-f は C-b と同じ
C-u -1 M-f は M-b と同じ
C-u -1 M-C-d は M-C-u と同じ
C-u -1 M-C-n は M-C-p と同じ
C-u -1 M-C-f は M-C-b と同じ

です。これ以降では明示的に説明しませんが、あるコマンドの「逆操作」は、そのコマンドの前に C-u -1 を付ければ実現しますし、ほとんどの場合、逆操作に相当するコマンドも用意されています。

Emacs の単語単位での移動は「単語を一つ飛び越す」操作です。一般的なエディタでの「次の単語の先頭に移動する」操作ではありません。次の単語の先頭に移動する操作は高い頻度で要求されるので追加的に用意してあります。

M-n	Move to next word (like conventional editor).

このコマンドの逆操作に相当するコマンドは用意されていません。

●対応するカッコへの移動

ポイントの直後の文字（＝カーソル位置の文字）がカッコであれば、対応するカッコに移動します。

M-p	Move to balanced parenthesis.

対応するカッコが存在しなければポイントは移動しません。カーソル位置の文字がカッコ以外であれば次のカッコに移動します。

●行・テキストの先頭・末尾への移動

line-edit は、無制限の文字数と無制限の行数を持つ「テキスト」を指定された文字幅の１行１列のウインドウを使って表示・編集します。このウインドウに表示される改行文字までの範囲を「行」、行の集まり全体を「テキスト」と呼んでいます。ウインドウ幅よりも長い行は必要に応じて横スクロールすることによって表示されます。

C-a と C-e が「行」の先頭と末尾に一気に移動するためのコマンドです。

C-a	Move to beginning of line.
C-e	Move to end of line.

コントロールをメタに変えた M-a と M-e が「テキスト」の先頭と末尾に一気に移動するためのコマンドです。

M-a	Move to beginning of text.
M-e	Move to end of text.

テキストがひとつの行だけからなっている（大部分の）場合には両者は同じ意味になります。

●履歴の検索と移動

履歴機能を提供する history-pkg.lsp と連動するように設定した場合、次のふたつのコマンドが使えます。

C-p	Search history backward (previous history).
C-n	Search history forward (next history).

C-p はテキストの先頭からポイントがある位置までの内容が一致する履歴を過去にさかのぼって検索し、一致するものを表示します。連続して C-p をタイプすれば順次、過去の一致する履歴が順に表示されます。C-n は履歴検索で行きすぎてしまった場合などに使用します。なお、標準では上向き矢印を C-p と同じ機能に、下向き矢印を C-n と同じ機能となるように割り当ててあります。

●スクロール機能

line-edit は、無制限の文字数と無制限の行数を持つ「テキスト」を指定された文字幅の１行１列のウインドウを使って表示・編集します。ウインドウに表示されていない文字範囲はポイントの移動に伴って自動的に表示されますが、明示的に表示範囲を変更したい場合には以下のコマンドを使用します。

C-v	Scroll down one line.
M-v	Scroll up one line (or C-^).
C-x <	Scroll left.
C-x >	Scroll right.
M-<	Move to oldest history.

M->　　Move to end of history (same as M-k).

左右への水平スクロール（C-x <と C-x <）のスクロール量は初期値ではウインドウ幅の 20%となっている。変更したい場合は hscroll-unit 関数でスクロール量を文字数単位で設定する。

> (hscroll-unit 20)

１回の左右へのスクロール量を２０文字分と設定している。この関数はキー・シークエンスと結合していないので上記のように直接実行する必要がある（もちろん、必要なら定義することもできる)。

●テキスト内での文字検索

ポイント位置以降の行内を対象にインクリメンタル・サーチを行います。

C-s　　Incremental search.

・探索文字に大文字を指定すると大文字と小文字を区別して探索します。
・探索文字を間違えた場合は C-h で取り消せます。
・探索文字列から(すべての)大文字を取り消すと、探索に対する大文字・小文字の区別を行いません。
・C-h によって探索開始位置までポイントが戻った場合、インクリメンタル・サーチも終了します。
・C-g は探索全体を取り消し、ポイントを探索開始時の位置に戻ります。
・探索文字の代わりに C-s をタイプすると探索文字列の次の出現個所を探索します。
・Enter, Linefeed, または Esc をタイプすると検索を終了します。

●テキストのキルとデリート

Line-edit では Emacs 同様にテキストのデリートとキルを使い分けています。分かりやすく言うと削除したテキストを復活できるのがキルで、復活できないのがデリートです。１文字より大きな単位を削除するコマンドは、すべてテキ

ストを「キル」しています。

キルされたテキストはキル・リング（kill-ring）にスタックされます。キル・リングは初期値では３０段の深さを持ちますが、kill-ring-max 関数にスタックの深さを与えることで、もっと深いスタックにすることもできます。スタックの深さに制限はありません。ただし、あまり深すぎても使いづらいので 30 段よりも深くすることはお奨めできません。

キル・リングにスタックされたテキストはヤンク・コマンド（C-y）でキル・リングからテキストの別の場所にペーストできます。

また、C-@（または C-o）をタイプした場所と現在のポイントに挟まれた範囲を領域（リージョン）と呼びます。C-@をタイプしたポイントをマークと呼びます。このリージョンを対象としてキルを行うこともできます。

C-d	Delete character (or DEL).
M-d	Kill word.
M-C-k	Kill symbolic-expression.
C-k	Kill line from cursor to end.
M-k	Kill whole line and reset search position.
BS	Delete backward character (or C-h).
M-BS	Backward kill word (or M-C-h).
M-z <c>	Kill characters until <c> (zap-to-char).
M-¥	Delete spaces and tabs around point.
M-s	Delete spaces and tabs around point, leaving one space.
C-@	Set region mark (or C-o)
M-@	Mark word (or M-o).
M-C-@	Mark symbolic-expression (or M-C-o).
C-x C-x	Exchange point and mark.
C-w	Kill-region (cut).
M-w	Kill-ring-save (copy).

M-C-w　　Force next kills to append to previous kill.

C-y　　Yank (paste from kill-ring).
M-y　　Paste from kill-ring 2'nd last, 3'rd last,..

●位置の入れ替え

タイプ・ミスや感違いによって書くべき位置を間違えることはよくあります。また、過去の履歴の一部の位置を入れ替えれば再利用できることもあります。このような場合には入れ替え（トランスポーズ）系のコマンドが便利です。入れ替えはポイントを基準に行われます。

C-t　　Transpose characters.

ポイントの前後の文字を入れ替えます。012354 というテキストでポイントが 4 の直前(カーソルが 4 の位置)にあるときに transpose-char を行うと 012345 に置き換わりポイントは 5 の直後に移動します。

指定回数が 0 のときはポイントの直後の文字とマークの直後の文字を入れ替えます。ポイント位置とマーク位置は入れ替わります。マークが設定されていなかったときは何もしません。

M-t　　Transpose words.

ポイントの前後の単語を入れ替えます。指定回数が 0 のときはポイントの直後の単語とマークの直後の単語を入れ替えます。ポイント位置とマーク位置は入れ替わります。マークが設定されていなかったときは何もしません。

M-C-t　　Transpose symbolic-expression.

ポイントの前後のＳ式を入れ替えます。Ｓ式とＳ式の間の空白文字類は、そのまま存在し続けます。ポイントは入れ替えたＳ式の直後に移動します。指定回数が 0 のときはポイントの直後のＳ式とマークの直後のＳ式を入れ替えます。ポイント位置とマーク位置は入れ替わります。

●レジスタ

英数字、および各種記号文字に対応する「レジスタ」と呼ぶテキストとは別の複数の領域にテキスト中のリージョンをコピーしたり、レジスタにコピーされたリージョンをテキストに挿入したり出来ます。

また、全レジスタの内容をファイルに保存することと、ファイルに保存されている全レジスタの内容を元のレジスタに読み込むことが出来ます。

これによって､たとえば､よく使う長い単語を特定のレジスタに保存しておき、使用する場合にはレジスタの内容を挿入する、といった使い方が出来ます。

C-x r s <R>	Copy region to register <R> (<R> = any character).
C-x r i <R>	Insert text from register <R>.

初期設定の状態では GCL が起動するときにファイルに保存されているレジスタが元のレジスタに読み込まれ、GCL が終了するときに全レジスタの内容がファイルに自動的に保存されます。

●ファイルの読み書き

編集中のテキストをファイルに保存したり、ポイント位置にファイルを挿入したり出来ます。

C-x i	Insert file just before cursor position.
C-x C-f	Discard line then insert file.
C-x C-w	Write current line to specified file.
C-x C-s	Write current line to file.

C-x C-w では書き出すべきファイル名を入力するようプロンプトが現れます。C-x C-s では日時を基に yyyymmdd-hhmmss.lsp というファイル名で自動的に書き出されます。

●単語の整形

単語の大文字・小文字を整えるためのコマンドもあります。

M-u	Convert word to upper-case (upcase-word).
M-l	Convert word to lower-case (downcase-word).
M-c	Capitalize word.

前置コマンド（C-u）によって指定された個数の単語を大文字・小文字化したり単語の先頭だけを大文字にします。負数が指定されたときは後ろ向きに指定個の単語を整形します。

●テキストの再表示

現在の行全体を書き直します。

C-l	Redraw line, delete control code.
M-r	Display point at column 0 (abbrev. of C-u 0 C-l).

#¥Newline 以外のすべての制御文字を削除し、ポイントが引数で指定されたカーソル位置に表示されるように行を表示し直します。

引数が nil の場合、(auto-scroll-offset)関数で
・数値が設定されていれば現在の表示範囲を変更せずに表示範囲全体を再描画します。
・nil でも数値でもない値が設定されていれば、横スクロールを行い、表示される情報量が最大となるようにポイント位置を調整して再描画します。

●コマンド実行の取り消し

実行すべきコマンドを間違えた場合、過去の変更（挿入、削除)）を undo 出来ます。テキストの入力を終了する前なら何段階でも遡って操作を取り消すことが出来ます。過去の undo を undo する場合は C-x U を使用します。

C-x u　　Undo previous changes (or C-_ or M-,).
C-x U　　Undo previous undo (or M-.).

undo 回数の初期値は無制限になっています。

●キーボード・マクロの定義と実行

何回も行う操作は「キーボード・マクロ」として記録して再利用することが出来ます。

キーボード・マクロの定義を開始します。

C-x (　　Start keyboard macro.
C-u C-x (　　Append to last macro defined.

反復回数が指定された場合は現在のキーボード・マクロを実行し、その定義にキーボードから入力するコマンドを追加します。

キーボード・マクロの定義を終了します。

C-x)　　End Keyboard macro.

反復回数が指定されると定義完了とともに指定した回数だけキーボード・マクロを実行します。ただし、定義しているときを（実行しているので）１回目の実行と数えます。

最新のマクロを実行します。

C-x e　　Call last keyboard macro.

マクロ定義にユーザへの確認応答を追加します。

C-x q　　Query user during kbd macro execution.

最新のマクロを定義ファイルに書き出し内容を編集する。

C-x C-k　　Edit a keyboard macro.

●その他のコマンド

C-x =　　print infomation on cursor position.
C-q　　Quote next character.
M-(　　Enclose following symbolic-expression in parenthesis.
C-g　　Abort command.
C-j　　Insert #¥Newline.
<ENTER>　　end input.

キー・シーケンスが与えられていない関数も多数あります。

(view-register)　　print registers which contains text.
(view-register #¥R)　　print contents of register R.
(save-registers)　　saves all register to file.
(restore-registers)　　restore all registers from file.
(registers-file)　returns current registers file-name.
(registers-file fname)　set registers file to 'fname'.
(undo-limit)　　returns current undo limit.
(undo-limit n)　　set undo limit to n.
(blink-paren-deley)　　blinking time in sec.
(blink-paren-deley n)　set parenthesis blinking time in sec.
(blink-paren-just-inserted mode)
　　blink parenthesis just inserted.
(blink-paren-just-inserted)
　　blink parenthesis just inserted ?
(last-kbd-macro)　　returns list of last keyboard macro.
(kbd-macro-query-time n)set how many sec. waiting for C-x q.
(kbd-macro-query-time)　　returns how many sec. waiting for C-x q.
(save-macro fname)　　saves kbd macro to fname.

(save-macro)	saves kbd macro to (macro-file).
(load-macro fname)	load kbd macro from fname.
(load-macro)	load kbd macro from (macro-file).
(macro-file fname)	set default load/save file for kbd macro.
(macro-file)	returns current load/save file for kbd macro.

これらのキー・シーケンスが与えられていない関数は通常は初期設定の際に使われる関数です。

●バッチ編集

上記のすべての機能は line-edit 関数がトップ・レベル・リードに組み込まれることで実現されていますが、line-edit をユーザ自身の関数定義の中で使用することもできます。line-edit はユーザからの入力を受け取り、その編集結果を文字列として返します。その場合、上記と同じ編集機能が利用できます。第１引数として文字列を指定した場合には、指定した文字列が初期文字列となります。

一方、line-edit 関数をエディタ・コマンドによって文字列を編集する文字列編集関数として使うこともできます。line-edit 関数の第２引数には emacs-mode では Emacs 互換の、vi-mode では VI 互換の編集コマンドを指定できます。編集コマンドの指定方法は、編集コマンドのキー・シーケンスを定義するのと同じ記法です。

C-a と指定する場合には ¥¥C-a、M-b と指定する場合には ¥¥M-b と記します。

編集用コマンドは文字列として指定しますが、この文字列中の空白は無視しますので、編集用コマンドが見やすいように書いて下さい。編集用コマンドに空白文字を指定したい場合は３種類の方法があります。ひとつめはエスケープ文字 '¥'を使う方法。ふたつめは下線（'_'）を使う方法。みっつめは縦棒（'|'）を使う方法です。

```
"¥¥ "
"_"
```

"| |"

いずれも、ひとつの空白文字を表しています。下線自体を記述する場合は "¥¥_" 、縦棒自体を記述する場合は "¥¥|"と記します。'¥'は Common Lisp のエスケープ文字でもあるので '¥'自体を記すために２つの '¥'が必要な点に注意して下さい。

第３引数には編集用コマンドのモードを明示的に指定できます。必要であれば、第４引数には編集用コマンドの定義ファイルが存在するパス名を指定できます。

つまり、emacs-mode であっても、明示的に vi-mode と指定することで VI 互換の編集コマンドでバッチ編集することも可能です。

上記の記法に従えば、たとえば

(line-edit "This is a pen." "¥¥C-u 3 ¥¥M-f | long|" "emacs-mode")

と

(line-edit "This is a pen." "3w i long_" "vi-mode")

は共に

"This is a long pen."

という文字列を返します。

以上。