コンピュータリテラシ＃5 – ファイルシステムとファイル操作

久野 靖 (電気通信大学)

2017.5.19

1　今回の目標

今回の目標は次の通りです。

- ファイルシステムやディレクトリ階層の概念について理解する— Unix 上でのファイルの保管/整理について知っていることは実験/研究を効率よくこなす上で必須です。
- ファイルのコピーや名前変更など基本的な操作について理解する— ファイルの操作ができることは実験データの保管や加工のために必須です。
- ファイルやディレクトリの保護モードとその設定について理解する — 実験等でデータを保護する必要がある場合もありますし、逆に保護設定をゆるめないと使えない場面もあります。

2　ファイルシステム

2.1　2 次記憶装置とファイルシステム exam

メモリ (主記憶) はコンピュータシステムにおける「主要な」記憶装置ですが、その容量は限られていますし、電源を切ると内容が消失します。一方、コンピュータシステムは電源が切られてもきちんと残って欲しい情報を沢山格納する必要があります。このため、次の要件を満たす **2 次記憶** (secondary storage) 装置が使用されます。

- 安定記憶 — 電源を切っても消えず、内容が勝手に書き変わったりしない。
- 容量/コスト — 大量のデータを格納でき、1 ビットあたりのコストが低い。

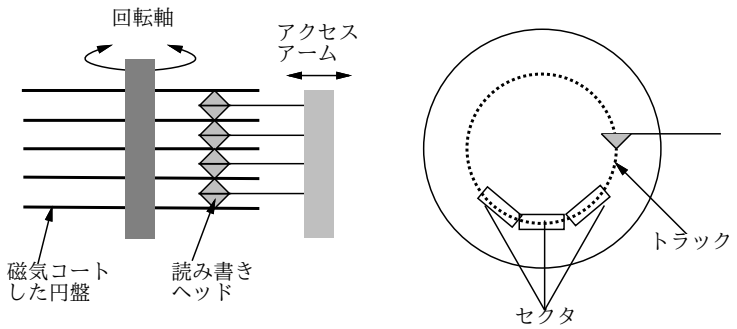

図 1: 磁気ディスク装置の構造

今日のコンピュータシステムで広く使われている 2 次記憶装置として、磁気コーティングした円盤に磁気的に情報を書き込み保存する**磁気ディスク装置** (hard disk drive、HDD) が代表的です (図 1)。このほか、電源を切っても内容が保持される半導体メモリである**フラッシュメモリ** (flush memory) もモバイル機器を中心に使われます。これはビット当たり単価はやや高めですが、HDD より高速なの

で、通常の PC でも磁気ディスク装置の代替として使用することもあり、その場合 **SSD**(solid state disk) と呼びます。[1]

2.2 情報の整理とファイルシステムの階層構造 exam

2 次記憶装置には大量の情報が入れられるので、それらに名前をつけて、構造を持たせて管理できるようにする必要があります。コンピュータシステムの中で、このような機能を提供するソフトウェアのことを**ファイルシステム** (file system) と呼びます。そしてこの名称は「沢山のファイルが集まった構造全体」という意味でも使われます。以下では Unix を例に、後者の意味でのファイルシステムを説明します。ファイルシステム中の主要な要素は、次の 2 種類のものです。

- **ファイル** (file) — 任意のビットのデータ (メモリ同様にバイト単位でアクセスするので長さもバイト単位)。
- **ディレクトリ** (directory) — 「登録簿」という意味の英語で、ファイルやディレクトリをいくつでも入れられる。

あなたが多数の情報 (書類など) を保管し、必要に応じて取り出したり追加するとしたら、どうますか? 書類が少量なら、適当に積み上げておくだけでも済むかも知れません。しかし、何百、何千もの書類となると、それでは無理ですね。普通は、書類をバインダーなどに綴じ込み、キャビネットなどに入れて保管することでしょう。このとき、キャビネットには「学校関係」「保険関係」など大まかな分類を記し、バインダーには「○○年度」「××生命」などさらに細かい分類を記すはずです。こうすれば、情報のありかは分類をたどって探すことができます。

この「大分類→中分類→小分類」と次第に細かく分類していく構造を**階層構造** (hierarchy) と呼び、広く使われています。情報の整理に階層構造を使うことには、次の利点があります。

- 新たな情報を置くべき場所を、明確に決めることができる。
- 入っている情報を取り出すとき、ありかが簡単に分かる。

このようなことから、コンピュータのファイルシステムでも、階層構造が一般的に使われています。つまり、ディレクトリが容れ物でその中にファイルやディレクトリが入るわけです (情報には物理的な大きさはないので、バインダやキャビネットなど複数種類の容れ物にする必要がないのです)。

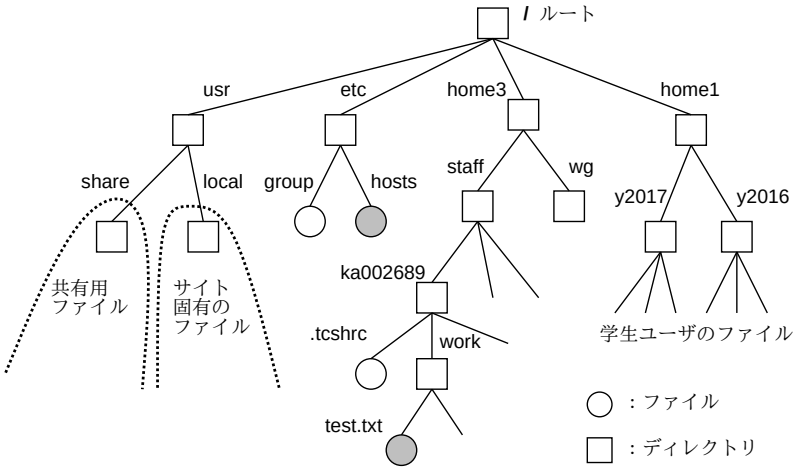

図 2: ファイルシステムの階層構造

図 2 に、sol のファイルシステムの階層構造を簡単化して示します。○で表したのがファイル、□がディレクトリです。一番上にあるディレクトリは**ルートディレクトリ** (root directory) と呼ばれます。これは、枝分かれした構造が木の根元 (root) を想像させるからです (植物の木と上下が反対ですが)。

[1] フラッシュメモリは書き換え回数の上限が数千回〜数万回程度で、それを超えると壊れてしまうので、実質何回でも書き換え可能なディスクの代わりに使うためには書き換え回数を押える工夫が不可欠になります。

ルートの下にはシステムのさまざまな共有ファイルを集めたディレクトリ usr、管理用ファイルを入れたディレクトリ etc、利用者のファイルを集めたディレクトリ home1〜home3 等があります。それぞれのディレクトリの下にはさらに、ディレクトリやファイルが多数置かれています。

2.3 パス名と現在位置 exam

次の問題は、多数あるファイル (やディレクトリ) をどのようにして「これ」と指定するか、です。このために Unix では**パス名** (pathnames) と呼ばれるものを使います。次の例を見てください。

```
/
```

これは、ルートディレクトリを表しています。そして、その次に名前を指定することで、そのディレクトリに直接置かれているものを指定できます。

```
/etc
/home3
```

これにより、管理用ファイルの置き場 etc と、教員のファイルの置き場 (の 1 つ)home3 が指定できます。さらに「/」で区切って名前を指定することで、それらの下のものが指定できます。

```
/etc/hosts
/home3/staff/ka002689/work/test.txt
```

これらは図 2 で灰色で塗って示したファイルです (指定方法はファイルでもディレクトリでも同じ形)。このような、「/」で区切った名前を並べてファイルやディレクトリを指定する記法がパス名です。そして上の例の「/」(ルート) から始まるものは**絶対パス名** (absolute pathnames) と呼びます。

Unix で動いているプログラムは、常にディレクトリのうちの 1 つを「現在位置」ないし**カレントディレクトリ** (current diretory) として覚えています。コマンド pwd は現在位置を表示します。

```
...$ pwd
/home3/staff/ka002689
```

各ユーザがログインした直後の現在位置を**ホームディレクトリ** (home directory) と呼び、各ユーザのファイルは基本的にその下に置かれます。このため、システム上に多数のユーザが居ても、それらのファイルが混ざることは無いわけです。

パス名は「/」以外に名前から始まる場合もあり、これを**相対パス名** (relative pathnames) と呼びます。

相対パス名は、ルートではなく現在位置から、それぞれの名前をたどる意味になります。たとえば、先のファイル test.txt や、ユーザ ka002689 のホームディレクトリにあるファイル .tcshrc は、現在位置が/home3/staff/ka002689 の場合、次のように指定できます。

```
work/test.txt
.tcshrc
```

現在位置は「cd ディレクトリ」により変更できます (単に「cd」だとホームディレクトリに移動)。

```
...$ cd work     (cd /home3/staff/ka002689/work と同じ意味)
...$ pwd
/home3/staff/ka002689/work
```

この場合、現在位置が変わったので、先の 2 つのファイルは次のように指定できます。

```
test.txt
../.tcshrc
```

「..」は「ディレクトリ階層における 1 つ上のディレクトリ」という意味になります。ついでに、現在位置のディレクトリは「.」で指定できます。

2.4 ls — ファイル名一覧の表示 exam

階層構造の全体像が分かったところで、今度は個々のファイル (やディレクトリ) について見て行きましょう。まず必要なのは、ディレクトリに入っているファイルやディレクトリの一覧を見ることでしょう。それには ls コマンドを使います。ただの ls では名前の最初が「.」で始まるものは表示されませんから、それらもあわせて表示させたい場合には、-a というオプションを指定します。どの使い方でも、ディレクトリ名を指定すればそのディレクトリの、指定しなければ現在位置のディレクトリの一覧を表示します。

- ls — ディレクトリにあるファイルの一覧を表示
- ls -F — 名前の末尾に種別を表す文字 (ディレクトリなら「/」等) を付加して表示
- ls -a — 「.」で始まるファイルも含めて表示

これらを使っているようすを見てみましょう。

```
...$ ls
CL16     IED_HOME  WindowsEdu  bin          test1.txt
Desktop  WWW       a.out       public_html  test2.c
...$ ls -F
CL16/     IED_HOME/  WindowsEdu/  bin/          test1.txt
Desktop/  WWW@       a.out*       public_html/  test2.c
...$ ls -a
.             .emacs.d         .lesshst         .viminfo    bin
..            .gconf           .mew.el          CL16        public_html
.Xauthority   .gconfd          .mozilla         Desktop     test1.txt
.cache        .gnome2          .profile         IED_HOME    test2.c
.ced          .gnome2_private  .sqlite_history  WWW
.dbus         .history         .ssh             WindowsEdu
.emacs        .iiim            .tcshrc          a.out
...$
```

「ls -F」の結果を見ると、ディレクトリは「/」つきで表示されることが分かります。また、実行可能なプログラムは「*」つきで表示されます (その他については後で)。「ls -a」の結果を見ると「.」で始まるファイルが多数あると分かります。これは、Unix では各種のプログラムごとに、固有のオプション設定などを「.」で始まるファイルに書く習慣があるためです。そして、いつもそれらが表示されているとうるさいので、-a を指定しない限り ls はそれらを表示しません。

2.5 cat、more、less — ファイルの中身を見る exam

どのようなファイルがあるか分かったら、次はその内容が見たいですね。そのためのコマンドを挙げておきます (less については、man コマンドのところでも説明しました)。[2]

- cat ファイル … — 1 個以上のファイルを次々に表示
- more ファイル — 長いファイルを 1 画面ずつ表示 ([SP] で次の画面、「q」で終了)
- less ファイル — 戻ることもできる more ([SP] で次の画面、「p」で前の画面、「q」で終了)
- od -t x1 ファイル — ファイルの各バイトを 16 進表記で表示
- file ファイル — ファイルの種別を表示

[2] file 以外のコマンドはファイルを指定し忘れるとキーボードから読み込もうとして入力待ちになります。そうなったときに強制終了させるには Ctrl-C を打ってください。

演習 1 ここまでに学んだコマンドを活用して次のことをやってみなさい (ファイル数が多い場合は出力を 1 画面ずつ見るために「`ls -F | less`」などを活用)。

a. 自分のホームディレクトリ以下にどのようなファイルやディレクトリがあるか調べてみる。
b. 以下のディレクトリ (およびその下) にどのようなファイルやディレクトリがあるか 2 つ以上調べてみる — `/usr/share/dict`、`/usr/share/X11`、`/usr/include`、`/etc`、`/bin`
c. 大変すぎない範囲で、sol のファイルシステムのおおまかなリストを作ってみなさい。

3 ファイルのさまざまな操作

3.1 ファイルの各種属性 exam

ファイルには名前に加えて、さまざまな属性がついています。それらの属性を表示するには、`ls` にもっと別のオプションを指定します。

- `ls -l` — 長さを始めとする詳しい情報を表示。

たとえば次のような内容を格納したファイル `test.txt` があるものとしましょう。

```
...$ cat test1.txt
This is a pen.
That is a dog.
```

そしてこれらの英字等はそれぞれ 1 バイトで格納できるものとします (実際できます)。そうすると、このファイルの長さはいくつになるでしょうか? (答えを見る前に数えてみること!)

```
...$ ls -l test1.txt
-rw-r--r-- 1 ka002689 faculty 30  6 22 16:12 2016 test1.txt
```

答えは「30 バイト」ですが、合っていましたか? まず、「空白」も文字として数えることは分かります。しかし空白を含めても、まだ目に見える文字の総数より 2 バイト多いですが、これは各の行の終りに**改行文字** (行の切れ目を表す文字) がくっついているためです。

さて、長さの右にあるのは、ファイルを最後に変更した日時です。変更した日時は記録されていないと、都合が悪そうですね? (たとえば最近変更したファイルだけ見たいなどの場合に。) 一方、長さの左にあるのは、そのファイルの「持ち主」「所属グループ」です。持ち主の情報はなぜ必要なのでしょう? それはもちろん、ファイルを持ち主だけが読めたりするように「保護」するためですね。さらに、Unix ではユーザの他にグループという概念があり、「一緒に作業する仲間」を表します。そして上の表示にあるように、久野の場合は「faculty(教員)」というグループに所属しています。その左の「1」はそのファイルが持つ名前の数 (次に述べる `ln`/`rm` で名前の数が増減します)、さらに左側は保護モードで、少しあとで説明します。

3.2 ファイルとディレクトリの操作 exam

ではいよいよ、ファイルやディレクトリの操作について取り上げます。その前に、ファイルを簡単に作る方法から説明しておきます。それには `echo` というコマンドを使います。

- `echo` 文字列… — 文字列を画面に表示

これだけでは画面に表示するのでファイルと関係ないですが、Unix ではコマンドの末尾に「>ファイル」という指定をすることで「画面に表示する代わりにファイルに保存」できるので、これでファイルを作れます。

```
...$ echo 'This is a pen.'
This is a pen.                    ←指定した文字列が画面に
...$ echo 'This is a pen.' >t1    ←今度は画面に現れない
...$ cat t1                       ←ファイルの中を見ると
This is a pen.                    ←その文字列が
```

次に cp、mv、ln、rm という 4 つのコマンドを説明します。これらのうち最初の 3 つについては、ファイル 2(名前 2) の代わりにディレクトリを指定することもでき、その場合はそのディレクトリの下にファイル 1 と同じファイル名が指定されたものとして扱われます。

- cp ファイル 1 ファイル 2 — ファイル 1 の内容をファイル 2 にコピー (ファイル 2 が無ければ新たに作られる)
- mv ファイル 1 ファイル 2 — ファイル 1 の名前をファイル 2 に変更する (パス名を指定することで別のディレクトリに移動できる)
- ln ファイル 1 名前 2 — ファイル 1 の「別の名前」として名前 2 を追加する (パス名を指定することで別のディレクトリに名前を登録することもできる)
- rm ファイル — ファイルを消去する (正確には名前を 1 つ消去し、名前が 0 個だと本体も消える)

実際に先の続きからやってみましょう。

```
...$ cp t1 t2  ←コピーする
...$ cat t2    ←確かに同じ内容
This is a pen.
...$ mv t2 t3  ←名前変更
...$ cat t2    ←古い名前は…なくなっている
cat: t2: そのようなファイルやディレクトリはありません
...$ cat t3    ←新しい名前はある
This is a pen.
...$ ln t3 t4  ←新しい名前をつける
...$ cat t3    ←古い名前もそのまま使える
This is a pen.
...$ cat t4    ←新しい名前でも同じ内容
This is a pen.
...$ rm t3     ←古い名前を消す
...$ cat t3    ←確かになくなっている
cat: t3: そのようなファイルやディレクトリはありません
```

実はファイルについては、mv は ln で別名をつけてから rm で古い名前を消すのとほぼ同等です。

ディレクトリについての操作も説明しておきます (このほか、ディレクトリも mv を使って名前変更や移動ができます)。

- mkdir ディレクトリ — 新しいディレクトリを作る
- rmdir ディレクトリ — ディレクトリを消す (空っぽである必要がある)

演習 2 ここまでに出て来たコマンドを使って次のことを調べてみなさい。

a. 「echo -n '...' >ファイル」のように echo の-n オプションを使って作ったファイルは使わない場合より少しだけ長さが短い。その理由を調べて説明しなさい。

b. cp でファイルをコピーしてから古いファイルを消すのと、mv で名前を変更するのとでは、どのような違いがあるか検討しなさい。できれば ln と rm の組合せでもやってみるとよい。また、ls に-i オプションを指定すると **i 番号** (i-node number) と呼ばれる固有番号のようなものが表示されるが、これについても検討するとなおよい。

c. ln でファイルの名前を増やすと ls -l で表示される名前の数が増えることを確認しなさい。またその複数の名前がファイルのコピーではなく「同じファイルの別の名前」であることを確認する方法を考え、実際に確認しなさい。できれば、ディレクトリについては名前の個数がどういう意味を持つかを検討するとなおよい。

ヒント: 単に「ls -l ディレクトリ」とすると、そのディレクトリに置かれているファイルの一覧が表示されてしまいます。ディレクトリ「そのもの」の情報を調べたい場合は「ls -ld ディレクトリ」のように「-d」オプションを指定してください。なお、ディレクトリは ln コマンドで名前を増やすことはできませんが、自動的に作られる「.」や「..」は「別の名前」の一種です。

3.3 ファイルの保護設定 exam

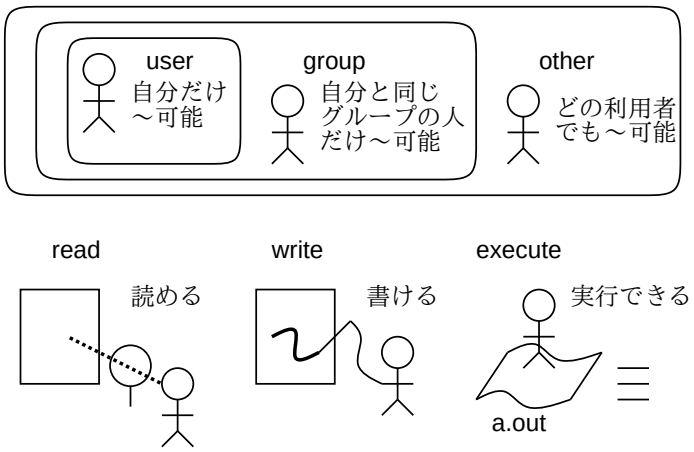

図 3: ファイルの保護設定

Unix では、ファイルの保護設定を**モード** (mode) と呼び、各ファイルごとに「user(持ち主)」「group (グループメンバー)」「other (その他の人)」それぞれについて「read(読む)」「write(書く)」「execute(実行する)」の可否を各々設定できます (図 3)。モード情報は ls -l の表示の最初の部分に含まれます。

```
-rw-r--r-- 1 ka002689 faculty 30  6 22 16:12 2016 test1.txt
```

これは先の例ですが、持ち主 (ka002689) は読み、書きは可能だが実行は不可、それ以外の人には読みだけ可という設定を意味しています。実行というのは、実行可能な (マシン語のプログラムの入った) ファイルに対してのものなので、C 言語で「hello.」と打ち出すプログラムを作って試します。

```
...$ echo 'main() { puts("hello."); }' >test.c
          ↑ 1 行の C プログラム
...$ gcc test.c  ←コンパイル
...$ ls -l a.out ←ファイル a.out に実行形式ができる
-rwxr-xr-x 1 ka002689 faculty 6626  6月 22 16:18 2016 a.out
...$ ./a.out  ←「現在位置の a.out」の指定で実行
hello         ←プログラムの出力
```

Unix ではコンパイラは「a.out」という決まった名前の実行形式ファイルを作るので、これのモードを見ると確かに実行可能になっています。次はまた別のファイルを見ます。

```
...$ ls -l t1
-rw-r--r-- 1 ka002689 faculty 15  6 24 15:56 2016 t1
```

ファイル t1 は、持ち主 (ka002689) は読み書きともに可能ですが、グループ faculty の人、および他の人には読むことのみ可能です。モードの変更は、**chmod** コマンドで次のような形で指定します。

- chmod 対象 (+|-) 許可 ファイル … — モードを設定する

「対象」は u、g、o どれか 1 つ以上 (それぞれ持ち主、グループメンバ、その他の人を表す)、「許可」は r、w、x どれか 1 つ以上 (それぞれ読み、書き、実行を表す) で、「+」はその許可を出し、「-」はその許可を取り除きます。たとえばさっきのファイルでやってみましょう (a は全員という意味で ugo と同じです)。

```
...$ chmod a-rwx t1      ←全員に対して「R」「W」「X」を OFF
...$ ls -l t1
---------- 1 ka002689 faculty 15  6 24 15:56 2016 t1
...$ chmod u+rx t1       ←自分に対して「R」「X」を ON
...$ ls -l t1
-r-x------ 1 ka002689 faculty 15  6 24 15:56 2016 t1
...$ chmod go+x t1       ←グループ/他人に対して「X」を ON
...$ ls -l t1
-r-x--x--x 1 ka002689 faculty 15  6 24 15:56 2016 t1
```

chmod ではもう 1 つの方法として、rwx の 3 ビットを 1 桁の 8 進表現で指定することもできます。8 進表現と 2 進表現 (3 ビット) と保護設定の対応を表 1 に示しておきます。実際には User/Group/Other

表 1: 8 進表現による保護モードの指定

8 進	ビット	保護設定
0	000	---
1	001	--x
2	010	-w-
3	011	-wx
4	100	r--
5	101	r-x
6	110	rw-
7	111	rwx

の順に 8 進表現 3 桁で指定をおこないます。

```
...$ chmod 750 t1   ← 7: rwx, 5: r-x, 0: ---
...$ ls -l t1
-rwxr-x--- 1 ka002689 faculty 15  6 24 15:56 2016 t1
...$ chmod 642 t1   ← 6: rw-, 4: r--, 2: -w-
...$ ls -l t1
-rw-r---w- 1 ka002689 faculty 15  6 24 15:56 2016 t1
```

このように、chmod コマンドを使うことで、さまざまな保護モードが自由に設定できます。

3.4 ディレクトリに対する保護設定 exam

ここまではファイルに対するモードを見て来ましたが、ディレクトリに対しても同じようにモードが設定されています。実際に見てみましょう。

```
...$ ls -ld bin
drwxr-xr-x 2 ka002689 faculty 16  5 18 14:45 2016 bin
```

これまでずっと「-」だったモードの先頭が「d」になっていますが、これが「ディレクトリである」ことを表しています。そして、このディレクトリは「自分には読むことも書くことも実行もできる」「グループや他の人には読むことと実行だけできる」というモードになっています。

しかしディレクトリに対する読み/書き/実行とは、どういう意味でしょう? それは次のようになります (順番を入れ替えました)。

- 実行 (x) — そのディレクトリの下にあるファイルやディレクトリを (パス名で指定して) アクセスできる。このモードが OFF だと、そこにあるファイルは対象の人 (一般/グループ/自分) がアクセスすることは一切ないので保護という点では安全になる (もちろん、自分がアクセスしたいときはディレクトリのモードを変更する)。
- 読み (r) — そのディレクトリの一覧を ls コマンドなどで表示できる。x モードが ON でも r モードが OFF だと、ファイル名を直接知っている人しかその中のファイルにアクセスできないので、知っている人だけにアクセスさせるという効果がある。(中のファイルを読める、書ける等はそのファイル自体のモードによる。)
- 書き (w) — ディレクトリの変更とは、ディレクトリに新しいファイルを追加したり、そこにあるファイルを削除すること。このため、このモードが OFF だと、ファイルを追加したり削除したりは不可能になる (中のファイルの読める、書ける等はそのファイル自体のモードによる)。

ディレクトリ自体に対する保護設定の変更は、ファイルと同様にして chmod コマンドで行えます。

```
...$ chmod go-rx bin
...$ ls -ld bin
drwx------ 2 ka002689 faculty 16  5 18 14:45 2016 bin
```

演習 3 自分の持っているファイルを ls -l で調べ、保護モードを確認しなさい。その後、以下の課題をやってみなさい。

a. 自分のファイルのモードを変更し、「読めない」ものの内容を (cat などで) 表示させようとしたり、「書けない」ものに (cp で内容をコピーするなどで) 書こうとした場合にどうなるか観察しなさい。「誰にでも読めるが自分には読めない」などの矛盾した設定の効果も調べたり、ディレクトリの場合はどうか調べたりできるとなおよい。

b. 他のクラスメートと協力して、他人のファイルやディレクトリについて、a. と同様に観察しなさい。グループや他人に対する保護設定のうまい活用例を考案し、実際にそれがうまくいくことを確認できるとなおよい。

c. 実行形式ファイルの「実行」について、その保護モードを OFF にするとどうなるか試しなさい。また再び ON にした場合どうなるか、実行形式でないファイルについて同様に ON にするとどうなるかも試しなさい。他のクラスメートと協力して、「その人には読めない (コピーできない) けれど実行できる」モードを実験してみて、それがどういう場合に役立つかを検討できるとなおよい。

4 テキストエディタ

ここまで、サンプルファイルを作るのには echo を使ったりしてきましたが、実際に Unix で作業をする場合はあまりそうはしません。まず、ファイルには人間が読める文字が入った**テキストファイル** (text file) と、それ以外の (任意のビット列が入った) **バイナリファイル** (binary file) とがあります。

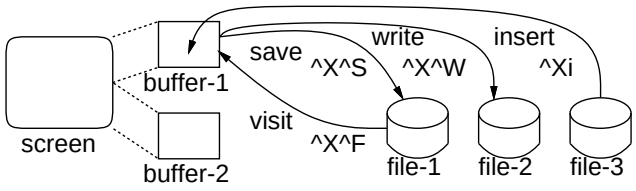

図 4: Emacs の編集モデル

そして、テキストファイルを作成したり修正するためのプログラムのことを**テキストエディタ** (text editor) と呼びます。

ここでは Unix で広く使われている Emacs というテキストエディタについて、その基本部分を取り上げます (後の回でもっと細かいことも扱います)。Emacs の基本的なモデルを図 4 に示します。ファイルを作ったり修正するのは、すべて**バッファ**に持って来て行います。Emacs は多数のバッファを備えていて複数のファイルを同時に編集できます (今回は簡単のためバッファを 1 つしか使いませんが)。バッファの一部 (編集している付近) が画面に常時見えています (このあたりは Word などと同じですが、実際には Word より Emacs の方が先輩です)。

Emacs の特徴の 1 つは、コマンドがすべて Ctrl-□や Alt-□ (Ctrl キーや Alt キーを押しながらどれかのキーを打つ) で始まることです。なぜかというと、エディタで一番多くやるのは「普通の文字を打ち込む」ことなので、文字キーをそのまま打ったら「その文字が入る」ことにしたからです。

Word も文字キーを打ったらそのまま入るのでそこは同じですが、ただし Word ではさまざまな機能はメニューやボタンで指示しますね? それをやると、マウス操作が必要になって時間が掛かり効率が落ちます。そこで Emacs では上述のように Ctrl-□や Alt-□を使うのです (Word でもよく使うコマンドは**キーボードショートカット**として同様になっていますが、Emacs は「キーボードショートカットしかない」わけです)。なお、以下では「Ctrl-□」を「^□」と表記します。また、Alt キーがない環境もありますが、Alt-□は常に「ESC キーを打ってから□のキーを打つ」ことで代替できます。

さて、再び図 4 を見てください。ファイルの読み書き関係のコマンドを説明します。

- `^X^F` ファイル名 `[RET]` (Find File) — ファイルの内容をバッファに読み込んで編集に備える。そのファイルがまだ無い場合は空っぽのバッファからはじまる。
- `^X^S` (Save File) — バッファからファイルに書き戻す (まだ無いファイルを指定していたならこのとき作られる)。
- `^X^W` ファイル名 `[RET]` (Write File) — バッファの内容を別の (ここで名前を指定している) ファイルに書き出す。
- `^Xi` ファイル名 `[RET]` (Insert File) — 指定したファイルの内容をカーソル位置に挿入する。このコマンドでは 2 文字目が Ctrl のない「ただの i」なのに注意。
- `^X^C` (Kill Emacs) — Emacs を終了する (ついでなので説明しておきました)。

では実際に Emacs を起動してみましょう。それにはコマンド「`emacs &`」を実行します (最後の「&」を忘れずに。そうしないと、Emacs が終るまで次のコマンドが打てません)。すると図 5 のような画面が現れるはずです。画面の一番下の行は**ミニバッファ**と呼び、ファイル名入力などのときに使われます。そしてその 1 つ上の行を**モードライン**と呼び、ここにはファイル名や文字コードなどが常時表示されています。

ここで上述のように`^X^F` でファイルを指定して編集を開始します。編集コマンドは今回は最低限しかやりませんが、矢印キー、`[BS]` キー、文字キーだけでも編集はできます (ただし矢印キーより Ctrl=□の方が高速です)。

- 通常の文字キー (self insert) — その文字がカーソル位置に挿入される。
- `^N`, ↓ (Next) — カーソルを 1 行下へ。

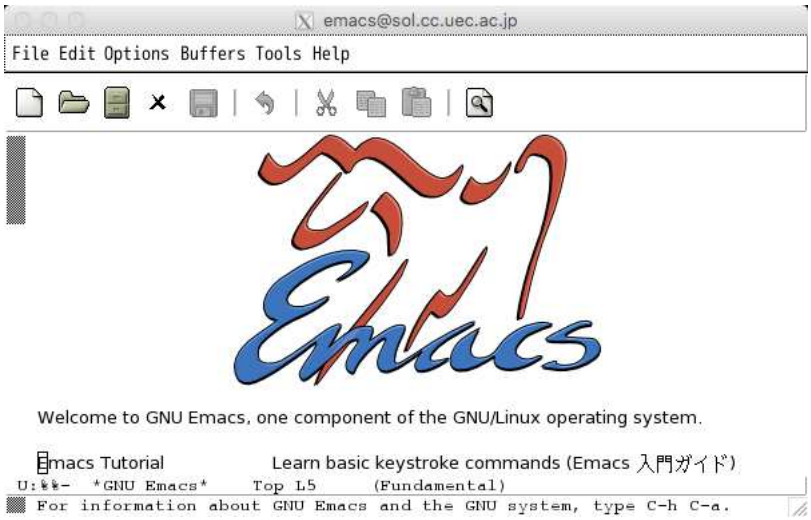

図 5: Emacs の初期画面

- `^P`, ↑ (Previous) — カーソルを 1 行上へ。
- `^F`, → (Forward) — カーソルを 1 文字右へ。
- `^B`, ← (Backward) — カーソルを 1 文字左へ。
- `^D`, `[DEL]` (Delete) — カーソル位置の文字を消す。
- `[BS]` — カーソル位置の直前の文字を消す。

演習 4 「`emacs &`」で Emacs を起動し、`^X^Ftest1.txt[RET]` でファイルの編集を開始しなさい。開始したら次の文章をタッチタイプで打ってみなさい (改行は適宜入れる)。

> Once upon a time, there was a man named Urashima-Taro. One day he saved a turtle on the beach, and the turtle took him to Ryuuguu-Jo under the sea, where Otohime-sama welcomed him. Upon leave, Otohime-sama gave him a box called Tamate-Bako.

一気に打つのでなく、時々`^X^S` で保存し、コマンドの窓でそのファイルの存在や内容を確認すること。間違いがあれば修正して保存・確認する。またストーリーが途中までなのを完成させるとなおよい。最後は保存してから`^X^C` で Emacs を終わること。

本日の課題 5A

本日の課題は「演習 1」「演習 2」「演習 3」に含まれる小問 (合計で 9 個) の中から 1 つ以上を選択し、結果をレポートとして報告して頂くことです。LMS の「レポート＃ 5」の入力欄に直接入力してください (別途作成したものをコピーペーストで貼っても構いません)。以下の内容がこの順に含まれるようにしてください。

- 冒頭に題名「コンピュータリテラシレポート＃ 5」、学籍番号、氏名、提出日付を書く。(グループでやったものはグループのメンバー全員の氏名も別途書く。)
- 課題の再掲を書く (どんな課題であるかをレポートを読む人が分かる程度に要約する)。
- レポート本体の内容 (やったこととその結果) を書く。
- 考察 (課題をやった結果自分が新たに分かったことや考えたこと) を書く。
- 以下のアンケートに対する回答。

Q1. ファイルシステムやファイルの属性についてどれくらい知っていましたか。新たに知って面白かったことは何ですか。

Q2. `ls` によるファイルの表示やその他のコマンドによるファイルの操作を (日付や保護モードなどの参照も含めて) GUI による操作と比較してどう思いますか。

Q3. リフレクション (今回の課題で分かったこと)・感想・要望をどうぞ。

なお、課題はグループでやって構いません。その場合も、(メンバー氏名を明記した上で) レポートは必ず各自で執筆してください。レポート文面が同一 (コピー) と認められた場合は同一であると認めた全員について点数にペナルティを科すことがあります。