# 続・大型計算機センターでのUNIX入門

南里, 豪志
九州大学大型計算機センター研究開発部

伊東, 栄典
九州大学大型計算機センター研究開発部

https://doi.org/10.15017/1470335

出版情報：九州大学大型計算機センター広報. 31 (3), pp.156-183, 1998-09. 九州大学大型計算機センター
バージョン：
権利関係：

解　　説

# 続・大型計算機センターでの UNIX 入門

南里 豪志 * 伊東 栄典**

UNIX は Windows95 等に比べて一見地味な OS です. しかし, 実は UNIX には痒いところに手が届くような機能が豊富に用意されています. 本記事では, UNIX で作業を楽にするためによく用いられる便利なコマンドやシステムを紹介します. 本稿は, 前回の広報記事 [1] の続編として書かれているので, UNIX の基本的な操作方法や用語についての知識を前提としています. 本稿で分からない点があれば, 参考文献 [1], [2], [3] や, man コマンドによるオンラインマニュアルを参照して下さい. 本稿の内容は, 基本的には本センターの汎用計算機 M1800/20U 上で動作する UXP/M を前提としていますが, 本センターの計算機を利用するために用意された特殊なコマンド以外については, 他のほとんどの UNIX でも利用できます.

## 目次



*九州大学大型計算機センター・研究開発部 E-mail: nanri@cc.kyushu-u.ac.jp
**九州大学大型計算機センター・研究開発部 E-mail: itou@cc.kyushu-u.ac.jp

# 1 ファイル管理

本節では, UNIXにおけるファイルの管理方法について解説します. ここで紹介する機能を使うと以下のようなことが出来ます.

- あるディレクトリについて研究室の内部のユーザーだけがアクセスできるようにする.
- 転送や保管のために複数のファイルを一つのファイルにまとめる.
- ファイル課金の負担を軽くするためにファイルのサイズを小さくする.

## 1.1 ファイルのアクセス権

UNIX のファイルには所有者及び所属グループという属性があります. また, 所有者, 同じグループに所属するユーザー及びそれ以外のユーザーに対して, それぞれファイルのアクセス権, すなわちどのような操作を許可するかが設定されています.

### 1.1.1 アクセス権の表示 — ls -l

ファイルのアクセス権は, ls コマンドに -lオプションを付けて実行することにより見ることができます[1].

```
kyu-cc% ls -l ↵
総ブロック数 1688
-rw-------  1 a79999a user    998 May 22 19:26 README
lrwxrwxrwx  1 a79999a user     21 Jan  7 15:18 VPP -> /KYU-VPP/usr9/a79999a
-rwxr-xr-x  1 a79999a user   4472 May 22 19:26 a.out
drwx------  2 a79999a user    512 May 22 19:18 archives
-rw-r--r--  1 a79999a user 833602 May 22 19:26 sample.ps
-rw-r--r--  1 a79999a user    145 May 22 19:26 sample.txt
kyu-cc%
```

図 1: ls -lの実行例

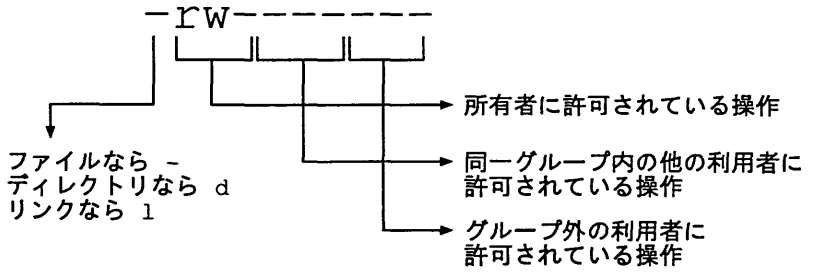



図 2: ファイルのモードの例

[1]本稿で利用する例は, a79999a という ID のユーザーが実行していると仮定しています. 例に書かれたコマンドを試される場合は, ユーザー ID やホームディレクトリ等を自分のものに置き換えて下さい.

図1では, 総ブロック数の下に続く6行で6個のファイルに関する情報が表示されています. これらの行において一番右の文字列がファイル名であり, 一番左の10文字がそのファイルの種類とアクセス権を示すモードと呼ばれる文字列です.

図2に示すように, モードの一番左の文字がファイルの種類を表します. この文字が - の場合は通常ファイル, d の場合はディレクトリであることを示します. また, l の場合はシンボリックリンクであることを示します[2].

モードの残りの9文字でそれぞれのファイルのアクセス権が表示されます. これは, 図2のように3桁ずつの3ブロックに区切られ, 各ブロックがそれぞれ, 所有者・同じグループに属する他のユーザー・それ以外のユーザーに対して許可されているアクセス権を表します. 各ブロックの一桁目で r(読み出し権), 二桁目で w(書き込み権), 三桁目で x(実行権) を表します. r, w, x の代りに -が表示されている場合, そのブロックが対象とするユーザーに対して, そのアクセス権を許可しないことを表します.

図2では, 図1 のREADME という名前のファイルのモードを示しています. まず, モードの最初の1文字が-となっていることから, これが通常ファイルであるということがわかります. 次に, 所有者のブロックで r と w が表示されているので, 所有者に対して読み出し権と書き込み権があることが分かります. それ以外のアクセス権については全て - であるため, その他のユーザーに対する全ての権利と, 所有者に対する実行権は許可されていません.

表 1: 通常ファイルのアクセス権の意味

| 桁 | 現れる文字 | 意味 |
|---|---|---|
| 第1桁 | r | そのファイルの読み出し権 |
| 第2桁 | w | そのファイルに対する書き込み権 |
| 第3桁 | x | そのファイルをコマンドとして実行する実行権 |

表 2: ディレクトリのアクセス権の意味

| 桁 | 現れる文字 | 意味 |
|---|---|---|
| 第1桁 | r | そのディレクトリにあるファイル一覧の読み出し権 |
| 第2桁 | w | そのディレクトリに対するファイルの作成権 |
| 第3桁 | x または - | そのディレクトリへの移動権 |

この, ファイルのアクセス権の意味は, 通常ファイルである場合とディレクトリである場合で多少違います. まず, 通常ファイルにおけるアクセス権の意味を表1に示します. 図1で表示されている通常ファイルのアクセス権は次のようになっています.

- sample.psとsample.txt (-rw-r--r--)
  所有者には読み出し権と書き込み権がある (rw-) . それ以外のユーザー (同じグループ内及びグループ外) には, 読み出し権だけがある (r--) .

- a.out (-rwxr-xr-x)
  所有者には読み出し権と書き込み権と実行権のすべてがある (rwx) . それ以外のユーザーには, 読み出し権と実行権だけがある (r-x) .

[2]シンボリックリンクはファイルへの間接的なポインタです. 例えば先の例の VPP はシンボリックリンクで, その実体は /KYU-VPP/usr9/a79999a にあることが示されています. 実はこのファイル (/KYU-VPP/usr9/a79999a) はkyu-vpp のホームディレクトリです. この VPP によってあたかも kyu-cc にあるディレクトリのようにアクセスできます.

次にディレクトリのアクセス権について説明します. それぞれのアクセス権の意味は表2に示す通りです. すなわち, 図1のディレクトリ archivesのアクセス権 (drwx------) は, 「所有者は, このディレクトリのファイルの一覧を見ることも, ここにファイルを作ることも, このディレクトリに移動することもできるが, それ以外のユーザー (同じグループ及びグループ外) には, 何の操作も許されていない」ということがわかります.

ここで, ディレクトリへの移動が禁止されている場合, コマンド cd で直接移動することはもちろん, そのディレクトリの下のファイルにアクセスすることもできません.

### 1.1.2 アクセス権の変更 — chmod

ファイルのアクセス権はchmod[3]コマンドによって変更することができます. ただし, ファイルのアクセス権を変更できるのはそのファイルの所有者とシステムの管理者だけです.

【コマンドの形式】

```
chmod [-R] [ugoa]{+|-}[rwx] target
```

- [ugoa] では変更の対象となるユーザーを指定します. 文字の意味は, u (所有者), g (同じグループのユーザー), o (グループ外のユーザー) , a (すべてのユーザー) です. 文字を続けて記述することにより複数のカテゴリのユーザーに対してアクセス権を変更することができます. また, 省略するとaを指定したと見做されます.
- {+|-} では変更の方法を表します. 文字の意味は, + (許可を与える, すでに与えられているときは変化なし) , - (許可を取り消す, すでにないときは変化なし) のいずれか1つを選びます.
- [rwx] のところは変更するアクセス権を表します. 文字の意味は, r (読み出し権), w (書き込み権), x (実行権) です. 文字を続けて記述することにより複数のアクセス権を同時に変更することができます.
- *target* がディレクトリである場合, オプション -R を指定することにより, そのディレクトリの下の全てのファイルについてアクセス権を変更します. 指定しなければ, ディレクトリ *target* に対してだけ変更が行われます.
- *target* で指定されるファイルのアクセス権を変更します.

大型計算機センターの UNIX でファイルを作成すると, 特に設定を変更しない限り, 自動的に所有者以外のユーザーに対して全てのアクセス権が禁止された状態に設定されます. これにより, 文書などの機密性が保たれます. しかし, 例えば, 実験データを複数のユーザーが共有する場合等, 他のユーザーにファイルへのアクセスを許可したい場合は, chmod によってアクセス権を変更します.

#### アクセス権の変更例

chmod を用いたアクセス権の変更例を紹介します.

- 所有者自身に実行権を与える

  ```
  kyu-cc% chmod u+x target ↵
  ```

  target が通常ファイルの場合, 所有者はそれをコマンドとして実行できるようになります. また, target がディレクトリの場合, 所有者はそのディレクトリの下に移動できるようになります.

[3]change mode

- 全てのユーザーに読み出し権・書き込み権・実行権のすべてを与える

```
kyu-cc% chmod a+rwx target ↵
```

- 一部のユーザー間でファイルを共有する

上の例のように, ファイルの読み出し権, 書き込み権を全てのユーザーに与え, さらにディレクトリについては実行権を与えることにより, ユーザー間でファイルを共有できます. しかし, これでは全てのユーザーに対してファイルへのアクセスを許してしまいます. そこで UNIX には, 一部のユーザーに対して特別なアクセス権を与えるためにグループという仕組みがあります. しかし, グループの作成や変更を行えるのはシステムの管理者だけです. そのため, 大型計算機センターの UNIX では利用できません.

このように, 大型計算機センターでは一部のユーザー間でファイルを共有することはできません. しかし, ファイルの場所を知っているユーザー間でファイルを共有することはできます. 以下にその方法を示します.

1. 共有するファイルのあるディレクトリを/home/user9/a79999a/workとする. 通常は所有者以外のユーザーはこのディレクトリにアクセスすることはできない.
2. 全てのユーザーに対して, ホームディレクトリ /home/user9/a79999a の実行権を与える.

```
kyu-cc% chmod a+x ~ ↵
```

これにより, 全てのユーザーはこのディレクトリに移動することができる. しかしこのディレクトリの下にあるファイル名を知ることはできない.

3. 全てのユーザーに対して, ディレクトリ /home/user9/a79999a/work の読み出し権と実行権を与える.

```
kyu-cc%  chmod a+rx ~/work ↵
```

これにより, ディレクトリ /home/user9/a79999a/workの存在を知っているユーザーだけが, その下のファイルにアクセスできるようになる[4] (図 3).

そのため, ディレクトリ /home/user9/a79999a/workの下のファイルについて適当なアクセス権を与えることにより, このディレクトリの存在を知っているユーザーだけがその下のファイルにアクセスできるようになります. このようにして, 擬似的に一部のユーザー間でファイルを共有することができます.

## 1.2　ファイルの保管

この節では, ファイルをまとめたり[5]圧縮したり[6]する方法を紹介します. これは, 以下のような場合に利用します.

- 複数のファイルを一括して他の計算機に転送する[7] .

[4]すなわち, このディレクトリ名がパスワードのような役割を果たします.
[5]まとめる とは, 複数のファイルを一つのファイルとして扱えるようにすることです.
[6]圧縮する とは, ファイル中の冗長な部分を無くすことにより, 情報量を減らさずにファイルのサイズを小さくすることです.
[7]大型計算機センターの UNIX の場合, 研究室の計算機にファイルを転送し, 大型計算機センターの計算機上のファイルをできるだけ削除することにより, ファイル課金を節約できます. なお, 転送には ftp を用います. 詳細はオンラインマニュアルを参照して下さい.

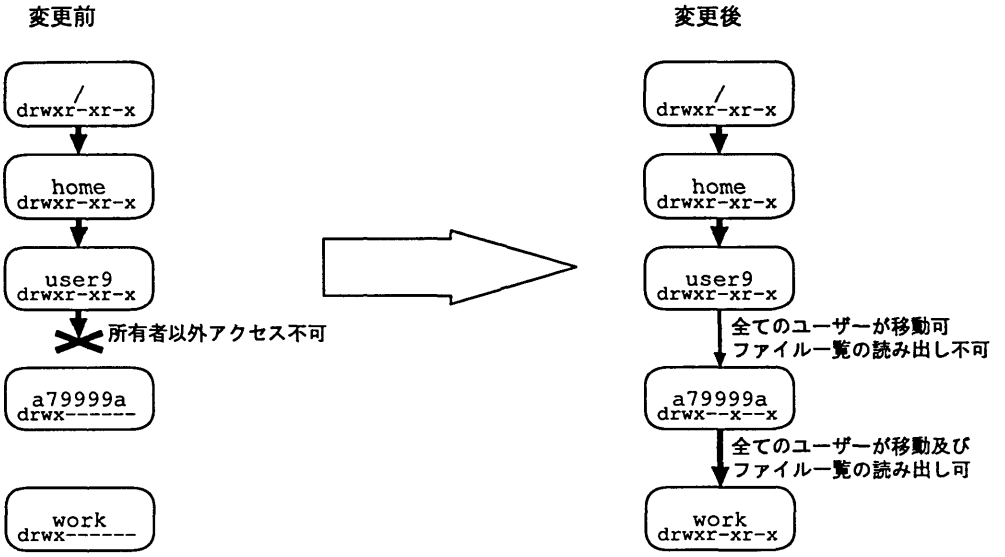



図 3: 一部のユーザー間でファイルを共有する

- ディスクサイズを小さくしてディスクを有効に利用する[8].
- バックアップを取る[9].

### 1.2.1 複数のファイルを一つにまとめる — tar

tar は, 複数のファイルを一つにまとめたり, まとめたファイルを抽出するコマンドです. tar によってファイルをまとめると, 新しく tar ファイルと呼ばれるファイルが作成され, そのファイルの中にまとめられる対象となった全てのファイルが格納されます[10]. tar でディレクトリをまとめる場合, そのディレクトリから下の全てのファイルが対象となります. また, 各ファイルの所有者, 所属グループ, アクセス権等の属性や, ディレクトリの階層構造は保持されます.

【コマンドの形式】

**tar** *key* [ *file1 file2 ...*]

- *key* では, tar コマンドの機能を指定します. *key* には, 基本的な機能を指定する機能文字と, その機能を修飾する機能修飾子を組み合わせて使用します.
- *file1 file2...* にはまとめたり抽出したりする対象となる個々のファイルを指定します. ディレクトリを指定した場合, そのディレクトリの下の全てのファイルが対象となります.

頻繁に用いられる機能文字と機能修飾子は以下の通りです.

[8] これもファイル課金の節約に寄与します.
[9] 磁気テープへのバックアップの方法については本記事では扱いません. 詳細はtar のオンラインマニュアルを参照して下さい.
[10] まとめられる対象となったファイルは削除されません.

| 機能文字 | |
|---|---|
| c | 新規に tar ファイルを作成する. |
| r | 指定したファイルを tar ファイルに追加する. |
| u | まだ tar ファイルに保存されていないか, 既に入っていてもその後ファイルに変更が加わっているファイルだけ保存する. |
| x | 指定したファイルを tar ファイルから抽出する. ファイル名の指定がなければ, tar ファイルに保存されているすべてのファイルを抽出する. |
| t | 指定したファイルを一覧表示する. 指定がなければ tar ファイルに保存されているすべてのファイルを表示する. |

| 機能修飾子 | |
|---|---|
| v | 機能文字によって指定された処理の対象となるファイル名を出力する. 機能文字tと併用すると, ファイルの詳細な情報が得られる. |
| w | 対象となる各ファイルについて, 指定された処理を行うかどうかユーザーに確認する. y を入力すれば処理し, それ以外の入力では処理しない. |
| f *file* | *file*を tar ファイルとして指定する. 機能文字が x で *file* が - の場合は標準入力からの入力を tar ファイルとして抽出を行う. また, 機能文字が c で *file* が - の場合は, まとめた結果を標準出力に出力する. |

通常, tar ファイルにまとめられるファイルは相対パスで指定します. これは, ファイル名を絶対パス名で指定すると, その tar ファイルからファイルを抽出するときも同じディレクトリに抽出するため, 誤ってファイルを上書きする危険性があるからです. これに対して, まとめるときに相対パスで指定されたファイルは, 抽出するときもカレントディレクトリからの相対パスに抽出するため, 一時的に別のディレクトリに抽出することができます.

以下にtarの使用例を示します.

【使用例 1】 カレントディレクトリのworkディレクトリ以下を一つのファイル (work.tar) にまとめる.

```
kyu-cc% tar cvf work.tar ./work ↵
```

【使用例 2】 【使用例 1】でまとめた tar ファイルの中身を表示する.

```
kyu-cc% tar tvf work.tar ↵
```

【使用例 3】 【使用例 1】でまとめた tar ファイルから中身を全てカレントディレクトリに抽出する.

```
kyu-cc% tar xvf work.tar ↵
```

【使用例 4】 【使用例 1】でまとめた tar ファイルから./work/test.txt というファイルだけをカレントディレクトリに抽出する.

```
kyu-cc% tar xvf work.tar ./work/test.txt ↵
```

### 1.2.2 ファイルの圧縮 — compress, gzip

ディスクを有効に利用するため, しばらく利用しないファイルを圧縮することによってサイズを小さくしておく事ができます. UNIX でファイルを圧縮するコマンドとしては, compress と gzip があります. compress はほとんどのUNIX に標準で付属しているのに対し, gzip は必ずしも利用できるとは限りません. これは, gzip がフリーソフトと呼ばれるソフトウェアであるため, システムの管理者がインストールする必要があるからです[11]. 一般に gzip の方がcompress よりファイルを小さく圧縮できます. ここでは両方の利用法を紹介します.

#### compress, uncompress

【コマンドの形式】

**compress** [-cfv] [ *filename...*]

**uncompress** [-cv] [ *filename...*]

compressは指定ファイル *filename* を圧縮[12]し, それぞれのファイル名に拡張子.Zを付けて置き換えます[13]. ファイルの所有者, 所属グループ, アクセス権などの属性は保持されます. ファイルを指定しない場合, 標準入力を圧縮して標準出力へ出力します. compressで圧縮されたファイル (ファイル名に拡張子.Zが付いたもの) は, uncompressを用いて圧縮を解除することができます.

| compress, uncompressのオプション | |
|---|---|
| -c | 処理結果を標準出力に出力します. そのため, ファイルは置き換えられません. |
| -f | そのファイルを圧縮してもサイズが縮小しない場合や, 圧縮の結果出来るファイル名と同じファイル名のファイルが存在する場合でも, 圧縮を実行します. |
| -v | 圧縮される各ファイルの縮小率を表示します. |

#### gzip

【コマンドの形式】

**gzip** オプション *filename* ...

gzipも, ファイルを圧縮するコマンドです. 圧縮の対象となったファイルは, .gzという拡張子を付けたファイル名で置き換えられます[14]. また, compress同様, ファイルの所有者, 所属グループ, アクセス権などの属性は保持されます. ファイル名を指定しないか, ファイル名として「-」と入力した場合は, 標準入力を圧縮して標準出力へ出力します.

gzipにより圧縮されたファイルは, gzip -dによって圧縮を解除することができます. また, 前述のcompressによって圧縮されたファイル (拡張子.Z の付いたファイル) もgzip -d によって解除できます[15] .

オプションの機能は次の通りです.

[11] kyu-vpp, kyu-cc, wisdomには既にインストールしてあります.

[12] 圧縮前後の比率は, ファイルの種類によって異ります. 冗長性の高いテキストファイルの場合, 圧縮によってサイズが圧縮前の50 ～ 10 % 程度になります. これに対して冗長性の低いバイナリファイルの場合, 圧縮の前後でほとんどファイルサイズが変化しないこともあります.

[13] 圧縮の対象となるファイル名に拡張子.Zを付けた名前のファイルが既に存在する場合はエラーとなります.

[14] 圧縮の対象となるファイル名に拡張子 .gzを付けた名前のファイルが既に存在する場合はエラーとなります.

[15] 逆に, gzip で圧縮された .gz の付いたファイルは, uncompress で解除することはできません

| gzipのオプション | |
|---|---|
| -c | 圧縮や解除の結果を標準出力に出力します. 元のファイルは変更されません. |
| -d | 圧縮を解除します. |
| -f | 圧縮や解除の結果出来るファイルと同じ名前のファイルが存在する場合でも圧縮や解除を実行し, 新しいファイルで上書きします. |
| -l | 圧縮されたファイルについて, 圧縮ファイルのサイズ, 非圧縮の場合のサイズ, 圧縮率, 解除した場合のファイル名を出力します. |
| -r ディレクトリ名 | 指定したディレクトリ下の全てのファイルを圧縮や解除の対象とします. |
| -v | 圧縮や解除時に各ファイルの圧縮率を表示します. |
| -*num* | *num*は, 1 から 9 までの整数で, 圧縮速度を規定します. 1 のときが最も高速で, 9 のときが最も低速となります. なお, 基本的に圧縮率は速度に反比例します. 何も指定しない場合の速度は 6 です. |

### tar と gzip (または compress, uncompress) を一括して実行する

あるディレクトリの保存や転送を行う際, そのディレクトリを tar コマンドでまとめた後gzip (または compress, uncompress) で圧縮するという操作は, 頻繁に用いられます. 以下は, ディレクトリ work とその下のファイルをtar でまとめ, その結果できた tar ファイルをさらに gzip で圧縮する手順です.

```
kyu-cc% tar cf work.tar ./work  ↵

kyu-cc% gzip work.tar  ↵
```

しかしこの手順では, ディレクトリ work と最終的に得られるファイル work.tar.gzの他に work.tar という tar ファイルが一時的に必要となります. そのため, もしディレクトリ workの下にあるファイルのサイズの合計が非常に大きいと, ディスクに入りきれず作業を行えない場合があります. これに対してパイプラインとリダイレクトを用いた以下の方法では, ディレクトリ work の下のファイルを tarでまとめながら圧縮するため, ディスクの空き容量が最終的に得られるファイルwork.tar.gz の大きさだけあれば作業を行うことができます.

```
kyu-cc% tar cf - ./work | gzip - > work.tar.gz  ↵
```

compress コマンドを用いる場合は以下のようにします.

```
kyu-cc% tar cf - ./work | compress - > work.tar.Z  ↵
```

元に戻すときも以下のようにすると圧縮の解除とファイルの抽出を一括して行えます.

```
kyu-cc% gzip -cd work.tar.gz | tar xf -  ↵
```

または

```
kyu-cc% uncompress -c work.tar.Z | tar xf -  ↵
```

## 2 プロセス管理

UNIX で実行されたコマンドはプロセスとして扱われます. この節では, プロセスの状態の確認や, 不要なプロセスの削除などを行う方法を紹介します. これにより, 実行途中のプロセスや, 正しく実行されていないプロセスを削除することができるようになります.

### 2.1 プロセスの状態の表示 — ps

【コマンドの形式】

```
ps オプション
```

psのオプション

| | |
|---|---|
| オプション無し | 現在使用中の端末[16]で現在実行されているプロセスの情報を表示します. |
| -f | 同じプロセス群について詳しい情報が表示されます. |
| -u ログイン名 | そのユーザーが実行したプロセスだけが表示されます. |

以下に実行例を示します.

オプション無し

```
kyu-cc% ps ↵
   PID TTY      TIME COMD
  4147 pts/2    0:00 ps
  4070 pts/2    0:00 utms_sh
  4078 pts/2    0:00 csh
kyu-cc% ■
```

PID はプロセスに与えられた番号, TTY はプロセスを制御している端末の名前, TIME はそれまでにそのプロセスが消費した CPU 時間, COMD はそのプロセスを起動したコマンド名を表します.

-f オプション

```
kyu-cc% ps -f ↵
     UID   PID  PPID  C    STIME TTY      TIME COMD
 a79999a  5665  4078  2 14:49:41 pts/2    0:00 ps -f
    root  4070  4068  0 14:17:31 pts/2    0:00 -utms_sh
 a79999a  4078  4070  0 14:17:37 pts/2    0:00 -csh
kyu-cc% ■
```

一番左のUID はそのプロセスを実行したユーザー, STIME はプロセスの開始時刻, PPID はそのプロセスを起動したプロセスの番号[17]を示します. また, C はスケジューリングのためのプロセッサ使用率を示します.

[16]ここでは ps を実行したウィンドウと考えて下さい.
[17]親プロセスと呼びます.

-u オプション

```
kyu-cc% ps -u a79999a ↵
   PID TTY      TIME COMD
  6456 pts/8    0:00 csh
  6497 pts/2    0:00 ps
  4078 pts/2    0:00 csh
kyu-cc% ■
```

## 2.2　不要なプロセスの消去 — kill

【コマンドの形式】

```
kill　オプション プロセス ID
```

| killのオプション |
|---|
| -KILL　プロセスの依存関係に関わらず強制的にプロセスを消去します. |

通常, 処理が正常に終了するとプロセスも消去されます. しかし, まだ終了していないプロセスを強制的に消去させたい場合もあります. kill は, 処理が終了する前に強制的にそのプロセスを消去させるためのコマンドです. 例えば以下のような場合に利用します.

1. 端末が正常に動作しなくなり, コマンドを受け付けなくなった.
2. 非常に長い時間動いているプログラムがあるが, もう待てないので, 今日のところは実行をやめてログアウトしたい.
3. さきほど起動したプログラムにどうやら無限ループがあるらしいので, 負担金を節約するためにこのプロセスを強制的に消去したい.

これらの場合, Ctrl-c (Control キーを押しながら c キーを押す) でプロセスを消去することができる場合もありますが, 出来ない場合は他の端末からログインし, kill コマンドを利用してプロセスを強制的に消去します. これは, 基本的には次のような手順で行います.

1. 他の端末あるいはウィンドウから UNIX に入り直し,

   ```
   kyu-cc% ps -u ログイン名 ↵
   ```

   として, 強制的に消去したいプロセスのプロセス ID を調べます. 以下の例では, psコマンドそれ自身はpts/2で動いていますから, それとは別の端末pts/8で動いている ID 6461 のa.outが強制的に消去したいプロセスです.

```
kyu-cc% ps -u a79999a ↵
   PID TTY      TIME COMD
  6456 pts/8    0:00 csh
  6497 pts/2    0:00 ps
  4078 pts/2    0:00 csh
  6461 pts/8    0:00 a.out
kyu-cc% ■
```

2. プロセスのIDがわかったら, kill コマンドでそのプロセスを終了させます. 例えば, 上記のプロセス6461を終了させるときのkillコマンドの使い方は以下のようになります.

```
kyu-cc% kill 6461 ↵
kyu-cc% ■
```

なお, killコマンドで終了させることができるのは, 自分のプロセスだけで, 他のユーザーのプロセスを終了させることはできません.

## 3 検索

ここでは, あるディレクトリの下から目的のファイルを捜し出すfindコマンドと複数のファイルから特定の文字列を捜し出すgrep, egrepコマンドを紹介します.

### 3.1 ファイルの検索 — find

【コマンドの形式】

**find** 検索開始パス名 検索条件

findは, 検索開始パス名で指定されたディレクトリの下にあるファイルのうち, 検索条件が真となるファイルを検索します[18]. 良く用いられる検索条件としては以下のものがあります. なお, 引数 $n$ の前に + をつけると $n$ より大きい数を, - をつけると $n$ より小さい数を示します. 例えば -atime +10 のように指定すると, ファイルが 11 日以上前にアクセスされた場合に真となります.

[18] より厳密に書くと, 検索開始パス名で指定されたディレクトリの下にある全てのファイルについて, 指定した検索条件が真となるか否かを判定します. 一般には, 検索条件が真となるようなファイルについて, 画面に表示させたり, あるコマンドを実行させたりします.

| findで指定する検索条件 | |
|---|---|
| -name *file* | ファイル名が *file* と一致した場合真です. |
| -atime *n* | ファイルが *n* 日前にアクセスされた場合に真となります. |
| -mtime *n* | ファイルが *n* 日前に変更された場合に真となります. |
| -ctime *n* | ファイルが *n* 日前に作成された場合に真となります. |
| -group *gname* | ファイルのグループが *gname* である場合真です. |
| -user *uname* | ファイルの所有者が *uname* である場合真です. |
| -exec *cmd* \; | *cmd* で示されるコマンドを実行した後, その終了ステータスが 0 の場合真です. 通常, この条件は他の検索条件と組み合わせて用います.<br>*cmd* の後に記号 {} を書くと, その時点で対象としているファイルのパス名で置き換えられます. 例えば, -exec ls -l {} \; と書くと, 条件に合致するファイルについて詳細な情報が表示されます. \; はコマンドの終了を示す記号です. \ の直前が空白でないと, *cmd* が正しく実行されません. 例えば, -exec rm {}\; のように空白を入れないと, 検索条件に関わらずそのディレクトリの下のファイル全てが消去されてしまいます. |
| -print | 常に真となります. 上記の -exec と同様に他の検索条件と組み合わせて用います. すなわち, -printより左の検索条件に合致するファイルのパス名を標準出力に出力します. |

また, 検索条件は, 下記の演算子を使用して組み合わせることができます. 下記では, 演算子の優先度の高い順に記述しています.

1. \( 検索条件 \)
   括弧で囲まれた検索条件が真である場合真になります. 括弧だけだとシェルによって別の意味で解釈されるため, \ を付けます.

2. ! 検索条件
   検索条件の否定を返り値とします. すなわち, 検索条件が偽の場合真となります.

3. 検索条件 [-a] 検索条件
   左右の検索条件の論理積 (AND) を返り値とします. -aは省略することができるため, 検索条件を単に列挙すると論理積と解釈されます.

4. 検索条件 -o 検索条件
   左右の検索条件の論理和 (OR) を返り値とします.

以下にfindの使用例を示します.

【使用例 1】 自分のホームディレクトリ以下のファイルの中から, test.c というファイルを探し, あった場合はそのパス名を表示します.

```
kyu-cc% find ~ -name test.c -print ↵
```

【使用例 2】 10 日以内にアクセスされたファイルを自分のホームディレクトリ以下から探します.

```
kyu-cc% find ~ -atime -10 -print ↵
```

【使用例 3】 /tmp以下で, ファイルの所有者がa79999a以外のファイルを出力します.

```
kyu-cc% find /tmp ! -user a79999a -print ↵
```

【使用例 4】 /usr/local/src/gzip以下のファイルで, ファイル名がREADMEのファイルを探し, あった場合はその内容をcat コマンドで表示します.

```
kyu-cc% find /usr/local/src/gzip -name README -exec cat {} \; ↵
```

【使用例 5】 ホームディレクトリ以下にある1週間アクセスされていないファイルのうち, 名前がa.outであるものと, 拡張子が.oであるものすべてを削除します.

```
kyu-cc% find ~ \(-name a.out -o -name '*.o'\) -atime +7 -exec rm {} \; ↵
```

【使用例 6】 カレントディレクトリ以下にある, 拡張子が .c であるファイルを tar ファイルcfiles.tarにまとめます.

```
kyu-cc% find . -name '*.c' -exec tar rf cfiles.tar {} \; ↵
```

### 3.2 パターン検索 — grep, egrep, fgrep

【コマンドの形式】

**grep** [-clnv] *limited-regular-expression* [ *filename...*]

**egrep** [-clnv] [ *regular-expression*] [ *filename...*]

**fgrep** [-clnv] *string* [ *filename...*]

ファイル中から特定の文字列パターンを検索するのに便利なコマンドとしてgrepがあります. パターン検索grepには, grep, egrep, fgrepの3種類があり, それぞれ以下のような特徴があります.

**grep** 検索するパターンとして限定された正規表現を使用可能です.

**egrep** 検索するパターンとして完全な正規表現を使用可能です.

**fgrep** 検索するパターンを固定の文字列とし, 検索の高速処理を行ないます. 正規表現は使えません.

grepコマンドは, 文字列をファイルの中から検索する簡単な操作から, 複雑な正規表現によるパターン検索まで, 幅広い使い方ができます.

grepの正規表現では, 以下のような特殊記号を用います.

| grepの正規表現 | |
|---|---|
| . | 任意の 1 文字. |
| * | 直前の文字の 0 回以上の繰り返し. |
| ^ | 行の先頭を意味します. |
| $ | 行の最後を意味します. |
| [文字] | 囲まれた文字が検索する文字集合であることを意味します. 例えば, [a-z] はすべての英小文字を表します. また, [^*string*] は, *string* 以外の文字を意味します. |

これらの特殊文字は, シェルに対しても特別な意味を持ちます. ですから, 通常正規表現全体を'...'のように単一引用符で囲みます.

また, egrep では以下の表現も利用できます.

| egrepの正規表現 | |
|---|---|
| + | 直前の文字の 1 回以上の繰り返し. |
| ? | 直前の文字の 0 回以上の繰り返し. |
| 正規表現 1 \| 正規表現 2 | 正規表現 1 または正規表現 2 と一致する文字列. |
| (正規表現) | 正規表現のグループ化. 例えばこの後に * の記号があると, この正規表現の任意回数の繰り返しと一致します. |

ただし, egerpでは, [a-z] のような文字の範囲指定はできません.

grep, egrep, fgrepの主なオプションとしては以下のようなものがあります.

| grep, egrep, fgrepのオプション | |
|---|---|
| -c | 一致したパターンを含む行の行数だけを出力します. 検索対象として複数のファイルが指定された場合は, 行数の前にファイル名も表示します. |
| -l | 一致した行が 1 行でもあるファイルのファイル名だけを, 1 ファイルずつ復帰改行で区切って出力します. パターンが 2 回以上見つかっても, ファイル名の出力は 1 回だけです. |
| -n | 一致したパターンを含む行の行番号を先頭に付けて, 行の内容を出力します. |
| -v | 一致するパターンを含む行以外の行を出力します. |

以下に使用例を示します.

【使用例 1】 カレントディレクトリ内の拡張子が.f90で表される Fortran 90 のソースプログラムからXOCLという文字列を含む行を検索する.

```
kyu-cc% grep XOCL *.f90 ↵
```

【使用例 2】 データ dat1~dat7 から comment という文字列を含まない行を検索し, new-data というファイルに保存する.

```
kyu-cc% grep -v comment dat[1-7] > new-data ↵
```

【使用例 3】 テキストファイルsampleの中で, 行の先頭が数字で始まる行を検索する.

```
kyu-cc% grep '^[0-9]' sample ↵
```

【使用例 4】 プログラム test.fの中から, 文字列 dat_a またはdat_b を含む行を検索する.

```
kyu-cc% egrep 'dat_(a|b)' test.f ↵
```

## 4 一括処理

同じ作業を繰り返すような場合のために, UNIX では一連の操作をファイルに記述しておいて, その内容を連続して実行することが出来ます. そのような方法として, ここではシェルスクリプトと make について説明します. シェルスクリプトの詳細については文献 [2, 3, 5] 等やオンラインマニュアルを参照して下さい. また, make の詳細については文献 [4] 等を参照して下さい.

### 4.1 シェルスクリプト

今まで, キーボードから文字を入力することによって UNIX にコマンドを実行させてきました. この, ユーザからのコマンドを受け付け, それを UNIX に渡すインターフェースの役割を果すものがシェル (shell) です. シェルにはこのような対話的な利用法の他に, シェルに渡す一連のコマンドをプログラムのようにファイルに書いて, 一括して実行させる利用法があります. このように一括して実行させる一連のコマンドをシェルスクリプトと呼びます. シェルには様々な種類があり, それぞれについてシェルスクリプトを利用できます. ここではシェルとして csh (C シェル) を取り上げ, csh で利用可能なシェルスクリプトについて説明します.

#### 4.1.1 シェルスクリプトの作成及び実行

シェルスクリプトは, 基本的にはコマンドを列挙したものです. シェルスクリプトの作成には, Fortran や C プログラムと同じようにテキストエディタを用います. また, ファイルの先頭に # あるいは#!/bin/csh を記述します. 以下に C シェルスクリプトの例 (ファイル名は test.sh とします) を示します.

```
#!/bin/csh
cd /tmp
ls
```

最初の行は C シェルスクリプトである事の宣言です. これ以外の行では, # 以降から行末までの記述は註釈として無視されます. 実際の処理は 2 行目から行なわれます. 2 行目に cd /tmp と書いてあるので/tmpディレクトリに移動します. 3 行目には ls と書いてあるのでls コマンドを実行します.

Fortran や C プログラムと異なり, 作成したシェルスクリプトはコンパイルせずに実行できます.

```
kyu-cc% csh test.sh ↵
```

また, スクリプトが記述されているファイルの実行権を所有者に与えると, そのファイルをコマンドとして実行できるようになります.

```
kyu-cc% chmod u+x test.sh ↵
kyu-cc% ./test.sh ↵
```

この test.sh スクリプトを実行すると, /tmp ディレクトリにあるファイルが表示されます. test.sh スクリプトの実行結果を次に示します.

```
kyu-cc % ./test.sh ↵
#dbwrk.12292/          daemonchk.17289          sa.adrfl
#dbwrk.20710/          daemonchk.709            work.17125
25097/                 jd_sockV4                work.17133
avssock19820           k71417a.19980506134711/  work.17647
```

### 4.1.2　変数と制御構造

シェルスクリプトが通常の対話的利用に比べて優れている点の一つは, 制御構造を利用して複雑な処理を容易に記述出来ることです. また, シェルスクリプトに渡す引数によって動作を変えることもできます.

◆ 変数

20 文字までの英文字, 数字の並びで, 先頭文字が英文字であるものを変数として扱うことができます. *name=value* とすることにより, 変数 *name* を値 *value* に置換することができます. 変数の値は, 名前に記号 $ を付けることにより参照できます.

```
#!/bin/csh
workdir=/home/user9/a79999a/work
cd $workdir
```

◆ 引数

C シェルスクリプトで作成したプログラムに, 引数を与える事ができます. 例えば, ファイルを処理するスクリプトで, 処理対象となるファイルを引数で与えるような場合に利用できます.

C シェルスクリプトの中で, 引数は $1, $2, … のように扱います. すなわち, $1が最初の引数を, $2が2番目というように $n$ 番目の引数を$n$ で表現します. ただし$0はシェルスクリプトのファイル名自身を表現します.

例えば以下のシェルスクリプトは, 引数として与えられた名前のファイルに対し grep コマンドで文字列 “member” を検索します.

```
#!/bin/csh
grep 'member' $1
```

◆ if 文

C シェルのスクリプトで条件文を記述するために if 文が用意されています. if 文の記述形式を以下に示します.

```
1.   if ( expression ) command
2.   if ( expression ) then
         command
     [ else if ( expression ) then
         command ]
     . . .
     [ else
         command ]
     endif
```

if文を使うスクリプトの例を示します. 先ほどのスクリプトの例では指定されたファイルが存在するかどうかをチェックしませんでした. この例ではスクリプトの引数で指定されたファイルがディレクトリ内に存在してかつ普通のファイルであるか, というファイル検査演算 (-f) を条件式に用いています.

```
#!/bin/csh
if ( -f $1 ) then
   grep 'member' $1
else
   echo "$1 doesn't exist."
endif
```

このように, ファイルの種類や属性を調べるファイル検査演算はシェルスクリプト中で頻繁に利用されます. 以下に C シェルで提供されているファイル検査演算を示します.

| ファイル検査演算 | |
|---|---|
| -r filename | ユーザが *filename* の読み取り権を持っていれば真すなわち 1 を, そうでなければ偽すなわち 0 を返します. |
| -w filename | ユーザが *filename* の書き込み権を持っていれば, 真です. |
| -x *filename* | ユーザが *filename* の実行権 (またはディレクトリの検索権) を持っていれば, 真です. |
| -e *filename* | *filename* が存在していれば, 真です. |
| -o *filename* | ユーザが *filename* を所有していれば, 真です. |
| -z *filename* | *filename* のサイズが 0 のとき, 真です. |
| -f *filename* | *filename* がプレーンファイルなら, 真です. |
| -d *filename* | *filename* がディレクトリなら, 真です. |

また, if 文の条件文である *expression* で数値や文字列を用いる場合, 関係演算子として>, <, >=, <=, ==, !=が使用できます.

### ◆ foreach 文

foreach 文を使うと, リスト中の要素ごとに処理を繰り返すことができます. foreach 文の記述形式を示します.

```
foreach name ( wordlist )
   commands
end
```

以下に, カレントディレクトリにある.c で終る全てのファイルのファイル名を, 後ろに .old がついた名前に変更するスクリプトを示します. 下記の例では, まずカレントディレクトリにある .cで終わるファイルを, コマンド ls を使って抽出します. その後, 抽出された各ファイルについてファイル名に拡張子 .oldを付けます.

```
#!/bin/csh
foreach  cfile (`ls *.c`)
   mv    $cfile  ${cfile}.old
   echo  $cfile  ${cfile}.old
end
```

### 4.1.3　バッチキューへのジョブ投入

大型計算機センターのバッチキューへのジョブ投入にもシェルスクリプトを用います．すなわち，処理の内容をシェルスクリプトで記述し，qsub コマンドで投入します．

例えば，「VPP700/56 利用の手引 第 2.0 版」では以下のようなシェルスクリプトが紹介されています．

```
#
cd EXAMPLE
frt -Ps -Wv,-m3 test.f90
a.out
```

この C シェルスクリプトは，

1. EXAMPLE ディレクトリへ移動
2. frt コマンドで test.f90 という名前のソースファイルを翻訳
3. a.out で翻訳したプログラムを実行，

という操作を行なっています．このスクリプトのファイル名を test.sh とすると，以下のようにしてバッチキューにジョブを投入できます．

```
kyu-cc % qsub -q s test.sh ↵
```

## 4.2　make

大きなプログラムを開発する場合，プログラムを一つのファイルに格納するのではなく，複数のファイルに分割した方が，プログラム作成やデバッグ等の作業が容易になります．

Fortran や C で作成プログラム全体が完成するまでには，ソースファイルの編集及びコンパイルを何度も繰り返す必要があるでしょう．しかしプログラムを複数のファイルに分割して作成しておけば，すべてのファイルに対してコンパイルを行う必要はありません．更新したファイルのみをコンパイルすれば良いからです．このようなプログラム群を効率良く管理するために make コマンドがあります．

【make コマンドの形式】

```
make [-f  make ファイル] [ターゲット]
```

makeコマンドは，**make ファイル** に記述された指示にしたがって処理を行います．make ファイルには，予め処理に必要なファイル名や具体的な処理を記述しておきます．make ファイルが指定されていない場合，カレントディレクトリの makefile または Makefileが利用されます．

また，**ターゲット** により，make ファイル中に記述された処理のうち実際に行う処理を指示します．ターゲットが省略された場合 all が指定されたものと見做されます．

### makeファイルの記述

例として以下の3つのファイルからなるプログラムを作っていると考えます.

- Fortran90 で書かれたファイル file1.f90, file2.f90
- C 言語で書かれたファイル test3.c

これらから実行ファイルを作成する場合, 以下のようなコマンドを実行する必要があります.

```
kyu-cc% frt -c file1.f90 ↵
kyu-cc% frt -c file2.f90 ↵
kyu-cc% cc -c file3.c ↵
kyu-cc% frt -o program file1.o file2.o file3.o ↵
kyu-cc%
```

もしプログラムに誤りがあり, ソースファイルの書き換えを行う必要がある場合, 上記のコマンドを毎回入力するのは面倒です. そこで次の様な make ファイルを作成します.

【make ファイルの例】

```
OPTS = -O
COPTS = -DN=100

program: file1.o file2.o file3.o
        frt $(OPTS) -o program file1.o file2.o file3.o
file1.o: file1.f90
        frt $(OPTS) -c file1.f90
file2.o: file2.f90
        frt $(OPTS) -c file2.f90
file3.o: file3.c
        cc $(OPTS) $(COPTS) -c file3.c
clean:
        rm -f file1.o file2.o file3.o *~
backup: ../files.tar.gz
../files.tar.gz: makefile file1.f90 file2.f90 file3.c
        tar -cf - ./makefile ./file1.f90 ./file2.f90 ./file3.c | gzip > ../files.tar.gz
```

make ファイル中の :(コロン) の左側の文字列がターゲットです. この make ファイルには 7 つのターゲットがあります. : の右側はそのターゲットを処理するために必要なファイルが記述されています. 例えば, ターゲット program を処理するためにはfile1.o, file2.o, file3.o が必要です. 実際の処理はその下のアクション行で指定されます. アクション行は必ずタブで始まります. エディタの画面上ではタブと空白との区別が付かないので, make ファイルを作成する時に気を付けて下さい.

プログラムのソースファイルと同一ディレクトリ内に make ファイルを作成し, そのディレクトリで以下を実行すると make が make ファイルを見て必要な作業を開始します.

```
kyu-cc % make program ↵
```

上記の例では program がターゲットとして指定されているので, ターゲット program を作成しようとします. ここで, make は依存関係から自動的に必要な処理を判断し, 実行します. 例えば, ターゲット program の処理にはfile1.o, file2.o, file3.oが必要です. このうち file1.o は, その下のターゲットとなっており, ファイル file1.f90 を必要とします. ここで make は, file1.f90 と file1.oのどちらが新しいかを調べ, file1.f90 の方が新しい場合のみ, すなわち, 前回のコンパイルから, file1.f90 の内容に変更が

加えられた場合のみFortran コンパイラによりコンパイルを行います. 同様にして必要なコンパイルを全て実行すると, program というファイルが作成されます.

もし前回のコンパイルから, プログラムのソースファイルに何の変更も加えずに make コマンドを実行しようとすると,

```
kyu-cc% make program ↵
'program' is up date
```

と表示され何も実行されません. このように, その時点で必要な処理を自動的に判断するという点が, 前節のシェルスクリプトと大きく異る点です[19].

C 言語や Frotran で記述したプログラムをコンパイルする場合, オプションに様々な指定を行う事ができます. それらのオプションなどを毎回記述するのは面倒です. また, オプションの変更をしたい場合, 全ての記述を変更する必要があります. そこで, 同じ記述を繰り返す場合にはマクロを使用すると便利です.

make ファイルでは以下のようにマクロを定義します.

```
COPTS = -g -ansi -Wall -Dlint
```

make ファイルの中ではマクロで定義した変数は,

```
$(COPTS)
```

や,

```
${COPTS}
```

といった記述で参照されます.

## 5　変換

### 5.1　バイナリファイルからテキストファイルへの変換 — uuencode, uudecode

UNIX の通常ファイルは, テキスト形式とバイナリ形式の二つに分類されます. テキスト形式のファイルは電子メールなどで送受信することができますが, バイナリ形式のファイル (画像ファイルや圧縮されたファイルなど) はそのままでは送受信出来ません. uuencodeは, 主に電子メールによる送信のためにバイナリファイルをテキストファイルに変換するコマンドです.

【コマンドの形式】

**uuencode** [ *source-file*] *decode-pathname*
**uudecode** [-p] [ *encode-file*]

*source-file* には, 変換したいバイナリファイルを指定します. この *source-file* の指定がない場合は, 標準入力からの入力になります. 次の *decode-pathname* には, 元のバイナリファイルに戻す際のファイル名を指定します[20].

uuencodeの処理結果は標準出力に出力されるので, 一般にはリダイレクトによって別のファイルに保存します. 例えば, gzip で圧縮されたファイル test.txt.gzをテキストファイルに変換して, ファイル test.txt.gz.uu に保存する場合, 以下のようにします.

```
kyu-cc% uuencode test.txt.gz test.txt.gz > test.txt.gz.uu ↵
```

すると, text.txt.gz.uu には以下のような内容が入ります.

[19]実はシェルスクリプトでも make と同じようなことができます. ただしスクリプトが非常に複雑になるので, このような処理には make が適しています.

[20]この *decode-pathname* の情報は, uuencodeで変換されたテキストファイルの第一行に記述されます.

```
begin 644 test.txt.gz
M'XL("/Q\XC4  W1E<WON='AT +U4WT_;5AA]]U_AQTT+44A'6_+ NA<>-DVJ
M2O\!AE!'I\%$B:;NO[GV5S82,T)^D<1)G.#$"4[L5-/4@2:%$:V4.&E1*4.J
M](4S+G3K.&TJH]8(XFN\WJ)8VH.JG9$NR7C??B.)W]S_>IC.T!_4(9Q>5$7$
M3&HZ92?C;],;)] 90G,*'=:5$NPN:9P#J&7JCOT_TJ_F\\?IW]1L2E.S:E+-
M\SX2Q5H:.5=.JWGV@YI2$P4%$<$XT4!L2[BTCI6Q.YA7B^)UK7Y!MM@"KMDQ
:NOQ !=':-OU\U%*KZOZ_??T'-U$\@*$&  "T

end
```

これはテキストファイルなので, メール中にコピーして送信することができます.

uudecodeは, uuencodeで変換されたテキストファイルを元のバイナリファイルに戻します. *encode-file* には, uuencodeで変換されたテキストファイルを指定します. この指定がない場合は, 標準入力からの入力になります. 通常は, uuencodeの際に指定したファイル名 *decode-pathname* のバイナリファイルに復元されますが, オプションスイッチ-pが指定された場合は, 標準出力へ出力され, パイプ処理などに利用できます.

```
kyu-cc% uudecode test.txt.gz.uu  ↵
```

uuencode や uudecode の対象となったファイルは, 処理の後もそのまま残ります.

### 5.2 漢字コード変換 — nkf

テキストファイル中の日本語は漢字コードと呼ばれる方法で数値化され, 保存されています. 漢字コードとして現在用いられているのは, JIS コード, EUC コード, シフト JIS コードの三種類です. アプリケーションによってはこれらの漢字コードのうち一種類しか扱えないものもあります. そのようなアプリケーションに異る漢字コードによるテキストファイルを処理させると, 正しい処理ができません. nkf はあるテキストファイルの漢字コードを別の漢字コードに変換し, 標準出力に出力します.

【コマンドの形式】

```
nkf [-jse] [ file]
```

| nkfのオプション | |
|---|---|
| -j | 元のファイルを 7 ビット JIS コードに変換します. |
| -s | 元のファイルを シフト JIS コードに変換します. |
| -e | 元のファイルを EUC コードに変換します. |

nkf の使用例

例えば test.txt を EUC コードに変換し, ファイル teste.txt に格納する場合以下のようにします.

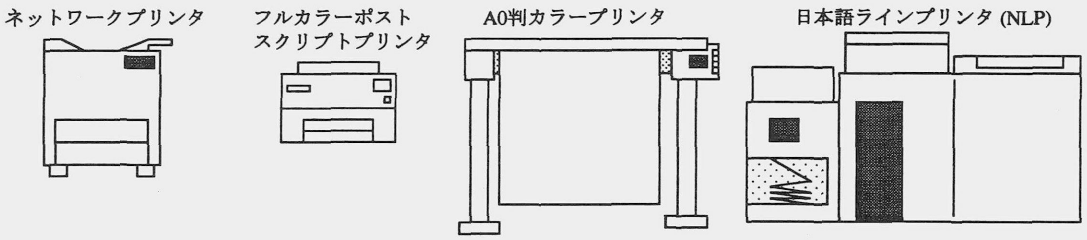
```
kyu-cc% nkf -e test.txt > teste.txt
```

## 6 印刷

九州大学大型計算機センターでは印刷装置として, ネットワークプリンタ, フルカラーポストスクリプトプリンタ, A0 判カラープリンタ, 日本語ラインプリンタ (NLP) を利用することができます. それぞれの設置場所, 利用可能な計算機及び出力コマンドを表 3に示します.

表 3: 大型計算機センターのプリンタ

| プリンタ | 設置場所 | 利用可能計算機 | 利用法 |
|---|---|---|---|
| ネットワークプリンタ | オープン端末室 | kyu-cc, kyu-vpp, wisdom | lp -d ps *options file* |
| フルカラーポストスクリプトプリンタ | 画像機器室 | kyu-cc, kyu-vpp, wisdom, medics | colorps *options file* |
| A0 判カラープリンタ | オープン端末室 | vhsgi | a0lpr *file* |
| 日本語ラインプリンタ | オープン端末室 | kyu-cc | utoprint *file* |




本節では, まず通常の UNIX システムにおけるプリンタの利用方法を紹介します. その後, 大型計算機センターに設置されている特殊なプリンタの利用方法を紹介します.

### 6.1　プリンタへの出力とその管理 — lp, lpstat, cancel

ここでは, テキストファイルや PostScript ファイルをプリンタへ出力する方法, 及び, プリンタ出力の確認や, プリンタ出力の要求取消しを行う方法を紹介します. ただし, 大型計算機センター 2 階のネットワークプリンタを利用することを前提としています. そのため, それ以外のプリンタを利用する場合, 指定方法の違うオプションや, 利用できないオプションがあります. また, システムによってはここで紹介する lp, lpstat, cancel ではなく, lpr, lpq, lprm を利用する場合があります. 大型計算機センター以外のプリンタを利用する場合は, 必ず事前にそのプリンタの利用法をシステムの管理者に問い合わせて下さい.

#### 6.1.1　ファイルをプリンタに出力する — lp

ファイルをプリンタに出力する lp コマンドの形式は以下のようになります.

- lp *options file*

*file* で指定されたファイルをプリンタに出力します. *options* のところではさまざまな指定が可能ですが, 通常以下のようなオプションを利用します.

-d プリンタ名
: 印刷するプリンタを指定します. 現在 kyu-cc や kyu-vpp, wisdom から lp コマンドで直接出力できるプリンタは大型計算機センター 2 階オープン機器室のネットワークプリンタのみです. このプリンタは ps というプリンタ名ですので, -d ps を指定して下さい.

-T ファイルの形式
: 印刷するファイルの形式を指定します. 印刷しようとするファイルが PostScript 形式の場合, 必ず -T ps と指定して下さい. プリントアウトしようとするファイルが PostScript 形式か否かは, more コマンドでファイルを見ると分かります. プリンタに出力しようとする内容が表示されれ

ばテキスト形式です. また, 以下のような行で始まる見慣れない内容が表示されればPostScript形式です[21].

```
%!PS-Adobe-2.0
%%Creator: dvips 5.528 Copyright 1986, 1994 Radical Eye Software
%%Title: paper.dvi
```

もし誤って -T ps オプションを付けずにPostScript 形式のファイルをプリントアウトしてしまった場合, この内容がそのまま印刷されます. その場合, 一般に大量の紙が使われてしまいますので, 後述する cansel コマンドを用いて印刷要求を削除して下さい.

-m
: 指定したファイルの印刷が終了すると, 通知のメイルを送るように指示します.

-w
: 指定したファイルの印刷が終了すると, 端末にメッセージを出力するように指示します. もしその印刷要求を出した利用者がその計算機にログインしていなければ, メイルが送られます.

-y side=both
: 両面印刷を指示します. side=both は間に空白を入れずに入力して下さい. このオプションは両面印刷が可能なプリンタに対してのみ有効です. 現在の大型計算機センターのネットワークプリンタは, 両面印刷が可能です.

### lpの使い方の例

以下の例は, 大型計算機センター 2階のネットワークプリンタ (プリンタ名 ps) に印刷する場合のものです.

- README という名前のテキストファイルを印刷する
  ```
  kyu-cc% lp -d ps README ↵
  ```
- sample.ps という名前の PostScript ファイルを印刷する
  ```
  kyu-cc% lp -d ps -T ps sample.ps ↵
  ```
- README という名前のテキストファイルを両面印刷する
  ```
  kyu-cc% lp -d ps -y side=both README ↵
  ```
- sample.ps という名前の PostScript ファイルを両面印刷する
  ```
  kyu-cc% lp -d ps -T ps -y side=both sample.ps ↵
  ```
- README という名前のテキストファイルを印刷し, 印刷が終了したらシステムからメッセージが送られるようにする
  ```
  kyu-cc% lp -d ps -w README ↵
  ```
- sample.ps という名前の PostScript ファイルを両面印刷し, 印刷が終了したらシステムからメッセージが送られるようにする
  ```
  kyu-cc% lp -d ps -T ps -y side=both -w sample.ps ↵
  ```

lpコマンドを実行すると, ただちにシステムから以下のようなメッセージが出ます.

[21] このファイルの内容は, プリンタに対する細かな印刷制御命令を文字列で表したものです.

```
kyu-cc% lp README ↵
要求 I Dは ps-3777 (ファイル1個) です
kyu-cc% ■
```

3777のところは, 実際にはそのときに応じて数字が変わります. これは, 受け付けられた印刷要求の番号です.

### 6.1.2 自分の出力要求がどうなったか確認する — lpstat

lpstatコマンドは, 現在出されている印刷要求を確認します. 例えば, lpコマンドを実行して大型計算機センター2階のネットワークプリンタのところに行ってみたがまだ印刷されていない, あるいは, lpコマンドで印刷要求を出してしまったがもうその印刷の必要がなくなった, というようなときに, 自分の印刷要求が残っているか否かを確認することができます.

lpstatコマンドの実行例を以下に示します. 通常の利用ではオプションを付ける必要はありません.

```
kyu-cc% lpstat ↵
ps-3782                 a79999a          833602   May 23 11:56
kyu-cc% ■
```

もし, この印刷要求を取り消したいときには, 次に述べるcancelコマンドを使用します.

### 6.1.3 自分の出力要求を取り消す — cancel

既に出した印刷要求を取り消したいときには, cancelコマンドで以下のように指示します.

```
kyu-cc% cancel ps-3785 ↵
要求 "ps-3782" は取り消されました
kyu-cc% ■
```

ここで, ps-3785 は要求 ID です. 印刷要求の ID を知るためには, 前述の lpstatコマンドを使います.

一方, 現在残っている自分の印刷要求をすべて取り消したいときは, 自分のログイン名が a79999a の場合, 次のようにします.

```
kyu-cc% cancel -u a79999a ↵
要求 "ps-3788" は取り消されました
要求 "ps-3789" は取り消されました
要求 "ps-3790" は取り消されました
kyu-cc% ■
```

要求 ID とログイン名による形式のどちらであっても, 一般の利用者が取り消すことができるのは自分が出した印刷要求だけで, 他の利用者の印刷要求を取り消すことはできません.

## 6.2 大型計算機センターの特殊なプリンタの利用

### 6.2.1 フルカラーポストスクリプトプリンタに出力する — colorps

大型計算機センターには, 200 ～ 400dpi で A4 サイズに出力可能なフルカラーポストスクリプトプリンタが用意されています. カラー画像を含むポストスクリプトファイルや, その他の形式のファイルをxvなどで

ポストスクリプト形式に変換したものを鮮明に印刷することができます. また, Macintosh や Windows パソコンなどで作成したカラーポストスクリプトファイルの出力も可能です.

medicsに必要なポストスクリプトファイルを転送しmedicsにログインしてからlpコマンドで出力することで, 印刷可能です.

### フルカラーポストスクリプトプリンタを利用するための準備

フルカラーポストスクリプトプリンタはメディア変換用ワークステーションmedics を用いますので, 予め利用登録をしておく必要があります. 利用登録は kyu-cc で行います. kyu-cc にログインして以下のコマンドを実行して下さい.

```
kyu-cc % touroku all ↵
```

また, kyu-ccからフルカラーポストスクリプトプリンタを利用するためには, その他に, medicsのホームディレクトリにあるファイル.rhosts に次の行を加える必要があります. (.rhostsがなければ, 新しく作成して下さい).

```
kyu-cc a79999a
```

medicsにログインして以下のようなコマンドを実行すると, この行が追加できます.

```
medics% echo kyu-cc a79999a >> ~/.rhosts ↵
```

なお, a79999aのところは自分のログイン名にして下さい. 上記の準備が終ったら, kyu-ccからフルカラーポストスクリプトプリンタに出力できるようになります. また, kyu-cc を kyu-vpp や wisdomに変えて同じ手続きを繰り返すことにより, kyu-cc, wisdom からも出力できるようになります.

フルカラーポストスクリプトプリンタへの出力はcolorps コマンドを利用します.

- colorps -d プリンタ名 *file*

  *file* で指定されたファイルをフルカラーポストスクリプトプリンタに出力します. ファイル名を省略すると, 標準入力をプリンタに出力します. なお, 出力するファイルの形式は PostScript 形式だけです. プリンタ名は, 以下のうちファイルの解像度に合うものを指定して下さい.

  npsf … 400dpi

  npsf320 … 320dpi

  npsf267 … 267dpi

  npsf200 … 200dpi

フルカラーポストスクリプトプリンタは, 専用のプリンタ用紙またはOHP フィルムに印刷できます. ただし, プリントアウトする前に, トレイにどちらが入っているかを確認する必要があります. そこで, フルカラーポストスクリプトプリンタに出力する場合, 必ず大型計算機センター 2 階のオープン端末室でプリンタのトレイを確認し, 必要に応じて取り替えてから使用して下さい. 詳しくは medicsに備え付けの説明書を参照して下さい.

### フルカラーポストスクリプトプリンタへの印刷例

- fullcolor.psという名前のポストスクリプトファイルを 400dpi で印刷する

  ```
  kyu-cc% colorps -d npsf fullcolor.ps ↵
  ```

- fullcolor.psという名前のポストスクリプトファイルを 320dpi で印刷する

  ```
  kyu-cc% colorps -d npsf320 fullcolor.ps ↵
  ```

### 6.2.2 A0 判カラープリンタに出力する — a0lpr

大型計算機センター 2 階の画像機器室には A0 判の大きさで印刷できるカラープリンタが用意されています. これも PostScript 形式のファイルを印刷するプリンタです.

A0 判カラープリンタへの印刷は, センターの可視化サーバvhsgi.cc.kyushu-u.ac.jp からのみ行うことができます. また, ソフトウェアのライセンスの関係で vhsgiはリモートログインを禁止しています. そのため, A0 版カラープリンタに印刷する場合, まず大型計算機センター 2 階の画像機器室にあるワークステーションvhsgi に直接ログインし, 印刷する PostScript 形式ファイルを FTP 等で取ってきて下さい.

A0 判カラープリンタへ印刷するためのコマンドは a0lprです.

- a0lpr *file*
  *file* で指定された PostScript 形式ファイルを A0 判カラープリンタに出力します.

A0 判カラープリンタへ印刷する際の詳細については, 広報記事 [6] や以下の WWW ページを参照して下さい.

```
http://www.cc.kyushu-u.ac.jp/system/A0printer/index.html
```

### 6.2.3 テキストファイルを NLP に出力する — utoprint

MSP システムの日本語ラインプリンタ (NLP) にテキストファイルを打ち出したいときには, utoprintコマンドを使います.

- utoprint *file*
  *file* で指定されたファイルを MSP システムの日本語ラインプリンタに出力します.

  *options* には以下のような指定が可能です.

  -p 印刷モード
  　印刷モードは以下の 4 種類です.
  　lp … 縦 61 行×横 136 文字
  　port … 縦 66 行×横 77 文字
  　land … 縦 46 行×横 110 文字
  　zoom … 縦 53 行×横 136 文字
  　省略時は lp になります.

  -s シートサイズ
  　シートサイズは以下の 4 種類です.
  　a4 … A4 サイズ
  　b4 … B4 サイズ
  　省略時は a4 になります.

コマンドの実行が終ったら, 大型計算機センター 2 階のオープン機器室に来て下さい. 2 台の NLP (A4 と B4) がありますが, その間のコンソールからどちらのプリンタに出力させるか選択するようになっています. 備付のマニュアルを参照して選択して下さい.

## 参考文献

[1] 南里 豪志, 大型計算機センターでの UNIX 入門, 九州大学大型計算機センター広報, pp.61–102, Vol.31, No.2, 1998.

[2] 坂本 文, たのしい UNIX – UNIX への招待 –, アスキー出版, 1990.

[3] 坂本 文, 続・たのしい UNIX, アスキー出版, 1990.

[4] Andrew Oram, Steve Talbott, (矢吹道郎 監訳, 菊地彰 訳), make 改訂版, オライリー・ジャパン, 1997.

[5] G アンダーソン, P アンダーソン 著, (落水浩一郎, 大木敦雄 訳), UNIX C SHELL フィールドガイド, パーソナルメディア, 1987.

[6] 渡部 善隆, 伊東 栄典, A0 版 PostScript データの作成方法, 九州大学大型計算機センター広報, pp.242–255, Vol.30, No.3, 1997.