エディタ emacs の使い方

プログラミング演習 1 #4
平成 26 年 5 月 2 日

1 UNIX でのプロセス管理 (前回の残り)

UNIX では，通常複数のプログラムが同時に動作しています．実行中のプログラムを，**プロセス (process)** と呼びます．カーネルはこれらのプロセスを管理するために，そのプログラム名ではなく，システム内で一意に定まるような一連の番号を付けて用いています．この番号のことを**プロセス ID (PID)** と呼びます．プロセスの状態表示，実行の一時停止，強制終了などの操作はすべて，このプロセス ID を指定して行われます．

また UNIX では，コマンド行から入力された 1 行のコマンド群をひとまとめにしたものを**ジョブ (job)** と呼んで，プロセスの概念と区別しています．ジョブは，**ジョブ番号**で区別されます．

ジョブの実行形態には，**フォアグラウンドジョブ**と**バックグラウンドジョブ**の 2 種類がありますが，デフォルトは，フォアグラウンドジョブです．コマンドをバックグラウンドジョブとして実行すると，シェルのプロンプトがすぐに表示され，次のコマンドを入力できるようになります．

バックグラウンドで実行させるには，コマンドの最後に `&` を付けます．

```
% prog1 &
```

```
% emacs &
```

上記のように，コマンドをバックグラウンドで実行させようとすると，そのコマンドにジョブ番号が与えられ，ジョブ番号とプロセス ID が表示されます．

フォアグラウンドジョブの実行中に，`C-z`[1]を入力すると，そのフォアグラウンドジョブをサスペンド (一時停止) できます[2]．サスペンドしたジョブをバックグラウンドに移して再開するには，`bg` コマンドを実行します．

```
% bg
```

また，`fg` コマンドを実行すれば，サスペンド中のジョブやバックグラウンドジョブを，フォアグラウンドに移して再開することができます．

```
% fg
```

バックグラウンドジョブを使うと，複数のジョブを同時に走らせることができます．`jobs` コマンドで現在のジョブの番号や状態，それぞれのコマンドが見れます．

```
% jobs
[1]  - Suspended                 emacs prog1.c
[2]  + Suspended                 evince article.ps
[3]    Running                   ~/johokagaku/prog2
```

[1] `Ctrl-z`(コントロール＋ z) のこと．

[2] `Ctrl-z` は**ジョブを「一時停止」したいときにだけ使う**こと．`Ctrl-z` で一時停止させたまま，新しくプログラムを起動していくと，最悪システムにトラブルが発生することがあります．本当にジョブを「終了」させたいときは，後で述べる `Ctrl-c` を使ってください．

「+」が付いているジョブは，**カレントジョブ (current job)** で,「-」が付いているジョブは，その前のジョブです.

bg コマンドと fg コマンドは，デフォルトのジョブの指定がカレントジョブになっています．カレントジョブ以外のジョブを指定するには,「%」を用います．例えば，ジョブ番号 3 のジョブをフォアグラウンドに移すときには，下記のように指定します.

```
% fg %3
```

ちなみに，下記のように「%」だけを指定すると，カレントジョブを指定した事になります.

```
% bg %
```

プロセスの状態を調べるには，ps コマンドを使います．プロセスの名前や，プロセス ID，プロセスがどのくらいの時間動いているかなどが分かります.

```
% ps
```

バックグラウンドで実行されているジョブが不要になった場合や，そのプロセスの実行に時間がかかりすぎていると判断した場合は，プロセス ID を知っていれば，kill コマンドで，そのプロセスを強制終了することができます.

```
% kill  PID
```

また，kill コマンドは，プロセスだけでなく，ジョブも強制終了させることができます.

```
% kill  %2
```

※ 知っておくと良いこと

- プロセスを強制終了しようとしたのに，kill コマンドでうまくいかない場合,

  ```
  % kill -KILL PID
  ```

  とやることで，うまくいく場合があります.（自分で起動したプロセスであれば，自分が権限を持っているケースがほとんどなので，大抵うまく行きます.）ただし，-KILL は有無を言わさず強制終了するので，処理中のデータが壊れてしまう可能性が高くなります．まずは-KILL を使わずに丁寧にお願いしてから，それでも終了しない場合で，どうしても終了させたい時，あるいは，書き込みを行なっていないことが明らかな時にのみ-KILL を付けます.

- (設定にもよりますが，) バックグラウンドジョブは，ユーザーがログアウトしても実行を継続します.

  たとえば，予定があって帰宅しないといけないが，実行を続けたいプログラムがある場合には，プログラムをバックグラウンドで走らせておいて，結果をファイルに出力させ，後日そのファイルを確認するということもできます．逆に，うっかり放置していたバックグラウンドジョブが迷惑になることもあるので，注意しましょう[3].

[3]現在の演習室システムの場合，普通は使用後電源を落とすので，気をつける必要はありませんが，他のシステムにログインしてバックグラウンドジョブを作った場合は気をつけましょう.

2 テキストエディタ Emacs

テキストエディタ (text editor) は，プログラムや文章などのテキストファイルを新しく作成したり，あるいは，既に存在するものを読み込んで編集するためのツールです．UNIX では vi と emacs が有名です．ここでは，emacs について紹介します．emacs はその名の通り，

```
$ emacs
```

で起動します．もしくは，引数にファイル名を指定して，

```
$ emacs diary.txt
```

のようにやれば，diary.txt を編集すべく起動します．もし，diary.txt がなければ，新規作成を行うことと仮定して emacs が起動します.

emacs は，ウィンドゥ，モード行 (画面下の反転表示されている行)，ミニバッファ(モード行の下の空白行)から成り立ってます．モード行では，ウィンドゥに現在表示されているバッファの状態 (入力モード, 漢字コード, 状態表示, バッファ名, モード, 表示位置など) が表示されますミニバッファでは，emacs からのメッセージや，利用者が入力したコマンド文字列が表示されます

emacs によるファイルの編集はすべて**バッファ**と呼ばれる一時的な作業領域で行われます．編集した内容を後で再利用するためには，それをファイルとして保存しなければなりません.

3 Emacs の操作

以降，“C-” と言う表記は,「[Control キー] を押しながら」を表します.

“M-” と言う表記は,「[Meta キー] を押しながら」です．演習室のキーボードでは，[Alt キー] が該当します．[Alt キー] や [Meta キー] がないキーボードの場合でも，[Esc キー] を押すことで，代用できます．ただし，[Esc キー] の場合は「押しながら」ではなく「押した後で一旦指を離して」という事になります．[4]

主なコマンドを下記に示します.

[4] [Esc キー] もない場合（あるいは [Esc キー] に指を伸ばすのが面倒な場合）は，“C-[” が [Esc キー] の代わりに使えます．iPad などから Linux にログインして Emacs を使う場合などに便利です.

コマンド	機能
C-x C-f	ファイルの読み込み
C-x C-c	emacs の終了
C-g	コマンドのキャンセル
C-x C-s	ファイルの保存 (上書き)
C-x C-w	ファイルの保存 (ファイル名を指定)
C-_ または C-x u	アンドゥ（直前の操作の取り消し）
C-s	文末方向への検索 (下方検索)
C-r	文頭方向への検索 (上方検索)
M-%	文字列の置換
C-x i	ファイルの挿入
C-d	1 文字削除 (カーソルがある文字)
C-h	1 文字削除 (カーソルの左にある文字) ... ただし皆さんの初期環境ではの話
C-k	行末までの削除

本日のやってみよう#1

1. diary.txt という引数をつけて emacs を起動する．
2. 次の文章を正確に打ち込む．
   - I study Linux. Since it is very easy for me, I will be a computer expert!
3. ファイルをセーブする．
4. 引数なしで emacs を起動する．
5. diary.txt をファイルから読み込む．
6. 開いた文章の “Since it is very easy for me, I will be a computer expert!” を削除し，“It is very difficult for me because I have no previous experience of Linux.” に書き換える．
7. 書き換えたファイルを diary2.txt というファイル名でセーブし，emacs を終了する．
8. diary.txt と diary2.txt の２つの引数を与えて emacs を起動し，内容を比較してみる．バッファを切り替えるには，後述のように C-x b を入力する．なお，Tools メニューの下の Ediff を実行すると，さらに高機能な比較ができる．

カーソル移動に関するコマンドは下記です．ざっと見た感じ，シェルや Anthy でのキー操作と同様であることが分かると思います．

コマンド	機能
C-f	1 文字右に移動
C-b	1 文字左に移動
C-n	1 文字下に移動
C-p	1 文字上に移動
C-a	行の先頭に移動
C-e	行の最後に移動
C-v	次のページに移動
M-v	前のページに移動
M->	ファイルの最後に移動
M-<	ファイルの先頭に移動

バッファとウィンドゥ操作に関するコマンドは下記です．ただしこれらのほとんどはメニューから選択可能ですし，メニューの該当箇所の右に実際のキー操作が書いてあるので，すぐに覚えなくとも忘れた時にそちらを参照して少しずつ覚えればよいでしょう.（もっとも，ちゃんと覚えておかないと，将来外部からログインして Emacs を使う場合にメニューが使えなくて困ることはあります.）

コマンド	機能
C-x b	バッファの切り換え
C-x k	バッファの削除
C-x C-b	バッファの一覧表示
C-x 2	ウィンドゥを上下に 2 分割
C-x 3	ウィンドゥを左右に 2 分割
C-x 1	現在カーソルのあるウィンドゥを残し，他のウィンドゥを削除
C-x 0	現在カーソルのあるウィンドゥを削除
C-x o	他のウィンドウにカーソルを移動
C-x 5 2	現在あるウィンドウはそのまま，新しく emacs のウィンドウを開く
C-x 5 0	カーソルのある emacs のウィンドウを閉じる

M-h t を入力すると，チュートリアルが表示されます．一度は見ておいて下さい．また設定ファイルは，皆さんの環境ではホームディレクトリの下の .emacs.el ファイルです．この設定ファイルは Emacs Lisp というプログラム言語で書かれており，多様な設定が可能となっています.

ファイルの読み込みを行うときに，ファイルではなくディレクトリを指定すると，Dired モードになります．Dired モードでは，メニュー形式でファイルのコピーや削除，バッファへの読み込みなどができます．Dired モードでのコマンドは下記です.

コマンド	機能
d	削除マークを付ける
u	削除マークの取り消し
x	削除を実行
c	ファイルのコピー
r	ファイル名の変更
f	ファイルの表示

4 バックアップファイル

emacs を使ってファイルを編集していると，

ファイル名~

やら

#ファイル名#

というファイルが，自動的に作成されます．これらはバックアップファイルです．

「ファイル名~」は，以前セーブした (つまり編集前の) ファイルが保存されています.「#ファイル名#」は，emacs の起動中に作成され，emacs は，編集中の文書を定期的に，このファイルにセーブしています．ですので，emacs が異常終了した場合，ファイルをセーブし忘れた場合などに，バックアップとして利用することができます．(正常に終了した場合は「#ファイル名#」というファイルは自動的に消えます．)

「ファイル名~」の場合，編集中のファイルと同様の形式で保存されているため，直接このバックアップファイルを Emacs で開くなりコピーするなりなどして利用できます．これに対して,「#ファイル名#」の場合，大抵は異なる文字コードで保存されるため，日本語を含む場合はそのままでは利用できません.「#ファイル名#」の内容を使って復活させるには，M-x recover-file というコマンドを用います.

5 Emacs の設定

Emacs の設定ファイルは、ホームディレクトリの.emacs ファイルです。.emacs ファイルは、シェルの起動時に自動的に読み込まれます。

これまで特にこの.emacs ファイルの設定は行いませんでしたが，とりあえず私のものをコピーしておくとよいかもしれません.

```
% cp ~tdh8025/.emacs ~
```

この.emacs というファイルを覗いてみると，何かしらカッコがたくさん書いてあります．これは Emacs Lisp という言語です．Emacs はこの Emacs Lisp という言語で動いており，このため Emacs の設定は Emacs Lisp で行います．[5]

もう少し詳しく見てみましょう.

```
(set-language-environment "Japanese")

(prefer-coding-system 'utf-8)
(set-buffer-file-coding-system 'utf-8)
(set-default-coding-systems 'utf-8)
(setq default-buffer-file-coding-system 'utf-8)

(load-library "anthy")
(setq default-input-method "japanese-anthy")

(add-hook 'latex-mode-hook
          (lambda ()
```

[5]ただし，最近のバージョンの Emacs では，customize という多少は一般人に分かりやすいフロントエンドを介して設定できるようになっています.

```
            (set-buffer-file-coding-system 'euc-jp-unix)
            ))

;; 標準のモードをテキストモードに
(setq default-major-mode 'text-mode)

;; テキストモード用設定
(setq text-mode-hook
      (function
       (lambda ()
         (set-fill-column 60) ;60 文字目で折り返し
         (turn-on-auto-fill)  ; 自動改行オン
         )))
    ...
```

各行の; 以降はコメント (読み込んで実行する時に無視される，人間が見てわかりやすくするための説明) です. それ以外の部分は,

```
( ... )
```

のような記述（S 式と呼ばれる）がたくさん並んでいます.

ちなみに，コメントを取り除いたあとは，改行は無視されます．たとえば,

```
( A
  B   C
      D )
```

は

```
( A B C D )
```

と同じ扱いとなります.

これら１つ１つは，Emacs に対する指示であり，上から順に実行されます.

```
( setq 変数名 内容 )
```

というのは,「変数名」に「内容」を割り当てるものです.「内容」としては，カッコが入り組んだ S 式のこともありますが，"文字列"，数，t (真)，nil (偽)，' 変数名などのカッコを含まない単純な値（「アトム」とか「原子式」とかいいます.）がよく現れます．より一般には,

```
( setq 変数名 1 内容 1
          ...
       変数名 n 内容 n
       )
```

つまり

```
( setq 変数名1 内容1 ... 変数名n 内容n )
```

という形式の場合もあります．この場合は，全ての $i = 1,...,n$ に対して「変数名 i」に「内容 i」を割り当てることになります．

実際のところ多くの設定項目がこの形をしています．ので，この単純な形式さえ理解していれば，

- `setq` で設定するところは真面目に設定
- それ以外は，設定方法をググって訳も分からず `.emacs` にコピペ

というやり方でたいていの設定はできてしまいます．(ただし，コピペする前に一応目を通すことにして，あまり安易に怪しいコードを走らせて変なものに感染させないように注意しましょう．)

本日のやってみよう#2

現在の設定では，Emacs のスクロールバーは左側に表示されるようになっています．他のウィンドウでは右側なので，無意識にマウスを左に持って行こうとしたりして不便です．これを右に移動してみましょう．

1. Emacs 上で`*scratch*`バッファに移動して下さい．`C-x b *sc[TAB] [Enter]` と打つか Buffers メニューから選択すればいけるはず．

   モード行に `Lisp Interaction` と書いてあることに気づくかもしれません．`*scratch*`バッファは，基本的には Lisp で遊ぶ (?) ためのバッファです．Lisp だと思ってみると，一番上に最初から書かれている 3 行が`;` で始まるコメントであることがわかりますね．

2. `(set-scroll-bar-mode 'right)` とタイプした後，`)` の右側にカーソルがある状態で `C-x C-e` と押してみましょう．あーらびっくり！

   なお，`C-x C-e` というのは「手前の S 式を実行せよ」という意味です．

3. これだけだと，emacs を再起動してしまえば元に戻ってしまいます．この設定が気に入ったのであれば，毎回 emacs を起動するたびにこの行を実行するために，`.emacs` の最後の行あたりにでも貼り付けておきましょう．

4. 他にも，`(setq scroll-conservatively 100 scroll-margin 0 scroll-step 1)` を実行するとスクロール幅が 1 行ずつになったりとか，いろいろ好きなようにいじれます．

6 Lisp 入門

と，いうわけで，Lisp が分からなくても Emacs の設定はできてしまうのですが，一応この科目は「プログラミング演習 1」という名前なので，もう少し Lisp をいじってみましょう．

Lisp において，

$$(f\ x_1\ ...\ x_n)$$

のような S 式（f, x_i は一般の S 式でよい）は，

$$f(x_1,...,x_n)$$

という数式のように解釈されます．[6]なので，たとえば $3+4$ は Lisp では `(+ 3 4)`，$2\times4\div3$ は `(/ (* 2 4) 3)`，-5 は `(- 5)` のようになります．(`-` と `5` の間の空白に注意．)

[6]ただし，ここでいう f は必ずしも純粋に数学的な意味での関数とは限りません．数学的な意味での関数の計算をするついでに，別の操作 (「副作用 (side effect)」と呼ばれる) を行う「関数」もあります．たとえば，先ほどの `setq` は,「変数への代入」という副作用を主な働きとする「関数」です．

本日のやってみよう#3
これらの S 式を実行してみましょう．間違いがなければ，結果はミニバッファ(最下段のぷちバッファ) に表示されます．なお，整数どうしの割り算は端数が切り捨てられます．

6.1 関数を定義してみる

ここまでだと単なる「カッコをたくさん入力しなければならない電卓」に過ぎません．プログラムの本質は関数にあるので，関数を定義してみましょう．

$$f(x_1,...,x_n) = E$$

(ただし，E は x_i を多分含む一般の式) となる関数 f に `foo` という名前をつけるには，

```
(defun foo (x1 ... xn) E)
```

のように書いて，やはり `C-x C-e` で実行します．たとえば，

$$f(n) = 3+n$$

であるような関数 f に `3+`という名前をつけるのであれば，

```
(defun 3+ (n) (+ 3 n))
```

を実行すればよろしい．(つまり，`3+`は 3 を足す関数．) こうして作った関数は，

```
(3+ (3+ 5))
```

のようにして実行することができます．

いちいち名前をつけてから実行するのがめんどくさければ，`defun` 名前の部分の代わりに `lambda` を使うと名前をつけずにその場で使えます．たとえば，`(lambda (n) (+ 3 n))` は上記の `3+`と同じものです．こうして作った関数は，

```
((lambda (n) (+ 3 n)) 5)
```

のようにその場で関数として使えます．

一般に，

`((lambda (`x_1 `...` x_n`) `E`) `E_1 `...` E_n`)` は，E の中の $x_1 ... x_n$ を $E_1 ... E_n$ で置き換えた式になります．この置き換えのことを難しい言葉で「β 簡約」といいます．この辺の話は後々「プログラミング言語論」という科目で出てくると思います．

6.2 高階関数を定義してみる

関数に与える引数 ($x_1,...,x_n$ の部分) は別にそれ自体が関数であっても構いません．このように，関数を引数とする関数を高階関数といいます．ただし，Emacs Lisp の場合，引数として与えられた関数を関数として使うためには，関数名の前に `funcall` というおまじないをつける必要があります．たとえば，

$$\circ(f,g,x) = f(g(x))$$

な $\circ$ (というか

$$(f \circ g)(x) = f(g(x))$$

な $\circ$ といったほうが分かりやすいかも．つまり，関数合成を行う高階関数) は

```
(defun o (f g x) (funcall f (funcall g x)))
```

のように定義できます.

このようにして定義された関数 o は，(o 3+ 3+ 4) のようにして使うことができます，と言いたいところですが，これだと Emacs Lisp の場合 3+の部分を関数でない変数と解釈してエラーになってしまいます．関数名を引数とする場合は，またまたおまじないとして

```
(o '3+ '3+ 4)
```

のように' をつけます．ただし，無名関数を引数として渡すときは

```
(o (lambda (n) (+ 3 n)) (lambda (n) (* 2 n)) 4)
```

のように' を省略できます.

本日のやってみよう#4

- (defun flip (f x y) (funcall f y x)) における flip がどういう働きをする関数か考え，(flip '/ 2 6) がどういう結果になるか考えてみよう．実行して確認してみよう.
- ((lambda (x) (funcall x x)) (lambda (x) (funcall x x))) がどういう結果になるか考えてみよう．実行して確認してみよう.

7 Emacs での日本語入力

先々週は，Linux デスクトップ環境での日本語入力を習いました．より具体的には，IBus を介してかな漢字変換エンジン Anthy と通信することにより，日本語入力を行うやり方を学びました.

同様の方法で Emacs での日本語入力を行うことも可能ではあるのですが，Emacs のコマンドの中には IBus がコマンドとして認識しないものがあるため，使っていてストレスを感じることがあります.（たとえば，バッファ切り替えのコマンドである C-x b を入力した場合，コントロールキーを用いる C-x の部分はさすがに IBus も Emacs コマンドとして認識しますが，b の部分は単独で押下されるため，地の文の一部と解釈されてローマ字かな変換のモードに入ってしまいます.）また，ネットワーク越しに使う場合も，IBus はしばしば不安定になります.

今週は emacs で Anthy.el という Lisp プログラムを介して日本語入力する方法を修得します．これを可能にするためには，emacs の設定ファイルである~/.emacs というファイルに以下の記述がなければなりません.

```
(load-library "anthy")
(setq default-input-method "japanese-anthy")
```

さて，それでは emacs で日本語変換を行う練習をしてみましょう．まずは，端末エミュレータから emacs を起動します．日本語を入力するためには，emacs 上から Anthy.el を呼び出す必要があります．これは，emacs 上から C-\（コントロール+バックスラッシュ）を押すことで起動されます.

Anthy.el を起動すると，モード行の左端に Anthy という表示が現れます．次に何かキーボードから文字を打ち込んでみましょう．Anthy はローマ字かな変換モードで起動し，ローマ字で文字を入力し，スペースキーで変換，リターンキーで確定，という動作になります.

8 Anthyによる日本語入力

それでは，先々週と同様に，日本語入力モードで次のように入力してみましょう．

```
はははははははとわらった
```

Anthy.el を用いた場合，画面上では入力した文字列の前に | という記号が付加され，下線付きで表示されています．この下線の引かれた文字列が日本語へ変換される対象であることを表しています．これらの文字列を日本語へ変換するには，やはりスペースキーを押します．すると，前回どう学習させたかによるとは思いますが，以下のように表示が変化するんじゃないかなと思います．

```
|[ははは]| ははは | と | 笑った
```

ここから先の操作は先々週に IBus を使った時と同様です．念のため，キー操作を表 1 にまとめます．

表 1: Anthy でのキー操作

キー操作	機能
C-\	日本語モード切り換え（トグル操作）(Anthy.el のみ)
スペース	変換実行
C-p	直前の変換候補に戻す
C-n	次の変換候補を表示する
C-f	変換対象文節を右へ移動
C-b	変換対象文節を左へ移動
C-a	行頭の文節へ移動
C-e	行末の文節へ移動
C-i	変換対象文節を縮める
C-o	変換対象文節を伸ばす
リターン (Enter)	変換結果の確定

なお，Anthy.el では，IBus で使えたようなファンクションキーによるカタカナ，英字，半角文字への変換はできません．カタカナ，英字を入力するには，まずそれらを入力するモードに入ってから入力することになります．

日本語入力の途中で l キーを押すと，モード行が<Anthy: A>になって半角英字入力モードになります．また，L キーを押すと，モード行が<Anthy: A>になって全角英字入力モードになります．再度日本語入力モードに戻るには，C-j を押しましょう．カタカナ入力モードとひらがな入力モードを行き来するには q を押します．

カタカナ変換は変換候補の最後に入っているので，変換を開始して C-p で選択する手もあります．この点は IBus と同様です．