令和 4 年度地球惑星物理学演習
UNIX その 1
— UNIX の基礎的概念・基本コマンドとシェル —

担当：橋本 恵一 *

テキスト: 堀田 (2004, 2005)
改訂: 吉武 (2006)、太田, 吉武 (2007)、渡邊 (2009)、
横田 (2010)、小寺 (2011)、仲谷 (2012)、
山上 (2013)、伊藤 (2014)、南原 (2015)、
宮寺 (2016)、植村 (2017)、鈴木 (2018)、
髙木 (2019)、湯本 (2020)、坂井 (2021)、
橋本 (2022)

実施日: 2022/4/6

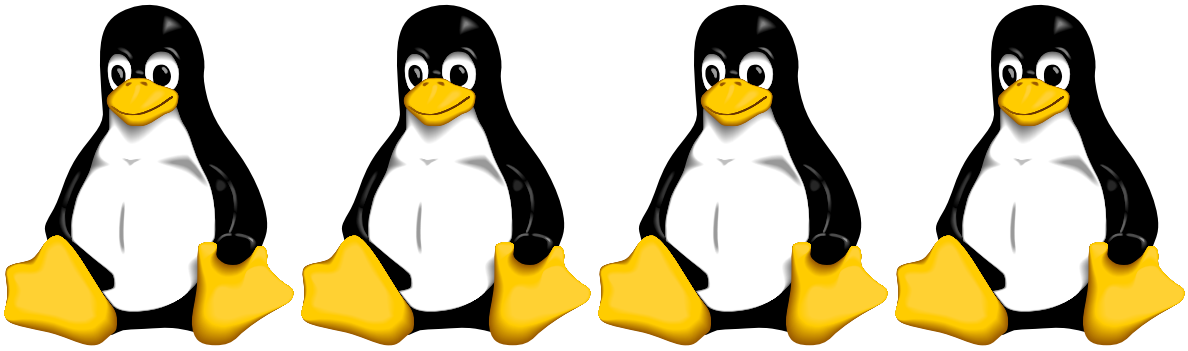

figure: **Tux**, the official muscot of the Linux kernel,
by *Larry Ewing* <lewing@isc.tamu.edu> thanks to *GIMP* (http://www.gimp.org/)

* 大気海洋科学講座 三浦研究室 M1, khashimoto@eps.s.u-tokyo.ac.jp

目次

1 はじめに

今回の目標は UNIX の基礎的な概念を理解し基本的なコマンドを使えるようになること、シェルの機能を利用して楽をする方法を知ること、UNIX について知らないことがあり困ったときに自力で調べて解決する方法を身に付けることです。

2 UNIX・Linux

UNIX とは 1971 年にアメリカの電話会社 AT&T のグラハム・ベル研究所で Ken Thompson と Dennis Ritchie が開発し、主に大学や研究機関で愛用され育ってきた OS です。

UNIX は高級言語である C 言語で書かれたために移植しやすく、現在の OS で基本となる機能 (マルチユーザ / マルチプロセス周辺機器への統一的なインターフェイス、一貫的なファイル / 階層的ファイルシステムなど) を数多く実装したため、広く利用されることとなりました。また、ソースコードが公開されたことで UNIX をベースに独自の機能を追加した派生バージョンが多数出現しました。

現在は UNIX という名前の OS があるわけではなく UNIX 系の OS を総称して UNIX と呼んでいます。以下でも UNIX という言葉はこの意味で使っています。この演習室で使っているのは **Linux** という UNIX の仲間です。

Linux は厳密には UNIX とは呼べないのですが、非常に UNIX に近い OS です。Linux が使えるようになれば他の UNIX を使いこなすのも容易なはずです。

Linux は日本では「リナックス」と読むのが一般的ですが、北米では「ライナックス」と読む人が多く、欧州では「リーヌクス」とか「リノー」とか、 Finnish (?) な読み方が多いそうです。

Linux は、1991 年にヘルシンキ大学の学生 Linus B. Torvalds によって UNIX をモデルに開発されました。Linus 氏は Linux のカーネルという中核部分のみを開発したのでカーネル部分のみを「Linux」と呼ぶのが本来は正しいのですが、現在では大抵、Linux カーネルで動いているシステム全体を指して「Linux」と呼んでいます。

2.1 GNU・Free Software Foundation・Richard Stallman 氏

演習室で動いている Linux のカーネルは Linus 氏が中心となって書かれたものですが、Linux で利用される基本コマンドやコンパイラなどはほとんど全て the GNU Project の製品です。

the GNU Project とは、1985 年の春、MIT の人工知能研究所にいた Richard M. Stallman という人物が「ソフトウェアは “free” であるべき」という彼の理想を実現するために立ち上げたプロジェクトで、彼の設立した Free Software Foundatoin (FSF) というボランティア団体により運営されています。Richard Stallman 氏が the GNU Project において目標としたのは全てが “free” なライセンスで提供される[*1]、UNIX 互換のシステムを作り上げるという大変なものでした。Richard Stallman 氏が書いたソフトウェアには、GNU Emacs やコンパイラ gcc[*2]、デバッガ gdb, bison, ld, /bin/sh など、皆さんがこれからお世話になるであろうソフトウェアが信じられない程沢山あります。

3 CUI と GUI

Microsoft Windows や macOS[*3]は GUI (**G**raphical **U**ser **I**nterface) と呼ばれる仕組みでの操作が中心となっているため、マウスでアイコンをクリックするなどの操作で多くのことが行えます。これに対して、UNIX ではこの GUI に加えて CUI (**C**ommand-line **U**ser **I**nterface あるいは **C**haracter **U**ser **I**nterface) という仕組みによる操作法も良く使われます。この CUI においては、コンピュータと主に文字のやりとりをすることでさまざまな操作を行います。GUI と比較すると CUI の操作方法は敷居が高いものですが、その分、慣れてしまうと GUI と比べて動作が軽快に行えるなどのさまざまな利点があります。たとえば作業の自動化・再利用などは GUI ではほとんど不可能ですが、CUI では実にスマートに実現できます。

逆に言えば、操作に慣れないままだと、単なる使いにくいコンピュータになってしまいます。使い方が分かってしまうまでは不便かも知れませんが、積極的にマシンに触って、早いうちに CUI の操作になれましょう。

CUI の操作はターミナルウィンドウとよばれるウィンドウの中で行われます。ターミナルウィンドウを起動してみましょう。

以後の作業は全てこのウィンドウの中で行います。以降、「ターミナル」とか「シェル」とかいう言葉はこれを指していると考えて下さい。

4 コマンド・オプション・引数

CUI ではコンピュータに**コマンド**と呼ばれる、文字列からなる命令をターミナルに打ち込むことでさまざまな操作を行います。

[*1] 現在 GNU Project は GPL(General Public License) と呼ばれるライセンスを用いていますが、このライセンスの求める “free” は必ずしも無料であることを意味しません。また著作権の放棄とも少し異なる概念です。ライセンスの条項は極めて長いですが、概ね、プログラムについて実行、改変、再配布を許可し、2 次生産物に対して同じ GPL ライセンスを適用することを義務付けることで、公的に開かれたプログラム群を作ろうとする枠組みだと把握しておけばいいと思います。こういったライセンスはかなり色々あります。

[*2] これから Fortran の演習で使う gfortran も gcc の一部です

[*3] 実は macOS も UNIX です。なので演習で習うコマンドは macOS でも使うことができます。しかし、UNIX コマンドには系統があって、Linux には GNU 系コマンドが入っている一方、macOS には（少し古めの）BSD 系コマンドが入っています。そのためオプションなどの挙動が異なることがあり、特に awk とか grep はかなり違っていたりします。計算機演習の講義資料はすべて GNU 系コマンドを前提に作られているので、計算機演習では macOS ではなく、Debian という Linux の asano という計算機を使うことにしています。

ターミナルウィンドウの中には

```
~$
```

のような表示があると思います。これを**プロンプト**といいます。プロンプトはターミナルが入力待ちの状態であることを示すものです。ターミナルウィンドウを終了するときにはプロンプトにつづけて

```
~$ exit
```

と入力しリターンキーを押します。この操作をしないでログアウトしたりすると後に述べるヒストリの保存が行われなかったりと、色々不都合がありますのでログアウトする前に必ず `exit` と入力する癖をつけるようにして下さい。

では早速コマンドを入力してもらいたいのですが、最初に以下のコマンドを入力してください（自前の PC で UNIX の仮想環境を使っている人は入力しないでください)。

```
~$ ssh asano
```

これによって、演習室にある asano という Linux マシンにログインします。初めて ssh でログインする際には

```
The authenticity of host 'asano (133.11.231.4)' can't be established.
ECDSA key fingerprint is ef:e0:a3:1d:06:ba:b5:16:05:1f:aa:eb:a2:83:70:6a.
Are you sure you want to continue connecting (yes/no)?
```

などと尋ねられますが、気にせず

```
 yes
```

と入力しましょう。すると、以下のようにパスワード入力を求められます。

```
 Warning: Permanently added 'asano,133.11.231.4' (ECDSA) to the list of
known hosts.
s222601@asano's password:
```

ここでマシンのログインパスワードと同じパスワードを入力してください。以上によって asano にログインすることができるようになりました。今後の演習でも、演習室の mac を使用する場合には必ず最初に ssh asano を入力し、asano にログインするようにしましょう。

それでは気を取り直して早速コマンドの練習をしていきましょう。プロンプトに続けて

```
~$ echo -e 'Hello \n World!'
```

と入力し、リターンキーを押して下さい。

```
Hello
 World!
```

のような出力が返ってきたと思います。今入力した `echo -e 'Hello \n World!'` の全体、あるいはその内の先頭の `echo` の部分をコマンドといいます。また、この例の場合、- (ハイフン) ではじまる `-e` の部分を**オプション**、最後の `'Hello \n World!'` の部分を**引数** (ひきすう、argument) と呼びます。**オプション**はコマンド

の動作に変化を加える目的で使われ、**引数**はコマンドの動作の対象 (文字列とか、ファイル、ユーザとか) を指定するのに使われます。

シェルはユーザからの入力を受けると、スペースで区切って (tokenize して) コマンドに渡します。従ってスペースはただの区切りの意味しか持ちません。但し、「'」や「"」によって囲まれた文字列は 1 つの "単語" (token) としてコマンドに渡されます。その際、両側の「'」や「"」は取り除かれます。

このように「␣';[]\!#$^&*()"<>?|~`」(␣ は半角スペース) の各文字および改行はシェル (正確には bash; シェルや設定によってこのような文字は異なる) にとって特別な意味を持っているので、これらをただの文字として認識させるにはその文字の前に「\」を置くことで エスケープ する必要があります。

また、「'」や「"」で囲んでも同様にエスケープすることができますが、「'…'」の中で「'」を含むことは (bash では) できません。さらに「"…"」の中で「"」と「$」を使う場合にもやはり「\」を前置する必要があります。

英語の命令文は例えば Hold me tight. のように、先頭に動詞 (hold), 続けて目的語 (me) や副詞 (tight), という形をしていますが、オプションは副詞に、引数は目的語に相当していると思うとよいでしょう。

以上まとめると、UNIX のコマンドは 1 つのコマンド、0 個以上のオプション、0 個以上の引数、をそれぞれ 1 つ以上のスペースで区切ったもの、という形をしています。最後にリターンキーを押せばコマンドが実行されます。

練習課題

先ほどの `echo -e 'Hello \n World!'` のオプションを-n にしたり無しにしてみましょう。また、エスケープシーケンスを\n から\c や\v に変えてみましょう。結果を見てこれらのオプションやエスケープシーケンスの意味を考えて下さい。ターミナルに `man echo` と入力すると、答えが分かります。

5 man コマンド

`man` コマンドは UNIX を使えるようになる上で非常に重宝するものです。

`man` は manual の略で、その名の通り各種コマンドのマニュアルを表示するコマンドです。例えば

```
~$ man echo
```

と入力してリターンを押せば `echo` のマニュアルが表示されます。マニュアルは `less` というプログラムによって表示され[*4]、スペースを押せば次のページにスクロールします。 `less` から抜けるには q キーを押します。`less` 画面の中で / (スラッシュ) をタイプし、続けて文字列を入力してリターンを押すとその文字列を検索することもできます。 n キーを押せば次の、Shift+n キーを押せば前の検索候補が表示されます。

コマンド名は分からないが知りたいキーワードがある、という場合には-k オプションをつけて

```
~$ man -k Shiritai-Keyword
```

と入力すれば *Shiritai-Keyword* に関係したコマンドの一覧が表示されます。たくさん表示されすぎて画面に表示しきれない場合は後に述べるパイプの機能を利用して

[*4] 正確には、環境によって異っており、`less` とは限りません。

```
~$ man -k Shiritai-Keyword | less
```

と入力して下さい。q キーを押せば抜けることができます。何か分からないことがあったらすぐに man でマニュアルを読む、という癖をつけて下さい。他にも echo --help, info echo という方法もあります。

6 ヒストリ機能・補完機能・コマンドライン編集機能

6.1 ヒストリ機能

コマンドに複雑なオプションやたくさんの引数をつけることで入力する文字数はどんどん多くなっていきます。そんな中でもう一度同じコマンドを入力したい、一部分変更して入力したい場合にもう一度はじめから入力するのは大変です。

そんなときは Ctrl+p[*5] をタイプしてみましょう。すると一つ前に入力したコマンドが表示されます。さらに Ctrl+p をタイプすることで、どんどん過去にさかのぼってコマンドの履歴をたどることができます（上下カーソルキーでもコマンドの履歴を遡ることができます)。逆に Ctrl+n をタイプすると、今度はコマンドの履歴を古いほうから新しい方にたどって表示させることができます。これを使うと入力が圧倒的に速くなるので積極的に使っていきましょう。

6.2 補完機能

例えば文字数の長いコマンドを入力する場合、またコマンド名をきちんと覚えていない場合、どのようにコマンドを入力したらよいでしょうか。ここでは Tab キーを利用します。ある程度までコマンドを入力した状態で Tab キーをタイプすると、全てのコマンドの中から、入力した部分を先頭の文字列にとるコマンドをリストアップしてくれます。つまりユーザが入力したいコマンドを予測してくれるわけです。例えばコマンドラインに「ech」とだけ入力した状態で Tab キーをタイプしてみましょう。echo と補完されたと思います。

ちなみに「e」とだけ入力した状態で Tab キーを押しても何も補完されません。これは他にも e から始まるコマンドが存在するためです。このような場合には Tab をもう一度押すと e から始まるコマンド全てが表示されます。

Tab キーはコマンドだけでなく、ディスク内にあるディレクトリやファイル名も予測してくれます。コマンドの引数としてファイル名を入力することが多くあります。その際にこの Tab 補完機能は非常に役に立ちます。

Tab キーを押しまくる癖を付けておけばかなりキータッチの回数を減らせます。タイピングミスや腱鞘炎の防止にもなりますので Tab キーを押しまくる癖を是非身に付けて下さい。

実は、ファイル・ディレクトリ名以外の引数やオプションも Tab キーで補完させるように設定することが可能です。その設定を自分で書くこともできますが、通常は用意されているものを使うのが簡単です。

bash の場合は、 ~/.bashrc に

```
if [ -r /etc/bash_completion ]; then
  source /etc/bash_completion
fi
```

[*5] p は previous、n は next の頭文字です。

などと記述して bash を立ち上げなおして下さい。例えば “ls -h” と打ち込んだ状態で Tab キーを 2 回押すと-h で始まるオプションが表示されると思います。また、“ssh␣” (␣はスペース) と入力した状態で Tab キーを 2 回押すとホスト名が、“cd␣” の後で Tab キーを押すとディレクトリだけが補完されるでしょう。

zsh や tcsh の場合にはさらに高機能な補完のシステムがあり、ユーザの作業効率を高めてくれます。

6.3 コマンドライン編集機能

GNU readline というライブラリの機能により、Emacs でのカーソル操作と同様のキーバインドでカーソルを動かしたりすることができます。たとえば Ctrl+a と打つとコマンドラインの先頭にカーソルを動かすことができますし、Ctrl+e と打つとコマンドラインの最後部にカーソルを動かすことができます。また Ctrl+w と打つと現在のカーソル位置から直前の空白までが、 Ctrl+k と打つとカーソル位置から行末までが切りとられ、Ctrl+y で直前に切り取られた文字列をペーストできます。[*6]

7 ファイルシステム

7.1 ファイルとディレクトリ

UNIX や Windows, Mac OS などほとんどの OS は様々なデータを整理するのに、**ファイル**と**ディレクトリ** (フォルダ) という構造を採用しています。文書や画像などの一連のデータはファイルと呼ばれていて、現実の世界ではルーズリーフに例えるとわかりやすいでしょうか。コンピュータを使い込んでいくと、すぐに大量のファイルがたまっていき収拾がつかなくなってしまいます。これらを整理するための入れ物がディレクトリです。ルーズリーフを束ねるバインダですね。たくさんのファイルを作っていくと、当然ディレクトリも大量に作られて、整理が必要になりますが、その場合は、これらのディレクトリ (子ディレクトリ) をまとめるためにさらにディレクトリ (親ディレクトリ) を作ります。このディレクトリはバインダをためる本棚に相当します。このように、ディレクトリはいくつでも入れ子状にすることができます。以上のように、ファイルとディレクトリの構造をとることによって、データを効率よく整理することができます。

7.1.1 ファイルの名前

全てのファイルには名前が付いています。自分でファイルを作る場合には好きな名前を付けることができますが、UNIX で付けられるファイル名は、多くの場合 255 バイト (半角英数字で 255 文字、日本語のような 1 文字が複数バイトの文字を使うともっと少なくなります) までという制約があります。あまり長い名前を付けても不便なだけなので、分かりやすい名前を付けるのがよいでしょう。

ファイルに名前を付けるときには大文字と小文字の英数字、そしていくつかの記号を使うことができます。大文字と小文字は区別されます。

ターミナルでファイルを操作する可能性があるなら、記号はできれば英数字、ハイフン (-), ピリオド (.), アンダースコア (_) だけを使うようにした方がよいでしょう。他の多くの記号はターミナルでコマンドを入力する際に特別な意味を持つため、ターミナルでファイル名のつもりで入力すると、コマンドが意図しない動作を起こしたりします。(第５節で書いたようにエスケープ記号を前に置けばただの文字として使うことはできますが、やめましょう)。ファイル名の先頭と末尾に記号を使うのも一般に控えたほうが良いとされます。[*7]。ま

[*6] ちなみに Mac の一部の GUI プログラムでもこの操作は可能です。

[*7] 例えば、ハイフン (-) を名前の最初につけるとコマンドの引数のつもりで入力してもオプションとして扱われてしまいます。

た、日本語の入ったファイル名を付けるのも避けた方がよいです。面倒なようですが、どのような場面でもリスクのない命名規則を使うよう心がけるべきです。[*8]

7.1.2 拡張子

ピリオド (.) はファイルの種類を明示したいときに使われることがあります。たとえば index.html という名前のファイルは、最後のピリオド以下の部分「.html」によって HTML 形式のファイルであることが明示されています。これをファイルの**拡張子**と呼びます。

7.1.3 隠しファイル (ドットファイル)

まず、ターミナルに ls /tmp/ と入力してみて下さい。何も表示されませんね。では、次にターミナルに ls -a /tmp/ と入力してみて下さい。今度はピリオド (.) で始まるファイルがいくつか表示されたと思います。

このように、名前がピリオドで始まるファイルはいわゆる隠しファイルとなっており、通常 ls コマンドなどでは表示されません。主にアプリケーションのユーザごとの設定ファイルなど、誤って消去しては困るようなファイルが隠しファイルになっています (e.g., .emacs)。隠しファイルを見るには ls -a とコマンドを入力する必要があります。

7.2 特別な意味・呼び名のあるディレクトリ

7.2.1 ホームディレクトリ

ホームディレクトリとは 1 ユーザにつき 1 つ確保される、そのユーザ専用のディレクトリのことです。ユーザは、特別な理由のない限り、自分の作成したプログラムやデータをホームディレクトリに置いておくのが慣習になっています。

自分のホームディレクトリを示すのにチルダ (~) という記号を使うことができます。皆さんの場合、~ と /home1/s2226??/ は同じディレクトリを意味します。

また、ユーザ X のホームディレクトリを示すのに ~X という記号を使うことができます。例えばユーザ sakai のホームディレクトリは /home2/sakai/ ですが、~sakai/ と書いても同じディレクトリを意味します。

7.2.2 カレントディレクトリ

カレントディレクトリとは現在作業しているディレクトリのことを指します。カレントディレクトリを確認するには pwd コマンド (**p**rint **w**orking **d**irectory の略) を使います。カレントディレクトリを示すのにドット (.) という記号を使うことができます。

7.2.3 親ディレクトリ

カレントディレクトリの一つ上の階層のディレクトリを親ディレクトリといいます。親ディレクトリを示すのに .. という記号を使うことができます。例えばカレントディレクトリが /home2/s222600/matrix/revolutions/ の場合を考えると ../../harry_potter/prisoner_Azkaban/ は /home2/s222600/harry_potter/prisoner_Azkaban/ と同じです。

[*8] https://xkcd.com/327/

7.2.4　相対パス・絶対パス

ディレクトリ名には 2 通りの指定方法があります。それは**相対パス**指定と**絶対パス**指定です。相対パス指定というのは、今自分がいるディレクトリから見たときの相対的なディレクトリの名前です。絶対パス指定というのは、そのファイルの本来の名前で指定することです。ファイルの本来の名前は、'/' (ルートと読みます。) という一番大きな部屋 (その中に、全てのファイルやディレクトリが含まれている) から見た時の名前です。例えばさっきの例を使うとカレントディレクトリが /home2/s222600/matrix/revolution/ の場合 ../../harry_potter/sorcerers_stone/ は相対パス指定、/home2/s222600/harry_potter/sorcerers_stone/ は絶対パス指定です。

練習課題

まず、カレントディレクトリを確認してみましょう。

続いて cd を用いて様々なディレクトリに移動し、pwd でどこにいるのかを確認してみましょう。cd は別のディレクトリを移動するコマンド (Change Direcrory) で、"cd < ディレクトリ名 >"のように使います（cd ./や cd ../ とするとどこに移動するでしょう)。

色々移動して遊んだら、最後に絶対パスでディレクトリ名を指定して最初にいたディレクトリに帰ってきましょう。おかえりなさい。

練習課題

この後の演習に必要なので、必ず実施してください！

この演習用端末にはバグがあり、必要なディレクトリの一つが存在していません。以下のコマンドを実行して作ってやりましょう。

```
~$ mkdir -p ~/Library/Logs/X11
```

さてここで、正しくディレクトリが作られたことを確認するにはどうしたら良いでしょう？

8　ファイル操作

これまでにファイル・ディレクトリの基本的な概念、用語を学習しました。また、man コマンドについても学びました。そこで、今まで使ったコマンドとそのオプションについて man コマンドで調べてみて、提出課題 1, 2 をやってみて下さい。

9　ユーザ・グループ

9.1　スーパーユーザ

UNIX には **スーパーユーザ** (**root**) という特殊なユーザがいます。root ユーザはシステム管理をするためのユーザで、他のユーザのファイルを消したり、ユーザそのものを追加したり消去したり、アカウントを停止したり、他のユーザやシステム全体のプロセスを止めたり、とにかく何でもできるユーザです。演習室では admin210 という学生の管理者集団がボランティアで 演習室マシンのシステムを管理しており、admin210 のメ

ンバーたちが root 権限を持っています。

9.2 ユーザとグループ

ユーザには**ユーザ名**と**所属グループ**という情報があります。ユーザ名は説明するまでもなく、自分のユーザ名です。所属グループは自分の所属するグループであって、ユーザは少くとも一つのグループに所属し、複数のグループに所属することもできます。ユーザの所属するグループのうち一つは特別なグループで**プライマリグループ**と呼ばれ、ファイルなどを作成した時にそのファイルの所有グループとして使われます。

グループへの所属登録自体は管理者 (root) が行います。また、ユーザ名やグループ名には数値で表された識別子 (ID) が関連付けられています。`id` コマンドを使うことで自分や他のユーザの情報を見ることができます。

```
~$ id
```

と打ってみて下さい。

```
uid=50600(s222600) gid=20000(student) groups=20000(student)
```

のような出力が返ってきたと思います。

これによって得られる表示のうち uid はユーザ ID とユーザ名、gid はプライマリグループ ID とプライマリグループ名、groups は自分の所属する全グループ ID とグループ名です。

このように、210 では学生ユーザはみな student というグループに所属しています。admin210 のメンバーになるとこの他 admin (gid=21000) というグループにも所属するようになります。また TA や教官は ta (gid=20090) というグループに所属しています。また、

```
~$ id username
```

でユーザ *username* の情報をみることができます。

9.3 ファイルの所有者

UNIX ではそのファイルが誰のものかを

1. owner (所有者)
2. group (所有グループ)
3. other (他人)

の 3 つに分けて区別しています。1. はそのファイルを所有する者、2. はそのファイルを所有するグループです。3. は所有者でなく 2. のグループにも属さないユーザを表わします。

自分の所有するファイルでも owner を変更することはできませんが (root にはできます)、自分の所有するファイルなら自分の所属するグループのいずれかへファイルのグループを変更できます。このときに利用するコマンドは `chgrp` です。

```
~$ chgrp groupname filename
```

これでファイル *filename* の所有グループが *groupname* に変更されます。ディレクトリの中身も全て所有グ

ループを変えたいのならば「-R」オプションをつけます。

```
~$ chgrp -R groupname dirname
```

但し、自分の所属しないグループを所有グループにすることはできませんので student グループにしか所属していない皆さんは chgrp コマンドを使うことはないでしょう。

9.4 パーミッション

ファイルやディレクトリに対するアクセス許可のことを**パーミッション**といいます。owner/group/other に対して

- r： 読み取り許可
- w： 書き込み許可
- x： 実行許可

を与えることができ、また、複数組み合わせることもできます。これらの意味するところはファイルかディレクトリかで異っており、次の表のようになります

	ファイル	ディレクトリ
r	ファイルの読み込み	ディレクトリ内のファイル一覧取得
w	ファイルへの書き込み	ディレクトリ内のファイル作成・削除
x	実行	ディレクトリ内へのアクセス

ディレクトリ内へのアクセスとは、そのディレクトリをカレントディレクトリにする、そのディレクトリの下のファイルを参照する、ということです。また、ディレクトリに r が付与されていなくても x が付与されていれば、その中のファイルにアクセスすることがでます。(もちろん、ファイルそのものに適切なパーミッション設定がされていなければなりません。)

パーミッションを確認するにはコマンド ls に -l (エル) オプションをつけて使います。

```
~$ ls -l
```

これで出てくる

```
-rw-rw-r-- ··· (省略)
```

のような表示がそのファイルに設定されているパーミッションになります。

この 10 個のパートは

最初の 1 文字が	d ならディレクトリ、- ならファイル
それ以降	3 文字 1 セットで解釈 最初の 3 文字　持ち主 次の 3 文字　グループ 最後の 3 文字　他人
	そのそれぞれの意味 r　読んで良い (read) w　書いて良い (write) x　実行して良い (excute) -　(その場所に応じて) 許可しない

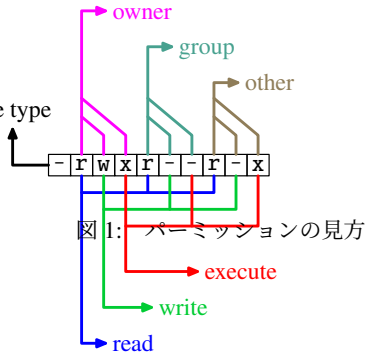

図 1:　パーミッションの見方

のように解釈します。パーミッションは owner/group/other ごとに

許可の種類	r	w	x	- (不許可)
表す数	4	2	1	0

の 3 数の和で表されることもあります (ディレクトリであるかどうかは評価されません)。例えば、-rw-rw-r-- や -rwx---r-x はそれぞれ

-rw-rw-r-- の場合	
owner : r (4) + w (2) + 0 =	6
group : r (4) + w (2) + 0 =	6
other : r (4) + 0 + 0 =	4

-rwx---r-x の場合	
owner : r (4) + w (2) + x (1) =	7
group : 0 + 0 + 0 =	0
other : r (4) + 0 + x (1) =	5

で 664 および 705 と表現されます。この 3 数の和で表現する方法はビット演算 (2 進数演算) を利用していることから、「owner に r ビットが立っている」などの言い方をすることもあります。

9.5　chmod

ファイルやディレクトリのパーミッションを変更するには chmod コマンド（change mode）を用います。使い方については man コマンドで調べてみましょう。

9.6　umask

umask 値はファイルやディレクトリを新規作成するときのパーミッションをいくつにするかという設定です。この値はファイルなら 666 の補数、ディレクトリなら 777 の補数であらわされます。どういうことかというと、umask=022 のとき新規作成されるファイルは (繰り下げを許さない) 8 進数の引き算で 666 - 022 = 644 で 644 となり、umask=022 のとき新規作成されるディレクトリは (繰り下げを許さない) 8 進数の引き算で 777 - 022 = 755 で 755 となる、といった具合です。

他のユーザに自分のファイルを見られたくないと思ったら umask 値を 044、他のユーザに自分のファイルを編集されたくないと思ったら umask 値を 022、見せたくもないし編集もさせたくないと思ったら 066、といった感じです。ちょっと複雑ですが、理解しておきましょう。

この umask 値はデフォルトでシステム規定の値となっていますが、umask コマンドでその値を知ることができます。また、umask コマンドに設定したい umask 値を引数で与えてやれば設定をかえることもできます。

```
~$ umask
```

とすると現在の umask 値が表示され、

```
~$ umask 002
```

とすると umask 値が 002 に設定されます。210 のデフォルトでは皆さんの ~/.bashrc ファイルで umask 値が 022 に設定されています（つまり所有者以外は編集ができない）。~/.bashrc ファイルが何か知りたい人は

```
~$ man bash
```

としてみて下さい。

10　プロセス管理

例えば、firefox で遊んでいるとき、突然 firefox が何の反応もしなくなり、ひどいときには画面全体が凍り付いてしまうことがあるかもしれません。何かトラブルがあったとき、いちいち管理者さんを呼んで処理してもらうようでは、どちらも結構面倒くさいですね。何かトラブルがあったときに、簡単な対応なら自分一人で出来るようになることを目指しましょう。

10.1　プロセス

UNIX では同時に複数のプログラムが動作することができますが、これら、動作中のプログラムは**プロセス**という単位で管理・処理されます。試しに

```
~$ ps
```

と打ってみましょう。

```
 PID TTY          TIME CMD
5252 ttyp1    00:00:00 bash
5254 ttyp1    00:00:00 ps
```

のような出力が得られます。各プロセスには**プロセス ID** (pid) という番号がつけられています。上の例では 1 列目がそれにあたります。また、プロセスにもファイルと同様に所有者がいます。通常は、そのプロセスを起動したユーザがそのプロセスの所有者になります。この情報は

```
~$ ps u
```

と打つことで明らかになります。

```
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
s222600  28161  0.0  1.1   2444  1128 ttyp0    S    18:39   0:00 -bash
s222600  28162  0.0  1.1   2436  1124 pts/0    S    18:39   0:00 -bash
s222600  28184  0.0  1.2   2904  1208 ttyp0    R    18:45   0:00 ps u
```

ただの ps より詳しく出力されました。

```
~$ ps ux
```

で、他のウィンドウで動作しているプロセスも表示されます。

```
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
s222600  28111  0.0  1.0   1996   968 ?        S    18:39   0:00 bash -f /home
s222600  28146  0.0  0.8   2132   792 ?        S    18:39   0:00 /usr/bin/ssh-
s222600  28152  0.0  1.3   2568 1312 ?        S    18:39   0:00 xclock -geome
                                                              ..... (省略) .....
s222600  28174  0.0  1.1   2436 1128 ttyp1    S    18:39   0:00 -bash
s222600  28176  0.0  1.4   2700 1328 ttyp1    S    18:39   0:01 ssh -l yoshit
s222600  28241  0.0  1.1   2436 1124 ttyp2    S    18:58   0:00 -bash
s222600  28246  0.0  1.2   2904 1208 ttyp0    R    19:01   0:00 ps ux
```

但し、これは途中で出力が切れています。切れないようにするには、オプション w をつけます。

```
~$ ps uwx
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
s222600  28111  0.0  1.0   1996   968 ?        S    18:39   0:00 bash -f /home
1/s222600/.xsession
s222600  28146  0.0  0.8   2132   792 ?        S    18:39   0:00 /usr/bin/ssh-
agent /home2/s222600/.xsession
s222600  28152  0.0  1.3   2568 1312 ?        S    18:39   0:00 xclock -geome
try 80x80+5+5 -fg black -bg white -hl black
s222600  28153  0.0  1.5   2616 1464 ?        S    18:39   0:00 xload -geomet
ry 80x80+90+5 -fg black -bg white -hl red -update 10 -jumpscroll 1
s222600  28154  0.0  1.9   3100 1848 ?        S    18:39   0:00 kterm -geomet
ry 78x5-5-5 -C -name login -n login -fg black -bg white
                                                        ..... (以下略) .....
```

他人のプロセスも見たいという時には、オプションとして、a を使います。例えば、

```
~$ ps au
```

としてみて下さい。

また、プロセスには親子関係があり、全てのプロセスはある親プロセスから起動されます。UNIX では init というプロセスがコンピュータの起動時に実行され、全てのプロセスはその親をたどっていくと init に辿り着きます。

この様子は pstree コマンドにより見ることができます.

10.2 kill

ここでは、まず、プロセスの強制終了の方法を説明します。ためしに、xload という、マシンにどれだけの負荷がかかっているかを表示するプログラムを動かして強制終了してみましょう。

```
~$ xload &
~$ ps x
```

と打ってみて下さい。

```
  PID TTY      STAT   TIME COMMAND
28111 ?        S      0:00 bash /home2/s222600/.xsession
28146 ?        S      0:00 /usr/bin/ssh-agent /home2/s222600/.xsession
28152 ?        S      0:00 xclock -geometry 80x80+5+5 -fg black -bg white
28154 ?        S      0:00 xterm -geometry 78x5-5-5 -C -name login -n xterm
28156 ?        S      0:02 emacs -geometry 75x38-5+5 -bg white -fg black
28159 ?        S      0:00 uim-xim
28160 ?        S      0:01 fluxbox
28161 ttyp0    S      0:00 -bash
28162 ttyp0    SN     0:00 xload
28163 ttyp0    SN     0:00 ps x
```

このように出力が得られます。ここで、xload の PID (プロセス ID) を見てみましょう。この場合は、28162 ですね。(皆さんの出力では違う数のはずです。注意。) プロセスを強制終了する為には、kill というコマンドを用います。

```
~$ kill (PID)
```

というように用います。ここでは、

```
~$ kill 28162
```

としてみましょう。xload が消えてしまいましたね。このように、kill コマンドは、あるプロセスを強制終了します。何かのプロセス (firefox、実行型ファイル等) が暴走してしまってそれを終了したいとき、よく使います。

firefox 等がピクリとも動かなくなってしまった時は、kill コマンドで終了するしかないわけですが、時と場合によっては kill コマンドをうってもプロセスを終了できないときがあります。その場合は、

```
~$ kill -HUP (PID)
```

と、オプション -HUP をつけてみて下さい。それでもだめなら、

```
~$ kill -KILL (PID)
```

とオプションを変えます。但し、-KILL は、最後の手段です。多用しないように注意して下さい。

また、あるプロセスを kill すると、その子プロセスも kill されます。

10.3 jobs、フォアグランド、バックグランド

先ほど、xload を消してしまったので、xload をもう一度出してみましょう。

```
~$ xload
```

すると、xload がまた現れましたね。しかし、コマンドを打ち込んだターミナルウィンドウにはプロンプトがかえってこず、そのターミナルウィンドウが使えなくなってしまいます。これは、プロセスが**フォアグラウンド**で実行されている (フォアグラウンドジョブと呼びます) 状況です。フォアグランドジョブの強制終了するには Ctrl+C とします。

フォアグラウンドがあるなら、**バックグラウンド**もあります。

```
~$ xload &
```

と、コマンドに & を付けると、ジョブをバックグラウンドに送ることができ、複数の作業を同時に行うことができます。今度は、

```
~$ jobs
```

と入力してみて下さい。

```
[1]+  running                 xload
```

のように出力されます。1 列目の数字はジョブ番号と呼ばれるもので、ジョブを管理する上で使われます。次の + (or -) は現在フォアグラウンドで実行されているジョブ (**カレントジョブ**) に対して +, その他のジョブに対して - が表示されます。特に指定が無ければ、以下のジョブの制御コマンドはカレントジョブに対して行われます。その次の Running / Suspended / Stopped というのは Running が実行中、Suspended, Stopped は一時停止中を意味します。これらは、日本語で出力されることもあります。

先ほどは、PID を用いて `kill` しましたが、ジョブ番号を用いて `kill` することもできます。`kill` のあと、`%` をつけてジョブ番号を指定します。

```
~$ kill %1
~$
[1]+  Terminated              xload
```

フォアグラウンドからバックグラウンドにジョブを送ることもできます。

```
~$ xload
```

としたあと、Ctrl + Z と押すと、フォアグランドジョブを一時停止できます。そこで、`jobs` を入力すると、

```
~$ jobs
[1]+  Stopped                 xload
```

となっていますから、この後に

```
~$ bg %(job-number)
```

と入力すると、このジョブはバックグランドで実行されます。具体的には、この場合は、

```
~$ bg %1
[1] xload &
```

となるはずです。逆に

```
~$ fg %(job-number)
```

とするとバックグランドジョブをフォアグランドに移行できます。その他、バックグランドジョブの一時停止には `kill` に `-STOP` オプションを付けて

```
~$ kill -STOP %(job-number)
```

というコマンド、先ほども説明しましたが、バックグランドジョブの強制終了には

```
~$ kill %(job-number)
```

を用います。このように、自分のプロセス、ジョブにたいしては、自分で制御することが出来ます。自分のプロセスが暴走してしまったようなときは、自分で責任を持って対処しましょう。また、他人のプロセスが暴走しているときもあります (上でも説明しましたが、`ps au` で他人のプロセスも見ることができます)。この場合は、そのプロセスを `kill` するよう、管理者に連絡して下さい。

10.4 top, w

現在のプロセス稼働状況を一覧するには、

```
~$ top
```

現在の 5w (who what where when how-long) を知るには、

```
~$ w
```

などというコマンドがあります。詳しくは `man` などで調べてみてください。

10.5 ジョブとプロセス

ジョブとプロセスはどちらも同じ UNIX 上の処理の 1 単位です。違いを簡単に言えばプロセスはカーネルにより管理される、処理の最小単位であり[*9]、ジョブは 1 つ以上のプロセスの集まりで、シェルにより管理されると言えます。例えば、あるプロセスが実行過程で別のプロセスを生成することがあります。この時、元のプロセスを “親プロセス”、生成されたプロセスを “子プロセス” といいます。この 2 つはプロセスとしては別のものですが、ジョブとしては同じジョブに該当します。

11 パイプ・リダイレクト

ここで紹介するパイプとリダイレクトの機能、特にパイプは UNIX の哲学の根幹にあるものです。しっかり理解しましょう。

11.1 標準入力・標準出力・標準エラー

UNIX の各プログラム (正確には、各プロセス) は**標準入力**、**標準出力**、**標準エラー** (標準エラー出力ともいう) と呼ばれるデータの入口と出口、非常口を持っています。例えば、 `a2ps` というプログラムは標準入力 (入口) からテキストファイルを取り込み、PS (**P**ost**S**cript) ファイルという型にして、標準出力 (出口) に出すというプログラムです。特に指定しない限り標準出力のデータはシェルの動いているターミナルに送られます。`ls` はディレクトリの内容を読み込んで、標準出力へと出力するプログラムですが、上の理由によって画面上にファイルの内容が表示されるのです。同様に特に指定しない限り標準エラーのデータもシェルの動いているターミナルに送られます。また、特に指定しない限りターミナルに対するキーボード入力が標準入力となり

[*9] 最近の UNIX ではプロセスより小さなスレッドという単位がありますが。

ます。

11.2 リダイレクト

標準出力を別の場所に送ることもできます。

```
~$ ls > ls.log
~$ cat ls.log
Mail/  News/  kadai.tex  kadai1.ps  kadai2.txt  kadai3.txt  kadai1.tex
kadai1.txt  kadai2.ps  (← ※本当は縦 1 列に並ぶはずです)
```

上のように '>' を使うことによって ls の標準出力が ls.log へと書き込まれました。このことを、標準出力の**リダイレクト**といいます。

```
~$ ls >> ls.log
```

のように、'> >' を使うとファイル ls.log がすでに存在するときは、その後ろに出力内容が追加 (append) されるようになります。ファイル ls.log がすでに存在するときに '>' を使うと、コマンドが何も出力しなかった場合でももとのファイルの中身は消えて新しくファイルが作られてしまうので注意して下さい。

また標準エラーを別の場所に送ることもできます。例えば

```
~$ awk --hoge > hoge.txt
```

としても hoge ファイルは空のままで出力が端末に出て来てしまいます。これは awk --hoge の出力が標準エラーに出ているためで、これをファイルに落とすには

```
~$ awk --hoge 2> hoge.txt
```

のようにします。後に述べるパイプに送る場合は 2>&1 として、先に標準エラーを標準出力にリダイレクトします。

同様にして、標準入力をリダイレクトすることも可能です。例えば、入力した文字列を逆さまにして出力するコマンド rev に対して

```
~$ rev < ls.log
```

として rev の標準入力に ls.log のファイル内のデータを送り各行の文字の順番を逆にして表示させます。

11.3 パイプ

今度は、次のようにして /etc/ ディレクトリの中身を表示させて下さい。

```
~$ ls -al /etc/
```

中身が多すぎて表示し切れません。このような時こそ、標準入出力の仕組みが役立つときです。次のようにすると、ls -al の標準出力を less の標準入力に送り込んで、少しずつ読むことができるようになります。less はファイル内容を 1 ページずつ見るようなコマンドです。less から抜けるときは q を押します。

```
~$ ls -al /etc/ | less
```

上のような ‘|’ を使った標準入出力のやり取りを**パイプ**と呼びます。パイプの便利さを実感するために次の例を見てみましょう。ホームディレクトリの容量を各ディレクトリごとに分けて計算し、その結果を大きい順に並べ替えて、上位 25 個を表示する、というなかなか複雑な操作も、パイプの機能を利用して次のように一行でできてしまいます。

```
~$ du -k ~/ | sort -nr | head -n 25
```

“du -k ~/” というコマンドが、ホームディレクトリ以下の各ディレクトリの容量を、キロバイト単位で計算し出力する。“sort -nr” は du の出力を入力として受け取り、数字ベースで降順で出力する。最後の “head -n 25” は sort の出力を入力として受け取り、その最初の部分 25 行だけを出力する。といった具合です。それぞれのコマンドの機能の詳細は、man コマンドで調べて下さい。

この例の sort のように、標準入力を読んで何らかの処理をし、結果を標準出力に書き出すプログラムをフィルタプログラム、あるいは単にフィルタと呼びます。UNIX 上で利用できるコマンドの多くは、フィルタとして使われることを意識して作成されています。

UNIX の哲学の一つとして、「KISS」 (Keep It Simple and Smart) とか「Small is beautiful.」というものがあります。一つ一つは小さな機能しか持たないコマンドをいくつか組み合わせて、いろいろな仕事を上手にこなそう、という考え方です。フィルタプログラムとパイプの機能は、この哲学を実現するために発明されたもので、まさに UNIX の哲学の根幹をなすものです。

また、先程述べたように、標準エラーをパイプに送る場合は 2>&1 として、先に標準出力にリダイレクトしなければなりません。例えば

```
~$ awk --hoge 2>&1 | less
```

のようにすると、 awk --hoge の出力を less で読むことができます。提出課題 3 をやってみてください。

12 シェル

12.1 シェルの働き

UNIX を使っていると、シェルという言葉をよく耳にしますが、このシェルというのはユーザーとコンピュータの間に立って、ユーザーの命令をうまくコンピューターに伝えてくれるプログラムのことです。UNIX など最近の OS ではたくさんのプログラムが集まって様々な命令を実行していますが、これらのプログラムは人間の言語を理解できません。そこで、人間の出す命令をプログラムがわかる言葉に翻訳して伝える働きをするのがシェルです。先ほど、皆さんが打ち込んだ ls や cp 等の命令はシェルを通して初めてコンピュータに伝わるわけです。また、シェルは単なる翻訳機の働きだけにとどまらず、ユーザーを助ける様々な機能を持っています。

12.2 シェルの種類

UNIX には、たくさんの種類のシェルがありますが、それらの多くに共通するのは、キーボードから ls や cp 等の文字列を打ち込んで命令を伝える方式をとっていることです。プリントの始めにも書きましたが、この仕組みは CUI と呼ばれていて、この CUI の流儀に慣れるのが、UNIX 上達の早道といえます。UNIX のシェルは大きく分けて B-shell 系と C-shell 系の二つのシェルがあります。

Bourne shell
: 最も初期に作成されたシェルで、B-shell の由来となっているプログラムです。以下にあげる新しいシェルと比較とすると機能が少ないので、普段から使っている人は見かけません。但し、現在でもたいていの UNIX 系の OS に標準で入っているので、多くのシェルスクリプトはこのシェルで動くように作られています。

C shell
: C-shell 系の由来となっているシェルで、シェルスクリプトの書き方が C 言語の文法と似ているのでこう呼ばれます。また、Bourne-shell と比較してユーザーのコマンド入力の手間を省いてくれる機能が強化されています。

tcsh (TENEX C Shell)
: C shell の機能強化版で、コマンド入力がさらに効率的に行えます。前回の講義で出てきた Emacs と似たような操作が行えるのも特徴のひとつです。BSD 系の UNIX の多くでは標準的に採用されているようです。最近まで主流となっていたシェルのうちのひとつですが、csh の文法に問題があることが分かり[*10], 現在では少しずつ人気がなくなってきているようです。

Bash (Bourne-Again SHell)
: その名のとおり、Bourne shell の機能強化版で、tcsh の機能などを取り入れています。Linux ではこのシェルが標準として採用されており[*11]、tcsh と人気を二分しています。

この他にも、ksh や zsh などの多くのシェルがあります。なかでも zsh は大変に高機能で便利ですので、bash に慣れてきたら zsh への移行をお薦めします。

演習室の環境でログインシェルを変更するには ypchsh コマンドを使います。ただ、変更する場合はその前に `chsh` と `shells` を `man` して下さい。

12.3 シェルの機能

最初の方で、Tab キーを押すとコマンド入力などの作業を助けてくれる便利な機能があると説明しましたが、このセクションでは bash を例にしてそれらの機能について解説していきます。

12.3.1 ヒストリ機能と補完機能

これは既に説明しました。

12.3.2 ワイルドカード '*', '?'

UNIX マシンを使って作業をしていると、同じ種類のファイルなどをまとめて処理してしまいたいという場面が現れるはずです。そのような場合には、ワイルドカードと呼ばれる文字、'*' と '?' を使うのが非常に便利です。例えば `kadai1.txt`, `kadai2.txt`, `kadai3.txt` というファイルをまとめて `kadais/` というディレクトリにコピーしたいとします。

[*10] http://www.faqs.org/faqs/unix-faq/shell/csh-whynot/ とか
[*11] 現在、asano のデフォルトで採用されているのもこの bash です。

```
~$ mkdir kadais
~$ cp kadai?.txt kadais/
~$ ls kadais/
kadai1.txt  kadai2.txt  kadai3.txt
```

‘?’ が入力されると、シェルは ‘?’ を任意の 1 文字に置き換える作業をしてから、他のプログラムに命令を伝えます。上の例では、‘?’ が ‘1’, ‘2’, ‘3’ の三つの文字に置き換えられ、cp が実行されています。次に、kadai1.ps, kadai1.tex, kadai2.ps, kadai.tex というファイルをまとめて kadais/ というディレクトリにコピーしたい場合を考えます。

```
~$ cp kadai* kadais/
~$ ls kadais/
kadai.tex   kadai1.tex   kadai2.ps    kadai3.txt
kadai1.ps   kadai1.txt   kadai2.txt
```

ここで現れた ‘*’ はシェルの働きによって任意の 0 字以上の文字列と置き換えて処理されます。つまり、この例では kadai という文字で始まるファイルが全て cp される対象となるわけです。同様に、ps という文字で終わるファイルを全て cp しようと思ったら、次のようにすればうまくいきます。

```
~$ mkdir psfiles
~$ cp *ps psfiles/
~$ ls psfiles
kadai1.ps   kadai2.ps
```

また、 kadai という文字で始まる全てのファイルを消去したい場合は、

```
~$ rm kadai*
```

とすればよいはずです。また、ワイルドカードはファイル名のみならず、ディレクトリの名前に対しても使用することができます。 test/test.txt, foo/foo.txt という異なるディレクトリの下にあるファイルをまとめて hogege/ と言うディレクトリにコピーしたいときは、

```
~$ cp */*.txt hogege/
```

とすればうまくいきます。

この様に、大変便利なワイルドカードですが、予期しないファイルを処理してしまうこともあります。特に、rm コマンドとあわせて使うときには注意して下さい。

なお、 ‘*cs’ や、 ‘*’ のように入力しても、 .emacs のように、 ‘.’ で始まるファイル名には変換されません。そうして欲しい場合には ‘.*’ のように、明示的に先頭に ‘.’ を付けます。

12.3.3 エイリアス (alias)

シェルの話に限らず、UNIX ではよく alias という言葉が良く出てくるのですが、alias とは別名のことです。bash などのシェルでは、コマンドに別名をつけて登録することができます。この別名登録をするためのコマンドが alias で、

```
alias 別名=’本来打ち込むべきコマンド’
```

として使います。

少し前に習った less を使ってホームディレクトリにある .aliases というファイルを開いてみて下さい。

```
~$ less ~/.aliases
```

自分の環境の中にこのファイルがあった人は、無事に開けたかと思います。なお、皆さんの環境によってはこの .aliases というファイルが無い場合もあります。その場合は以下の例を見るようにしてください。.aliases の中には、例えば以下のような記述が見つかるかと思います。

```
alias   md='mkdir'
alias   rd='rmdir'
if [ "$TERM" = "dumb" -o "$TERM" = "emacs" ]; then
  alias ls='ls -F'
else
  alias ls='ls -F --color=auto'
fi
alias   lf='ls -F'
alias   la='ls -a'
alias   ll='ls -l'
alias   l.='ls -ld .*'
alias   rm='rm -i'
alias   ..='cd ..'
                              ..... (以下略) .....
```

実は、皆さんが演習室のマシンにログインすると .aliases 内のこれらのコマンドが自動的に実行される設定となっています。したがって、rm と入力すると実際には rm -i が実行されることになります。rm では通常は削除してよいかどうかの確認をしてくれないのですが、このおかげで削除の確認がされるのです。

また、長いコマンドを何度も打ち込む時には、別名登録をしておき簡単な名前で利用するようにしておくことで仕事がはかどることもあります。詳しくは次の講義で取り扱います。楽しみにしておいてください。

12.4 シェルスクリプト

UNIX では UNIX のコマンドやシェルの組み込みコマンドを用いてプログラムし、ユーザが独自の便利な機能を作ることができます。これをシェルスクリプトと呼びます。シェルスクリプトを書けるようになると冗長な作業を繰り返す必要がなくなり、非常に効率よく作業ができるようになるでしょう。

13　提出課題

提出期限は一週間後の 2022 年 4 月 13 日 (水) としますが、何か問題を抱えていて提出が間に合わない場合は期限までに khashimoto@eps.s.u-tokyo.ac.jp へ連絡いただければ各人の状況に合わせて提出期限を延長いたします。柔軟に対応いたしますので相談ください。

touch コマンドを使って `s2226??.txt` (s2226??はみなさんの学籍番号) というテキストファイルを作成し、以下の課題の解答を emacs 等のエディタを使って入力してください。入力し終わりましたらテキストファイルをメールに添付し、件名を「s2226??_Unix1 課題提出」として khashimoto@eps.s.u-tokyo.ac.jp へ送ってください。

課題 1.　— man —

シンボリックリンク (symbolic link) とは何かを調べ、シンボリックリンクを作成するにはなんというコマンドを用いればよいのかと共に、簡単な説明を書いて下さい。

ヒント: `man -k` あるいは `apropos`

課題 2.　— ファイルの操作 —

表にあるコマンドを使って次の操作を行い、どのようなコマンドを入力したかを記入して下さい。但し、日本語で説明を加える必要はありません。１行目にカレントディレクトリの名称を（絶対パスで）書き、２行目以降に `ls` や `cd ???/` 等、入力したコマンドを１行ずつ書いて下さい。

1. ホームディレクトリに移動する。
2. ディレクトリ `dir/` を作成する。
3. 存在するファイルの一覧を表示して作成した `dir/` が存在することを確かめる。
4. `dir/` に移る。
5. 隠しファイルも表示するオプションをつけてファイルの一覧を表示して「`.`」と「`..`」しかないことを確かめる。
6. `hoge.txt` という空ファイルを作成する。
7. 存在するファイルの一覧を表示して作成した `hoge.txt` があることを確認する。
8. 親ディレクトリに移る。
9. ディレクトリ `dir/` を削除する。失敗した場合はどうして失敗したか、どうすれば削除できるか考えてみる。
   （ヒント： `rm` に適切なオプションをつける）

課題 3.　— パイプ —

`/etc` ディレクトリに `.conf` で終わる名前のファイルがいくつあるかを、コマンドライン一行で数えるにはどうしたらよいか考え、その説明を書いて下さい。そして、その方法で数えた結果を２行目に書いて下さい。但し、"一行" というのはコマンドが "List" ではなく、"Simple Command" または "Pipeline" という意味です。

ヒント: ls, wc, パイプ、ワイルドカード

課題を解くにはこのレジュメに書かれていない事柄が必要になるかもしれません。その場合は man コマンドや Google先生 をフル活用して調べてみて下さい。
今回はここまでです。おつかれさまでした。

14 Appendix

14.1 ファイルシステム (詳細)

今回の演習では UNIX のファイルシステムとして、ファイルやディレクトリ、さらにその指定という基礎的な内容について取り扱いました。しかし UNIX をしっかり活用して研究を行う際にはパーティションやそのディレクトリ構造についても知識を得ておく必要があります。この Appendix では簡単にこれらの内容についても取り扱います。

14.1.1 パーティション

ハードディスクには、物理的には一つのドライブを、論理的に複数の領域 (**パーティション**) に分割して使用する機能が用意されています。分割できる個数や容量はマザーボードの BIOS[*12] などにより異なっています。

多くの UNIX のシステムでは (Windows 等でも) この機能を利用して一台のハードディスクを複数の領域に区切って、あたかも複数台のハードディスクがあるかのように利用しています。こうして、それぞれのパーティションについて「システム用の領域」「ユーザのデータ用の領域」のように使い分けておくことで、ファイル情報が消失するなどの事故が起きても、失うデータを小さくすることができます。

また、パーティションごとに異なる OS をインストールして、複数の OS を一台のハードディスクの中に共存させ、マシンの起動時に立ち上げる OS を選択する、という使い方もあります。(いわゆるデュアルブートなど。)

14.1.2 UNIX 的ディレクトリ構造

UNIX ではディレクトリの名前の付け方や利用の仕方にある慣習が存在します。これを知っておくと見通しがよくなり、UNIX への理解が深まるかも知れません。図 2 は UNIX の代表的なディレクトリと、そこに何が格納されているかを示したものです。

[*12] コンピュータに接続されたディスク、キーボード、ビデオカードなどの周辺機器を制御するプログラム群。通常マザーボードや拡張カード上の ROM に書き込まれいるが、最近のパソコンでは更新することが可能となっている。

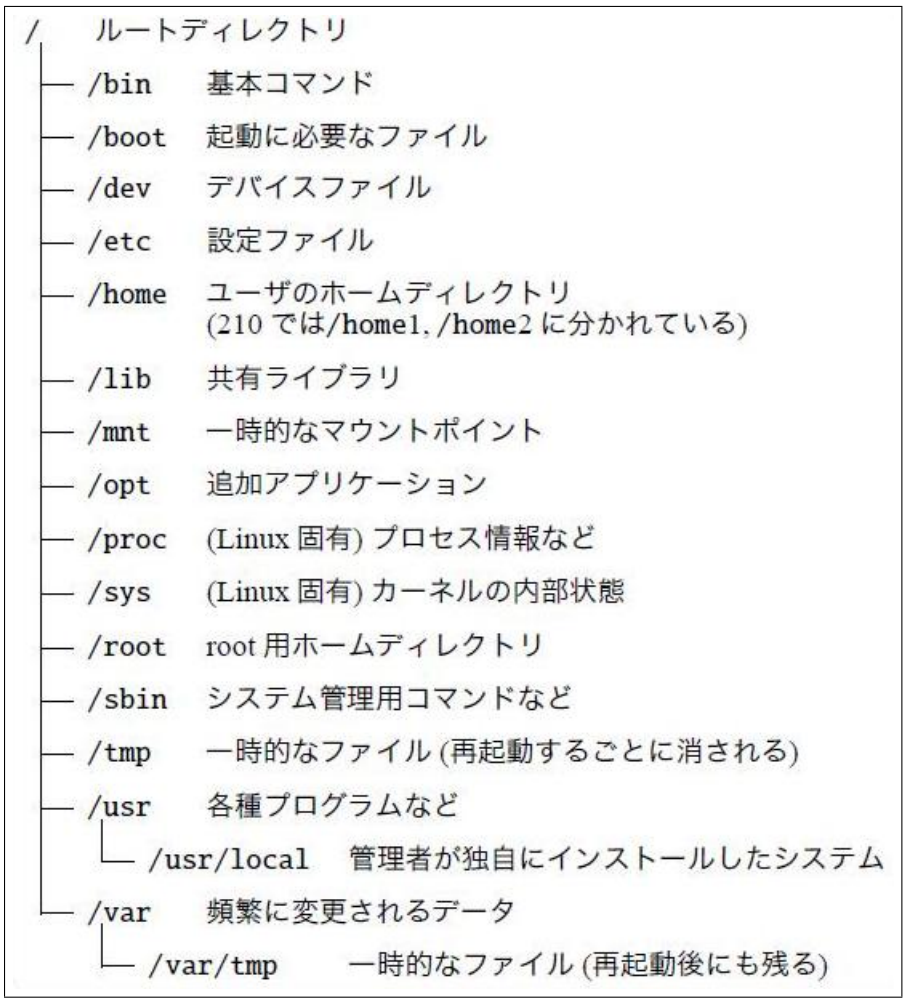

図 2: UNIX での代表的なディレクトリとその中身