プログラミング環境 第3回

久野 靖 *

1991.10.15

5 ファイルシステムと名前空間

5.1 ディレクトリ

これまで、login したらそこで ls というと自分のファイルが見える、というのを当たり前のように考えてきたが、これはよく考えてみるととても不思議なことではないだろうか?例えば ruri さんが login して自分の仕事をした後で hisako さんが同じ機会に login しで仕事ができる、ということから、同じ機械の中に両方の人のファイルがともに存在することは明らかである。でも、**なぜ二人のファイルはまぜこぜになってしまわないのだろう?**

実際、昔私が使っていたシステムでは、全ての利用者のファイルはごちゃまぜに格納されていた。だから、各ファイルは必ずそれぞれの人の固有番号を頭につけて「U5235.TEST.DATA」などのような名前にすることになっていた。[1]これが間違えて「U5545.TEST.DATA」に書いてしまうと簡単に他の人のファイルを壊してしまう、という具合だった (保護機構さえ無かった!)。たとえ保護機構があったとしてもこれでは煩わしいことこの上ない。

実は、Unix ではファイルはディレクトリ (=登録簿、というような意味) という単位にまとめられて管理されている。ディレクトリにはいくらでもファイルを登録しておくことができる。すべてのプロセスには「現在位置」(カレントディレクトリ) が対応していて、特に指定しなければ新しくファイルを作ればそのディレクトリにできるし、ls もそのディレクトリにあるファイルの一覧を表示する。従って、上の質問の答えは「二人のいるディレクトリが違うから」である。この様子を図 1 に示す。なぜ違うか?それは login を担当する部分の処理が、ユーザ名に応じてそれぞ

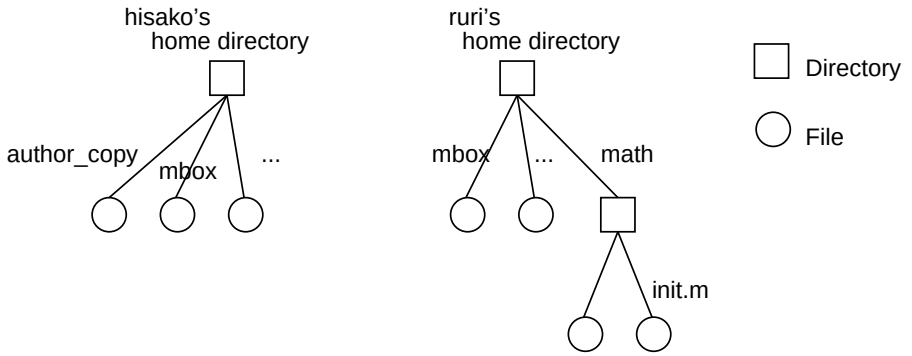

図 1: ディレクトリの概念

*筑波大学経営システム科学専攻

[1]さらに困ったことに、自分がどのようなファイルを現在持っているか知る方法はなく、1 日 1 回 30 分間かかるジョブが動いて各自のファイルの一覧表を打ち出してくれるだけだった。だからその日にどんなファイルを作った/消したかはそのリストに赤鉛筆で書き込んで置かないと分からなくなってしまうのだった!

れに固有のディレクトリを現在位置にしてから csh(コマンドインタープリタ) を起動するからである。この、login した時の現在位置を各自の「ホームディレクトリ」とよぶ。

ところで、実はディレクトリにはファイルだけでなく他のディレクトリを入れることもできる。ディレクトリの中に入っているディレクトリを「サブディレクトリ」という。例えば図 1 では ruri さんは math というサブディレクトリを持っていて、その下に 2 つファイルを持っている。さらにサブサブ、サブサブサブ、… といくら作ってもよい。サブディレクトリを作る/消すには

```
mkdir ディレクトリ名    -- ディレクトリを作る
rmdir ディレクトリ名    -- ディレクトリを消去する
```

による。ディレクトリは `ls -l` ではモード表示の最初に「d」と表示されるので分かる。あるいは、`ls -F` による表示では名前のあとに「/」がついて表示される。

5.2 ディレクトリの木構造

ところで、Unix のファイルシステムは図 1 のようなホームディレクトリがふわふわ沢山浮かんでいるものだ、と思う人はあんまりいないだろう。すでにご想像の通り、実は各自のホームディレクトリ群も、とあるディレクトリの「子」(=サブディレクトリ) だったりする。ということは、「親」がいるわけだ。実は、Unix では全てのディレクトリには「親」がいることになっている。すると、無限に「親」を遡れることになってしまうが...?

またまた実は、Unix のファイルシステムには一つだけ「ルートディレクトリ」と呼ばれるディレクトリがあり、これが文字通りディレクトリの木の「根」になっていて、全てのディレクトリやファイルはこの「子孫」である。ではルートの親は...?あんまり論理的でないが、ルートの親はルート自身、ということになっている。図 2 に我々のサイトのシステムにおけるディレクトリの木構造の概要を示す。このように、我々のシステムでは各ユーザのホームディレクトリはルートの直下にある ua というディレクトリの下にまとまっている。(Q. このような木構造になっていることの利点は何だろうか?)

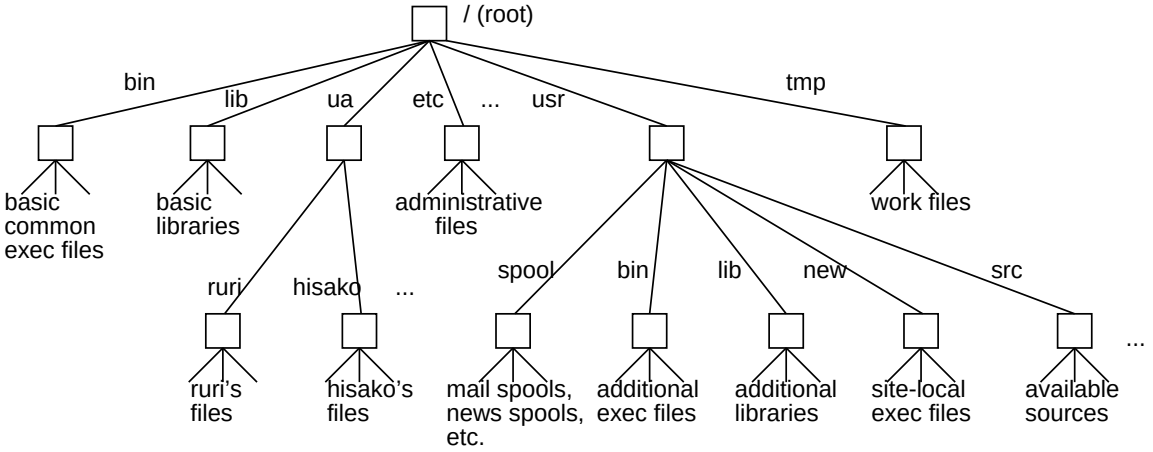

図 2: ディレクトリの木構造

5.3 パス名

ところで、図 2 の中には lib とか bin という名前のディレクトリが複数あるし、もちろん各自が同じファイル名を考えつくこともある。もちろんこれらはディレクトリの木の中で別の場所にある別のものなわけだが、それらを区別して指定する方法が必要である。それには**パス名**というものを使う。パス名には次の 2 通りがある。

```
/名前/名前/.../名前   -- 絶対パス名
名前/名前/.../名前    -- 相対パス名
```

絶対パス名というのは、ルートから始めて指定した名前を順番にたどることで目的のファイルやディレクトリの位置が示されることを意味している。例えば

```
/ua/hisako/mbox -- hisako さんのホームディレクトリにある mbox
/ua/ruri/math/init.m -- ruri さんの math サブディレクトリ下の init.m
/ -- ルートディレクトリそのもの
```

のような具合である。次に、相対パス名というのは、ルートの代わりに現在位置から始めてどうようにたどることを示す。だから、現在位置がhisakoさんのホームディレクトリであれば単に「mbox」でhisako さんのホームディレクトリの下のファイル「mbox」を意味することになる。つまり、これまで「ファイル名」と思っていたのは実は「パス名」の特別な場合だったのだ。ところで、特別な名前として

```
.    -- そのディレクトリ自身
..   -- そのディレクトリの一つ上
```

というのが使える。例えば同じくhisakoさんのホームディレクトリにいる場合だと次のような具合である。

```
.                    -- 自分のホームディレクトリ
../ruri/math/init.m  -- ruri さんの math サブディレクトリ下の init.m
../..                -- ルートディレクトリ
```

相対パス名は絶対パス名より短く指定できるので、ある場所にあるファイル等をたくさん操作する場合はそこへ現在位置を移動してから作業するのが一般的である。そのための指令として次のものがある。

```
pwd         -- 今いる所 (現在位置) の絶対パス名を表示する
cd パス名   -- 指定した場所に行く (つまり、現在位置にする)
cd          -- ホームディレクトリに行く
```

5.4 名前空間

このようにUnixでは全てのファイル/ディレクトリは一つの木に構成されているのだが、実際には一つのディスクに全てのファイルを収めることはとうてい不可能であるし、そうしてしまうとネットワーク経由で共有されている部分、自分固有の部分などを自由に混ぜることができない。そこで、Unixではディスク (正確にはディスクをいくつかに区切って使う、その1区画) ごとにディレクトリの木が存在し、それを張り合わせる (マウントする、という) ことで一つの木に構成するようになっている。この様子を図3に示す。ネットワーク共有もこのディスク単位で行なえる。(Q. OSによってはこのように一つの木にせず、「どのディスクの」を併せて指定するものもある。MS-DOSなどはそうである。そういうのはどういう欠点があると思うか?) どのようなディスクがマウントされているかを知るには次のような指令が利用できる。

```
mount -- ディスクのマウントの状況を表示する
df -- 併せて、ディスクの容量、現在使用量を表示する
```

ところで、このように一つの名前空間 (というのは、名前を指定することでそこにある個々のものが一意に指定できるようになっているもの) を提供することもOSの重要なサービスであるといえる。例えば/tmp/lockという名前のファイルは一つのシステムには一つしか存在し得ないわけだから、このファイルの有無で何かの情報を表したり、このファイルに共有の情報を格納しておいて参照するなど、一つのプロセスで済まないような場合の情報交換にファイルが利用できる。Unixでは単にデータを格納するだけでなく、このようなファイルの利用方法が多い。

このようにせっかくディレクトリによる名前空間があるのだから、システムの中にある様々な「もの」をこれによって参照できると便利である。例えばプロセスもわけの分からない数字で参照するよりは、

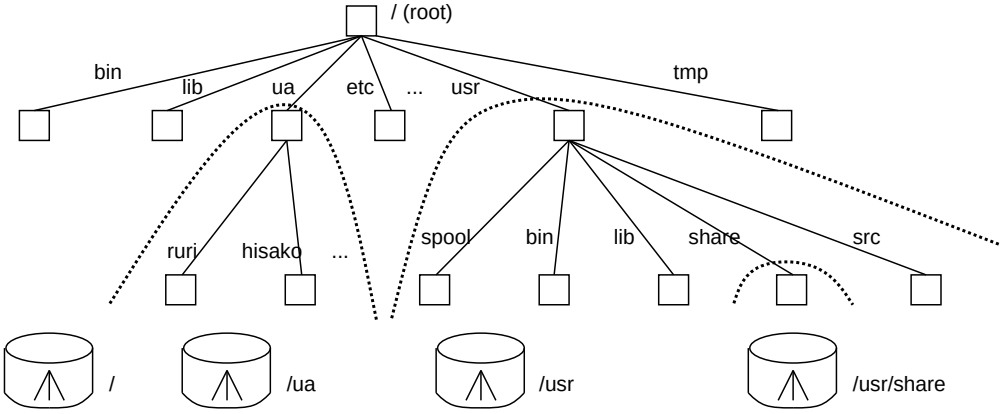

図 3: ディスク装置との対応関係

「ls /process/system/daemon/mailer」などといって表示させられたら嬉しい。残念ながら我々のシステムはそうなっていない[2]が、磁気テープ装置、カートリッジテープ装置、端末回線などの入出力デバイスは Unix の初期からディレクトリの木に含まれるようになっている。具体的にはこれらのデバイスはすべて/dev というディレクトリの下にあり、「/dev/rmt8」とか「/dev/rst8」などパス名で指定できる。そして、例えば MT からデータを読み出すのを「cp /dev/rmt8 t.data」のように普通のコピー指令で行なうこともできる (が、実際は色々な制御をしてくれる専用の指令を使うことが一般的ではある)。また、「/dev/tty」は常に現在使用中の端末デバイスに対応するようになっている。

5.5 再び、ファイルの名前について

ここまでずっと、ファイルに名前という属性がある、という説明をしてきたが、実はそれは**真っ赤な嘘である** (驚きましたか?)。というのではあんまりだから、もう少しおだやかな質問をしてみよう。**Q. ファイルの名前は、どこに格納されていると思うか?**

もちろん、これに答える前にその他のファイルの属性一般はどこに格納されているかを知らないといけない。図 4 に示すように、個々のディスクはそれぞれ i-領域とデータ領域に分かれている。そして、前者には i- ノードと呼ばれるレコードが順番に詰め合わさって入っていて、その一つ一つがファイルまたはディレクトリに対応している。ファイルもディレクトリも持ち主、モードなどの属性を持っているが、これらはこの i- ノードに格納されている。一方、ファイルの中身は?それはデータ領域にある、データブロックに格納されていて、i-ノードにこのファイルのデータブロックはどれとどれ、という情報が入っている。Q. なんでこういう風に分けてあるのだと思うか?

さて、話を戻して、ファイルの名前が他の属性と同じように i-node の中に格納されているとすると、いろいろ困ったことが起こる。(なにか?)　例えば長さがひどく長くてもよいのでその最大の長さぶんの場所を取ると大変である (代案もあるが、あんまりよくない。お分かりかな?)。おまけに ln で名前が二つつけられる場合はどうするか?そして何よりも、ある名前のファイルを探す時にどうするか (まさか、i-領域を頭から順番に見ていくわけには行かないだろう)。

というわけで、ファイルの名前というのはファイルの中には入っていない。ではどこに入っているか?(もうお分かりですね?) ディレクトリである。図 5 左に示すように、ディレクトリは実はファイル名と i-番号の対応を記した表にすぎない。が、これでちゃんと右側に対応する情報が記されているわけである。ここまでずっと、ファイル名やディレクトリ名を○や□の上ではなく、そ

[2]Unix システムの一部ではそのような機能を持たせたものも実在する

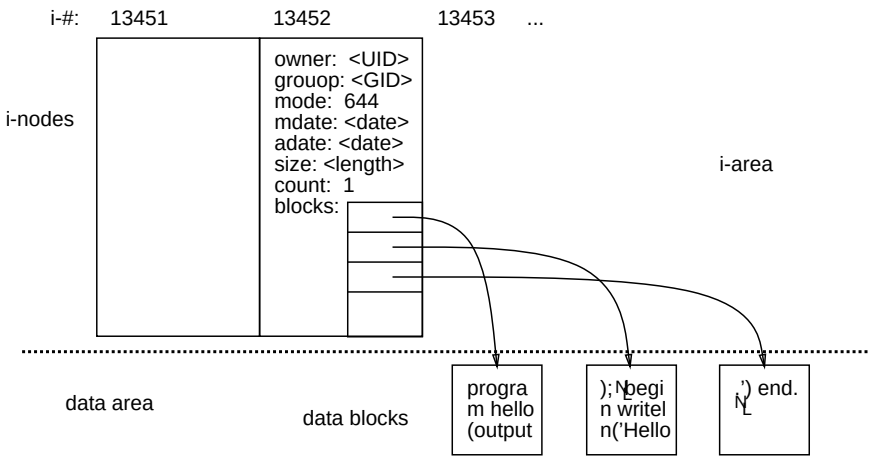

図 4: i-ノード、i-領域とデータ領域

れらを結ぶ線の上に描いてきたことにお気づきだろうか。つまり、これらの名前はファイルやディレクトリについているのではなく、それに向かってたどるリンクについているというのが真実である (普段はあまり意識する必要はないが)。また、. とか.. は自分自身、および親へのリンクだということになる。

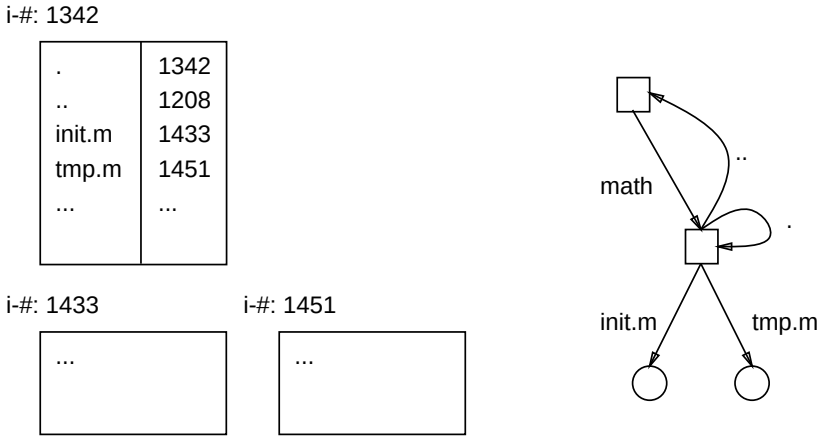

図 5: ディレクトリの中身

ここまで来てようやく ln や rm や mv の意味がちゃんと説明できる。図 6 にあるように、ln というのは既にあるファイルを指すリンクを新しく余計に作る、という意味である。また、rm はリンクを切る指令であるが、もしその結果ファイルを指すリンクが一つもなくなればそのファイルは本当に削除され、その領域は回収される。そして mv は新しいリンクを作ったあとで古いリンクを消すので図の A と B が同じディレクトリなら結果的に名前が変更されたことになる。また違うディレクトリであればファイルの位置が移ったことになる。mv では同様にしてディレクトリの名前や位置を変更することもできるが、これは ln+rm ではできない。というのは、Unix ではディレクトリに複数名前をつけることは許されないようになっているからである。

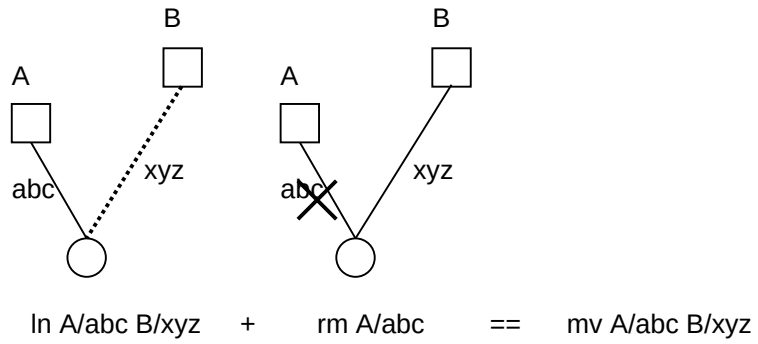

図 6: ln と rm と mv の関係

5.6 ディレクトリとファイルに関するその他の指令

5.6.1 シンボリックリンク

ところで、先にも述べたようにリンクはディレクトリに対して張ることはできない。さらに、普通のファイルでもディスクが違う場合には張ることができない。こういう制限はもっともなことではあるが (なぜもっともかな?) 不便でもある。

これに対して、シンボリックリンクの場合はその「名前を覚えている」だけだから、ディスクが違おうが、ディレクトリに対してだろうが張ることができる。図 8 にシンボリックリンクの概念図を載せておく。

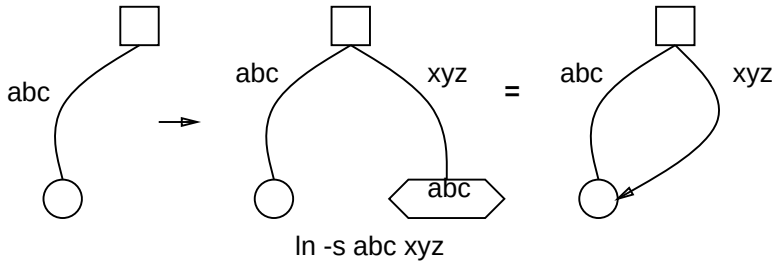

図 7: シンボリックリンクの概念

シンボリックリンクは上述のようにディレクトリを指すことができ、またディスクをまたがってもいいのでファイルの構造を整理するのに便利である。

5.6.2 ディレクトリ単位の操作

上で述べたように、mv を使えばディレクトリごと場所を移動できるのでファイルの整理に便利である。またそれ以外にも

```
mv ファイル... ディレクトリ -- 複数のファイルをいちどに移動
cp ファイル... ディレクトリ -- 複数のファイルをいちどにコピー
```

という使い方もできる。これらの場合は名前そのものはもとのまま、ということになる。またディレクトリの木構造をそのままにそっくりコピーしたり、木構造をそっくり削除するには

```
cp -r ディレクトリ 行き先
rm -r ディレクトリ
```

が使える。間違って自分自身の下にコピーしようとすると無限コピーが始まるので注意。

5.6.3 領域管理、探索

自分のファイルが増えてくるとその管理も大変である。まず、自分がどれだけファイル領域を使用しているかを知るには

```
du ディレクトリ     -- そのディレクトリ以下にあるファイル量合計を示す
du -a ディレクトリ -- 同様だが、個々のファイル名と大きさも表示
```

によるとよい。

大きさだけでなく、様々な条件を指定してそれに合致するファイルを木構造の中で探してくれるのが find である。これには色々なオプションがあるがいくつかの例を挙げる:

```
find ディレクトリ -mtime n -print  -- n 日前に変更したものを探す
find ディレクトリ -size nc -print  -- 大きさ n バイトのものを探す
find ディレクトリ -type x -print   -- 種類 x のものを探す。
                        ただし x は d:ディレクトリ、f:ファイルなど。
```

A　演習および課題

A.1　5 節の演習

5-1. 自分以外の人の持ちものであるファイルを自分のディレクトリの下に置くことは、その人の協力がなくても可能であると思うか。実地に試せ。[3]

5-2. そうやって他人のもののまま持ってきたファイルの中身を書き買えることは、その人の協力がなくても可能かどうか。実地に試せ。[4]

5-3. ディレクトリの書き込みを禁止する、というのは具体的にはどういう効果があるか、実例で示せ。例えばあるファイルを書き込み可能にしておいて、それを含むディレクトリは書き込み不可にしておいた場合、そのファイルの内容を消してしまうことはできるか。逆にあるファイルは書き込み不可だが、それを含むディレクトリは書き込み可能な場合間違ってそのファイルを消してしまうようなことは起きるか。

5-4. ディレクトリの場合は「実行可」というモードは「そのディレクトリをたどる許可」に対応している。読めないがたどることができる、という状態はどういう意味/用途があるか実例で示せ。

5-5. `ln abc xyz` のようにしてシンボリックリンクを作った後、このシンボリックリンク `xyz` を `mv` で他のディレクトリに移すとどうなるか? また、`ln .. up` のようなものではどうか? 実際に試せ。またこのように `mv` したとき変わってしまうという問題を避けるにはどうすればいいかも考えてみよ。

5-6. `cp -r` はシンボリックリンクがあってもそれがシンボリックリンクだと気づかづにそれが指している先のファイルをコピーしてしまう。そのため `cp -r` でディレクトリ構造をそっくりコピーしたつもりでも、コピーでできた構造がもとのと全然違うことがある。そのような例を実際に試して報告せよ。特に、あるシンボリックリンクがそのシンボリックリンクを含んでいるディレクトリ (あるいはさらにその親...) を指しているとどういうことが起きるか。

A.2　3 回目の課題

本日の課題から報告を提出する場合は、上記 5-1〜5-6 のうちから最低２ つ以上選択して下さい。

[3] コピーしてしまったらできたファイルは自分のものになってしまうことに注意。
[4] 誰にでも書けるファイルが必要なら、例えば/ua/kuno/anyone というファイルを用意したのでお使い下さい。