UNIX の階層ディレクトリとコマンド

この資料において「**別資料**」とは、資料「情報科学実験 1・新出担当分—シェルとシェルスクリプト—」を指す。

1　階層ディレクトリ

UNIX (Linux を含む。以下同様) では, 階層構造の**ディレクトリ** (他の OS では「フォルダ」と呼ぶ場合あり) によるファイル管理が行われている。図 1 がその例である。同じファイル名 `a1.txt` でも、異なるディレクトリの下にあれば違うファイルを指す。

階層構造 (木構造) の根にあたるディレクトリを「**ルートディレクトリ**」と呼び、「/」という名で表す。

本資料は、2018 年度からの奈良女子大学生活環境学部・生活情報通信科学コースの「計算機実験 1」のシェルスクリプトの回で使用しているものであるため、同コースで使用している計算機環境や資料に依存する部分があります。

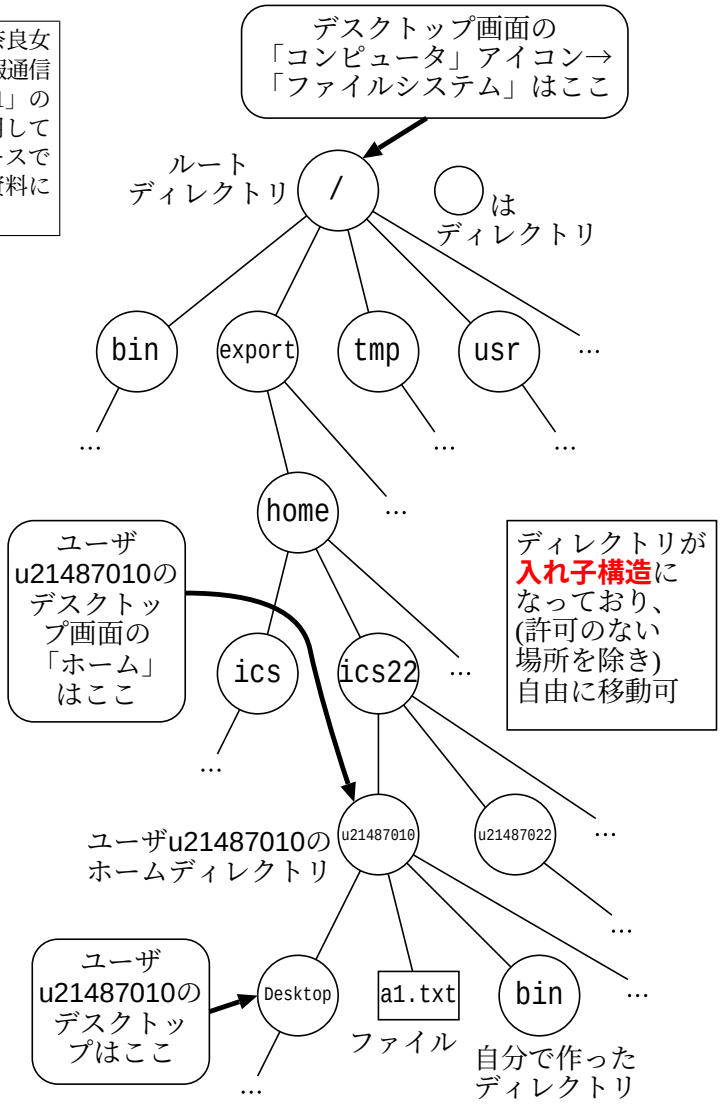

図 1: 階層ディレクトリの例 (G 棟システムの Linux の場合)

1.1　絶対パス

木構造のどこかにあるディレクトリやファイルを特定するには、ルートディレクトリから下へとたどって行って、経由するディレクトリやファイルの名を「/」をはさみながらつなげた名前を用いる。これを「**絶対パス**」と呼ぶ (「フルパス」とも呼ぶ[1])。例えば図 1 の下の方の真ん中へんにあるファイル `a1.txt` の絶対パスは `/export/home/ics21/`

[1]「絶対パスとフルパスは異なる」としている Web ページなどもあるが、それはファイル名ではないものの話をしている場合である。ファイル名の話をしている場合は、絶対パスとフルパスは同じものである。

u21487010/a1.txt であるが、同図の右下にあるファイル a1.txt の絶対パスは /export/home/ics21/u21487010/misc/a1.txt というように、同名のファイルでも**絶対パスが異なる**ので区別できる。なお、絶対パスは**必ず「/」で始まる**。

1.1.1 ホームディレクトリ

各ユーザには「**ホームディレクトリ**」と呼ばれるディレクトリが 1 つ決まっている (その決まり方はシステムによって異なる)[2]。図 1 では、例えばユーザ u21487010 のホームディレクトリの絶対パスは /export/home/ics21/u21487010 となっている。

自分のホームディレクトリがどこであるか確認するには、端末エミュレータで「echo ~」というコマンドを実行すればよい。確認しておこう。学科システム**以外**の場合は、先頭が「/export/」で始まらないことが多い。

自分が作成したファイルは、基本的にはホームディレクトリの下 (あるいはその下のディレクトリのさらに下) に作られている。

1.1.2 絶対パスによるファイルの指定

端末エミュレータで実行する各種コマンドや、その他のソフトでファイルを指定する時、絶対パスを使ってファイルを指定することができる (1.2 節で述べる「相対パス」で指定することもできるが、それは後で)。

> 以下の例を実際に試す場合は、「/export/home/ics21/u21487010」の部分を、1.1.1 節で確認した**自分のホームディレクトリ**に変えて試そう。

例えば、

```
$ emacs /export/home/ics21/u21487010/a1.txt↵
```

とすれば (先頭の「$」はプロンプトを表すので、実際には打ち込まないこと。以下も同様)、絶対パス /export/home/ics21/u21487010/a1.txt のファイルを Emacs エディタで作成することができる (既に存在する場合は編集できる)。

> 端末エミュレータでコマンド行を打ち込む際は、**コマンド行補完機能**を積極的に活用しよう。例えば「emacs /ex」まで打ち込んでタブ (Tab) キーを押すと「/export」まで自動補完される。また、同じコマンドをもう 1 回実行したい場合は、上向き矢印キーで前回のコマンドが再表示される (左右矢印キーで編集もできる)。

ファイル作成後は

```
$ ls -l /export/home/ics21/u21487010/a1.txt↵
```

で、作成されたことを確認してみよう。ls -l コマンドは指定したファイルの各種情報 (サイズ、更新時刻など) を出力するものである (そのファイルが存在しなければエラーメッセージが出る)。

ディレクトリを**自分で作る**には mkdir コマンドを使う。例えば

```
$ mkdir /export/home/ics21/u21487010/misc↵
```

とすれば、そのディレクトリが作られる (既にそのディレクトリが存在する場合、その旨のエラーメッセージが出る)。そうしてから

```
$ emacs /export/home/ics21/u21487010/misc/a1.txt↵
```

として、今度は絶対パス /export/home/ics21/u21487010/misc/a1.txt のファイルを作ってみよう。

なお、ファイルを作るには、ルートディレクトリからそのファイルに至るまでの途中のディレクトリが全て存在していなければならない。例えば、ディレクトリ /export/home/ics21/u21487010/misc が存在しないうちにファイル /export/home/ics21/u21487010/misc/a1.txt を作ることはできない (実際には、Emacs でファイルを作ろうとすると、途中のディレクトリがなかった場合、そのディレクトリを作るかどうかを Emacs がユーザに問い合わせてくれるので、Emacs であれば結果的にはファイルを作れる)。

ls, mkdir などのコマンドについては 2 節も参照。

[2] 厳密に言うとここの話は、「ホームディレクトリ」と「ログインディレクトリ」を混同して書いているのだが、標準的にはこの 2 つは同じである。この資料ではこの 2 つを区別しないで、「ホームディレクトリ」とだけ呼ぶことにする。

1.1.3 ホームディレクトリの略記法

端末エミュレータでコマンドを打ち込むときなどには、自分の**ホームディレクトリ**を「~」、他人のホームディレクトリを「~ユーザ名」と略記できる[3]。例えば図 1 の右下のファイル a1.txt の絶対パス /export/home/ics21/u21487010/misc/a1.txt は、ユーザ u21487010 から見れば ~/misc/a1.txt と略記でき、他のユーザから見れば ~u21487010/misc/a1.txt と略記できる。よって、ユーザ u21487010 にとっては

```
$ emacs ~/misc/a1.txt↵
```

とするのは、$ emacs /export/home/ics21/u21487010/misc/a1.txt↵とするのと同じ意味になる。

「echo ~」コマンドで自分のホームディレクトリが確認できる (1.1.1 節) のも、この略記法による。echo が引数をそのまま出力するコマンドであり、かつ「~」がホームディレクトリの略記だからである。

1.2 カレントディレクトリと相対パス

端末エミュレータを操作している場合は、端末エミュレータ 1 つ毎に[4]、図 1 のようなファイル構造の中のどこか 1 つのディレクトリが「**カレントディレクトリ**」として設定されている。そして、「カレントディレクトリ」を起点にしてファイルを特定するのが「**相対パス**」である。

例えばファイル /export/home/ics21/u21487010/misc/a1.txt の相対パスは、カレントディレクトリが /export/home/ics21/u21487010 であるときは misc/a1.txt となる。カレントディレクトリから misc→a1.txt とたどってそのファイルまで行けるからである。このように、あるファイルの相対パスは、カレントディレクトリからそのファイルまでたどる時に経由するディレクトリやファイルの名を、間に「/」をはさみながらつなげたものになる。なお、相対パスは**「/」で始まらない**ので、最初の文字を見れば絶対パスと相対パスは必ず区別できる[5]。

同じファイル /export/home/ics21/u21487010/misc/a1.txt でも、カレントディレクトリが /export/home/ics21/u21487010/misc であるときは、相対パスは a1.txt となる。また、カレントディレクトリが /export/home/ics21 であるときは、相対パスは u21487010/misc/a1.txt となる。このように、同じファイルでも、相対パスはカレントディレクトリがどこにあるかによって**異なる**。

逆に、相対パスが同じであっても、カレントディレクトリが違えば**違うファイルを指す**。例えば、相対パスが a1.txt のファイルは、カレントディレクトリが /export/home/ics21/u21487010/misc であるときは /export/home/ics21/u21487010/misc/a1.txt だが、カレントディレクトリが /export/home/ics21/u21487010 であるときは /export/home/ics21/u21487010/a1.txt である。

1.2.1 カレントディレクトリを知る・変更する

端末エミュレータを使っていて、**カレントディレクトリを知りたい**場合には、pwd コマンドを実行すればよい。

G 棟システムの場合は、プロンプトの右半分 (「]$」より前の部分) に、カレントディレクトリの絶対パスの最後の部分 (例えばカレントディレクトリが /export/home/ics21/u21487010/misc の場合は「misc」) が表示されているので、これによってもカレントディレクトリの見当をつけることができる。ただし、カレントディレクトリがホームディレクトリであるときだけは、「~」が表示される。
なお、プロンプトはシステムによって異なるので、どのシステムでもこうなっているとは限らない。

端末エミュレータを起動した直後のカレントディレクトリは、ホームディレクトリである。
カレントディレクトリを**変更**するには cd コマンドを使う。例えば

```
$ cd /export/home/ics21/u21487010/misc↵
```

とすると、ホームディレクトリの下の misc ディレクトリがカレントディレクトリになる (そのディレクトリが存在していないとエラーになる)。ユーザ u21487010 の場合、$ cd ~/misc↵でも同じである。こうしてから

```
$ emacs a1.txt↵
```

---

[3] 使っているシェルの種類などによっては、そうでないこともある。
[4] 正確には「プロセス毎に」である。
[5] ただし、~/misc/a1.txt や ~u21487010/misc/a1.txt は**絶対パスである**。一見、先頭が「/」で始まっていないように見えるが、略記を元に戻せば先頭が「/」で始まっているからである。

とすると、絶対パス /export/home/ics21/u21487010/misc/a1.txt のファイルを編集することになる。

cd コマンド自体でも相対パスはもちろん利用できるので、例えばユーザ u21487010 が端末エミュレータを**起動した直後**に

```
$ cd misc↵
```

とすると、$ cd /export/home/ics21/u21487010/misc↵ としたのと同じことになる。なぜなら、端末エミュレータを起動した直後のカレントディレクトリは、u21487010 のホームディレクトリ /export/home/ics21/u21487010 であり、従ってその時の相対パス misc は絶対パスとしては /export/home/ics21/u21487010/misc を指すからである ($ cd ~/misc↵ あるいは $ cd ~u21487010/misc↵ とするのとも同じである)。

絶対パスは長くなりやすいので、カレントディレクトリを適切に変更した上で**相対パス**を使う方が便利なことが多い。特に、端末エミュレータの起動直後はホームディレクトリがカレントディレクトリなので、相対パスを指定してファイルを作れば基本的にホームディレクトリの下にファイルが作られる。

また、ホームディレクトリのすぐ下に直接ファイルを作るのではなく、ディレクトリを作ってその下にファイルを作る方が、**ファイルの整理**がしやすい。

1.2.2 親ディレクトリ

例えばカレントディレクトリが /export/home/ics21/u21487022 であるときに、ファイル /export/home/ics21/u21487010/a1.txt の相対パスを決めようとすると、一旦 u21487022 ディレクトリの 1 つ上の /export/home/ics21 ディレクトリまで行ってから、u21487010→a1.txt とたどらなければならない。

このとき、「**1 つ上へ行く**」は相対パスでは「..」と表す。従って、この場合の相対パスは「../u21487010/a1.txt」となる。

「..」で表されるディレクトリ、すなわちそのディレクトリの 1 つ上のディレクトリを、そのディレクトリの「**親ディレクトリ**」と呼ぶ。「..」を使えば、カレントディレクトリがどこであっても、どのファイルの相対パスも決めることができる。

そのディレクトリ自身を表す「.」という記法も覚えておこう。例えばカレントディレクトリが /export/home/ics21/u21487010 であるときに、./misc/a1.txt はカレントディレクトリから見て「自分自身」→misc→a1.txt とたどったファイル、つまり相対パスが misc/a1.txt のファイルと同じものを指す (カレントディレクトリが他のところであっても同様である)。同様に、./a1.txt は相対パスが a1.txt のファイルと同じものを指す。従って例えば $ emacs a1.txt↵ とするのと $ emacs ./a1.txt↵ とするのとでは、同じファイルを編集することになる。

特に、**カレントディレクトリ自身**の相対パスは「.」と約束する。例えばカレントディレクトリが /export/home/ics21/u21487010 であるときに、/export/home/ics21/u21487010 自身の相対パスは「.」である。

1.3 ファイルのパーミッション

各ファイルには「**パーミッション**」(許可) が決められていて、ファイルの読み出し・書き込み・(プログラムとしての) 実行のうち、許可のない操作はできない。

通常、他人のファイルには書き込みはできない。他人のファイルを読めるかどうかはパーミッションの設定によって異なる。また、ファイルの持ち主は、chmod コマンドなどでパーミッションを変更できる。

例えば、ユーザ u21487022 が $ emacs ~u21487010/a1.txt↵ のようにすると、ユーザ u21487010 のホームディレクトリ下にある、(おそらくユーザ u21487010 が持ち主の) a1.txt というファイルを書き換えようとすることになるが、これは通常できない。読むことができるかどうかは、場合によって異なる (ので、Emacs の画面にはそのファイルの内容が表示されることもある)。いずれも、ファイルの持ち主は chmod で許可の有無を変更できる。

パーミッションについての詳細は、別資料 0.3 節「ファイルの属性について」を参照。

2　UNIX の基本コマンド

この節では、別資料の 0.1 節で述べる主要コマンド群の中でもさらに超基本的なものの使い方と、主なオプション引数について説明する。

2.1　標準入力・標準出力・標準エラー出力

標準入力・標準出力・標準エラー出力とは、C 言語でいう stdin, stdout, stderr のこと。

コマンドの処理結果の標準の出力先が「**標準出力**」、エラーメッセージなど人間との対話のための標準の出力先が「**標準エラー出力**」で、これらは何もしなければ**端末画面** (端末エミュレータの場合はそのエミュレータのウィンドウ) だが、出力先をファイルなどに切り替えることができる。また、コマンドが起動後に何か処理用の入力データを受け取るための標準の入力元が「**標準入力**」で、これは何もしなければ**キーボード**だが、入力元をファイルなどに切り替えることができる。これらの**入出力先の切り替え**については別資料 0.2.3 節参照。

2.2　オプション引数

オプション引数 (単に「オプション」と呼ぶこともある) とは、コマンドに与える引数のうち、コマンドの動作を少し変更するために指定するものをいう。「`-`」の後ろに一文字 (例えば「`-a`」) の形、あるいは「`--`」の後ろに数文字 (例えば「`--help`」) の形をしていることが多い (どちらでもないものもまれにある。本資料に出てくるのは全て前者)。また、オプション引数は**引数のうち最初** (オプション引数以外の引数よりも前) に指定されることが多い (オプション引数を**最初に指定**しなければオプションと認めてくれないコマンドと、**そうでないもの**がある。いずれも多数[6]。前者は echo など、後者は cat などが該当)。

『「`-`」の後ろに一文字』の形のオプションは、いくつか指定する場合には**まとめて指定** (例えば「`-l`」と「`-a`」をまとめて「`-la`」のように) できるコマンドも多い (が、そうでないものもある)。

2.3　基本コマンド群

cat: ファイルの出力

使い方: `cat` ファイル名$_1$ ファイル名$_2$ … ファイル名$_n$
主なオプション: `-n` (行番号つきで出力)

ファイルの内容を出力する。例えば `$ cat a.c b.c`↵でファイル a.c と b.c の内容を出力する。`$ cat -n a.c b.c`↵だと行番号つきで出力。

ファイル名を 1 つも指定しない場合、**標準入力**を読んで標準入力に出力する。例えば `$ cat`↵とすると、キーボードから入力したものを端末画面に出力する。この時、**キーボードからの入力の終了** (EOF, End of File) をコマンドに知らせるためには、Ctrl-D を押せばよい (これは、キーボードから入力する場合には他のコマンドでも共通)。

less: ファイルの 1 画面毎出力

使い方: `less` ファイル名
主なオプション: `-S` (長い行を画面幅で折り返さない)

ファイルの内容を、**1 画面毎に停止**しながら出力する[7]。例えば `$ less a.c`↵で、ファイル a.c の内容を 1 画面毎に停止しながら出力する (1 画面に納まらないほど長いファイルで試してみよ)。その際、f (またはスペース) キーと b キーで画面を上下、↓キーと↑キーで 1 行ずつ上下し、q キーで終了できる。

オプション `-S` を指定するのとしないのとでどう違うかは、1 行が画面の横幅より長いファイルを less で表示させてみればわかるはず。

[6] システムにもよる。昔の UNIX や、現在でも一部のシステムでは、ほとんどのコマンドが前者に該当する場合もある。
[7] 実はファイル名を複数指定することもできるが、本資料では略。

ファイル名を指定しないこともでき、その場合は**標準入力**を読んで表示する。これが便利なのは、別資料 0.2.3 節で述べる「**パイプ**」を使う時。例えば `$ cal 2018 | less`↵ で、「`cal 2018`」というコマンド (2018 年のカレンダーを出力する) の出力を 1 画面毎に停止しながら見ることができる。

cp: ファイルのコピー

使い方その 1: `cp` ファイル名$_1$ ファイル名$_2$
使い方その 2: `cp` ファイル名$_1$ ファイル名$_2$ … ファイル名$_n$ ディレクトリ名
主なオプション: `-i` (対話的)、`-p` (更新時刻やパーミッションなどを元のファイルと同じにする)、`-R` (ディレクトリの中まで一気にコピー)、`-n` (既存ファイルを上書きしない)、`-v` (コピーの実行状況を表示)

「使い方その 1」は、あるファイル 1 つを**他のファイルにコピー**する。例えば `$ cp a.c b.c`↵ のようにする。
「使い方その 2」は、いくつか (1 つでも可) のファイルをまとめて**他のディレクトリにコピー**する。例えば `$ cp a.c b.c enshu`↵ のようにする。この場合、`enshu` は**ディレクトリ** (既に作られているもの) でなければならない。この使い方のもう 1 つの例として、例えば `$ cp ~nide/jugyo/sample/a.c .`↵ とすれば、ファイル `~nide/jugyo/sample/a.c` をディレクトリ「`.`」に、すなわちカレントディレクトリにコピーする。
いずれの場合も、コピー先のファイルが既存であれば、そのファイルを**上書きしてしまう** (そのファイルが書き込み禁止属性でない限り) ので要注意。`-n` オプションを指定するとこれを防げるが、コピー先のファイルが既存であってもエラーメッセージは出ない。
`-p` オプションを指定すると、コピーして新たにできた方のファイルの更新時刻は、**コピー元の更新時刻と同じ**になる。`-p` オプションを指定しない限り、コピーして新たにできた方のファイルの更新時刻は**現在**になる。
`-R` オプションを指定すれば、**ディレクトリとその下を丸ごと**コピーすることができる。例えば `enshu` というディレクトリがあるとして、`$ cp -R enshu enshu.bak`↵ とすれば、`enshu` ディレクトリの下全体を丸ごとコピーできる (このとき、`enshu.bak` というディレクトリが既にあると、そのさらに下に `enshu` ディレクトリが作られてそこにコピーされるので注意)。`-R` オプションを指定しない限り、ディレクトリをコピーすることはできない。

mv: ファイルの改名または移動

使い方その 1: `mv` ファイル名$_1$ ファイル名$_2$
使い方その 2: `mv` ファイル名$_1$ ファイル名$_2$ … ファイル名$_n$ ディレクトリ名
主なオプション: `-i` (対話的)、`-n` (既存ファイルを上書きしない)、`-v` (移動の実行状況を表示)

「使い方その 1」は、あるファイル 1 つの**名前を変える**。例えば `$ mv a.c b.c`↵ のようにすると、ファイル `a.c` の名前は `b.c` に変わる。
「使い方その 2」は、いくつか (1 つでも可) のファイルをまとめて**他のディレクトリに移動**する。例えば `$ cp a.c b.c enshu`↵ のようにする。この場合、`enshu` は**ディレクトリ** (既に作られているもの) でなければならない。こうすると、`a.c` と `b.c` が `enshu` ディレクトリの中へ移動する。
移動 (あるいは改名) 先のファイルが既存であると、そのファイルが**消されてしまう**ので要注意。`-n` オプションを指定するとこれを防げるが、移動 (改名) 先のファイルが既存であってもエラーメッセージは出ない。
指定したファイルがディレクトリであっても通常のファイルであっても、同じ操作で改名や移動ができる[8]。

rm: ファイルの削除

使い方: `rm` ファイル名$_1$ ファイル名$_2$ … ファイル名$_n$
主なオプション: `-f` (強制)、`-i` (対話的)、`-r` (再帰的)、`-v` (削除の実行状況を表示)

ファイルを削除する。例えば `$ rm a.c`↵ で `a.c` が削除される。
`-r` オプションをつけると、**ディレクトリを丸ごと削除** (その下のファイルやディレクトリも含めて) することができる。例えば `$ rm -r enshu`↵ で `enshu` ディレクトリ以下が全て削除される。`-r` オプションをつけないと、ディレクトリを削除することは rm コマンドではできない。

[8] 昔の UNIX では、場合によっては mv コマンドでディレクトリの移動はできなかったことがあった。

-f オプションを指定すると、書き込み禁止のファイルでも確認せずに削除する。このオプションがないと、**書き込み禁止のファイル**を削除しようとした場合には「削除しますか」と**確認を求めてくる**。もちろん、削除する権限のないファイルは「削除しますか」に y と答えても削除はできない。-f オプションの指定が有効なのは、例えば自分のファイル a.c を chmod コマンドで自分でも書き込み禁止に設定してあるとき、$ rm -f a.c↵とすると確認を求められずに削除できる、というような場合である。

ls: ファイルの一覧

> 使い方: ls ファイル名$_1$ ファイル名$_2$ … ファイル名$_n$
> 主なオプション: -a (ドットファイルを省かない)、-d (ディレクトリをそのままリストする)、-F (ファイル種別マーク付加)、-l (詳細リスト)、-R (再帰的)、-r (逆順)、-t (時刻順)

指定したファイル名を**一覧**する (一覧するのはあくまでファイル名であって、ファイルの中身は出力されない)。

指定したファイルの中にディレクトリが混じっていた場合は、そのディレクトリ自身ではなく**そのディレクトリのすぐ下にあるファイル名**を一覧する。また、ファイルが 1 つも指定されなかったら、**カレントディレクトリのすぐ下にあるファイル名**を一覧する。

例えば、引数何もなしで単に $ ls↵とすれば、カレントディレクトリにあるファイル名を一覧できる。

ディレクトリの中を一覧する場合[9]は、通常、名前が「.」で始まるファイル (正式な用語ではないが「**ドットファイル**」と呼ぶことがある) が一覧から省かれる[10]。しかし -a オプションを指定すると、名前が「.」で始まるファイルも一覧に含めるようになる。例えば $ ls -a↵とすれば、カレントディレクトリにあるファイル名をドットファイルも含めて一覧できる。$ ls↵の出力と比較してみよう。特に、「.」(そのディレクトリ自身) と「..」(親ディレクトリ) が一覧に含まれることに注意。

-d オプションが指定されると、引数にディレクトリが混じっていても、そのディレクトリの中を一覧するのでなく、単にそのディレクトリ自身の名が表示される。例えば $ ls -d ~↵と $ ls ~↵の出力を比較してみよう。-d がなければカレントディレクトリのすぐ下のファイル名の一覧が得られるが、-d があるとカレントディレクトリそのものの名前が出力される。

-F オプションが指定されると、一覧されるファイル名のうちディレクトリの後ろには「/」、実行可能属性が与えられているファイル (ファイルの属性については別資料 0.3 節参照) の後ろには「*」がついて出力される。

-l オプションが指定されると、ファイル名だけではなくそのファイルの種別 (種別についても別資料 0.3 節参照) と属性、持ち主やグループ、サイズ、更新時刻など詳細情報が出力される。例えば $ ls -l↵とすれば、カレントディレクトリにあるファイルを、詳細情報込みで一覧する。-a オプションと合わせて $ ls -la↵とすれば、ドットファイルも一覧に含めることができる。

-R オプションが指定されると、ディレクトリ内を再帰的にリストアップする。

-r オプションが指定されると、表示が逆順になる。また、-t オプションが指定されると、表示が更新時刻順になる (-l オプションと併せて試せばそのことがよくわかる)。

mkdir: ディレクトリの作成

> 使い方: mkdir ディレクトリ名$_1$ … ファイル名$_n$
> 主なオプション: -p (深いディレクトリを一気に作成)

ディレクトリを作成する。例えば $ mkdir enshu↵で enshu というディレクトリが作成される。本資料の 1.1.2 節にも別な例が登場している。

-p オプションを指定すると、2 階層以上のディレクトリを一気に作成することもできる。例えば、enshu というディレクトリがまだなかったとしても、$ mkdir -p enshu/part1↵とすれば、enshu ディレクトリとその下の enshu/part1 ディレクトリを一度に作れる。-p オプションには、ディレクトリの作成に失敗してもエラー扱いにならないという作用もある (ディレクトリの作成に失敗するのは、そのディレクトリが既に作成済みの場合などである)。

---

[9] 詳しく言うと、引数に指定したディレクトリの下を一覧する場合、および引数が指定されずカレントディレクトリの下を一覧する場合。

[10] 別資料 0.2.2 節で述べる「ワイルドカード」にも似たような働きがある。すなわち、ワイルドカードの展開結果には、名前が「.」で始まるファイルは含まれない。

rmdir: ディレクトリの削除

使い方: `rmdir` ディレクトリ名$_1$ … ファイル名$_n$　　主なオプション: 略

ディレクトリを削除する。ただし、空のディレクトリ (その下に別のディレクトリやファイルが 1 つもないディレクトリ) でないと削除できない。空でないディレクトリを削除したければ、空にしてから削除するか、`rm -r` を使う。

cd: カレントディレクトリの変更

使い方: `cd` ディレクトリ名　　主なオプション: オプションはあまり使わない

カレントディレクトリを変更する。例えば `$ cd enshu`↵で、現在のカレントディレクトリの下の `enshu` というディレクトリが新たなカレントディレクトリとなる。この時、`enshu` ディレクトリは既に存在していなければならない (なければ `mkdir` コマンドなどで前もって作る)。また、`$ cd ..`↵とすると、現在のカレントディレクトリの親ディレクトリが新たなカレントディレクトリとなる。本資料の 1.2.1 節にも別な例が登場している。

引数を指定しないこともできる。その場合、**ホームディレクトリ**が新たなカレントディレクトリとなる。

pwd: カレントディレクトリの確認

使い方: `pwd`　　主なオプション: オプションはあまり使わない

現在の**カレントディレクトリが出力**される。なお、1.2.1 節で述べた通り、G 棟システムではプロンプトを見ることによっても、カレントディレクトリの絶対パスの最後の部分だけなら知ることができる。

echo: 引数をそのまま出力

使い方: `echo` 引数…　　主なオプション: `-n` (直後で改行しない)

各引数を、空白 1 つで区切りながら**そのまま出力**。例えば `$ echo abc def`↵で「`abc def`」と出力する。

`$ echo -n abc def`↵だと、「`abc def`」と出力した後に**改行しない**という違いがある。ただし、システムによっては echo コマンドに `-n` **オプションがない**ものもあるので注意。

echo コマンドは、オプションを最初に指定しないとオプションと認めないコマンドの一例である。例えば `$ echo abc def -n`↵とすると、「`abc def -n`」と表示して改行する。`-n` がオプション引数扱いされていない。

2.4　オプション引数との衝突防止

オプション引数の形をした引数を、**オプション引数でない引数**として指定したい場合が時にある。

例えば、`~nide/jugyo/sample/a.c` というファイルをカレントディレクトリの `-e` というファイルにコピーしたいとする。`$ cp ~nide/jugyo/sample/a.c -e`↵では「オプションが違います」という意味のエラーになる。cp は、オプション引数が先頭になくてもオプション引数と認めるコマンドなので[11]、最後の「`-e`」がオプション引数の形をしているためにオプション引数と見なされ、しかも cp には `-e` というオプションはないので「誤ったオプション引数が指定された」という扱いになるからである。

このような場合、「`--`」という引数を余分に指定すれば、それより後の引数は**オプション引数と見なさない** (しかも「`--`」という引数自身は無視される)、という約束を持ったコマンドが多い (しかし、そうでないものもある)。cp はそのような約束を持つコマンドの 1 つである。そこで

```
$ cp -- ~nide/jugyo/sample/a.c -e↵
```

とすれば、`-e` は `--` より後にあるためオプション引数と見なされず、しかも `--` が無視されるため `~nide/jugyo/sample/a.c` が `-e` に正しくコピーできる。

「`--`」に関する上述の特別な約束を持たないコマンド (例えば echo がそうである) で同様のことをしたい場合は、コマンド毎に個別に工夫するしかない。例えば echo コマンドで「`-n`」とだけ出力するのはなかなか大変である。`$ echo -- -n`↵としても、echo は `--` を特別扱いしないので「`-- -n`」と出力されてしまう。

[11] システムによってはそうでないこともある。