Linuxで利用するコマンド

プログラミング演習1 #3
平成26年4月25日

1 ファイルとディレクトリの操作

emacsで作成するプログラムなどは，UNIX上ではすべて「ファイル」として扱われます．基本的にファイルとは，ハードディスク上に書かれた（保存された）文書やプログラムのソースやバイナリ（コンパイル後の実行ファイル）のことを指します．ファイルが沢山作られてくると，lsコマンドでファイルの一覧を表示させた時に見通しが悪くなります．そこで，関連のあるファイルをまとめて置いておけるように「ディレクトリ」[1]とよばれる仕組みが導入されています．

皆さんが演習室の端末にログインすると，自動的にあるディレクトリに移動しています．このディレクトリは**ホームディレクトリ（略してホームとも呼ばれる）**といい，皆さんがどの端末からログインしても，ユーザに対応するホームディレクトリが自動で設定され，即座に操作ができるようになっています．ホームディレクトリは~（チルダ，ニョロ）という特別な記号で表すことができます．自分がいまいるディレクトリ名はpwdコマンドで表示できます．

引数を与えないでcdコマンドを実行すると，ホームディレクトリへ移動できます．cd ~としても同じことになります．一方，ログイン名がAさんのホームディレクトリへ移動するには，cd ~Aとします．この違いをまとめると以下のようになります．

```
% cd ~A      （Aさんのホームディレクトリに移動）
% cd ~/A     （自分のホームディレクトリにあるAというディレクトリに移動）
% cd ~       （自分のホームディレクトリに移動）
% cd         （自分のホームディレクトリに移動）
% pwd        （現在自分がいるディレクトリ名を絶対パスで表示する）
```

1.1 ディレクトリの作成

新しくディレクトリを作成するコマンドは，mkdirです．コマンドに続けて作成したいディレクトリの名前を続けます．なお，日本語のディレクトリ名やファイル名は基本的に扱えないものと思ってください[2]．それでは，練習としてプログラミング演習1で作成するファイルを入れておくディレクトリ，Penshu1を作成してみましょう．

```
% mkdir Penshu1
```

これで，~/Penshu1というディレクトリが作成されました．~/Penshu1/の意味は皆さんのホームディレクトリの下にあるPenshu1というディレクトリ，を指します．このように，ディレクトリ間の区切りには/（スラッシュ）を使います．したがって，スラッシュをファイル名などに使わないようにして下さい．

[1] Windowsではフォルダといいます．
[2] OSによっては可能なものとそうでないものがありますが，いずれにせよ(Windowsも含めて)日本語ファイル名はトラブルの原因になるので使わないことを推奨しておきます．特に，電子メールに日本語ファイル名のファイルを添付するのは迷惑なので避けましょう．

1.2 ファイルのコピーと移動

ファイルのコピーには cp コマンド，移動には mv コマンドを使います．書式は以下の通りです．

```
% cp コピー元ファイル名 コピー先ファイル名
% mv 移動元ファイル名 移動先ファイル名
```

どちらも，移動したいファイル名が先，移動先が後になっていることを覚えておいて下さい．英語の from～to～というイディオムと同じ形です．from が先，to が後，です．cp コマンドでは，コピー元のファイルはそのまま残りますが，mv コマンドでは移動元のファイルは無くなり，移動先のディレクトリ内にのみ存在することになります．

たとえば，diary.txt というファイルを，今作成した Penshu1 ディレクトリに移動するには，以下のようにします．

```
% mv ~/diary.txt ~/Penshu1/
```

上のコマンドは,「ホームディレクトリにある diary.txt というファイルを，同じくホームディレクトリにある Penshu1 というディレクトリの下に，同じ名前で移動する」という意味になります．もし名前を変更したかったら,「~/Penshu1/新しい名前」とすれば，新しい名前で移動されます.（つまり，移動元と移動先で同じディレクトリ，異なるファイル名を指定すると，単なる名前変更になります.）

一般に，移動先・コピー先として既にあるディレクトリ名を指定すると，そのディレクトリの下に現在のファイル名で移動またはコピーされますが，その場所に存在しない名前を指定すると，指定したファイル名でその場所に移動またはコピーされます．このため，もし~/Penshu1/というディレクトリは存在するけれども~/Pensyu1/というディレクトリは存在しない場合，

```
% mv ~/diary.txt ~/Penshu1
```

と書くと~/Penshu1/というディレクトリの下に diary.txt というファイルができますが，これをうっかり

```
% mv ~/diary.txt ~/Pensyu1
```

と書いてしまうと，特にエラーになるわけでもなく Pensyu1 というファイルに名前変更されてしまいます．このようなミスを防ぐには，ディレクトリの下に移動・コピーするときは必ず

```
% mv ~/diary.txt ~/Penshu1/
```

のように最後にスラッシュを付ける習慣を付けておくとよいでしょう．

1.3 ファイルの削除

ファイルの削除には，rm コマンドを使います．標準の rm コマンドでは，実行したら即座にファイルを消してしまいます．rm コマンドを-i オプション付きで実行すると，指定したファイルを消すまえに確認をするようにできます．このとき，y と答えると実際に削除が実行され，n と答えるとファイルはそのまま残ります．なお，rm コマンドには「ごみ箱」機能はありませんので，一旦削除したり，同じ名前でセーブしてしまったファイルを復活させる方法はないと思っておいてください[3]．常にファイルを削除するかどうか確認するようにしたい場合は，自分のホームディレクトリに.cshrc というファイルを作成し，その中に下のように記述しておくと，常に-i オプションを付けた状態で rm コマンドが実行されるようになります[4]．

[3]実は，運がよければコストをかけることで結構な割合で復活させる方法はありますが，演習室の環境では対応できません．なお，ハードディスクを廃棄あるいは譲渡するときなどにおいて，削除の必要のある個人情報や，どうしても見られたくない恥づかしいファイルがある場合，shred コマンドで完全削除できます.

[4]既に設定済みのはずです.

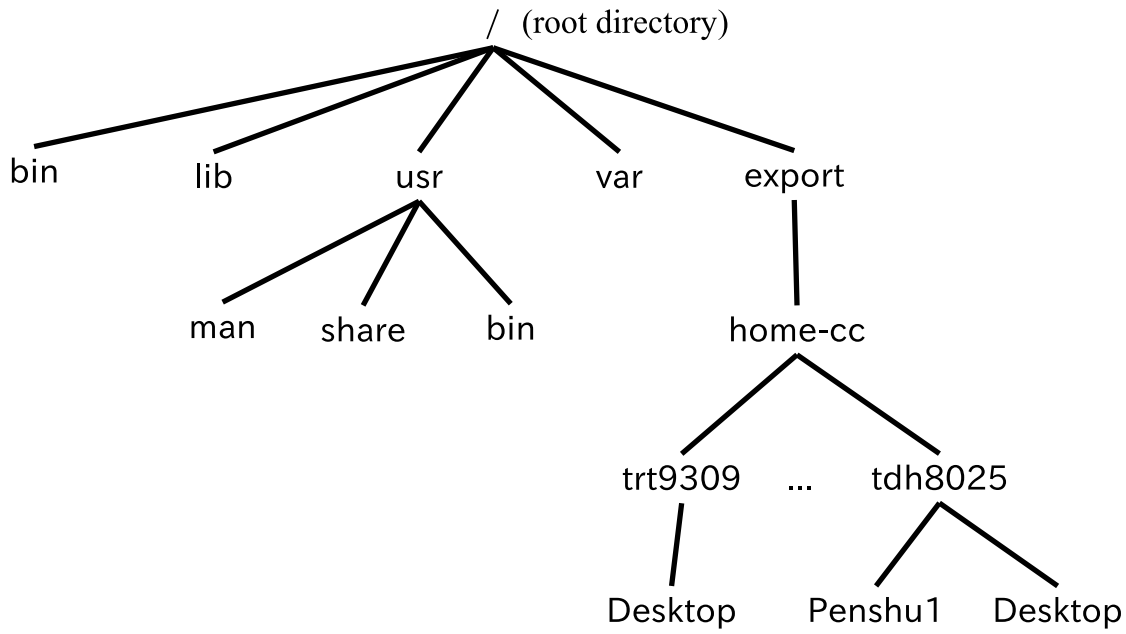

図 1: UNIX でのディレクトリ構造の例

```
alias rm 'rm -i'
```

1.4 相対パスと絶対パス

UNIX が用いるファイルシステムでは，ディレクトリが図 1 のように階層的に構築されています．

ファイル構造の元締となっているディレクトリは「ルートディレクトリ」と呼ばれ，/で表されます．図に従って説明すると，私（tdh8025）のホームディレクトリは，上から辿って/export/home-cc/tdh8025/となることが分かります．このように，あるディレクトリの位置をルートディレクトリから表す書き方を**「絶対パス」**と呼びます．

一方，今いるディレクトリ（カレントディレクトリ）からの相対的な位置関係で目的のディレクトリを示す方法を「相対パス」と呼びます．このとき，カレントディレクトリを示すのに“.”（ピリオド 1 つ），一つ上の親ディレクトリを示すのに“..”（ピリオド 2 つ）を使います）．例えば図 1 において，カレントディレクトリが/export/home-cc/tdh8025/Penshu1 であるとし，ここから/export/home-cc/trt9309/へ移動する方法としては，絶対パスを使った方法，相対パスを使った方法では次のようになります．

表 1: メタキャラクタの使い方の例

書式	意味
*.txt	最後が.txt で終わる全てのファイル
kadai?.txt	kadai で始まり，1 字を挟んで.txt で終わるファイル
kadai??.txt	kadai で始まり，2 字を挟んで.txt で終わるファイル
*	すべてのファイル
[a-c]*	先頭の文字が “a”，“b”，“c” であるすべてのファイル

```
% cd /export/home-cc/trt9309/（絶対パスを使った方法）
% cd ../../trt9309/（相対パスを使った方法）
```

相対パスを使った例では，現在いるところから 2 つ上で辿り（一つ上が tdh8025，2 つ上が home-cs，そこから trt9309 に 降りてくることになります.

1.5 メタキャラクタを使ったファイル操作

ファイルを移動させるときなど，最後の拡張子が.txt であるファイル全てを移動させたい，というようなときがあります．このような処理はメタキャラクタを使って実現できます．メタキャラクタの主なものには，?と*の 2 つがあります．?は任意の一文字，*は任意の文字列に対応します．先の「拡張子がすべて.txt であるファイル」は“*.txt” で表現できます．課題で作成したテキストファイル（kadai1.txt，kadai2.txt，kadai3.txt，kadai4.txt）をまとめて表現するのは，kadai?.txt で OK です．メタキャラクタを使った例を表 1 に示します.

emacs を使っていると増えてくるバックアップファイル（ファイル名の最後に~が付くファイル）をまとめて消すには次のようにします.

```
% rm *~
```

これは，最後に~が付くファイルを rm（ファイルの消去）コマンドに渡すことを意味します.

1.6 あのファイルは何処に？

あるファイルを探したいのに，どのディレクトリにあるのか分からなくなってしまうことがままあります．一つ一つディレクトリを探っていくことも一つので手ですが，UNIX ではそのようなときのための便利なコマンド find があります．find コマンドには様々な機能がありますが，簡単な使い方を紹介します.

1. ホームディレクトリ以下のどこかにあるファイルを探したいとき

```
% find ~ -name "foo.txt" -print
```

- ホームディレクトリ（~）以下にあるディレクトリ中で，foo.txt という名前のファイルを探し，その過程を表示します.
- -print というのは文字通り「結果を表示せよ」という意味ですが，結果に対する指示は省略すると-print を指示した場合と同じ結果になります．ので，単に

  ```
  % find ~ -name "foo.txt"
  ```

と書いても同じ意味です.

2. 特定の名前を持つファイルを一気に削除したいとき

```
% find ~ -name "*~" -print -execdir rm "{}" \;
```

- ホームディレクトリ以下にある，末尾が~で終わるファイルをすべて探しだし，過程を表示しつつそれを削除（rm）します.
- 古い文献では-exec オプションが使われていますが，このオプションはあまり安全ではないので，代わりに-execdir オプションを使いましょう.
- 途中で止めても，既に消されたファイルを復活できませんので，使用には十分注意して下さい．うっかり全てのファイルを消しても責任は持てません.

2 ファイルの保護と共有

ある利用者が作ったファイルを，別の利用者が勝手に覗いたり，書き込んだり，削除したりできないようにするために，UNIX では，ファイルごとに保護モードを持っています．保護モードとは，ファイルの利用者を，「ファイルの所有者」,「ファイルの所属するグループ」,「他人」の 3 種類に区別し，ファイルのアクセス方法を，「読み出し」,「書き込み」,「実行」の 3 種類に区別し，それぞれの利用者に対して，どのアクセス方法を許可するか (しないか) を設定したものです．ファイルごとの保護モードの表示には ls コマンドの l オプションを使います.

```
% ls -l
```

上記のコマンドを実行すると，ファイルの作成日や大きさなどの詳細が行ごとに表示されます．各行の先頭は，10 文字で区切られていて，下記のような表示がされていると思います.

```
-rw-r--r--
```

この 10 文字の最初の 1 文字は，ファイルの種別を表しています．特殊なファイルは，ここに，その種類を表す記号が表示されます.

記号	意味
-	ファイル
d	ディレクトリ
l	リンク

続く 9 文字が，ファイルの保護モード (アクセス権) を表しています．この 9 文字は，3 文字ごとに意味が分かれていて，それぞれ,「ファイルの所有者」,「ファイルの所属するグループ」,「他人」を表しています．また，その 3 文字は，それぞれ 1 文字ごとに，ファイルに対するアクセス許可を以下のように表しています.

記号	意味
r	(Read) ファイルの内容を読むことができる
w	(Write) ファイルを修正したり，ファイルの削除ができる
x	(eXecute) ファイルがコマンドとして実行できる

該当する場所に – が表示されていれば，その許可がないことを表しています．これは約束ですが，ディレクトリには実行許可 (x) がないと，そのディレクトリに移動することができません．

ちなみに,「ファイルの所有者」,「ファイルの所属するグループ」は，上に書いた

```
% ls -l
```

のコマンドで，ファイルの保護モードの表示と同時に表示されます．

ファイルの所有者は，ファイルの保護モードを変更することができます．保護モードの変更には chmod コマンドを使います．指定の仕方は，2 種類あります．1 つは，ファイルの所有者を u で，ファイルの所属するグループを g で，他人を o で，全員を a で表し，

```
% chmod a+x ファイル名

% chmod u-rw ファイル名

% chmod go-r ファイル名
```

のように指定する方法です．もう 1 つは，ファイルの利用者別，つまり，上記の 3 文字ごとに 2 進数で計算し，

```
% chmod 400 ファイル名

% chmod 755 ファイル名
```

のように指定する方法です．初めて作ったファイルの保護モードは，-rw-r--r--(2 進数だと 644) がデフォルトになってます[5]．

他人に見られたくないファイルがもしあれば，そのファイルについて，他人の読み取り許可をなくせば，他人からは読めないファイルになります．

```
-rw-------
```

自分の所有するファイルを，他人に対して読み取りなどの許可を出すことを「パーミッション (permission) を出す」．許可が出てない状態を「パーミッションがない」と言うことがあります．

3 標準入力と標準出力

UNIX のコマンドの多くは，あるファイルなどからデータを入力し，処理した結果を出力します．コマンドの引数にファイル名を指定していなければ，シェルはキーボードから打たれたものをコマンドに対する入力と見なします．この時，キーボードはコマンドの**標準入力**となっています．

コマンドを実行すると，処理の結果は，普通ディスプレイ (画面) に表示されます．この時，ディスプレイはコマンドの**標準出力**になっています．

つまり，特に何も指定しなければ，コマンドは標準入力から入力データを受け取って，標準出力に結果を表示します．標準入力のデフォルトがキーボードに，標準出力のデフォルトがディスプレイになっているわけです．

[5]正確には，シェルの初期設定ファイル内にある umask コマンドで設定されています．

さて，このような標準入力と標準出力の割り当てを変更することができます．これを，**リダイレクト (リダイレクション)** と呼びます．入力を指定する場合は,「<」を，出力を指定する場合は,「>」を使います．

```
% cal > now.txt
% cat now.txt
% wc < now.txt
```

ここで,「>」，つまり出力のリダイレクション記号を使う場合には，既に存在するファイルを間違って上書きしないように注意する必要があります．

「>」の代わりに,「>>」を使えば，既存のファイルの末尾に出力結果を追加することができます．

```
% cal > now.txt
% date >> now.txt
% cat now.txt
```

名前の付いたファイルに入出力をリダイレクトするだけでなく，2 つのコマンドを結び付け，あるプログラムの出力を次のプログラムの入力にすることができます．これを**パイプ**と呼び,「|」で指定します．コマンドの間にパイプを指定すると，パイプ記号の左側にあるコマンドの標準出力が，パイプ記号の右側にあるコマンドの標準入力になります．

```
% ls -l | cat -n

% ls /usr/local/bin | lv
```

UNIX コマンドの中には，`wc`，`sort`，`grep` など，標準入力と標準出力を同時に使用できるものがあります．これらのコマンドでは，コマンドの前にも後ろにもパイプを指定することができます．このようなコマンドを，**フィルタ**と呼びます．フィルタを使ってファイルを加工することを,「フィルタにかける」と言うことがあります．下記のように，パイプを使ってコマンドをいくつもつないでいくことができます．

```
% ls -al | cut -c -10 | sort | uniq -c

% ps aux -w | grep root | lv
```

ところで，出力にはもう一つ，**標準エラー出力**があります．これは，エラーメッセージを出力させるためのものです．デフォルトはディスプレイです．これも変更することができます．

4 パイプを使ったプリンタ出力

情報システム工学科演習室では，ポストスクリプトというページ記述言語を備えたプリンタが 4 台あります．ページ記述言語とは，紙 1 枚を単位にしてどの位置にどのような文字や図形を描くのかを「プログラム言語」のように記述したものです．ポストスクリプト（略して PS などと呼ばれます）言語はアドビ社（フォトショップなどのグラフィックソフトで有名です）の開発になるもので，皆さんが時々目にする PDF（Portable Document Format）もポストスクリプトを元にしています．

さて，演習室のプリンタでファイルを印刷するためには，プリンタが解釈できるポストスクリプト形式にファイルを変換してやる必要があります．皆さんが `emacs` で作成するファイルは（プレイン）テキストファイル[6]というもので，こうしたテキストファイルをポストスクリプト形式に変換するコマンドには `u2ps` があります[7]．

[6]特定のプログラム（例えば Microsoft Word）でしか閲覧できないファイルはバイナリファイルと言うことがあります．

[7]`u2ps` と同様の機能を持つ有名なコマンドに `a2ps` がありますが，こちらは日本語に対応していません．演習室の `CentOS` では `u2ps` が用意されていますが，環境によっては `a2psj`, `a2ps-j`, `e2ps` などのコマンドが代わりに用意されていることがあります．

```
% u2ps -X utf-8 diary.txt
```

上記のコマンドを打ち込むと，カレントディレクトリにある diary.txt というファイルをポストスクリプト形式に変換し，**標準出力**に結果が出力されます．[8]ポストスクリプト形式のファイルを画面で確認するためには，演習室の CentOS では evince というコマンドを利用します．始めに，evince への入力となるポストスクリプト形式の**ファイルを作成**し，その後，そのファイルを引数として evince を起動します．ファイルを印刷する前には，紙の無駄遣いを防ぐ意味でも，画面上で内容をよく確認するようにしましょう．

```
% u2ps -X utf-8 diary.txt > diary.ps
% evince diary.ps
```

上の例で使用している「>」記号は，出力先を標準出力（画面）ではなく，後に記述している名前のファイル（diary.ps）に変更することを意味します．入力元を標準入力からファイルに切り替えるには，リダイレクト記号「<」を使います．

```
% COMMAND1 < FILE1 > FILE2
```

という記述があった場合，始めに FILE1 を入力として COMMAND1 を実行し，その結果をファイル FILE2 に記述することを表します．

では次に，作成した**ポストスクリプト形式のテキストファイル**を実際に印刷してみます．その前に，誰が印刷したのか分かるように，diary.txt の中に自分の学生番号と氏名を記述し，改めてセーブしておいて下さい．印刷には，すでに御存知の方もいると思いますが，lpr コマンドを使用します．

```
% u2ps -X utf-8 diary.txt > diary.ps
% lpr diary.ps
```

画面で内容を確認するまでもなく即座に印刷する場合は，リダイレクトで一旦ファイルを作成せず，パイプで直に lpr コマンドに継いでやります．

```
% u2ps -X utf-8 diary.txt | lpr
```

上の例では，diary.txt というファイルを u2ps コマンドでポストスクリプト形式に変換したあと，その出力をそのまま lpr コマンドに渡すことで印刷を実行しています．

このように，一旦ファイルに落として印刷する方法，パイプで直接印刷する方法の両方に慣れておいてください．

本日のやってみよう

次のページにある「やりたいこと」を実現するためには，どのようなコマンドを，どのようなオプションと共に実行すればよいか，オンラインマニュアルも見ながら考えよう．

[8]最近の a2ps(u2ps ではない) は標準出力ではなく直接標準プリンタに印刷してしまうので注意が必要です

UNIX 操作の練習 [4/25]

	やりたいこと	解答（操作）
1.	cp コマンドのオンラインマニュアルを閲覧するには？	
2.	tdh8025 のホームディレクトリにある Lecture/H26-2014 にある PE1 ディレクトリを，**ディレクトリごと**自分のホームディレクトリにある Penshu1 の下に一度にコピーするには？	
3.	カレントディレクトリから，自分のホームディレクトリにコピーした Penshu1/PE1/ディレクトリに一度に移動するには？	
4.	chap で始まる名前のファイルを一度にすべて消すには？	
5.	ファイル teachers.txt の内容を 1 頁づつ閲覧するには？	
6.	特定の文字列が含まれるファイルを探す grep コマンドのオンラインマニュアルを見るには？	
7.	ディレクトリ PE1 内で文字列 yama を含むファイルを探すには？	
8.	ディレクトリ PE1 内で文字列 Yama を含むファイルを探すには？	
9.	ディレクトリ PE1 内で拡張子が txt であるファイルの中から，文字列 yama を含むファイルを探すには？	
10.	ディレクトリ PE1 内で文字列 Katayama を含むファイルの何行目にその文字列があるかを調べるには？	
11.	ファイルから特定の文字列を切り出す cut コマンドのオンラインマニュアルを見るには？	
12.	ディレクトリ PE1 内のファイル teachers.txt の，各行の先頭 2 文字目までを表示するには？	
13.	ディレクトリ PE1 内のファイル teachers.txt の，日本語の名字だけを表示するには？	
14.	ディレクトリ PE1 内のファイル teachers.txt の，アルファベットで書かれたファーストネーム（名）のみを表示させるには？	
15.	アルファベット順などにテキストを並び替える sort コマンドのオンラインマニュアルを見るには？	
16.	ディレクトリ PE1 内のファイル teachers.txt の各行を，名字のアルファベット順に並べて表示するには？	
17.	ディレクトリ PE1 内のファイル fruits.txt の各行を，果物の数が多い順に並べて表示するには？	
18.	ディレクトリ PE1 内のファイル fruits.txt の内容を，果物の数が多い順に並び替えたファイル fruits2.txt をホームディレクトリに作成するには？	
19.	図 1 において，自分のホームディレクトリにある Penshu1 から，usr ディレクトリの下にある bin に相対パス指定で一度に移動するには？	
20.	図 1 において，自分のホームディレクトリから~tdh8025 以下の Penshu1 に絶対パス指定で一度に移動するには？	

5 シェルの基本機能

シェル csh の設定ファイルは、ホームディレクトリの.cshrc ファイルです。.cshrc ファイルは、シェルの起動時に自動的に読み込まれるので、パス (path) の設定やプロンプト (prompt) の設定に代表される様々な設定を、このファイルで行ないます。

5.1 シェルの補完機能

コマンドを打ち込んでコンピュータを操作しようとする場合，長いコマンド名やファイル名を入力しなければならないことが多々あります．こうした長いコマンドやファイル名の入力を補助するため，シェルには「補完機能」があります．まずは，カレントディレクトリにある長いファイル名を持つファイルを emacs で編集する場合を考えましょう．

カレントディレクトリが~/Penshu1/PE1/であるとします．先ほどの課題がちゃんとできていれば，hana というファイルがあるはずです．emacs からこのファイルを編集しようとする場合，

```
% emacs hana
```

とキーボードから入力します．ここで，実際には以下のように入力してみましょう．

```
% emacs h[TAB]
```

ここで，[TAB] はタブキーを押すことを意味します．すると，シェルは自動的に h で始まるファイル名をカレントディレクトリから選びだし，もし候補が一つしかなければ即座に全体を補完してくれます．

同様に，ファイルの内容を閲覧する lv コマンドを対象に練習してみましょう．キーボードから以下のように入力してみてください．

```
% lv f[TAB]
```

カレントディレクトリには，f で始まるファイルが複数あるので，この段階ではシェルは補完をしてくれません．ここで，もう 1 度 [TAB] キーを押すと，figure.eps,figure.obj,fruits.ps,fruits.txt と 4 つ（3 つかも知れません）の候補が表示されます．そこで，f に続いてもう一文字，r を入力してやります．

```
% lv fr[TAB]
```

すると，シェルは fr から始まるファイルをカレントディレクトリから探しだしてくれますので，自動的に

```
% lv fruits.
```

まで補完してくるはずです．あとは拡張子部分を入力してやれば，ファイル名の指定は完了です．

これらの補完機能はファイル名を入力するときだけでなく，コマンドを入力するときにも使えますので是非有効に活用してください．

なお，同様の補完は emacs でも可能です．

つまづいたらとりあえず [TAB]

ど忘れしたらとりあえず [TAB]

面倒くさくなったらとりあえず [TAB]

ってこってす．

5.2 シェルのヒストリ機能

シェルには、コマンドを記憶しておく**ヒストリ (history (履歴))** 機能があります。ヒストリ機能を使うには、前もって以下のような設定が必要です。

```
% set history = 100
```

これは、現在から過去にさかのぼって、100 個のコマンドを記憶しておくことを意味してます。この設定は、シェルの起動後に 1 回読み込めば良いので、通常 .cshrc ファイルに定義しておきます。

```
% set savehist = 100
```

と設定すると、ログアウト時に、自分のホームディレクトリに.history ファイルが生成され、履歴が 100 個保存されます。この履歴は、次回ログインした時に、利用することができます。
現在までのヒストリ (コマンドの履歴) を見るには、history コマンドを使用します。

```
% history
```

history コマンドは、記憶している全ての履歴を表示します。ここで、引数に数字を指定すれば、過去その数字分だけの履歴を見ることができます。例えば、過去 10 回分のヒストリを知りたければ、次のコマンドで見ることができます。

```
% history 10
```

history コマンドの実行結果での、各行の先頭に付けられている数字は、ヒストリ機能が付けたコマンドの番号です。ヒストリを参照してコマンドを実行させるには、「!」を用います。例えば、 12 番のコマンドを実行するには、

```
% !12
```

で実行することができます。ヒストリの参照コマンドには以下のようなものがあります。

ヒストリのコマンド	意味
!!	直前に実行したコマンド
!n	n 番目のコマンド
!-n	n 個前のコマンド
!文字列	最も直前に実行した、指定した文字列で始まるコマンド

また，**C-p で一つ前のコマンド**を呼び出せますし，**C-n で一つ先**（C-p で戻り過ぎたときなど）のコマンドを呼びだすこともできます．一般に，皆さんが使っている csh や bash では，**emacs** と同じキーボード操作でカーソル移動ができます．たとえば，ちょっとだけ間違えてコマンドを実行してしまったときなど，C-p で間違えたコマンドを呼び出し，C-a（行頭へ移動）や C-e（行末に移動），C-b（一文字分左へ移動）や C-f（一文字分右へ移動）でカーソルを移動させ，C-d（一文字分削除）や C-k（カーソル以降の文字を全部削除）したあと，新しい文字を入力することで，長いコマンドを最初から入力する手間を省くことができます．また，C-r を使って過去に入力した文字列を検索し，そのまま，あるいは，編集して，実行することができます．

5.3 シェルのエイリアス機能

よく使うコマンドや、引数が長いコマンドは、**エイリアス (alias(別名))** 機能を利用し、別の名前で定義しておくと便利です。エイリアス機能によって、新たな名前を付けることも、コマンドを再定義することもできます。

```
% alias 別名 コマンド
```

で定義できますが、エイリアスは、通常 .cshrc ファイルに前もって定義しておくのが普通です。

```
% alias
```

とだけ実行すると、現在のエイリアスの一覧が見れます。既にあるコマンドをエイリアス機能で再定義した場合に、一時的にそのエイリアスを無効にしたい時には、「\」をコマンドの先頭に付けることによって、無効にできます。

```
% \ls
```

また、エイリアス機能によって定義した別名を取り止めたい時には、unalias コマンドを使用します。

```
% unalias 前もって定義した別名
```

なお、最初に書きましたが、.cshrc ファイルは、シェルの起動時に自動的に読み込まれます。逆に言えば、.cshrc ファイルの一部を書き換えたとしても、起動時でないと読み込まれません。.cshrc ファイルを編集した場合に、その内容を今すぐ有効にするには、source コマンドを使います。

```
% source .cshrc
```

6 UNIX でのプロセス管理

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[9]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[10]．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．

[9] Ctrl-z(コントロール＋ z) のこと．

[10] 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．

```
% 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
```

「+」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[11].

[11]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.