コンピュータ・リテラシーI

1~10章 要点まとめ

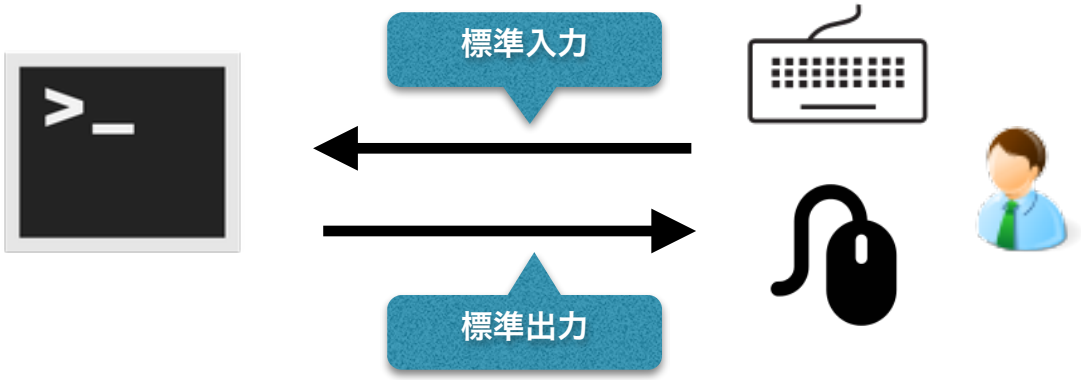
会津大学
d8161105 渡部未来 2014 5/8
資料公開URL : https://github.com/ababup1192/literacy

目的

会津大学の授業コンピュータ・リテラシーでは、主に*Unix*と*Unix*上で動くソフトウェアについて学ぶ授業である。コンピュータ・リテラシーIのハンドアウトは、内容がとても良く卒業まで使えるものである。しかし、復習などの際には、情報量が多すぎるため正しく情報整理が行えない場合がある。そのため、本資料では、独自に1~10章までの内容を簡単にまとめる。さらに、理解が捗るように補足などを加えて説明を行う。

1 ターミナル

*Unix*で作業を行う多くの場合は、ターミナルと呼ばれる文字を主体とした環境 (コマンドラインユーザインターフェース)を扱う。ターミナルでは、キーボードからコマンドを打ち込んだり、結果をターミナルで確認したりする。キーボードやマウスなどから、コンピュータに伝える信号を**標準入力**と呼び。ターミナルに表示された結果を**標準出力**と呼ぶ。(本資料では、標準出力とコマンドの区別を付けるために、コマンドの前には$記号を添える)

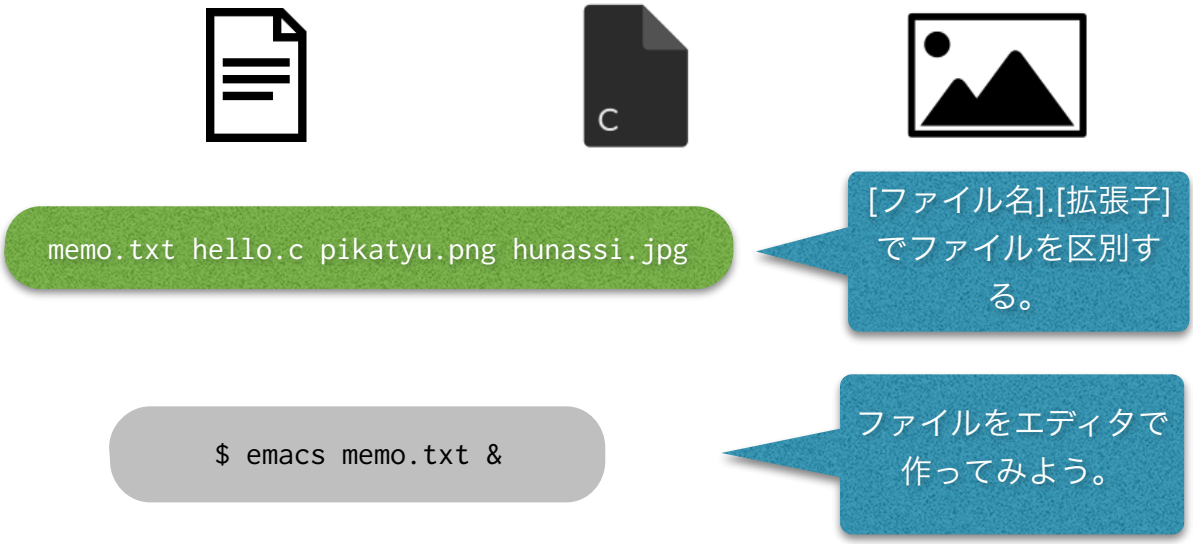


2 Unixファイルシステム

*Unix*上で、作業を行うときは、常にファイルと作業場所 (**カレントワーキングディレクトリ**)を把握し、管理する必要がある。それらの仕組みをファイルシステムという。本章では、ファイルシステムの仕組みを説明していく。

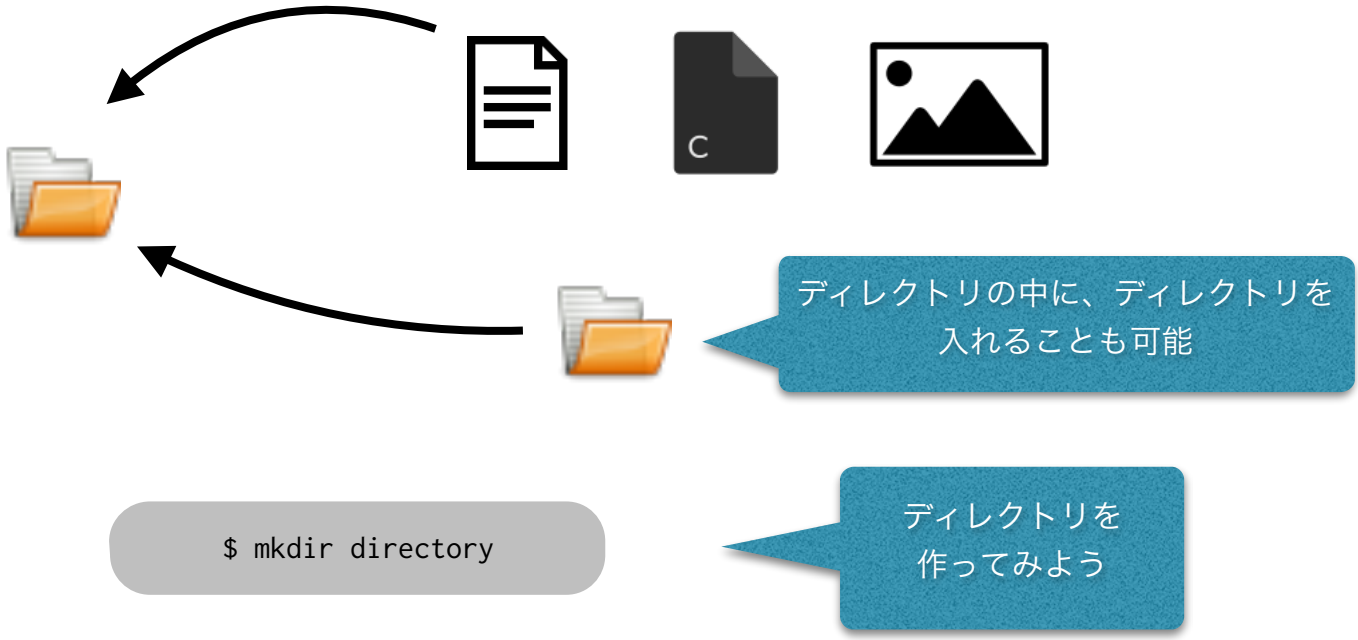
2.1 ファイル

*Unix*では、テキストデータ・画像データ・設定データなどは、全て**ファイル**として扱う。ファイルには必ず、名前が付く。ファイルの名前は自由だが、ファイルの種類を判別するために、ファイルの最後に**拡張子**が付けられる場合が多い。ファイルを作るときは、ファイルの種類が「何」であるかと、見て何が書いてあるかを考えて名前を付けよう。テキストファイルを作るには、**エディタ**と呼ばれるソフトウェアを使う必要がある。会津大の環境では、*emacs*, *vi*, *gedit*などのエディタが用意されている。



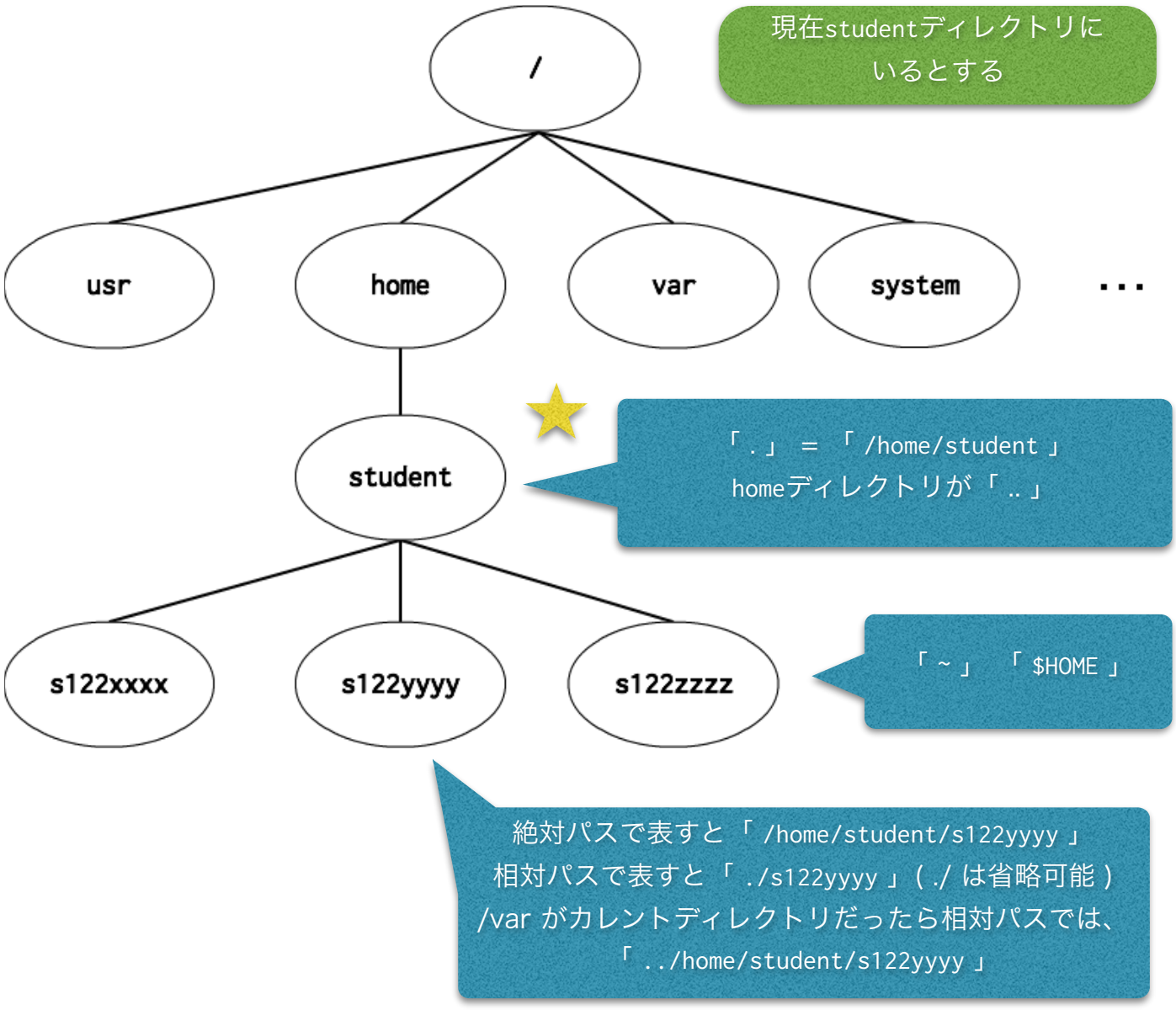
2.2 ディレクトリ

前節2.1では、ファイルとテキストファイルの作り方について学んだ。扱うファイルが多くなった場合や用途によってファイルを分別したい場合がある。*Unix*ではファイルを管理するための入れ物、**ディレクトリ**が用意されている。ディレクトリは、その内部にファイルとディレクトリ自身も収納することができる。ディレクトリを作るには、**mkdir**コマンドを使うことでできる。



2.3 パス

前節1.2では、ファイルを管理する入れ物ディレクトリについて学んだ。ディレクトリは、ディレクトリの中に収納できる。また、*Unix*系統の*OS* (オペレーティング・システム)では、各*OS*ごとにディレクトリの構成は若干の異なりがあるが、必ず最初は**ルートディレクトリ**から始まる。ディレクトリはルートディレクトリ(根っこ)から、用途に合わせた様々なディレクトリ(葉っぱ)から構成される、このようなディレクトリの構造をツリー構造と呼ぶ。ディレクトリの場所を記述したものを**パス**という。ルートディレクトリからスラッシュ区切りで表記したものを**絶対パス**、自分の現在いるディレクトリから指定する方法を**相対パス**と呼ぶ。作業ディレクトリのことを**カレントワーキングディクレクトリ** (*Current working directory*)と呼び、作業ディレクトリをドット「 . 」で表すことができる。 1つ上のディレクトリを**親ディレクトリ**、自分の中にあるディレクトリを**子ディレクトリ**という。親ディレクトリは、ドット2つ「 .. 」で表すことができる。ユーザは、ユニックスシステム上で作られた時点で自分の作業ディレクトリを与えられる、このディレクトリを**ホームディレクトリ**と呼ぶ。ホームディレクトリは、チルダ「 ~ 」や、**Unixグローバル変数**の「**$HOME**」で表すことができる。



2.4 コマンド

本資料に関わらず、*Unix*の利用やコンピュータ・リテラシーを学んで行く上で、コマンドは広く多く使われる。それらのコマンドを利用する前に、コマンドの実体と使い方を覚えておく必要がある。コマンドとは、ターミナル上で動くプログラムのことである。コマンドを実行する際には、本来はプログラムを実行するためのパスを指定する必要がある。

```
$ gcc hello.c
$ ./a.out
```

これは、「.」というカレントワーキングディレクトリを表す記号を利用しているので、相対パスでa.outという実行プログラムを指定していることがわかる。次に、これを絶対パスで以下のように指定して実行をしてみる。

```
$ /home/student/s122xxxx/Prog0/Ex01/a.out
```

このように実行しても同じ結果が得られることがわかる。しかし、普段コマンドを実行するときは、「cd」,「ls」のように絶対パスも相対パスも指定していない。これは、**コマンドサーチパス**という、コマンドプログラムの置き場所 (パス)をあらかじめ設定しているため、パスの指定はしなくて良いのである。サーチパスを確認するには、以下のコマンドを叩くことで確認できる。

```
$ echo $PATH
/usr/local/gcc/bin:/usr/local/gcc3/bin:/usr/local/perl5/bin:/usr/
local/texlive/bin:/usr/local/bin:/usr/local/gnu/bin:/usr/local/
java/jdk/bin:/usr/openwin/bin:/usr/bin:/usr/sbin:/usr/local/LPRng/
bin:/usr/local/SolarisStudio/bin:/usr/ccs/bin:/usr/dt/bin
```

$HOMEと同様に、$PATHもUnixグローバル変数に登録されており、これらの変数は、echoコマンドの引数に渡すことで、確認することができる。コマンドサーチパスでは、コロン「:」区切りで、コマンドプログラムが置いてある絶対パスを列挙している。コマンドの絶対パスを確認するには、whichコマンドを使用する。

```
$ which ls
/usr/local/gnu/bin/ls
```

lsの絶対パスのディレクトリが上の、サーチパスに含まれていることが確認できる。もちろん絶対パスを指定することで、コマンドを使用することもできる (普段はもちろん使わない)。

```
$ /usr/local/gnu/bin/ls
```

サーチパスから実行したコマンドと同じ結果が得られるはずだ。

続いて、コマンドの利用方法について述べる。基本的にコマンドは、以下のような形になっている。

```
コマンド [-オプション…] [引数…]
```

多くのコマンドは、使い方が単一でない場合が多い。引数の数、引数の種類、オプションの有無によって異なる。引数の順序は重要だが、基本的にオプションと引数は順不同でも問題はない。具体例としてmvコマンドを挙げる。

```
$ mv［移動したいファイルパス］［移動したいディレクトリパス］
```

これが、基本的なmvコマンドの使い方である。ファイル名やディレクトリを直接記述する場合は、相対パス、もちろん絶対パスも指定することもできる。続いて引数の数が違う場合を確認する。

```
$ mv［移動したいファイルパス］…［移動したいディレクトリパス］
```

これはつまり、移動したいファイルが複数あっても良いことを表している。重要な点は、最後が必ず、移動したいディレクトリになっていれば問題ない点である。続いて、引数の種類が違う場合を確認する。

```
$ mv［変更したいファイル名］［変更したいファイル名］
```

これは第2引数が、ディレクトリ名ではなくファイル名だった場合は、移動ではなく名前を変更するコマンドに変化する。mvコマンドの本質を、あるパスからあるパスへの変更・書き換えであると考えると、納得がいくと思われる。最後に、オプションがある場合を確認する。

```
$ mv -i［変更したいファイル名］［既に存在するファイル名］
```

mvコマンドでは、名前を変更する (リネーム)際に、変更するファイルの対象が既に存在する場合、上書きして名前を変更してしまう。つまり、元のデータが消えてしまう。そういったケースを防ぐ場合は、i オプションを使うことで、名前が被ってしまった場合に確認をしてくれる。

コマンドの利用についての最後に、コマンドを利用する際にそのコマンドがどのような入力を受け取り、どのような出力をするのかを把握することが重要である。例えば、catの引数なしのコマンドの場合、実行すると、標準入力を求められ、キーボードから何かを打ち込み改行を押すことで、打ち込んだ内容が標準出力される。platexコマンドは、latexファイルのパスを引数から入力として受け取り、結果としてdviなどの形式のファイルを出力する。コマンドを使う場合には、標準入力から値を受け取るのか、引数として与えるのか、結果として、標準出力に結果が出力されるのか、特定のファイルに結果が書き込まれるのかを考える必要がある。

2.5 ファイルシステムのためのコマンド

　前節まで説明したファイルシステムを扱うためのコマンドを紹介していく。コマンドは表の形で、実行の仕方や、実行結果の例を載せている。

コマンド	引数	オプション	実行例	実行結果
pwd	なし	なし	pwd	現在いるパス (カレントワーキングディレクトリ)を絶対パスで、標準出力に表示。print working directoryの略。
ls	なし	なし	ls	カレントワーキングディレクトリにある、ファイル・ディレクトリを標準出力に表示。listの略。
ls	なし	l (エル)	ls -l	カレントワーキングディレクトリのファイル・ディレクトリの詳細情報 (ファイルかディレクトリか、ファイルサイズ、グループ、作成日時など)を標準出力に表示。
ls		a	ls -a	カレントワーキングディレクトリにある隠しファイル・ディレクトリ(ファイル名・ディレクトリの先頭がドットから始まるもの)も含めて表示。
ls		F	ls -F	カレントワーキングにあるファイル・ディレクトリをわかりやすく表示。ディレクトリは、末尾に/ が付く。リンクは@が付くなど。
cd	なし	なし	cd	ホームディレクトリに移動。change directoryの略。
cd	ディレクトリパス	なし	cd literacy cd /home/student/s122xxxx/literacy	指定したディレクトリパスに移動。相対パスでも絶対パスでも指定可能。「 . 」の場合はカレントワーキングディレクトリに移動(つまり移動しない) 「 .. 」の場合は、親ディレクトリに移動。 ../../ の場合は、2つ上の親ディレクトリへ移動
cat	ファイルパス	なし	cat hoge.txt	指定したファイルの中身を標準出力に表示。ターミナル上に一気に出力される。concatenate (結合)の略。
cat	ファイルパス…	なし	cat hoge.txt foo.txt	指定した複数のファイルを結合し、表示。
more	ファイルパス	なし	more hoge.txt	指定したファイルの中身を標準出力に表示。ただし、ページャーが開くため、長いファイルを見る場合はcatコマンドより優れている。q で終了。
wc	ファイルパス	なし	wc hoge.txt	指定したファイルの文字数、単語数、行数を標準出力に表示。word countの略。
rm	ファイルパス	なし	rm hoge.txt	指定したファイルを消す。removeの略。

コマンド	引数	オプション	実行例	実行結果
rm	ディレクトリパス	r	rm -r directory	指定したディレクトリを消す。
mkdir	ディレクトリパス	なし	mkdir directory	指定したパスにディレクトリを作成する。
rmdir	ディレクトリパス	なし	rmdir directory	指定したパスのディレクトリを消す。rm -r と同じ動作。

以上は、ファイルシステムを扱う上で、最低限のコマンドなので完全に理解すること。補足として、コマンドは、ターミナル上でマニュアルを見ることができる。そのためのコマンドは`man`コマンドである。

```
$ man ls
```

マニュアルは、英語で表示されるため抵抗があるかもしれないが、コマンドの基本の利用方法を把握して、要点だけを見れば英語をあまり見なくても読むことができる。是非積極的に使っていただきたい。`ls`を`man`コマンドで調べると、 *ls - list contents of directory* が正式名であることがわかる。ディレクトリの内容物を列挙する。という平易な英語で記述されており、コマンドに対する理解が深まることがわかる。このコマンドもmoreと同様にページャなので、qをタイプすることで終了する。

2.6 グループとパーミッション

ワークステーションシステムを使う場合には、多くの人(会津大学の場合は、教員・職員・大学院生・学部生)とファイルシステムを共有することになる。その場合に、他人とファイルを共有したい場合、他人にファイルを見られていけない場合、様々なケースが想定される。ファイルシステムのユーザを分けたものを**グループ**と呼び、ファイルの使用権限(所有権)を**パーミッション**と呼ぶ。ファイルやディレクトリを作った場合には、グループとパーミッションを意識することが大切である。以下は、とあるディレクトリで ls -l をした結果である。

```
-rw-r--r-- 1 s122xxxx student 31143 May  3 11:10 ConstitutionOfJapan.txt
drwxr-xr-x 2 s122xxxx student  4096 May  8 22:48 directory/
-rw-r--r-- 1 s122xxxx student   239 Apr 17 08:46 grade.csv
```

一番左がパーミッション、その次がハードリンクの数 (解らなければ気にしなくてもよい)、その次が所有者、その次が属しているグループである。学部生は、*student*というグループに属している。続いてパーミッションの見方について説明をする。

```
drwxr-xr-x
```

一番左は、ディレクトリである場合は、dと表示される。rは読み込み (read)権限、wは書き込み(write)権限、xは実行(executable)権限である。ただし、ディレクトリの場合、xはcdの許可の可否を表す。これらのrwxが３つ並んでいることがわかる。1つ目は、所有者 (user)の権限、2つ目は、グループ (group)の権限、3つ目は、他の人 (other)の権限を表している。-は許可が降ろされていないことを表している。学部生の場合はgroupはstudent、otherは、院生や教員、職員などを指している。

2.7 パーミッション変更コマンド

ファイルやディレクトリのパーミッションについて前節2.6で学んだ。実際に、パーミッションを変更するコマンドchmodと使い方を紹介する。

```
$ chmod a-x file
```

chmodは、*change the permissions mode of a file*. の略である。引数は２つで、一つ目はパーミッション変更命令、２つ目はパーミションを変更する対象のファイルを記述する。パーミッション変更命令について詳しく見ていく。

```
a-x
```

左は、グループを表す記号である。rはall全てのグループ、後は前節で説明した、u g o などである。真ん中は、許可する場合は、+記号を、拒否をする場合は、-記号を記述する。右は、対象となる権限を表している。権限はr w xである。グループを表す記号、権限を表す記号は複数書くことも可能である。

3 Xwindowシステム・プロセス

今まで説明したコマンド類は、全てターミナル上で動く*CUI* (コマンドライン・ユーザ・インターフェース)で実装されたプログラムであった。他にも文字だけでなく、アイコンや画像を使った*GUI*(グラフィカル・ユーザ・インターフェース)で実装されたアプリが存在する。*emacs*, *xeyes*, *xcalc*, *sylpheed*, *firefox*などが*GUI*プログラム(アプリケーション)などである。これらのプログラムをターミナルで実行するときは、注意が必要である。仮に以下のようにコマンドをタイプしてみよう。

```
$ emacs memo
```

*emacs*は、memoという名前のファイルを編集するために起動する。このとき、ターミナルを見てみるとコマンドを受け付けなくなっているのがわかる。これは、emacsがターミナル上で起動しつづけているためだ。この状態を**フォアグラウンド**と呼ぶ。次に&を末尾に付けて起動してみる。

```
$ emacs memo &
```

この場合、emacsが立ち上がっているにも関わらず、ターミナルはコマンドを受け付けるようになった。このemacsの状態を**バックグラウンド**と呼ぶ。このように*Unix*上で実行されるプログラムは**プロセス**という単位で扱われる。現在動いているプロセスを確認するには、ps (プロセスステータス)コマンドやjobsコマンドを使用する。psでは、プログラムは**プロセス番号**(5桁などの数字)、jobsでは、プログラムは**ジョブ番号**が割り振られていることがわかる。プログラムを殺す(止める)には、killコマンドを使用する。

```
$ kill %ジョブ番号
$ kill プロセス番号
```

psコマンドで全てのプロセスを確認する場合は、ps auxやps -elfで情報を確認できる。

3.1 フォアグラウンド状態のアプリを制御する

前節3で説明したように、フォアグラウンド状態のアプリが、ターミナルで起動しているとターミナルは操作を受け付けなくなってしまう。しかし、ショートカットキーを駆使することで、フォアグラウンドアプリを制御することができる。

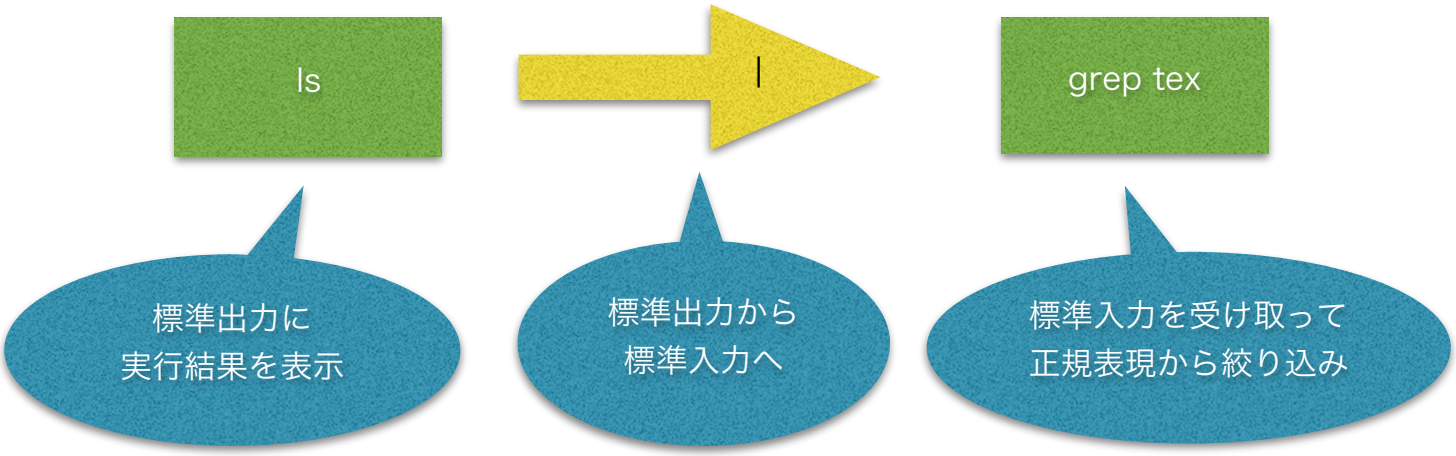
```
$ emacs memo
```

再び、emacsにフォアグラウンド状態で稼働してもらう。このとき、C-c (コントロールを押しながらc)をターミナルでタイプすると、emacsが殺される(停止)ことが確認できる。しかし、これでは編集中のファイルの内容が消えてしまう。続いてC-zを試す。これはSuspendedもしくは、Stop(一時停止)状態にすることができる。この状態でジョブ番号を調べ、bg ジョブ番号とタイプすることで、emacsをバックグラウンド状態で動かすことができる。すると、ターミナルがコマンドを受け付け、emacsが再び稼働することが確認できる。

4 パイプとリダイレクト

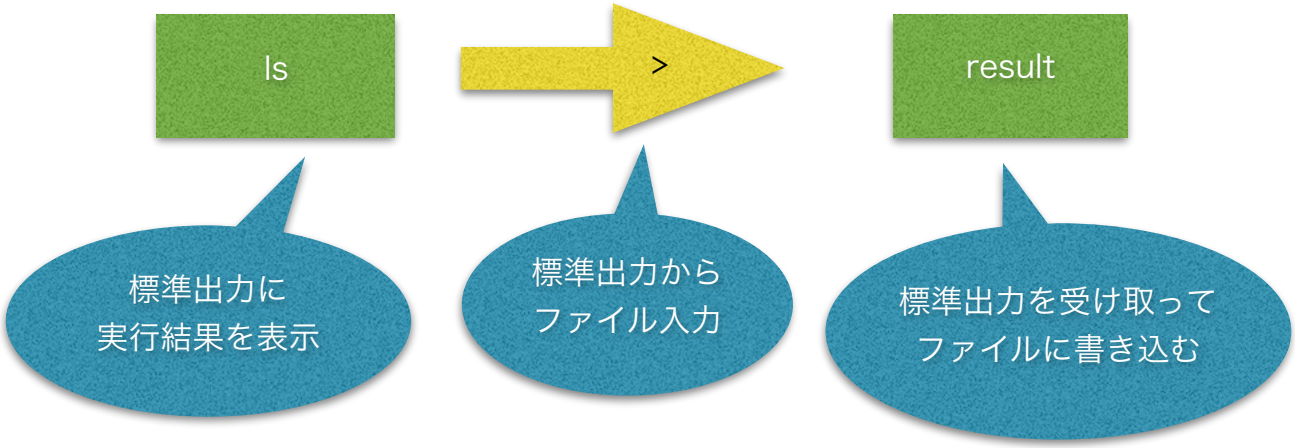
今まで紹介したコマンドの1つ1つは、１つの仕事しか全うしない。これは、*Unix*のSmall is beautiful. という哲学に従っているからである。これは、１つ1つのコマンドを小さくし、他のコマンドと連携しやすくするための仕組みである。例えば、grepというコマンドは、標準入力によって得られたデータから、正規表現にマッチした行を表示するコマンドである。このコマンド単体では、あまり意味を成さない。そこで**パイプ**と呼ばれる仕組みを使うことで、標準出力を次に実行するコマンドの標準入力として扱うことができる。literacyディレクトリからlatexファイルのみを抽出する場合は、以下のように実行する。

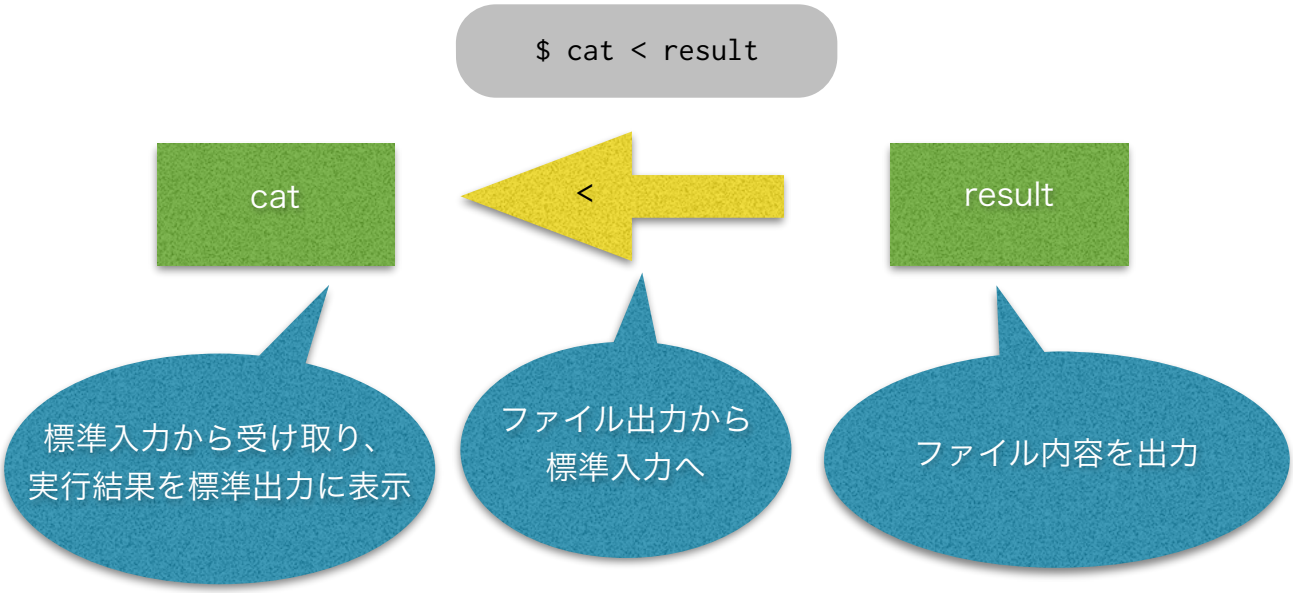
```
$ cd ~/literacy
$ ls | grep tex
```



また、標準入力の結果をファイルに書き込んだり、ファイルの内容を標準入力として、読み込ませたい場合がある。その場合は、**リダイレクト**という仕組みを利用する。

```
$ ls > result
```





ファイルに出力する場合、>> にするとファイルを上書きではなく末尾に追記することになる。これを**アペンド**と言う。

5 本資料にない重要なこと

体力・時間などの問題により、本資料はここまでにする。（ごめんなさい) そのため最後に、本資料で述べていない重要なポイントを列挙する。

- emacsの使い方 (保存、ファイル読み込み、コピー、カット、ペーストなどのショートカットキーは絶対に覚えること！)
- Sylpheedの学内ニュース、u-aizu.course.lit1 と wantedくらいは覚えておくこと！
- 印刷(印刷に必要なコマンド、lpr, lpqなど)や印刷に必要なファイルps形式への変換方法を必ず覚えること！ (latexからps, テキストファイルからpsファイル a2ps k2psなど)
- latexからpdfの作り方 (dviはgvコマンドやdisplayコマンドで確認できる！など！)
- pdfの閲覧方法(acroread, evinceなど！)
- sedと正規表現とgrep (特定の行の抜き出し、特定の文字の置換、検索など!)
- awkの基礎的な使い方(BEGIN{}{}END{}など!)

本資料の内容を完璧に押さえれば、5~6割 上の箇条書きに書いた内容を完璧に押さえれば8~9割は硬いと思われる。しかし、点数にこだわらず、Unixの考え方を押さえることで、様々な教科の基本になり、理解や効率的に課題を解く力に繋がるので、しっかり理解することをお勧めする。それでは、健闘を祈る。