第2回 UNIX ゼミ

ゼミ担当者　：　折戸　俊彦，荒久田　博士，鈴木　和徳
指導院生　　：　片浦　哲平，谷口　義樹
開催日　　：　2003 年 4 月 18 日

ゼミ内容: 本ゼミでは，研究活動での UNIX の使用において，最低限必要となる知識および操作のスキル取得を目的とする．本研究室では最適化の研究に並列計算機を用いることが多いが，その際に，並列計算機の利用および管理のための LINUX の知識が必要不可欠となる．そこで，本ゼミではそれらの利用が可能となるように，LINUX 上でディレクトリ操作，ファイル操作，パーミッション設定，エディタの利用方法などについて学ぶ．

1　UNIX 上での基本コマンド群

- whoami
  whoami と入力すると自分が誰なのかがわかる．

```
arakuta@mikilab:~$ whoami
arakuta
```

- pwd
  pwd と入力すると自分が今どの場所にいるのかがわかる．下記の例では自由にファイルやディレクトリを作ったり，削除したりできるホームディレクトリと呼ばれるところに居る．

```
arakuta@mikilab:~$ pwd
/home/arakuta
```

- ls
  ls と入力すると，自分が今いるホームディレクトリにどんなファイルがあるのかを知ることができる．

```
arakuta@mikilab:~$ ls
Maildir  foo  test.c  xyz.java
```

- mkdir
  ディレクトリを作成するコマンドである．作成したいディレクトリ名を入力する．自分が現在居る場所に新しいディレクトリが作成される．

```
arakuta@mikilab:~$ mkdir sample
arakuta@mikilab:~$ ls
Maildir  foo  sample  test.c  xyz.java
```

- cd
  別のディレクトリに移動するコマンドである．「cd 移動したいディレクトリ名」と入力する．指定したディレクトリへ移動できる．移動したかどうかは pwd で確認できる．

```
arakuta@mikilab:~$ cd sample
arakuta@mikilab:~/sample$ pwd
/home/arakuta/sample
```

- rm
  「rm ファイル名」でファイルの削除ができる．
  「rm -r ディレクトリ名」でディレクトリの削除ができる．ls コマンドで確認できる．

```
arakuta@mikilab:~$ ls
Maildir  foo  sample  test.c  xyz.java
arakuta@mikilab:~$ rm -r sample
arakuta@mikilab:~$ ls
Maildir  foo  test.c  xyz.java
```

ほかに rmdir というディレクトリを削除するコマンドもあるが，このコマンドでは空のディレクトリのみが削除でき，ファイルがディレクトリ内にあるときはこれらのファイルを削除した後でないとディレクトリは削除できない．

- more
  ファイルの１ページスクロール表示を行うコマンドです.「more 表示したいファイル名」というように入力すると表示することができる．エンターキーで１行ずつの表示が行え，スペースキーで次の１ページの表示が行うことができる．また”q”を押すと処理を中断できる．また，more コマンドの他により高機能な less というコマンドがあります.

```
arakuta@mikilab:~$ more test.c
include<stdio.h>
include<math.h>
```

- cp
  ファイルのコピーをするコマンドである.「cp file1 file2」と入力すると，file1 を file2 にコピーすることができる.

```
arakuta@mikilab:~$ ls
Maildir  foo  test.c  xyz.java
arakuta@mikilab:~$ cp foo foo2
arakuta@mikilab:~$ ls
Maildir  foo  foo2  test.c  xyz.java
```

- mv
  ファイルを移動したり，ファイル名を変更するコマンドである.「mv test sample」と入力すると，test というファイル名を sample というファイル名に変更することができる.

```
arakuta@mikilab:~$ ls
Maildir  foo  foo2  test.c  xyz.java
arakuta@mikilab:~$ mv foo2 foo3
arakuta@mikilab:~$ ls
Maildir  foo  foo3  test.c  xyz.java
```

- ln -s
  ファイルにシンボリックリンクを貼るコマンドである.「ln -s foo bar」と入力すると foo というファイルへを bar というシンボリックリンクを貼ることが出来る.

```
arakuta@mikilab:~$ ls
Maildir  foo  foo3  test.c  xyz.java
arakuta@mikilab:~$ ln -s foo bar
```

2　ファイル・ディレクトリについて

2.1　ファイル情報の意味

ls コマンドで-l オプションをつけると各ファイルについて詳細な情報を見ることができる.

“ls -l” というコマンドを実行した場合，Fig. 2 のような画面が表示される.

はじめの行の “合計 16” というのは一覧に表示されたファイルやディレクトリの合計ブロック数を表す．ブロック数というのは，ディスク中の占める割合で，ファイルサイズとしてはその容量を満たしていなくても，実際のディスク割り当てではその容量分の場所を占めていることになる.

それでは，表示の中の各意味について右側から順に説明する.

- ファイルの種類

  はじめ一文字は，そのファイルがディレクトリ (d) か，シンボリックリンク (l) か，それともふつうのファイル (-) かを表すものである.

- パーミッション

  次の “rwx” の意味であるが，“r” が読み込み権，“w” が書き込み権，“x” が実行権を表す．これは，所有者，グループ，その他の人に分かれていて，それぞれに権利を設定することができる．この部分が，“-” の場合はその権利が与えられていないことを意味する．これは，通常のファイルとディレクトリでは多少意味が異なる．通常のファイルではほぼそのままの意味だが，ディレクトリの場合は，読み込み権とはそのディレクトリの一覧を表示する権利であり，書き込み権とはそのディレクトリの中にファイルを作成する権利であり，実行権とは，そのディレクトリをカレントディレクトリとする権利である.

- ハードリンク数

  次の数字はハードリンク数と呼ばれるものである．これは，このファイルに対して何個ハードリンクさ

```
合計 16
-rwxr-xr-x    1 arakuta   mikilab      2988 Apr 15 22:58 a.out
lrwxrwxrwx    1 arakuta   mikilab         3 Apr 15 21:59 bar -> foo
-rw-------    1 arakuta   mikilab        23 Apr 15 23:49 foo
-rw-r--r--    1 arakuta   mikilab        63 Apr 15 22:03 hello.c
drwxr-xr-x    2 arakuta   mikilab      4096 Apr 16 09:25 testdir
```

Fig. 1　ディレクトリ内のファイル

れているかを示している．ハードリンクとはあるファイル実体に対して，どれだけの参照があるかを表すものである．

- ファイルの所有者

  ファイルの所有者を示す．通常はファイルを作成したユーザになる．

- 所有グループ

  ファイルを所有しているグループを示す．通常はファイルを作成したユーザが属するグループになる．

- ファイルのサイズ

  バイト単位でのファイルのサイズを示す．

- 更新日時

  ファイルが更新された日時を示す．

- ファイルの名前

  ファイルの名前を表す．

  このファイル名で，

  ```
  bar -> foo
  ```

  となっているものがある．これは，シンボリックリンクと呼ばれるもので，“bar” というファイルを指定することで，実際には，“foo” というファイルが指定されるということを表している．シンボリックリンクとは，Windows のショートカットのようなもので，元のファイルを指定してもシンボリックリンクのファイルを指定しても同一のファイルが指定されるというものである．

2.2　所有権とパーミッション

2.2.1　所有権の設定方法

UNIX では，全てのファイルについて，そのファイルを所有しているユーザとグループを設定できる．

ファイルの所有者を変更するには，chown コマンドを使用する．具体的には，

```
chown 新しい所有者 変更するファイル
```

とする．

所有グループを変更するには，chgrp コマンドを使用する．具体的には，

```
chgrp 新しい所有グループ 変更するファイル
```

とする．

2.2.2　パーミッションの設定方法

UNIX でファイルのパーミッションを設定するには，chmod コマンドを使用する．

chmod では，ファイルのパーミッションを設定する場合に 2 つの方法がある．

一つ目の方法は，どのユーザに対してどのパーミッションを割り当てるということを文字で指定する方法である．この方法では，Table 3 にあげられている文字を組み合わせることでパーミッションの設定を行う．たとえば，所有者以外の読込を禁止するには，

```
chmod go-r file
```

とする．

Table 1　chmod で用いる文字

対象	u(所有者), g(グループ), o(その他), a(全て)
操作	+(追加), -(削除), =(設定)
モード	r(読込), w(書込), x(実行)

もう一つの方法は，8 進数を用いてパーミッションを一括設定する方法である．ファイルのパーミッションには，“rwx” の 3 種類がある．そこで，“rwx” の一文字一文字を 2 進数に見立てる．たとえば，rw-の場合は，‘-’ を 0，それ以外を 1 に対応させると，110 となるので，8 進数では 6 になる．これが，所有者，グループ，その他についてのパーミッションがあるので，全体で 8 進数 3 桁となる．Table 4 に，数値指定での数値の意味を示す．

たとえば，

Table 2　chmod での数字の意味

0	一切の権限なし	4	読込
1	実行	5	読込＋実行
2	書込	6	読込＋書込
3	書込＋実行	7	読込＋書込＋実行

```
chmod 600 file
```

とすることで，所有者以外は読むことも書くこともできなくなる．

2.3　ファイルの検索方法

2.3.1　find の使い方

ファイルを検索するには，find コマンドを用いる．find コマンドを用いれば，指定したディレクトリ以下から再起的に，ファイルの名前，ファイルの更新日時，ファイルのサイズといった様々な条件から目的とするファイルを検索することができる．

find でファイルを検索するには，

```
find 検索ディレクトリ... オプション...
```

とする．

Table 5 によく使うオプションを示す．

Table 3　find で使うオプション

オプション名	意味
-name	ファイル名で検索
-size	ファイルサイズで検索
-mtime	作成日時で検索
-atime	アクセス日時で検索
-ctime	ファイル属性変更日時で検索
-perm	ファイルの許可属性で検索

たとえば，カレントディレクトリ以下から foo という名前のファイルを探したかったら，

```
find . -name foo
```

とする．

2.3.2　locate の使い方

ファイルを検索するには，find のほかにも，locate というコマンドを用いる方法ができる．locate は，updatedb コマンドによってあらかじめ作られたデータベースの中から，引数で指定されたパターンにマッチする文字列を含むファイルを検索するものである．

find は与えられた条件に合致するものをディレクトリを再帰的に調べて目的のファイルを見つける．一方，locate はデータベース中から検索するために名前からファイルを検索したいという場合は，find と比べて高速に検索できる．

たとえば，ファイル名に foo という部分を含むファイルを検索したい場合は，

```
locate foo
```

とします．この場合，データベース中に存在するファイルで “foo” という部分があるファイルがすべて表示される．これは，ファイル名だけではなく，ディレクトリの一部に “foo” という文字列が含まれている場合も含めて表示される．

3　UNIX 上のエディタの使い方

UNIX 上の多くのプログラムはその設定をテキスト形式のファイルで保存してる．よって，その設定ファイルを編集することでプログラムの様々な設定を行うことができる．そのためには，UNIX 上でファイルを編集する方法がわからないといけない．

UNIX では一般的に vi と Emacs というそれぞれ特徴のあるエディタが使われている．この節では vi エディタでファイルを編集するための基本的な使い方を説明する．

3.1　vi の使い方

vi は多くの UNIX に標準で添付されているエディタである．vi は Windows に標準添付されているメモ帳やこの後で説明する Emacs などの現在一般的に使われているエディタと違い，操作方法が多少変わっている．まず，それについて説明する．

3.1.1　コマンドモードと入力モード

vi には「コマンドモード」と「入力モード」という 2 つのモードがある．

コマンドモードではファイル中の編集したい位置 (以後，カーソルと呼ぶ) を移動したり，ファイルを保存する，テキストを検索するといった Windows のエディタではメニューバーやツールバー上から行う操作を行う．それに対して，入力モードでは実際に入力したい文字を入力する．

vi は起動したときにはコマンドモードになっている．この状態で ‘i’ と入力すると，入力モードに移り，カーソルのある位置に文字を入力できる．入力が終わったら，ESC キーを押すことで，コマンドモードに戻る．これを図で表すと，Fig.1 のようになる．入力モードに切り替えるにはほかにもいくつかの種類があるが，とりあえず，‘i’ を押せば，文字を入力できる，ということを覚えておけばよい．vi ではコマンドモードと入力モードを行き来することでテキストを編集していく．

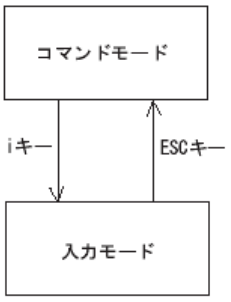

Fig. 2　コマンドモードと入力モードの移り変わり

3.1.2　vi でのファイル編集

vi でファイル編集のための操作を行うときはコマンドモードになってから行う．現在のモードがコマンドモードなのか，入力モードなのかわからないときは，ESC キーを押すことで，コマンドモードになることができる．

この後の説明中のアルファベットは大文字小文字が区別されるので，注意してほしい．

vi でファイルを開くためには，“:e filename” と入力する．あるいは，vi の起動時にコマンドライン上で “vi filename” とすることでファイルが読み込まれる．

ファイルを保存するためには，“:w” と入力する．“:w filename” と入力することで別ファイルに保存することが可能である．

入力された文字を消去するには，消したい文字の上にカーソルを移動して ‘x’ を入力する．“dd” と打ち込むと，カーソルのある一行全体が消去できる．

“/検索したい文字列” とすることで文字列検索することができる．‘/’ の代わりに ‘?’ を使うとファイルの先頭の方に検索を実行する．

vi を終了するためには，“:q” を打つ．編集したテキストを破棄したい場合は，“:q!” と入力する．“:wq” あるいは，“ZZ” とすると保存と同時に終了することが可能である．

4　シェルの活用

4.1　シェルとは

シェル (shell) とは UNIX のコマンドインタプリタで，ユーザ端末から入力された文字列を解釈し，その指示に従って仕事をするプログラムである．しかし，シェルは決して特殊プログラムではない．シェルも他のツールと同様に UNIX 上の１つのコマンドに過ぎない．シェルが他の多くのプログラムと違う点は自分自身がある特定の仕事をするのではなく「他のコマンド類のまとめ役」として機能することだ．シェルには B シェル，C シェルといったように，いくつかの種類があるが，本ゼミでは Linux の標準のシェルとして一般によく利用されている bash について説明する．

4.2　シェルを使う理由

シェルからコマンドを使用すると，作業をかなり速く行えることがあるからである．GNOME や Debian のファイルマネージャでファイルを開き，次にディレクトリを移動してファイルを作成，削除，修正するような作業でも，シェルを使うといくつかのコマンドですばやく作業できる．

4.3　シェルの位置づけ

UNIX は，本来のオペレーティングシステムとしての働きをするカーネルと呼ばれる部分と，ユーザとカーネルの仲介をするシェルと呼ばれる部分からなる．カーネルとはオペレーティングシステムの中枢で，メモリ，ファイル等の管理を行う部分である．

シェルは，ユーザからのコマンド入力を受け取ると，Fig. 3 のように，ヒストリーリストの展開，エイリアスの置き換え，リダイレクション解釈などを行い，カーネルに処理を依頼する．

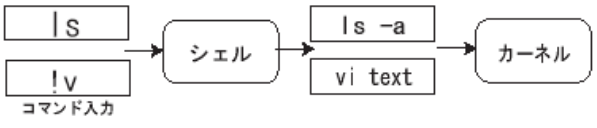

Fig. 3　シェルの仕事

4.4　シェルの機能

シェルの持つ機能は以下で示すようなものである．

- プログラムの実行
- ファイル名の置換
- 入出力の切り換え
- パイプ機能

4.5　コマンドラインの編集

コマンドラインを編集するためには，通常，左右キーでのカーソル移動や，BackSpace による文字削除，上下キーによる履歴の参照などが使われる．

ここでは，それ以外に覚えておくと便利な bash のキー機能について説明する．なお，キー操作の表示に特別な記法を用いるので，その記法に対するキー操作の対応表を Table 6 に示す．

- 行頭・行末への移動
  C-a で行頭に移動．C-e で行末へ移動する．
- 単語単位の移動
  M-b で１つ左の単語の先頭に移動．M-f で１つ右に移動する．

- 単語単位の削除
  M-C-h でカーソルの左の単語を削除．M-d で右の単語が削除できる．

- 行単位の削除
  C-k でカーソルから行末までを削除する．

- 削除した文字列の挿入
  削除された文字列は bash が一時的に記憶している．その記憶している文字列を C-y でカーソル位置に挿入することができる．

Table 4　コマンドとー操作の対応表

C-a	Ctrl キーを押しながら ‘a’ を押す
M-b	Esc キーを押した後，‘b’ を押す
M-C-h	Esc キーを押した後，Ctrl キーを押しながら ‘h’ を押す

4.6　履歴機能

bash はユーザが入力した機能を覚えている．履歴を参照したい場合には「history」コマンドを用いる．最も最近に入力されたコマンドが一番最後に示される．

- 直前のコマンドの実行
  「!!」と入力すると直前のコマンドが実行される．

- 履歴番号からの実行
  「!番号」で履歴のリストの中から，その番号のコマンドが実行される．

- 履歴の検索による実行
  履歴の中のコマンドをある文字列で検索して実行する場合には,「!l」というようなコマンドを実行する．この場合は,「l」で始まるコマンドを履歴の中から検索して実行する．

4.7　補完機能

bash には補完機能というファイル名の省略機能があります．補完機能とは，ファイル名やディレクトリ名をすべて入力しなくても，途中で「TAB」を押すことで補完することができる．

候補が複数ある場合には「TAB」を 2 回押すと候補一覧が表示される．

4.8　ファイル名の置換

シェルはコマンドラインを解析し，実行すべきコマンドや引数を決定する前に，特殊文字を見つけるとその特殊文字部分をファイル名に置き換える．(Table 7 参照)

Table 5　メタキャラクタ

*	任意の文字列を示す
?	任意の 1 文字を示す
[文字 1 文字 2…]	文字 1，文字 2，文字…のいずれか 1 文字を示す
{文字列 1，文字列 2…}	文字列 1，文字列 2，文字列…のいずれかを示す

4.8.1　メタキャラクタの使用例

メタキャラクタの簡単な使用例として，カレントディレクトリに「test12」と「test-ho」というディレクトリを作る．

```
arakuta@mikilab:~$ mkdir test{12,-ho}
arakuta@mikilab:~$ ls
intro  public_html  test-ho  test12
```

次に，今作成されたディレクトリをメタキャラクタを用いて削除する．

```
arakuta@mikilab:~$ rm -r test[12]
arakuta@mikilab:~$ ls
test
```

4.9　入出力の切り換え

シェルは起動したコマンドの入出力先を切り換える機能（I/O リダイレクト）を持っている．シェルはコマンドラインを解釈し，リダイレクトを表す特殊な文字が見つかるとそれに従った処理を行う．(Table 8 参照)

Table 6　リダイレクション記号

リダイレクション記号	役割
<ファイル名	入力を指定したファイルから行う
>ファイル名	出力を指定したファイルに行う
>&ファイル名	エラー出力を含めて，出力を指定したファイルに行う

4.9.1　リダイレクションの使用例

リダイレクションの簡単な使用例として，キーボードより入力した文章を直接ファイルに出力する．

```
arakuta@mikilab:~$ cat > test
This is sample text
```

C-d を入力

```
arakuta@mikilab:~$ more test
This is sample text
```

また，今作成した「test」というファイルを用いて，test1 に出力させることにより，コピーできる．

```
arakuta@mikilab:~$ cat test > test1
arakuta@mikilab:~$ ls
intro  test  test1
arakuta@mikilab:~$ cat test1
This is sample text
```

4.10 パイプ機能

シェルはリダイレクト文字や正規表現をコマンドラインから解釈すると同様にパイプ記号「|」も識別する．シェルはパイプ記号「|」をコマンドラインに見つけるとその前にあるコマンドの標準出力をその後ろにあるコマンドの標準入力に結合させる．そしてシェルは両方のコマンドを同時に実行させる．その例を以下に示す．

```
arakuta@mikilab:~$ who | wc -l
      7
```

まず，シェルはコマンドラインを解析して who と wc の間にあるパイプ記号「|」を見つける．次にシェルは最初のコマンド who の標準出力をそれに続くコマンド wc[1]の標準入力と結合させて 2 つのコマンドの実行を開始する．

その結果，コマンド who はログインしているユーザのリストを標準出力に書き出す．who の標準出力は wc の標準入力につながれてるため端末に who の出力は表示されない．who と同時に起動された wc はファイル名の指定がないので標準入力からの行数を数えるので，who の処理結果の行数を数えることになる．

また，パイプ機能は複数組み込んで使うこともできる．その例を以下に示す．

```
arakuta@mikilab:~$ cat .bash_history | cut -d\  -f1
| sort | uniq -c | sort -rn | cat -n
     1      141 ls
     2       59 cd
     3       29 less
     4       29 cat
     5       21 rm
     6       20 ps
     7       20 mkdir
     8       18 exit
```

今まで使ったコマンドの中で，どれが一番使われているかを順位付けして表示できる．

[1]ファイルのバイト数，文字数，行数を数えるコマンド

4.11 エイリアス機能

bash では「alias」コマンドを使ってコマンドに別名をつけることができる．たとえば,「alias h=”history”」とすれば，それ以降において「h」だけで「history」コマンドを実行することができる．一度設定したエイリアス機能の解除には「unalias h」のようにする．また，設定しているエイリアスの一覧を見るには「alias」とする．以下に，例を示す．

```
arakuta@mikilab:~$ alias less=jless
arakuta@mikilab:~$ alias
alias less='jless'
```

jless とは日本語を読める **less** コマンドであり，このように設定することによって，普通の **less** コマンドでも日本語が読めるようになる．

4.12 ジョブの実行と管理

UNIX では複数のジョブを同時に実行することができる．そのためにも，あるコマンドが終了しないうちに，別のコマンドを実行することができる．

すぐに別のコマンドを実行したい場合には，コマンドの末尾に「&」を付ける．こうするとシェルはプログラムをバックグランドで実行し，そのプロセス ID を表示し，すぐにプロンプトを表示してユーザからの次の入力を受け付け状態になる．ここで表示されるプロセス ID はバックグランドで実行されているコマンドを識別する唯一のものである．

また，普通に何も記述しないでコマンドを実行した場合は，フォアグランドジョブとして扱われ，そのジョブが終了するまで，次のコマンドは実行できない．

リモートログインで UNIX を用いる場合，ログアウト後も UNIX に処理をさせておきたいという場合がある．その場合は，コマンドの頭に「nohup」を記述し，バックグラウンドで実行させておいてからログアウトする．ログアウト後も UNIX は処理を行う．