

コンピュータ・リテラシーI

1~10章 要点まとめ(中間テスト対策)

会津大学

d8161105 渡部未来 April 11, 2015

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、**OS**(オペレーティング)の一つである**Unix**の基本的な使い方について学ぶ授業である。コンピュータ・リテラシーのハンドアウトは、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ケースがある。そのため、この資料では、1~10章の重要な点をいくつか要約している。加えて理解が捗るように補足などをその都度加えて説明を行う。

1. Unixの主な操作方法

Unixで行う作業の多くは、ターミナルと呼ばれる文字を主体とした対話環境を扱う。このような環境(図1)を**CUI(Character User Interface)**や**CLI(Command line Interface)**などと呼ぶ。ターミナルでは、キーボードから**コマンド**と呼ばれる命令を打ち込んだり、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。ターミナルは、コマンドが受け付け状態であることを表すために、ドル記号「\$」や大なり記号「>」を使うことが多い。この記号を**プロンプト**と呼び、本資料ではドル記号を使う。なので、コマンドを試す際に間違ってドル記号を打たないように気をつけて欲しい。

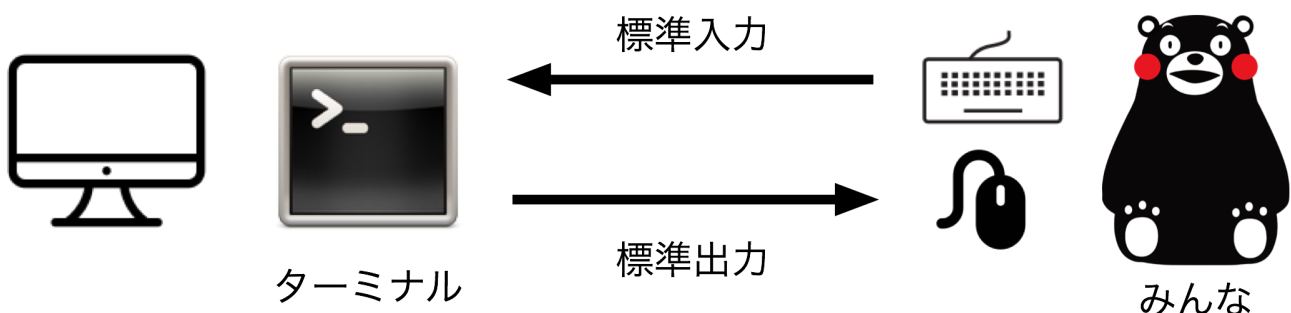


図1: CUIにおけるコンピュータとユーザの対話関係

2 Unixファイルシステム

コンピュータ上では、さまざまなデータをファイルという形で管理する。この章では、**Unix**におけるファイル管理方法について説明をおこなう。課題だけでなく、コンピュータを扱う基礎的な事項なので、しっかり仕組みを理解する用に心掛けて欲しい。

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**と言う単位で扱う。ファイルには必ず名前が付く。ファイルの種類を判別するために、ファイルの最後にドットを使った**拡張子**(ex01.txt, ex02.c など)が付けられる場合が多い。課題などでテキストファイルやプログラムファイルを作るには、**テキストエディタ**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などのテキストエディタが用意されている。memo.txtのファイルをemacsで作成・編集する場合は以下のようにコマンドをターミナルに打つ。

```
$ emacs memo.txt &
```

2.2 ディレクトリ

前節2.1では、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**を収納することができる。ディレクトリに新たなディレクトリを入れることができることを**入れ子**と呼ぶ。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。

```
$ mkdir dir1 dir2
```

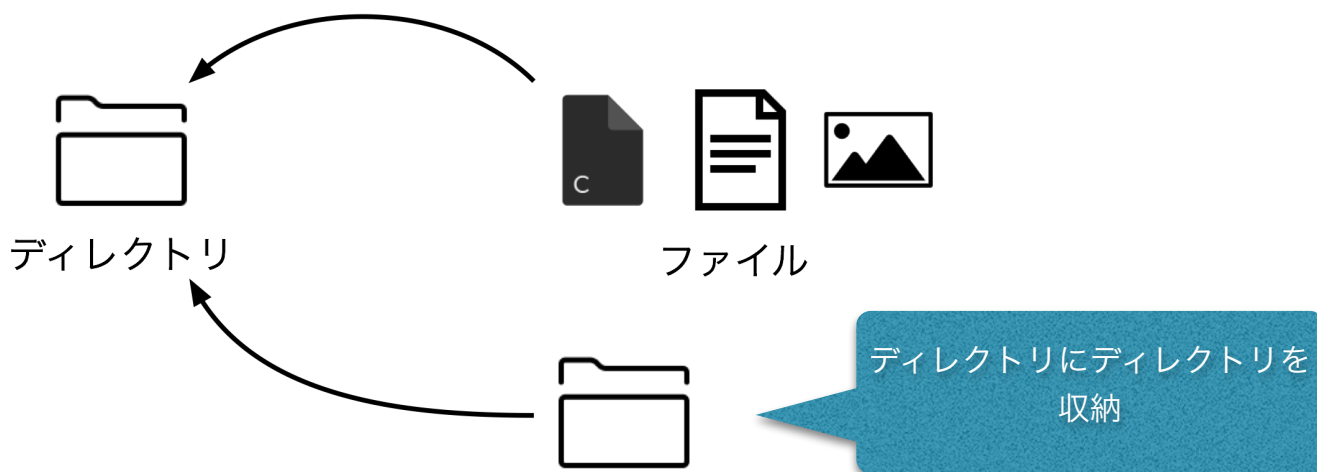


図2: ディレクトリの役割

2.3 パス

前章では、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。これは、ディレクトリが図3のように**階層構造**になっていることを表している。Unix系統のOS (オペレーティング・システム)では、必ず最初がスラッシュ記号「/」で表される**ルートディレクトリ**始まり、そこから様々なディレクトリが枝分かれすることから**木構造**(ツリー)と呼ばれる。ディレクトリの場所を**パス**と言いディレクトリ名をスラッシュ記号で区切って表記する。

スタート地点である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**(/home/student/s123xxx/literacy/introduction.tex)。現在いるディレクトリから別のディレクトリを指定する方法を**相対パス**(literacy/introduction.tex)と呼ぶ。現在自分が作業ディレクトリのことを**カレントワーキングディレクトリ** (*Current working directory*)と呼び、ドット記号「**.**」で表すことができる。自分自身が収納されている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**と言う。親ディレクトリは、ドット2つ「**..**」で表すことができる。生徒は、*Unix*システム上でアカウントを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**(/home/student/s123xxx)と呼ぶ。ホームディレクトリは、チルダ記号「**~**」や、**Unixグローバル変数**の「**\$HOME**」で表すことができる。

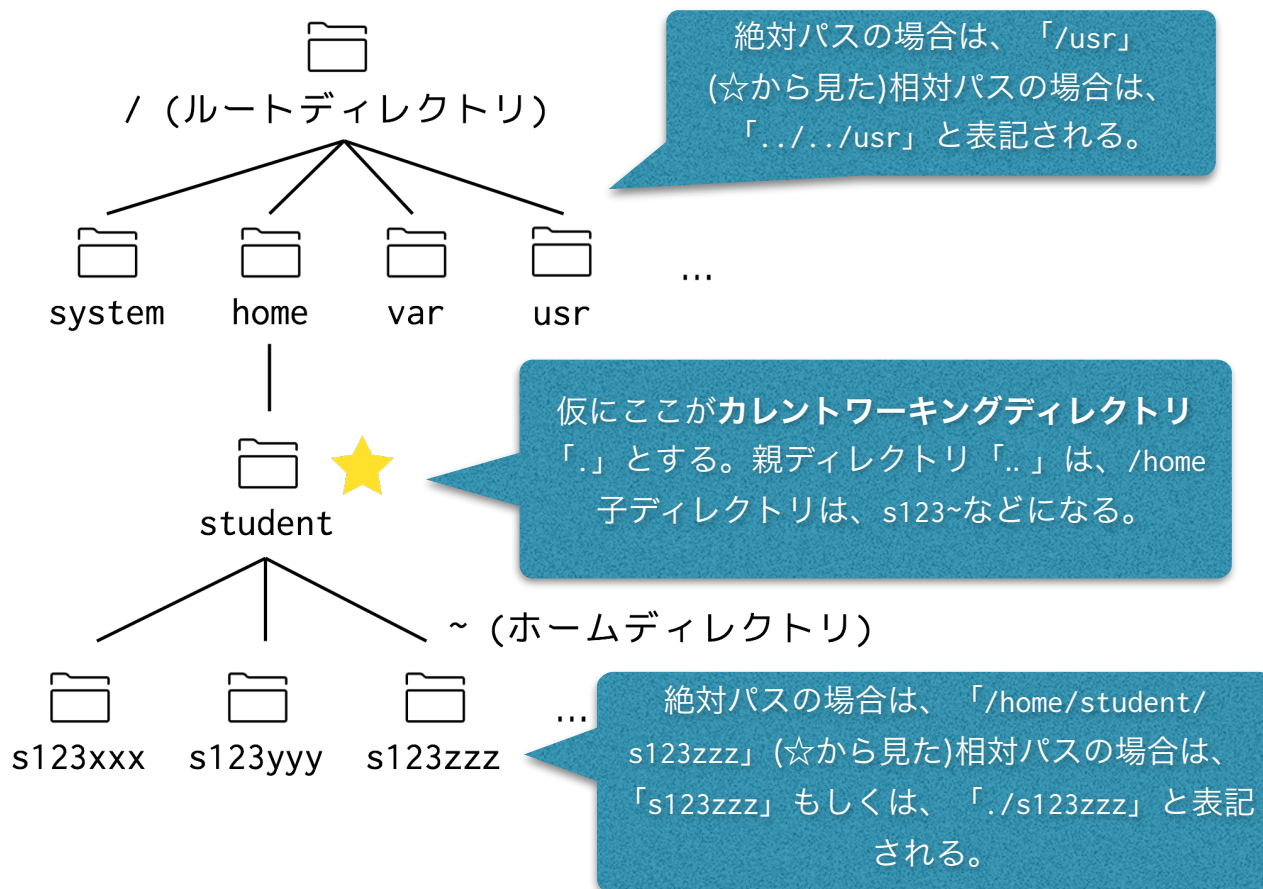


図3: ディレクトリのツリー構造

3 コマンド

*Unix*でコンピュータと対話するためには、**コマンド**と呼ばれる命令プログラムをターミナルに打つ必要がある。コマンドは、単体で動くものもあれば、ファイル名やディレクトリ名などの文字列を必要とするものがある。コマンドに渡される文字列のことを**コマンドライン引数**と呼ぶ。また、ハイフン記号「**-**」と文字列を渡し、コマンドの動きをコントロールできるものもある。この仕組みを**オプション**と呼ぶ。引数やオプションは、空白で区切ることで複数入力することが可能である。ここから先は、ファイルやディレクトリを操作するコマンドを例に、コマンドの使い方を学んでいく。

コマンド名 [-オプション…] [コマンドライン引数…]

3.1 コマンドの基礎

コマンドを実行する際には、2.3章で学んだカレントディレクトリのパスを意識する必要がある。今自分がどこのディレクトリにいるかを確認するコマンドはpwdコマンドである。ターミナルを開いたあとにコマンドを実行すると、自分のホームディレクトリの絶対パスが確認できるはずだ。

```
$ pwd  
/home/student/s123xxx
```

カレントディレクトリに収納されているファイルやディレクトリを確認するコマンドは、lsコマンドである。

```
$ ls  
Downloads Mail Pictures toSSB.pdf Desktop  
Movies Prog0 test ...
```

しかし、これでは、どれがファイルで、どれがディレクトリかの区別がつかない。そこでlsの-Fオプション

```
$ ls -F  
Downloads/ Mail/ Pictures/ toSSB.pdf Desktop/  
Movies/ Prog0/ test ...
```

プションを利用するとディレクトリ名の最後に/が付くようになる。

自分のカレントワーキングディレクトリを変える(別なディレクトリに移動する)コマンドはcdコマンドである。cdコマンドは単体で使用すると、ホームディレクトリに移動する。成功すると特に画面には何も表示されないため、移動できたか確認するにはpwdコマンドを続けて打つ。

cdコマンドは別なディレクトリに移動するコマンドなので、行き先のディレクトリパスを指定するのが本来の使い方である。ここでパスは絶対パスと相対パスの2種類があったことを思い出していただきたい。以下は、すべて同じディレクトリに行くためのコマンドである。

```
$ cd  
$ pwd  
/home/student/s123xxx
```

最初の2つのコマンドはセットで、まずはじめにホームディレクトリに移動してから、次のコマンドでliteracyディレクトリへ移動している。次のコマンドは絶対パスでliteracyディレクトリへ移動している。最後のコマンドは、ホームディレクトリを表すチルダ記号を使ってliteracyディレクトリへ移動している。

```
$ cd
$ cd literacy
$ cd /home/student/s123xxx/literacy
$ cd ~/literacy
```

最後に、今まで学んだオプションとコマンドライン引数を組み合わせた場合の例を見る。ファイルやディレクトリを確認するコマンドlsを例に取る。オプションとして、aとlオプションを渡す。aオプションは、ドット「.」から始まる隠しファイルを表示するオプションでlオプションは、ファイルの詳細情報を表示するオプションである。引数には、チルダ記号と同じホームディレクトリを表すグローバル変数\$HOMEを渡している。

```
$ ls -al $HOME
total ***
-rw-r--r--  1 s1230xxx student   117 Apr  6 14:55 #test#
drwxr-xr-x 40 s1230xxx student 4096 Apr 10 21:02 ./
dr-xr-xr-x  7 root      root      7 Mar  9 11:44 ../
drwx-----  2 s1230xxx student 4096 Apr 10 14:26 .Trash/
-rw-----  1 s1230xxx student  231 Apr 10 21:02 .Xauthority
...
```

3.2 よく使うコマンド

前章でコマンドの使い方の基礎を学んだので、良く使用するコマンドを表に列挙する。コマンドは英単語を縮めて表記する場合が多い。フルスペルとその意味を学ぶことで、コマンドの動きなどを推測できるようになる。丸暗記ではなく、意味を捉えて覚えることが重要だ。

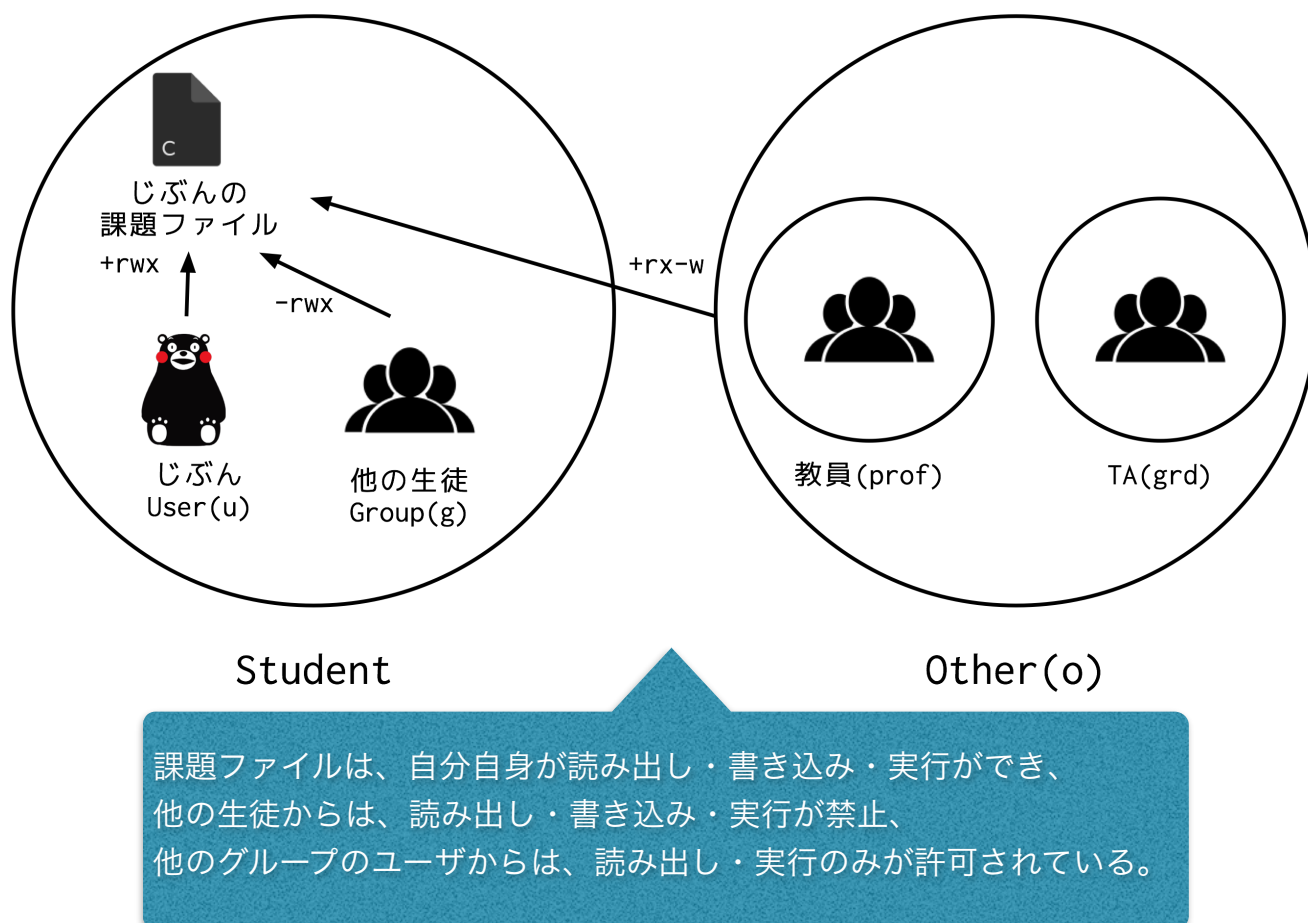
| コマンド | 引数 | オプション | 使用例 | 実行結果 |
|------|----------------|---------|----------------------------------|--|
| pwd | なし | なし | \$ pwd | カレントワーキングディレクトリを絶対パスで、標準出力に表示。print current working directoryの略。 |
| ls | なし ディレクトリパス | alF ... | \$ ls \$ ls -al \$ ls -a ~ | カレントワーキングディレクトリもしくは、指定したディレクトリパスにある、ファイル・ディレクトリを標準出力に表示。listの略。 |
| cd | なし ディレクトリパス | なし | \$ cd \$ cd ~/Prog0/Ex01 | ホームディレクトリもしくは、指定したディレクトリパスへ移動。change working directoryの略。 |

| コマンド | 引数 | オプション | 使用例 | 実行結果 |
|-------|---------------------------|-------|---|--|
| echo | 文字列 グローバル変数 | なし | \$ echo "aiueo" \$ echo \$HOME \$ echo \$PATH | 文字列を出力するコマンド。グローバル変数を指定することでグローバル変数に格納された値を出力することもできる。非常に応用が効くコマンド。 |
| ps | なし | elf | \$ ps \$ ps -elf grep s123xxx | 現在動いているプログラム(プロセス)の一覧を表示することができる。プロセスには特定するまでのID、PIDが付いている。process statusの略。 |
| kill | PID | なし | \$ kill 123 | psコマンドなどで確認したPIDでプロセスを止めることができる。emacsなどが動かなくなった場合に止めることが可能。 |
| cat | ファイルパス… | なし | \$ cat memo1 \$ cat memo1 memo2 | 指定したファイルの中身を標準出力に表示。複数ファイルを指定した場合、縦に結合されて表示される。concatenate (結合)の略。 |
| more | ファイルパス | なし | \$ more long_memo | 指定したファイルの中身をビューアで確認できる。中身が長いファイルの場合には、上から少しずつ見ることができる。qで終了。 |
| wc | なし ファイルパス | なし | \$ wc memo \$ ls wc | 標準出力、もしくは指定したファイルの文字数・単語数・行数を標準出力へ表示する。word countの略。 |
| mkdir | ディレクトリパス… | なし | \$ mkdir dir1 \$ mkdir dir1 dir2 | ディレクトリを作成する。複数のディレクトリを一気に作ることも可能。make directoryの略。 |
| mv | ファイルパス … ディレクトリパス … | なし | \$ mv memo dir \$ mv dir1 dir2 \$ mv memo memo2 | ファイルやディレクトリを別なディレクトリに移動する。移動元 移動先の順で指定する。古いファイル名 新しいファイル名と指定することで、リネーム(名前の変更)をすることもできる。moveの略。 |
| cp | ファイルパス… ディレクトリパス… | r | \$ cp memo memo2 \$ cp -r dir dir2 | ファイルを複製するコマンド。コピー元 コピー先の順で指定する。rオプションでディレクトリのコピーもできる。copyの略。 |
| rm | ファイルパス ディレクトリパス | rfi | \$ rm memo \$ rm -ri dir | 指定したファイルを削除できる。また、rオプションを付けることでディレクトリを削除することができる。removeの略。 |
| man | コマンド名 | なし | \$ man ls \$ man cd \$ man rm | コマンド名を引数に渡すことで、そのコマンドの説明を見ることができる。moreと同じでqで終了。manualの略。 |

これらは、非常に良く使うコマンドのごく一部なので、いち早く使い方をマスターすることをおすすめする。manコマンドは非常に便利なので、わからないコマンドが出てきたら使うと良い。

4 パーミッションとグループ

普段使うパーソナルコンピュータではなく、**ワークステーションシステム**を使う場合には、多くのユーザとファイルシステムを共有することになる。その場合に、他人とファイルやディレクトリを共有したい場合、他人にファイルを見られていけない場合などの様々なケース(期末テストのデータ流出を防ぐなど)が想定される。各ファイル・ディレクトリのアクセス権のことを**パーミッション**と言う。パーミッションを適切に設定することで、他人とのファイル共有が円滑に進む。また、複数のユーザをまとめる**グループ**と言う仕組みがある。会津大学の場合は、教員(prof)・職員・大学院生(grdm, grdd)・学部生(student)・ゲストなどのグループが存在する。



4.1 パーミッションの見方

ファイルに設定されているパーミッションや所属するグループを確認するには、「ls -l」コマンドを実行する。パーミッションを確認するには、rwxやマイナス記号「-」が並んだ列とファイルの**所有者(owner)**・所有グループを確認すると良い。パーミッションの一番左の列はdであればディレクトリ、マイナス記号であれば、ファイルであることを表している。対象がファイルである場合は、rは読み出し(read)、wは書き込み(write)、xは実行(execute)を表している。対象がディレクトリである場合は、rはディレクトリの中身の閲覧、wはディレクトリ内のファイル作成・削除、xはディレクトリ内のファイルにアクセスを表している。マイナス記号の場合は、アクセス許可が出されていないことを表す。

| ディレクトリ | パーミッション | ハードリンク数 | 所有者 | 所有グループ | ファイルサイズ | 最終更新日時 | ファイル名 |
|------------|---------|----------|---------|--------|--------------|-----------|-------|
| drwx---r-x | 17 | s1230xxx | student | 4096 | Apr 10 10:58 | Prog0/ | |
| -rw-r--r-- | 1 | s1230xxx | student | 120 | Apr 13 15:45 | myprofile | |
| drwx---r-x | 2 | s1230xxx | student | 4096 | Apr 8 10:30 | literacy/ | |

ls -lの結果をパーミッション部分のみ抜き出すと以下のようになる。すると合計6つのアルファベット、もしくはマイナス記号があるのが確認できる。これらは3文字ごと左から順に、ユーザ(自分自身、user)・所属グループ(学部生、student group)・他のグループ(other)に対するアクセス権を表している。以下の例では、自分自身が読み出し・書き込み、他の学生とその他のグループに読み出しのみの許可が出ていることが確認できる。

```
|u||g||o|
|r w -|r - -|r - -|
```

4.2 パーミッションの設定

前章まででパーミッションの見方について学んだ。この章では、ファイルにパーミッションを設定する方法について説明する。まずはじめにパーミッションを実験するための空のファイル生成する。touchコマンドは更新日時を変更するコマンドだが、ファイルが存在しない場合は、空のファイルを生成してくれる便利なコマンドだ。ファイルを作成した状態では、ユーザに読み込み書き込み許可、所属グループとその他のグループには読み出し許可が出ていることが確認できる。

```
$ touch test
$ ls -l test
-rw-r--r-- 1 s1230xxx student 0 Apr 13 17:30 test
```

それでは、全てのユーザに対して実行許可を出し、その後に所属グループに対してのみ実行を禁止する。パーミッションを変更するにはchmod(change mode)コマンドを利用する。

```
$ chmod a+x test
$ ls -l test
-rwxr-xr-x 1 s1230xxx student 0 Apr 13 17:30 test*
$ chmod g-x test
$ ls -l test
-rwxr-xr-x 1 s1230xxx student 0 Apr 13 17:30 test*
```


総合すると、chmodコマンドは、以下の法則に従ってパーミッションを設定できることがわかる(aはall)。

```
chmod (u|g|o|a|)(+|-)(r|w|x) file
chmod (対象グループ)(可否)(対象権限) ファイル名
```

しかし、上記の方法ではアクセスの可否どちらかしか指定できなかったり、対象権限を絞る必要がある。chmodコマンドは一度にすべてのパーミッション設定する方法が用意されている。rwxに対して、2進数のフラグ(可・否)を設定する。そして10進数に直した場合の数字を各グループごとに用意する。2進数がわからない場合は、左から順に4,2,1の数字を用意し許可を与える部分の数字のみを残し、その和を計算すると良い。

| | |
|-------|---------|
| (権限) | rwx |
| (二進数) | 111 |
| (十進数) | 4+2+1=7 |

```
$ chmod 715 test
$ ls -l test
-rwx--xr-x 1 s1230xxx student 0 Apr 13 18:01 test*
421 001 401
7 1 5
```

5 Xwindowシステム・プロセス

これまで説明したコマンド類は、すべてターミナル上で動く文字を使ったCUIプログラムであった。CUIと対をなす図・アイコン・画像などを使った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ケーションが存在する。Unixでは、emacs, xeyes, xcalc, sylpheed, firefoxなどが代表的なGUIアプリケーションである。これらのアプリケーション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
```

すると、emacsを開いている間はターミナルが使用できない。ターミナルは起動したプログラムが終了するまで、コマンドを受け付けることができない。この状態を**フォアグラウンド**と呼ぶ。通常GUIアプリケーションはターミナルで結果を見たりするわけではないので、フォアグラウンドで動かし続ける必要はない。ターミナルの裏でプログラムを動かし続ける状態のことを**バックグラウンド**と呼ぶ。プログラムをバックグラウンドで動かすには、最後にアンド記号「&」を付ける。

```
$ emacs memo &
```

仮に&をつけ忘れてしまった場合は、ターミナル上でC-c(コントロールキーを押しながらc)をタイプするとアプリケーションを**終了**することができる。C-zをタイプすることで、一時停止することができる。この状態のことを**サスペンド**と呼ぶ。一時停止した状態のときにbgコマンドを打つことで、一時停止したアプリケーションをバックグラウンドで動かすことができる。fgコマンドを打つことでフォアグラウンド状態でアプリを動かすこともできる。

```
$ emacs memo
C-z
$ bg
```

ユーザが動かしているプログラムを確認するコマンドには、psやjobsコマンドがある。動いているプログラムを特定するために、プログラムには**プロセスID**や**ジョブ番号**が割り振られる。例えば以下の例では、emacsに26098というプロセス番号と1というジョブ番号が割り振られている。

```
$ ps
PID TTY          TIME CMD
26081 pts/12      0:00 bash
29920 pts/12      0:00 ps
26098 pts/12      0:01 emacs-24
$ jobs
[1]+  Running                  emacs test &
```

GUIアプリケーションは障害が起きた時に終了ボタンなどが押せなくなってしまう場合がよくある。その場合は、プロセスIDやジョブ番号をターミナル上から特定し、killコマンド(Unixはジョークで物騒なコマンド名が多い。)によりアプリケーションを終了させることができる。プロセスIDの場合は、そのままの数字。ジョブ番号の場合は%を数字の前に付けることで、アプリケーションを終了させることができる。

```
$ kill 26098
もしくは
$ kill %1
```

6 パイプとリダイレクト

今まで習った多くのコマンドは、一つ一つはとても小さな機能しか持っていなかった。それは、「一つのことを行い、またそれをうまくやるプログラムを書け。協調して動くプログラムを書け。」というUnixの哲学に基づいているからだ。それでは、プログラムを協調させるには、どのようにすればいいのだろうか。答えは、**標準入出力**と**パイプ**、**リダイレクト**などの仕組みを使うことだ。今まで習ってきたコマンドで結果をターミナル上に出力するようなものは言い換えれば標準出力を行っている。例えば、ls, echo, pwdなどのコマンドだ。それに対し、何らかの文字列を受け取り実行されるようなコマンドは、言い換えれば標準入力を受け取っている。

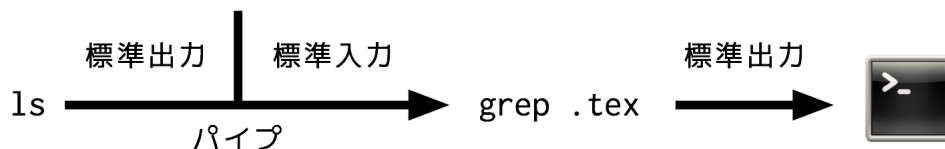
例えば、cut, wc, grepなどのコマンドだ。これらのコマンドを組み合わせることで非常に高機能な一つのプログラムに変化する。例として、カレントワーキングディレクトリにあるファイル・ディレクトリの数を確認するには、lsコマンドで標準出力にファイル・ディレクトリ名を列挙したあとに、その結果を標準入力として受け取った、wcコマンドで行数を確認すれば良い。

```
$ ls | wc
24      24      396
```

カレントワーキングディレクトリのパーミッション情報のみを取り出したい場合は、ls -l コマンドで結果を標準出力したあとに、その結果を標準入力として受け取った、cutコマンドで必要な列を切り出せば良い。

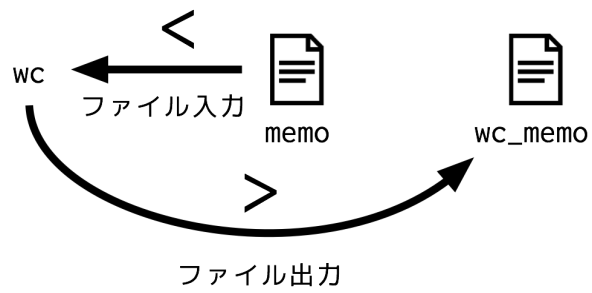
```
$ ls -l | cut -d' ' -f1
total
-rw-r--r--
drwx-----
drwx-----
drwx-----
-rw-r--r--
...
```

カレントワーキングディレクトリからtexファイルだけ取り出す場合は、ls | grep .tex というコマンドを用いると言いだろ。



標準入出力は、ターミナル上でやりとりされる一時的な出来事だ。ここで得られた結果をファイルに保存したり、ファイルから標準入力として読み込みたい場合も出てくるだろう。ここで使われるのがリダイレクトという仕組みだ。小なり記号「<」は、ファイルから標準入力を受け取る。大なり記号「>」は、標準出力をファイルへ出力することを表している。

```
$ wc < memo > wc_memo
$ cat memo
5 5 24
```



7 今後の予定

もし、要望があれば、awk, sed 印刷の仕方などの、他の重要な内容を続きとして書く・・・かもしれない。