第2回 UNIXゼミ

ゼミ担当者　：　谷口　義樹，永松　秀人，澤田　淳二
指導院生　　：　上川　純一，片浦　哲平
開催日　　：　2002 年 4 月 23 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の取得を目的とする．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 上でのディレクトリ操作，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　ファイル・ディレクトリについて

1.1　ディレクトリ構造

ファイルを格納する場合に一つの場所だけに全てのファイルを格納しようとすると，それぞれのファイルを識別するためにファイル名を全て別のものにしないといけなくなってしまい，ファイルの管理が非常に大変になります．さらに，UNIX のように複数の人が使うシステムではそれがより面倒なことになってしまいます．

そこで，ディレクトリという概念が出てきます．ディレクトリとは,「京都府の京田辺市の興戸に住んでいる山田さん」といったように，あるものを指定する際に大分類から開始して階層に分けて指定するものです．こうすることで，物事の管理がしやすくなります．ディレクトリが別なら同じファイル名のファイルを作成することも可能です．

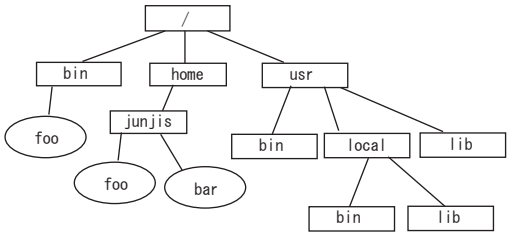

Fig. 1　ディレクトリ構造の例

Fig. 1 の “/” で表されているディレクトリをルートディレクトリといい，全てのディレクトリの起点となります．

現在自分がいるディレクトリをカレントディレクトリといい，あるディレクトリの一つ上のディレクトリを親ディレクトリ，一つ下を子ディレクトリといいます．ユーザがログインしたときに最初にいるディレクトリをそのユーザのホームディレクトリといいます．これは通常は “/home” ディレクトリ以下にユーザ名と同じ名前で，“/home/junjis” のように作成されています．一般ユーザは通常このディレクトリ以下に対してのみ，ファイルを作成する権限が与えられています．

1.1.1　絶対パスと相対パス

ファイルやディレクトリを指定するには，ファイルシステム上での絶対的な位置を指定する方法と自分が今いる位置からの相対位置を指定する方法があります．

絶対的な位置を指定する場合は，“/” を先頭に置いて，ルートディレクトリからの位置を “/home/junjis/sample/test” のように指定します．

相対的な位置を指定するには，先頭に “/” をつけずに，現在のディレクトリを起点として，“sample/test” のようにします．こうすることで，たとえばいま，“/home/junjis” にいたとすると，全体として，“/home/junjis/sample/test” というファイルが指定されることになります．もし，“/home/other” というディレクトリにいたとすると，全体として，“/home/other/sample/test” という別のファイルが指定されることになります．

相対的な位置指定の場合によく用いられるものとして，“.” でカレントディレクトリを指し，“..” で親ディレクトリを指すというものがあります．

1.2　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ます．

“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ます．

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します．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ります．

それでは，表示の中の各意味について右側から順に説明します．

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

Fig. 2　ディレクトリ内のファイル

- 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 (-) かを表すものです.

- パーミッション

  次の “rwx” の意味ですが，“r” が読み込み権，“w” が書き込み権，“x” が実行権を表します．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ます．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します．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ります．通常のファイルではほぼそのままの意味ですが，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す.

- 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す．これは，このファイルに対して何個ハード リンクされているかを示しています．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す.

- ファイルの所有者

  ファイルの所有者を示します．通常はファイルを作成したユーザになります.

- 所有グループ

  ファイルを所有しているグループを示します．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ります.

- 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します.

- 更新日時

  ファイルが更新された日時を示します.

- ファイルの名前

  ファイルの名前を表します.

  このファイル名で，

  ```
  bar -> foo
  ```

  となっているものがあります．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ます．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す.

1.3　所有権とパーミッション

1.3.1　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ます.

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

```
chown 新しい所有者 変更するファイル
```

とします.

所有グループを変更するには，chgrp コマンドを使用します．具体的には，

```
chgrp 新しい所有グループ 変更するファイル
```

とします.

1.3.2　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 コマンドを使用します.

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

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

す．この方法では，Table 1 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います．たとえば，所有者以外の読込を禁止するには，

```
chmod go-r file
```

とします．

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す．ファイルのパーミッションには，“rwx” の 3 種類があります．そこで，“rwx” の一文字一文字を 2 進数に見立てます．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 になります．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りますので，全体で 8 進数 3 桁となります．Table 2 に，数値指定での数値の意味を示します．

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

たとえば，

```
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ります．

1.4　ファイルの検索方法

1.4.1　find の使い方

ファイルを検索するには，find コマンドを用います．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ます．

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...
```

とします．

Table 3 によく使うオプションを示します．

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，

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

```
find . -name foo
```

とします．

1.4.2　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ます．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す．

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ます．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ます．

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ます．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ます．

1.5　プロセス

コマンドを指定して実行すると，それは「ジョブ」と呼ばれる実行単位になります．たとえば，**man cat** というコマンドを実行すると，それは，**man cat** という仕事の単位になります．現在のジョブは **jobs** コマンドで調べることができます．

また，このジョブというものは，端末ごとに見た仕事の単位です．実際には UNIX 本体は **man cat** をもっと小さな仕事の単位で処理しています．**man** コマンドを実行すると，**pager** コマンドが起動されるのがわかります．ユーザが **man cat** を入力すると，UNIX 内部では，

```
20820 pts/5    00:00:00 man
20823 pts/5    00:00:00 sh
20825 pts/5    00:00:00 pager
```

というように 3 つのジョブが起動しています．この小さな実行単位を「プロセス」と呼んでいます．通常は 1 つのジョブに対して 1 つのジョブが対応しています．

現在のプロセスを見るには **ps** コマンドを使います．また，プロセスを強制的に終了させるためには

```
kill プロセス番号
```

2　シェルの活用

2.1　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す．しかし，シェルは決して特殊プログラムではありません．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ません．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です．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りますが，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 について説明します．

2.2　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す．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ます．

2.3　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ります．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す．

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します．

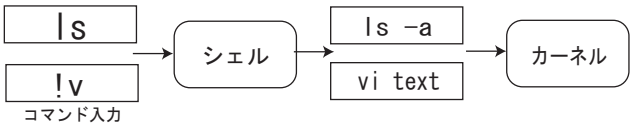

Fig. 3　シェルの仕事

2.4　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す．

- 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

2.5　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ます．

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します．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 4 に示します．

- 行頭・行末への移動
  C-a で行頭に移動．C-e で行末へ移動します．
- 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 つ右に移動します．
- 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ます．
- 行単位の削除
  C-k でカーソルから行末までを削除します．
- 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ます．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ます．

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

2.6　履歴機能

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

- 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ます．
- 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ます．

- 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します．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します.

2.7 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ます.

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ます.

2.8 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ます.(Table 5 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

2.8.1 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ってみましょう.

```
shuto@mikilab:~$ mkdir test{12,-ho}
shuto@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してみましょう.

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

2.9 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）を持っています．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います.(Table 6 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

2.9.1 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してみます.

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

C-d を入力

```
shuto@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，コピーが行えます.

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

2.10 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「|」も識別します．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ます．そしてシェルは両方のコマンドを同時に実行させます．その例を以下に示します.

```
shuto@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「|」を見つけます．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します.

[1]ファイルのバイト数，文字数，行数を数えるコマンド

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します．who の標準出力は wc の標準入力につながれていますから端末に who の出力は表示されません．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ますので，who の処理結果の行数を数えることになります．

また，パイプ機能は複数くみこんで使うこともできます．その例を以下に示します．

```
shuto@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
```

これは，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してくれます．

2.11 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ます．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ます．一度設定したエイリアス機能の解除には「unalias h」のようにしてください．また設定しているエイリアスの一覧を見るには「alias」としてください．いかに例を示します．

```
shuto@mikilab:~$ alias less=jless
shuto@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ります．

2.12 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ます．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ます．

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「&」を付けます．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ります．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す．

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ません．

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ります．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してください．ログアウト後も UNIX は処理を行います．

2.13 シェルスクリプト

UNIX のシェルには，ユーザがプロンプトに対して入力したコマンドを実行するほかに，シェルスクリプトと呼ばれる独自のプログラミング言語を実行する機能を備えています．Windows でいうバッチファイルに似ていますが，いくつかの相違点があり，シェルスクリプトのほうがはるかに複雑で高機能です．ここでは，いくつかの簡単なシェルスクリプト例を示しながら、書き方や変数，制御構造について簡単に学びます．

2.13.1 シェル変数と環境変数

シェルスクリプトでは，シェル変数という変数を利用してプログラムを実行できます．シェル変数はコマンドラインから簡単に設定できます．以下に例を示します．

```
yoshiki@mikilab:~$ NAME=taniguchi
yoshiki@mikilab:~$ mkdir $NAME
yoshiki@mikilab:~$ ls
taniguchi/
```

まず，NAME という変数を定義し，その値として taniguchi を設定します．次に，値が格納されたシェル変数を「$ 変数名」で参照することができます．

```
yoshiki@mikilab:~$ set
```

set コマンドで，シェル変数の一覧を表示します．いま自分が設定した変数のほかにも，あらかじめいくつかの変数に値がセットされていると思います．

```
yoshiki@mikilab:~$ unset NAME
```

unset コマンドでシェル変数を削除できます．ここで，シェル変数はその起動しているシェルにのみ有効な変数であり，それ以外[2]においては使用することはできません．したがって，使用できるようにするには，

```
yoshiki@mikilab:~$ export NAME
```

とします．こうすることでシェル変数は環境変数になります．ここで，環境変数とは UNIX 全体の環境における共通の変数であり，どこでも使用することができます．

[2]例えば，bash から起動された bash においてなど

2.13.2 シェルスクリプトの文法

どのコマンドを用いて，そのシェルスクリプトを解釈するかということを１行目に書くのが最低限のルールです．以下では，if 文，for 文，while 文の記述の仕方を説明します．

2.13.3 if 文

if 文の書式は次のようになります．

```
if 条件文
then
    実行文
elif 条件文
then
    実行文
else
    実行文
fi
```

例を次に示します．

```
#!/bin/sh

num=‘who | wc -l‘
if [ $num -le 10 ]
then
echo "There are few people in MIKILAB"
else
echo "There are many people in MIKILAB"
fi
```

2.13.4 for 文

for 文の書式は次のようになります．

```
for 変数 in リスト
do
  実行文
done
```

例を次に示します．

```
#!/bin/sh

for USER in yoshiki shuto junjis
do
  cp /home/$USER/intro ¥
       /home/yoshiki/introduct/$USER.intro
done
```

2.13.5 while 文

while 文の書式は次のようになります．

```
while 条件文
do
  実行文
done
```

例を次に示します．

```
#!/bin/sh

NUM=0
i=0

while [ $i -lt 10000 ]
do
   NUM=‘expr $i + $NUM‘
   i=‘expr $i + 1‘
done

echo finish! answer’=’$NUM
```

2.13.6 シェルスクリプトの実行

それでは，実際にシェルスクリプトを実行してみます．例として，先ほどの if 文で挙げた例を用います．

```
yoshiki@mikilab:~$ ls -l
-rw-r--r--   …   sample_script

＊「sample_script」に実行属性をつけます．
yoshiki@mikilab:~$ chmod 755 sample_script
yoshiki@mikilab:~$ ls -l
-rwxr-xr-x   …   sample_script*

＊「sample_script」を実行します．
yoshiki@mikilab:~$ ./sample_script
There are few people in MIKILAB
```

2.14 環境の自動設定

環境変数の設定やエイリアスの設定などはログアウトすると無効になります．常に同じ環境で作業をしたい場合には，ログインするたびに同じ設定を繰り返し行われなければなりません．そこで，これらの設定をログイン時に自動的に行う機能を説明します．

2.14.1 .bashrc ファイル

ホームディレクトリにある「.bashrc」ファイルの内容は，ユーザが bash を起動したときに，自動的に読み込まれます．内容の記述方法は，通常のコマンド行の入力と同じように１行ずつ記述するだけです．

```
alias ll=’ls -l’
alias la=’ls -a’
```

この.bashrc ファイルはほとんどの場合，システム管理者があらかじめ各ユーザのホームディレクトリに作成してあります．内容もそれぞれのシステムにあったものが記述されています．特に必要のない限り，元からある行は変更せずに必要な設定を追加してください．もし，ホームディレクトリに「.bashrc」がない場合には，vi エディタなどで作成してください．

2.14.2 .bash_ profile ファイル

ホームディレクトリの「.bash_ profile」は,「.bashrc」と同様にコマンドを記述しておくと，ログイン時に自動的に読まれて実行されます．.bashrc もログイン時に実行したい場合には，.bash_ profile ファイルの最後に 1 行を書いておきます．

```
source ~/.bashrc
```

2.14.3 実用例

```
yoshiki@mikilab:~$ ll
bash: ll: command not found
yoshiki@mikilab:~$ vi .bashrc
alias ll='ls -l'
```
を追加.

```
yoshiki@mikilab:~$ vi .bash_profile
source ~/.bashrc
```
を追加.

<<再びログインしなおす>>

```
yoshiki@mikilab:~$ ll
drwxr-xr-x    ⋯  Apr 19 21:33 ./
drwxr-xr-x    ⋯  Apr 11 12:52 ../
-rw-r--r--    ⋯  Apr  8 15:11 .alias
-rw-------    ⋯  Apr 22 10:11 .bash_history
-rw-r--r--    ⋯  Apr  8 15:11 .bash_logout
-rw-r--r--    ⋯  Apr  9 18:17 .bash_profile
-rw-r--r--    ⋯  Apr 22 10:08 .bashrc
```
（詳細表示）
になる.

2.14.4 PATH について

現在どこに PATH が通っているかを調べるには，

```
yoshiki@mikilab:~$ echo $PATH
/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/games
```

というように，環境変数 PATH を参照します．[3]また，新たに PATH を追加したいときには，次のような作業を行います．

```
yoshiki@mikilab:~$ export PATH=/sbin:$PATH
yoshiki@mikilab:~$ echo $PATH
/sbin:/usr/local/bin:/usr/bin:/bin:  ¥
/usr/bin/X11:/usr/games
```

[3]どのような環境変数があるかは env コマンドで一覧表示が可能

3　エディタの使い方

viでの代表的なキー操作をTable 7に示します．各コマンドの前に数字を入力すると，その回数分だけ同じ動作を繰り返します．たとえば，“5x” とすると，カーソル位置から5文字分を削除できます．

Table 7　viのキー操作

:q	viを終了
:q!	変更があってもセーブせずに終了
:w	ファイルをセーブ
:wq	セーブして終了
i	現在のカーソル位置から挿入
R	現在のカーソル位置から上書き挿入
I	現在行の先頭から挿入
A	現在行の末尾に挿入
O	現在行の前に空行を挿入
o	現在行の次に空行を挿入
0	行頭へ移動
$	行末へ移動
Enter	次の行の先頭へ
Ctrl+f	次の画面へ
Ctrl+b	前の画面へ
1G	文頭へ
G	文末へ
nnG	nn行目へ
x	1文字削除
cw	1語変更
c$	カーソル位置から行末までを変更
dw	1語削除
d$	カーソル位置から行末までを削除
/正規表現	前方検索
?正規表現	後方検索
n	次の候補
N	前の候補
:1,$s/正規表現/置き換え文字列/g	文書内の全ての「正規表現」を「置き換え文字列」に置換
yy	コピー
dd	カット
p	カーソル位置にペースト
P	カーソル位置の次の行にペースト
.	直前の変更操作の繰り返し
u	直前の変更操作の取り消し
:r filename	ファイル内容を読み込む
:r! command	コマンドを実行し，結果を挿入
:h	マニュアルを表示する

Emacsでのキー操作をTable 8に示します．

Table 8　Emacsのキー操作

C-g	操作のキャンセル
C-x C-c	終了
C-x C-f	ファイルを開く
C-x C-s	ファイルを保存する
C-x C-w	ファイルを別名保存する
C-~~¥~~	日本語IME起動
C-a	行頭へ移動する
C-e	行末へ移動する
M-<	ファイルの先頭へ移動する
M->	ファイルの末尾へ移動する
C-d	カーソル位置の文字を削除
C-k	カーソル位置から行末までの文字列を削除
C-s	文字列の前方検索．C-sで次の候補を検索
C-r	文字列の後方検索．C-rで次の候補を検索
M-x replace-string	カーソル位置以降の文字列を全て置換
M-x replace-regexp	上と同様だが，正規表現が使える
M-x query-replace	置換ごとに確認を求める
M-x query-replace-regexp	上と同様だが，正規表現が使える
C-SPC	領域選択開始
M-w	コピー
C-w	カット
C-y	ペースト
C-x u	アンドゥ
C-x 2	バッファを縦に二分割
C-x o(オー)	別のバッファに移動
C-x 0(ゼロ)	現在のウインドウを消去
C-x 1	別のウインドウを消去
C-x C-b	バッファリストを表示
C-x k	現在のバッファを削除
C-x k バッファ名	指定のバッファを削除