UNIX 実習マニュアル

version 1.04
2002 年 7 月 25 日（木）

http://www.gin.or.jp/users/daikoku/

1　UNIX の基礎

1.1　オペレーティングシステム

これからみなさんは、UNIX というものに関する実習をすることになるわけですが、まず最初に、そもそも UNIX というのはいったい何なのか、ということについて説明しておくことにしましょう。

UNIX というのは、「オペレーティングシステム」と呼ばれるものの一種です（オペレーティングシステムは、OS という略称で呼ばれることもあります）。オペレーティングシステムというのは、コンピュータの基本的な動作をコントロールするソフトのことです。たとえば、パソコンでよく使われている Windows というソフトも、オペレーティングシステムの一種です。UNIX は、パソコンで使われることもありますが、どちらかと言うと、ネットワークを経由してさまざまなサービスを提供する「サーバー」と呼ばれるコンピュータでよく使われているオペレーティングシステムです。

UNIX という名前は、ひとつの特定のオペレーティングシステムを指すものではなくて、いくつかのよく似たオペレーティングシステムの総称です。UNIX に属するオペレーティングシステムとしては、AIX、HP–UX、Solaris、Linux、FreeBSD、MacOS X などがあります。ちなみに、この実習マニュアルは Linux を使って実習することを想定して書かれているのですが、ほかの UNIX を使う場合も、違っているところはそんなに多くありません。

オペレーティングシステムというソフトは、さまざまなプログラムから構成されています。オペレーティングシステムを構成しているプログラムのうちでもっとも基本的な核となるものは、「カーネル」と呼ばれます。そして、カーネル以外の大多数のプログラムは、人間にとって便利な道具（ツール）として使うことができるように作られています。これから始まる UNIX の実習の目的は、UNIX というオペレーティングシステムに付属しているさまざまな道具の使い方を習得することです。

1.2　ログイン

UNIX は、あらかじめ登録されている人間だけにしか使えないように作られています。そのような、UNIX にあらかじめ登録されている人間のことを「ユーザー」と言います。

UNIX に登録されているユーザーは、「ログイン名」または「ユーザー名」と呼ばれる、英字や数字などでできている名前によって識別されます。この実習を受講されているみなさんも、すでにユーザーとして UNIX に登録されています。したがって、みなさんも自分のログイン名を持っているわけですが、それがどんな名前なのかというのは、実習の担当者から説明があるはずです。

UNIX を使うためには、まず最初に、「ログイン」と呼ばれる操作をする必要があります。ログインというのは、これから UNIX を使うのがどのユーザーなのかということを UNIX に知らせることです。

UNIX は、起動した直後に、「ログイン画面」と呼ばれる画面を表示します。ログインの操作というのは、ログイン画面が出ているときに自分のログイン名を入力する、とい

うことです。入力されたログイン名が、あらかじめ登録されているものと一致した場合、UNIX は、そのユーザーに対して自分を使用する許可を与えます[1]。

それでは、実際にログインしてみましょう。キーボードを使って自分のログイン名を入力してください。もしも間違った文字を入力してしまった場合は、Backspace というキーを押すことによって、その文字を取り消すことができます。正しく入力できたら、Enter というキーを押してください。

テンキー（キーボードの右端にある電卓のようなキー）を使って数字の入力をしたい場合は、あらかじめ、テンキーの中にある Num Lock というキーを押しておく必要があります。

もしも、入力したログイン名が正しくなかった場合は、パスワードの入力を要求する画面が出ます。もしもそうなってしまった場合は、もう一度 Enter キーを押してください。すると、ふたたびログイン画面に戻ります。

ログインしてからしばらくすると、画面の上にさまざまなものが表示されます。ここで、画面の上に表示されているそれぞれのものの名称について説明しておくことにしましょう。

デスクトップ　画面の上に表示されているさまざまなものの背景になっている部分。
アイコン　小さな絵。
パネル　画面の一番下にある、アイコンなどが表示された細長い長方形の領域。
ウィンドウ　枠のある長方形の領域。

1.3 ログアウト

UNIX の使用を終了するときは、そのことを UNIX に知らせる必要があります。これで使用を終了するということを UNIX に知らせることを、「ログアウト」と言います。

それでは、ログアウトの操作について説明しましょう。まず最初に、パネルの左端にある足跡のアイコンをクリックしてください。するとメニューが表示されますので、その一番下にある「ログアウト」という項目をクリックしてください。そうすると、

本当にログアウトしますか？

と表示されたウィンドウが現われます。このときに「はい」と書かれたボタンをクリックすれば、ログアウトの操作は完了です。

ユーザーがログアウトをすると、UNIX は、起動した直後と同じ状態、つまり、誰かがログインするのを待っている状態に戻ります。

1.4 シャットダウン

コンピュータの電源を OFF にしたいときは、電源スイッチを押すのではなくて、「シャットダウン」という操作をする必要があります。シャットダウンというのは、稼動している状態にあるさまざまな機能を穏やかに終了させて、電源が OFF になっても大丈夫な状態にすることです。

シャットダウンをするための操作は、途中まではログアウトと同じです。まず最初に足跡のアイコンをクリックしてメニューを出して、次に「ログアウト」という項目をクリックします。

そのときに表示されるウィンドウの中には、三つの選択肢があります。ログアウトではなくシャットダウンをしたいときは、その選択肢の中にある「シャットダウン」をク

[1] パスワードが設定されている場合、UNIX は、パスワードをユーザーに入力させて、それが一致した場合にのみ、自分の使用を許可します。

リックしてから、「はい」をクリックします。すると、シャットダウンが実行されて、それが完了すると、電源が自動的に OFF になります。

1.5 シェル

オペレーティングシステムにはさまざまなプログラムが道具として付属しているわけですが、「シェル」と呼ばれるプログラムもその中のひとつです。「シェル」という言葉の本来の意味は、貝などが持っている「殻」のことですが、オペレーティングシステムのシェルというのは、カーネルを貝殻のように包み込んで、ユーザーとカーネルとのあいだの橋渡しをする役目のプログラムのことです。

文字が並んでできているデータのことを「テキスト」と呼びます。UNIX のプログラムの大多数は、自分と人間とのあいだで情報を受け渡しするためのデータとしてテキストを使うように作られているのですが、シェルも、その例外ではありません。

人間がプログラムに対して指示を与えるために入力するテキストは、「コマンド」と呼ばれます。シェルというプログラムが持っているもっとも重要な機能は、入力されたコマンドを解釈して、その結果をカーネルに伝えることです。

プログラムが人間に対して何らかのテキストの入力を促すために画面に表示するテキストのことを「プロンプト」と言います。シェルもやはり、コマンドを入力してほしいということを人間に伝えるために、プロンプトを出力します。

シェルは、

(1) プロンプトを出力する。
(2) ユーザーが入力したコマンドを読み込む。
(3) コマンドの意味を解釈して、その結果をカーネルに伝える。

という動作を延々と繰り返します（ただし、`exit` というコマンドが入力された場合は、自分の動作を終了させることになります）。

シェルを使いたいときは、パネルの上にあるモニターの形のアイコンをクリックしてください。すると、「GNOME 端末」という名前のプログラムが起動して、ひとつのウィンドウを表示します。この「GNOME 端末」というのは、シェルを動作させるための環境を作り出す、「ターミナルエミュレーター」と呼ばれるプログラムのひとつです。

1.6 コマンド

シェルに入力する「コマンド」は、基本的には、

単語　単語　単語　・・・　単語

というように、いくつかの単語を空白で区切った形になっています。コマンドの先頭の単語を「コマンド名」と呼び、2 個目以降のそれぞれの単語を「引数」または「パラメーター」と呼びます。

コマンドによっては、マイナス (-) という文字で始まる単語をコマンド名のうしろに書くことによって、コマンドがあらわしている動作のバリエーションを指定することができるようになっています。そのような、コマンド名のうしろに書くマイナスで始まる単語は、「引数」ではなく「オプション」と呼ばれます。

コマンドは、「内部コマンド」と「外部コマンド」に分類されます。内部コマンドというのは、シェルの内部の機能を実行するためのコマンドです。たとえば、シェルを終了させる `exit` は、内部コマンドの一例です。

外部コマンドというのは、何らかのプログラムを起動するコマンドです。外部コマンドのコマンド名は、プログラムの名前と同じものです。言い換えれば、プログラムを起動したいときは、そのプログラムの名前をコマンド名とするコマンドをシェルに入力す

ればいい、ということです。たとえば、UNIX には、カレンダーを出力する cal というプログラムがあるのですが、それを起動したいときは、

```
cal
```

というコマンドをシェルに入力すればいいわけです。それでは、実際にこのコマンドを入力してみてください。間違って入力してしまった文字を消したいときは、Backspace というキーを押してください。

引数を何も書かずに cal を起動した場合、 cal は、現在の月のカレンダーを出力します。過去や未来のカレンダーを出力させたい場合は、引数で月と年を指定する必要があります。たとえば、

```
cal 4 1970
```

というコマンドで cal を起動すると、 cal は、1970 年 4 月のカレンダーを出力します。

2 ファイル

2.1 ファイルについての基礎知識

コンピュータの内部には、大量のデータを貯えておくことのできる、「ハードディスク」と呼ばれる装置があります。

ハードディスクの中にデータを貯えるためには、「ファイル」と呼ばれる箱を作る必要があります。データは、ファイルという箱の中に格納されることになります。

個々のファイルは、「ファイル名」と呼ばれる名前によって識別されます。ファイル名は、英字、数字、ドット (.) などの文字を組み合わせることによって作ります。

ファイル名の末尾には、普通、そのファイルの中のデータの種類を示す、ドット (.) で始まる接尾語を付けます。そのような接尾語は、「拡張子」と呼ばれます。たとえば、テキストが格納されているファイルの名前には、

```
mikan.txt
```

というように、 .txt という拡張子を付けるのが一般的です[2]。

2.2 標準出力のリダイレクション

UNIX の上で動作するプログラムで、基本的なツールとして使われるものの多くは、「標準出力」と呼ばれるところへデータを出力するように作られています。また、ファイルの内容を読み込むプログラムの多くは、引数でファイル名を指定しなかった場合、「標準入力」と呼ばれるところからデータを読み込むように作られています。

通常は、標準入力はキーボードに割り当てられていて、標準出力はモニターに割り当てられているのですが、それらをファイルに切り換えるということも可能です。標準入力や標準出力をキーボードやモニター以外のものに切り換えることを、「リダイレクトする」と言います（名詞は「リダイレクション」）。

標準出力をファイルにリダイレクトしたい場合は、

コマンド > ファイル名

というように、コマンドの右側に「大なり」(>) という文字を書いて、その右側にファイルの名前を書きます。そうすると、大なりの左側のコマンドによって起動されたプログ

[2]ただし、特殊な言語によって書かれているテキストを格納しているファイルには、その言語を示す拡張子を付けます。

ラムによって標準出力に出力されたデータが、指定されたファイルに格納されます（指定されたファイルが存在しない場合は、そのファイルが新しく作られることになります）。

それでは、リダイレクションを実際に試してみましょう。まず、

```
cal 9 1752
```

というコマンドを入力してみてください。すると、1752 年 9 月のカレンダーが出力されます。

では次に、このカレンダーを sep1752.txt というファイルに保存してみましょう。つまり、 cal の出力を sep1752.txt にリダイレクトするということですから、

```
cal 9 1752 > sep1752.txt
```

というコマンドを入力すればいいわけです。

ちなみに、 cal が出力する 1752 年 9 月のカレンダーは、2 日の次の日が 14 日になっています。その理由は、イギリスとその植民地が歴法をユリウス歴からグレゴリオ歴へ変更したときに、ユリウス歴によって生じていたズレを修正するために 11 日間が存在しないものとされたからです。

2.3 ファイル名の一覧の出力

それでは、 sep1752.txt というファイルが本当にできているかどうかを確かめてみましょう。ファイルがあるかどうかを確かめたいときは、 ls というプログラムを使います。 ls は、ファイル名の一覧を出力するプログラムです。シェルに対して、

```
ls
```

というコマンドを入力すると、 ls が起動して、ファイル名の一覧が出力されます。

ls を起動するコマンドに、

```
ls -l
```

というように、 -l（マイナスと小文字のエル）というオプションを付けると、 ls は、ファイルの名前だけではなく、データの大きさ（単位は byte）や、最後に更新された日付と時刻などのさまざまな情報を出力します。このように、コマンドに付けるオプションというのは、プログラムに対する付加的な指示をあらわしています。

2.4 ファイルの内容の出力

sep1752.txt というファイルができているということが確認できたら、次は、そのファイルの内容がどうなっているかということを調べてみましょう。ファイルの内容を調べたいときは、 cat というプログラムを使います。

```
cat ファイル名
```

というコマンドで cat を起動すると、 cat は、引数で指定されたファイルの内容を出力します。ですから、

```
cat sep1752.txt
```

というコマンドを入力すれば、 sep1752.txt の内容が出力されるはずです。

cat を起動するコマンドに、

```
cat -n ファイル名
```

というように、 -n というオプションを付けると、 cat は、出力するそれぞれの行の先頭に行番号を付けます。ですから、

```
cat -n sep1752.txt
```

というコマンドを入力すれば、sep1752.txt の内容を構成するそれぞれの行の先頭に行番号を付けたものが出力されるはずです。

引数でファイルを指定しないで cat を起動したとすると、cat は、標準入力からデータを読み込みます。試しに、

```
cat
```

というコマンドを入力してみてください。標準入力は普通はキーボードに割り当てられていますから、この場合、cat は、キーボードから読み込んだデータを画面に出力することになります。

標準入力からの読み込みを終了させたいときは、Ctrl というキー（「コントロールキー」と呼びます）を押しながら D のキーを押してください。

cat を使って、キーボードから入力したものをファイルに格納する、ということもできます。それをしたいときは、

```
cat > ファイル名
```

というコマンドを入力します。たとえば、

```
cat > ichigo.txt
```

というコマンドを入力すると、キーボードから読み込んだデータが ichigo.txt という名前のファイルに格納されます。本当にそうなったかどうかは、

```
cat ichigo.txt
```

というコマンドを入力して確かめてください。

2.5 引数の出力

シェルの内部コマンドのひとつに、echo というコマンドがあります。これは、コマンドのすべての引数を標準出力に出力するという意味のコマンドです。たとえば、

```
echo higashi nishi minami kita
```

というコマンドを入力すると、その引数が、

```
higashi nishi minami kita
```

というように出力されます。

echo コマンドによって出力されたデータも、ファイルにリダイレクトすることができます。たとえば、

```
echo haru natsu aki fuyu > kisetsu.txt
```

というコマンドを入力すると、出力されたデータは、kisetsu.txt というファイルに格納されるはずです。本当にそうなったかどうか、cat を使って確かめてみてください。

2.6 パイプ

プログラムが標準出力に出力したデータは、ファイルにリダイレクトすることができるだけではなくて、別のプログラムの標準入力に送り込む、ということも可能です。

プログラムが標準出力に出力したデータを別のプログラムの標準入力に送り込みたいときは、「パイプ」と呼ばれるものを使って、プログラムとプログラムとを接続します。パイプを使いたいときは、

```
コマンド 1 | コマンド 2
```

というように、縦棒 (|) という文字で二つのコマンドを連結します。そうすると、コマンド 1 で起動されたプログラムの標準出力が、コマンド 2 で起動されたプログラムの標準入力に接続されます。

それでは、実際に二つのプログラムをパイプで接続してみましょう。 cal というプログラムは、カレンダーを標準出力に出力します。また、 cat というプログラムは、引数によってファイルが指定されなかった場合は、標準入力からデータを読み込みます。ですから、それらのプログラムをパイプで接続すれば、 cal の出力を cat の入力に接続することができるはずです。

それでは、

```
cal | cat -n
```

というコマンドを入力してみてください。すると、今月のカレンダーに行番号を付けたものが出力されるはずです。

2.7 ファイルのコピー

すでに存在するファイルと同じ内容を持つ新しいファイルを作ることを、ファイルを「コピーする」と言います。ファイルをコピーしたいときは、 cp というプログラムを使います。 cp を起動するコマンドは、

cp [ファイル名 1] [ファイル名 2]

と書きます。ファイル名 1 というのがコピー元になるファイルの名前で、ファイル名 2 というのがコピーによって新しく作られるファイルの名前です。

それでは、新しいファイルを作って、それをコピーすることによって同じ内容のファイルをもうひとつ作る、ということをやってみましょう。まず、コピーの元になるファイルを作るために、

```
echo maguro > maguro.txt
```

というコマンドを入力してください。そうすると、 maguro.txt というファイルができて、そこに maguro というテキストが格納されるはずです。それでは次に、そのファイルをコピーすることによって、 maguro2.txt という名前の新しいファイルを作りましょう。そのためのコマンドは、

```
cp maguro.txt maguro2.txt
```

ということになります。それでは、本当に新しいファイルができたかどうかを ls で確認して、さらに、その内容が maguro.txt と同じかどうかを、 cat で確認してください。

2.8 ファイルの削除

不要になったファイルを削除したいときは、 rm というプログラムを使います。

rm [ファイル名]

というコマンドで rm を起動すると、 rm は、ファイル名で指定されたファイルを削除します。

それでは、先ほどのコピーの実習のときに作った、 maguro.txt というファイルを削除してみましょう。

```
rm maguro.txt
```

というコマンドを入力すれば、 maguro.txt が削除されるはずです。本当に削除されたかどうか、 ls を使って確かめてみてください。

2.9 ファイル名の変更

ファイル名を変更したいときは、mv というプログラムを使います。

mv [ファイル名 1] [ファイル名 2]

というコマンドで mv を起動すると、mv は、ファイル名 1 で指定されたファイルの名前を、ファイル名 2 に変更します。

それでは、mv を使ってファイルの名前を変更してみましょう。まず、

```
echo namako > namako.txt
```

というコマンドで namako.txt という名前のファイルを作って、ちゃんとできたかどうか確認してください。ちゃんとできていたら、次は、そのファイルの名前を namako2.txt に変更してみます。

```
mv namako.txt namako2.txt
```

というコマンドを入力すれば、namako.txt の名前が namako2.txt に変更されるはずです。ファイル名がちゃんと変更されたかどうか、ls と cat を使って確認してみてください。

3 ディレクトリ

3.1 ディレクトリについての基礎知識

ハードディスクの中には、容量が許す限り何個でもファイルを作ることができるのですが、ファイルの数が多い場合、それらを整理しないと、取り扱いが不便になります。そこで、ファイルを整理するために、「ディレクトリ」と呼ばれる箱が使われます。ディレクトリの中には、ファイルを何個でも入れることができます。また、ディレクトリの中にディレクトリを入れるということも可能です。

個々のディレクトリは、「ディレクトリ名」と呼ばれる名前によって識別されます。ディレクトリ名も、ファイル名の場合と同じように、英字や数字などを使って作ります。なお、ディレクトリ名には、普通、拡張子は付けません。

ディレクトリの中にいくつかのディレクトリがあって、それらのディレクトリの中にもいくつかのディレクトリがある、というような構造は、いちばん外側のディレクトリが根で、内側にあるディレクトリで枝別れして、葉のところにファイルがある、という木のような構造だと考えることもできます。

しばしば、ファイルやディレクトリの関係を説明するときに、「上」とか「下」という言葉を使うことがあるのですが、これは、ディレクトリの構造を木だと考えたときの位置関係を示しています。ただし、ディレクトリの木というのは、普通の木とはさかさまに、根がいちばん上にあって、下に向かって伸びていると考えるのが普通です。ですから、「上」というのが「外側」という意味で、「下」というのが「内側」という意味になります。

3.2 ルートディレクトリ

ディレクトリの木には、かならず根に相当するディレクトリがあって、そのようなディレクトリのことを「ルートディレクトリ」と呼びます（「ルート」というのは「根」という意味です）。UNIX の場合、ルートディレクトリは、スラッシュ(/) という名前で識別されます。

ls を起動するとき、

```
ls ディレクトリ名
```

というように、ディレクトリ名を引数として書くと、lsは、引数で指定されたディレクトリの下にあるものの一覧を出力します。

それでは、lsを使って、みなさんのコンピュータのルートディレクトリの下にどんなものがあるか、調べてみましょう。

```
ls /
```

というコマンドを入力してみてください。そうすると、lsは、ルートディレクトリの下にあるものの一覧を出力します。

lsは、ファイルの名前だけではなく、ディレクトリの名前も出力します。lsが出力した名前がファイルの名前なのかディレクトリの名前なのかということを知りたい場合は、lsを起動するコマンドに、

```
ls -F ディレクトリ名
```

というように-Fというオプションを付けます。そうすると、lsは、出力するものがディレクトリの名前だという場合、その右側にスラッシュ(/)という文字を付けます。

それでは、

```
ls -F /
```

というコマンドを入力してみてください。そうすると、lsによってルートディレクトリの下にあるものの一覧が出力されるわけですが、その一覧の中で、名前の右側にスラッシュが付いているものは、ファイルではなくてディレクトリです。

3.3 ディレクトリの作成

新しいディレクトリを作りたいときは、mkdirというプログラムを使います。

```
mkdir ディレクトリ名
```

というコマンドでmkdirを起動すると、mkdirは、引数で指定された名前を持つディレクトリを作ります。

それでは、実際にディレクトリを作ってみましょう。

```
mkdir kabocha
```

というコマンドを入力してください。そうすると、kabochaという名前のディレクトリができるはずです。本当にできているかどうか、

```
ls -F
```

というコマンドを入力して確かめてください。

3.4 ディレクトリの名前の変更

ディレクトリの名前を変更したいときは、ファイルの名前を変更するときに使ったのと同じプログラム、つまりmvを使います。

```
mv ディレクトリ名1 ディレクトリ名2
```

というコマンドでmvを起動すると、mvは、ディレクトリ名1で指定されたディレクトリの名前を、ディレクトリ名2に変更します。たとえば、

```
mv kabocha jagaimo
```

というコマンドを入力すると、先ほど作ったディレクトリの名前が、jagaimo に変更されます。

3.5 ディレクトリの削除

不要になったディレクトリを削除したいときは、rmdir というプログラムを使います。

rmdir ディレクトリ名

というコマンドで rmdir を起動すると、rmdir は、引数で指定された名前を持つディレクトリを削除します。

それでは、先ほど名前を変更した jagaimo というディレクトリを削除してみましょう。jagaimo は、

```
rmdir jagaimo
```

というコマンドを入力することによって削除することができます。それでは、本当に削除されたかどうか、ls で確かめてください。

なお、rmdir は、内容が空のディレクトリしか削除することができません。中身のあるディレクトリを削除するためには、それに先立って、その中身をすべて削除する必要があります。

3.6 カレントディレクトリ

シェルはいつでも、特定のディレクトリを自分の現在の位置として認識しています。そのような、シェルによって認識されている現在の位置のことを「カレントディレクトリ」と呼びます。

ls のような、引数でディレクトリを指定することができるプログラムは、普通、引数が省略された場合、カレントディレクトリが指定されたと解釈します。

カレントディレクトリがどこなのかということを知りたいときは、pwd というプログラムを使います。

```
pwd
```

というコマンドで pwd を起動すると、pwd は、カレントディレクトリの名前を出力します。

カレントディレクトリを別のディレクトリに変更したいときは、cd というコマンドを使います（cd はシェルの内部コマンドです）。

cd ディレクトリ名

というコマンドをシェルに入力すると、シェルは、カレントディレクトリを、引数で指定されたディレクトリに変更します。

それでは、実際にカレントディレクトリを変更してみましょう。まず、

```
mkdir renkon
```

というコマンドで、renkon というディレクトリを作ってください。そして次に、

```
cd renkon
```

というコマンドを入力してください。そうすると、カレントディレクトリが renkon に変更されます。

カレントディレクトリをひとつ上のディレクトリに変更したい場合は、cd の引数として、「ひとつ上のディレクトリ」という意味を持つ、ドットドット (. .) というものを書きます。ですから、カレントディレクトリが renkon になっているときに、

```
cd ..
```

というコマンドを入力すると、カレントディレクトリは、renkonのひとつ上に移動します。

cdコマンドとlsを使えば、自分のコンピュータの中を自由に動き回って、どこにどんなものがあるかを調べることができます。それでは、みなさんのコンピュータの中を探検してみてください。

3.7 ホームディレクトリ

UNIXのユーザーには、一人にひとつ、「ホームディレクトリ」と呼ばれるものが与えられています。それぞれのユーザーは、自分のホームディレクトリの下に、ファイルやディレクトリを自由に作ることができます。また、シェルが起動した直後は、自分のホームディレクトリがカレントディレクトリになっています。

コマンドの中にチルダ (~) という文字を書くと、それはホームディレクトリを意味することになります。たとえば、

```
ls ~
```

というコマンドを入力すると、カレントディレクトリがどこであろうとも、ホームディレクトリの下にあるものの一覧が出力されます。

引数のないcdコマンド、つまり、

```
cd
```

というコマンドは、カレントディレクトリを自分のホームディレクトリに変更する、という意味になります。ですから、ホームディレクトリ以外のディレクトリがカレントディレクトリになっているときに、カレントディレクトリを自分のホームディレクトリに戻したいときは、引数のないcdコマンドを入力すればいいわけです。

3.8 相対パス名

特定のファイルやディレクトリを指定するための記述で、それがディレクトリの木の中のどういう位置にあるかという情報を含んでいるものことを、「パス名」と呼びます。

パス名というのは、

名前/名前/・・・/名前

というように、ディレクトリの名前をスラッシュ(/)で区切って並べていって、右端に、指定したいファイルまたはディレクトリの名前を書いたもののことです。

パス名には、「相対パス名」と「絶対パス名」という2種類の書き方があります。

相対パス名というのは、カレントディレクトリを出発点にして、木の枝に沿ってディレクトリの名前を並べていって、最後に、指定したいファイルまたはディレクトリの名前を書いたもののことです。たとえば、

```
bunbougu/hasami
```

というのは、カレントディレクトリの下にあるbunbouguというディレクトリの下にあるhasamiというもの（ディレクトリまたはファイル）を指定する相対パス名です。

カレントディレクトリのすぐ下にあるファイルやディレクトリを指定する相対パス名は、そのファイルやディレクトリの名前だけになります。たとえば、カレントディレクトリのすぐ下に、

```
ichigo.txt
```

というファイルや、

```
renkon
```

というディレクトリがあるとすると、それらを指定する相対パス名は、それら自体の名前を書くだけです。

カレントディレクトリの下にあるものではなくて、そこへ到達するためには上に向かって進む必要がある、というものの位置を相対パス名で記述したい、というときは、ひとつ上のディレクトリをあらわすドットドット (..) という記法を使います。たとえば、

```
../shoseki/jiten
```

というのは、カレントディレクトリのひとつ上にあるディレクトリの下にある shoseki というディレクトリの下にある jiten というもの（ディレクトリまたはファイル）を記述している相対パス名です。

相対パス名を書くとき、ディレクトリの木に沿って上へ向かってどんどん進んでいきたい、というときは、

```
../../../../..
```

というように、ドットドット (..) をスラッシュで区切って並べていきます。たとえば、

```
../../nomimono
```

と書くことによって、カレントディレクトリの二つ上にあるディレクトリの下にある nomimono というもの（ディレクトリまたはファイル）を記述することができます。

それでは、カレントディレクトリのひとつ上のディレクトリや二つ上のディレクトリの下にあるものの一覧を ls に出力させてみましょう。たとえば、

```
ls -F ../..
```

というコマンドを入力すると、カレントディレクトリの二つ上のディレクトリの下にあるものの一覧が出力されます。

3.9 絶対パス名

相対パス名というのがカレントディレクトリを出発点とするパス名であるのに対して、絶対パス名というのは、ルートディレクトリを出発点とするパス名のことです。ルートディレクトリの名前はスラッシュですから、絶対パス名はかならずスラッシュで始まることになります。なお、ルートディレクトリの名前（スラッシュ）と、その下にあるものの名前とのあいだには、区切り文字のスラッシュを書きません。

たとえば、

```
/tabemono/yasai/ninjin
```

という絶対パス名は、ルートディレクトリの下にある tabemono というディレクトリの下にある yasai というディレクトリの下にある ninjin というもの（ディレクトリまたはファイル）を記述していることになります。

プログラムに対してファイルやディレクトリを指定するとき、コマンドの引数としては、どんなパス名を書いてもかまいません。これまで、コマンドの引数は主として相対パス名で書いてきましたが、絶対パス名を使って指定することも可能です。たとえば、

```
cd /usr
```

というコマンドを入力すると、カレントディレクトリは、ルートディレクトリの下にある usr というディレクトリに移動します。同じように、

```
ls -F /etc
```

と入力すれば、ルートディレクトリの下にある etc というディレクトリの下にあるものの一覧が出力されます。

3.10 ディレクトリ間のファイルのコピー

cpを使うことによって、もとのファイルがあるディレクトリとは別のディレクトリにファイルのコピーを作る、ということも可能です。それをしたいときは、

cp [ファイル名] [ディレクトリ名]

というように、2個目の引数としてディレクトリの名前を書きます。そうすると、cpは、1個目の引数で指定されたファイルのコピーを、2個目の引数で指定されたディレクトリの下に作ります。この場合、新しく作られるファイルの名前は、もとのファイルの名前と同じになります。

ディレクトリ間のファイルのコピーを実際に試してみましょう。まず、自分のホームディレクトリがカレントディレクトリになっている状態で、

```
mkdir seafood
```

というコマンドを入力して、seafoodという名前のディレクトリを作ってください。次に、

```
echo hamachi > hamachi.txt
```

というコマンドで、hamachi.txtという名前のファイルを作ってください。

それでは、hamachi.txtをseafoodにコピーしてみましょう。

```
cp hamachi.txt seafood
```

というコマンドでcpを起動すると、cpは、seafoodの下にhamachi.txtのコピーを作ります。本当にコピーができたかどうか、lsとcatを使って確かめてください。

```
ls seafood
```

というコマンドを入力すると、seafoodの下にあるファイルの一覧が出力され、

```
cat seafood/hamachi.txt
```

というコマンドを入力すると、seafoodの下にあるhamachi.txtの内容が出力されます。

ところで、カレントディレクトリではないディレクトリの下にあるファイルをカレントディレクトリの下にコピーしたいというときは、いったいどうすればいいのでしょうか。

そのためには、カレントディレクトリそのものを相対パス名で記述する必要があります。カレントディレクトリを記述する相対パス名は、ドット (.) を 1 個だけ書いたものです。

それでは、実際に試してみましょう。まず、

```
echo tarako > seafood/tarako.txt
```

というコマンドで、seafoodの下にtarako.txtというファイルを作ってください。それができたら、次に、

```
cp seafood/tarako.txt .
```

というコマンドで、ikura.txtをカレントディレクトリにコピーしてください。そして、コピーできたかどうか、lsとcatで確かめてください。

3.11 ファイルとディレクトリの移動

ファイルやディレクトリは、ディレクトリからディレクトリへ移動させることが可能です。ファイルやディレクトリを移動させたいときは、mvというプログラムを使います。mvは、ファイルやディレクトリの名前を変更するときに使うプログラムですが、実は、mvの本来の機能は、ファイルやディレクトリを移動させることなのです。

mvを使ってファイルやディレクトリを移動させたいときは、

mv パス名 1 パス名 2

というコマンドで mv を起動します。そうすると、mv は、パス名 1 で指定されたファイルまたはディレクトリを、パス名 2 で指定されたディレクトリへ移動させます。

それでは、mv を使って実際にファイルを移動させてみましょう。まず、

```
echo ikura > ikura.txt
```

というコマンドで、ホームディレクトリの下に ikura.txt というファイルを作ってください。次に、

```
mv ikura.txt seafood
```

というコマンドで、ikura.txt を seafood の下に移動させてください。そして、ちゃんと移動したかどうか、ホームディレクトリの下と seafood の下の両方を、ls を使って調べてください。

さて、今度は、mv を使ってディレクトリを移動させてみましょう。まず、

```
mkdir food
```

というコマンドで、food というディレクトリを作ってください。次に、

```
mv seafood food
```

というコマンドで、seafood を food の下に移動させてください。そして、ちゃんと移動したかどうか、ホームディレクトリの下と food の下の両方を、ls を使って調べてください。

4 vi の操作

4.1 エディター

ここから先の実習では、「エディター」と呼ばれるプログラムの使い方について学習していきます。エディターというのは、テキストを編集する（つまり入力したり修正したりする）ためのプログラムのことで、「テキストエディター」と呼ばれることもあります。

一口にエディターと言ってもさまざまなものがあるのですが、この実習で使うのは vi と呼ばれるエディターです。ほかのエディターに比べると、vi というのは、慣れないうちはものすごく操作しにくいと感じるかもしれません。しかし、vi 以外の操作しやすいエディターというのは、状況によっては使えないこともありますので、vi に慣れておくことはけっして無駄にはなりません。特に、UNIX の管理という仕事をする場合には、どのような状況のもとでもテキストを編集することができないといけませんので、vi の操作ができることは必要不可欠とさえ言っていいほどです。

4.2 vi の起動と終了

vi を起動したいときは、

vi ファイル名

というコマンドをシェルに入力します。そうすると、vi が起動して、指定されたファイルからテキストを読み込んで、それを画面に表示します。存在しないファイルのパス名が指定された場合、vi は、そのパス名を持つファイルを新しく作って、入力されたテキストをそのファイルに保存します。

それでは、実際に vi を起動してみましょう。

```
vi ningen.txt
```

というコマンドで、vi を起動してください。そうすると、画面の大部分が空白になって、画面の左上にカーソルが現われます。このとき、大多数の行の左端にチルダ (~) が表示されていますが、このチルダは、「この行はテキストの一部分ではありません」ということを意味しています。

それでは次に、vi を終了させてみましょう。vi を終了させたいときは、まず、コロン (:) を入力します。すると、画面の左下にコロンとカーソルが表示されます。この状態のときに wq と入力して Enter を押すと、vi が終了して、画面にシェルのプロンプトが表示されます。

4.3 vi のコマンド

人間が、カーソルを移動させたりデータを保存したり、というような操作を vi に対して指示するためには、その指示を意味するテキストを入力する必要があります。そのような、vi に対する指示を意味しているテキストのことを、「コマンド」と呼びます。

vi が理解することのできるコマンドは、大きく二つのグループに分類することができます。それぞれのグループは、「vi コマンド」と「ex コマンド」と呼ばれます。

vi コマンドというのは vi に固有のコマンドのことです。それに対して、ex コマンドというのは、ex というエディターで使うことのできるコマンドのことです。vi で ex のコマンドを使うことができるというのは奇妙なことだと思われるかも知れませんが、実は、vi と ex というのは別々のエディターではなくて、ひとつのエディターの別々の側面なのです。

vi には、入力されたコマンドが vi コマンドなのか ex コマンドなのかということを区別することができるように、「ex コマンドを入力する場合は先頭にコロン (:) を付けないといけない」という約束があります。また、大多数の vi コマンドはコマンドだけを入力すれば実行されるのですが、ex コマンドは、最後に Enter キーを押さないと実行されません。

さて、もうすでにお気付きだと思いますが、vi を終了させる :wq というコマンドは、ex コマンドの一例です。

4.4 テキストの挿入

vi というエディターには、「コマンドモード」と「入力モード」と呼ばれる 2 種類の状態があります。コマンドモードというのは、キーボードから入力したテキストがコマンドだと解釈される状態のことです。それに対して、入力モードというのは、キーボードから入力したテキストが編集中のテキストの中に挿入される状態のことです。ちなみに、vi が起動した直後の状態は、コマンドモードになっています。

コマンドモードから入力モードへ移行したいというときは、何種類かあるそのためのコマンドのうちのひとつを入力します。そして、それとは逆に、入力モードからコマンドモードへ移行したいときは、 Esc というキー（「エスケープキー」と呼びます）を押します。ちなみに、コマンドモードで Esc を押した場合、変化は何も起きません。

vi をコマンドモードから入力モードへ移行させるためのコマンドのうちでもっとも基本的なのは、 i というコマンドです。

それでは、もう一度、

```
vi ningen.txt
```

というコマンドで vi を起動して、それから i を入力してみてください。そうすると、vi の状態がコマンドモードから入力モードに移行します。それでは、

```
Uemura Shouen
Ogura Yuki
Higashiyama Kaii
Hirayama Ikuo
```

というような感じで、自分の知っている人の名前を 20 人分ほどローマ字で入力してみてください（改行は Enter キーで入力することができます）。

入力モードではコマンドの入力ができないわけですから、カーソルを移動させることはできません。しかし、 Backspace を押すことによって入力した文字を取り消すことは可能です。

名前の入力が終ったら、 Esc を押してください。そうすると、vi のモードが入力モードからコマンドモードに移行しますので、次に、 :wq を入力して vi を終了させてください。そして、 cat を使って、 ningen.txt の内容がどうなったかを調べてみてください。

4.5 カーソルの移動

vi の画面には、かならずどこかに、「カーソル」と呼ばれる黒い長方形が表示されています。vi のカーソルは、キーボードから入力された文字がテキストのどこに挿入されるかということを示しています。

カーソルは、コマンドを入力することによって自由に移動させることができます。たとえば、カーソルを下へ移動させたいときは j というコマンドを入力します。同じように、 k で上、 l で右、 h で左へカーソルが移動します。このように、カーソルを上下左右に移動させるコマンドは、右手をほとんど動かす必要のない位置にあるキーに割り当てられています。

w と b というコマンドを使うと、カーソルを単語単位で移動させることができます。 w は前方への移動、 b は後方への移動です。 l と h が改行を越えてカーソルを移動させることができないのに対して、 w と b は改行を越えてカーソルを移動させることができます。

行の先頭や末尾へカーソルを移動させるコマンドもあります。 0 は行の先頭へカーソルを移動させ、 $ は行の末尾へカーソルを移動させます。

次に、 G というコマンドを使ってみましょう。まず最初に、ただ単に G を入力してみてください。そうすると、カーソルが、編集の対象となっているテキストの最後の行へ移動するはずです。

G を使うことによって、移動先の行の番号を指定してカーソルを移動させる、ということもできます。あらかじめ行の番号を入力してから G を入力すると、カーソルは、入力された番号の行へ移動します。たとえば、 4G と入力すると、カーソルが 4 行目へ移動し、 31G と入力すると、31 行目へ移動します。

G というコマンドを使うときは、行の番号が画面に表示されるようにしておくと便利です。行番号は、

```
:set nu
```

という ex コマンドを入力することによって画面に表示させることができます。なお、行番号が表示されているときに、その表示を消したいときは、

```
:set nonu
```

という ex コマンドを入力します。

また、 Ctrl-G というコマンドを入力することによって、カーソルがある行の番号などを画面のいちばん下に表示させる、ということもできます（ Ctrl-G というのは、 Ctrl キーを押しながら G のキーを押すという意味です）。

4.6 検索によるカーソルの移動

それでは次に、文字またはテキストを検索することによってカーソルを移動させるコマンドを使ってみましょう。そのようなコマンドとしては、f と / という二つのものがあります。

f は、カーソルの右側にある文字を検索するコマンドです。f を入力してから検索したい文字を入力すると、vi は、その文字を検索して、最初に見付かった文字の位置へカーソルを移動させます。文字が見付からなかった場合、カーソルの位置は変化しません。このコマンドで検索の対象となる範囲は、カーソルの右隣の文字から、同じ行の末尾までです。

f が文字を検索するコマンドであるのに対して、/ はテキストを検索するコマンドです。検索の範囲は、編集の対象となっているテキストの全体です。

/ を入力すると、画面の左下に / とカーソルが表示されます。この状態のときに、検索したいテキストを入力して、それから Enter を押すと、vi は、入力されたテキストをカーソルの位置から下へ向かって検索して、最初に見付かった位置へカーソルを移動させます。テキストの末尾まで検索しても一致するものが見付からなかった場合は、テキストの先頭へ行って、そこからさらに下へ向かって検索を続行します。

/ を使ってカーソルを移動させたあと、n というコマンドを入力すると、ふたたび同じテキストを同じ方向で検索します。N というコマンドを入力すると、同じテキストを逆の方向で検索します。また、/ の代わりに ? を使うことによって、最初からテキストの上へ向かって検索させることもできます。

4.7 行の末尾へのテキストの追加

カーソルが行の末尾の文字の上にあるときに i で入力モードに移行すると、vi は、行の末尾の文字の左側に、新しく入力されたテキストを挿入します。vi のカーソルは、行の末尾の文字よりも右へ移動させることはできませんから、i というコマンドでは行の末尾にテキストを追加することができない、ということになります。

行の末尾にテキストを追加するためには、a というコマンドを使う必要があります。a は、i と同じように vi をコマンドモードから入力モードへ移行させるコマンドですが、a の場合は、カーソルを 1 文字分だけ右へ移動させてから入力モードに移行します。ですから、行の末尾の文字の上にカーソルを置いた状態で a を入力することによって、行の末尾にテキストを追加することができます。

4.8 テキストの保存

ファイルの中に格納されているテキストを変更するためには、エディターを使ってテキストを編集するという作業をしたのち、そのテキストをファイルに保存するという操作をする必要があります。実は、以前に説明した :wq という ex コマンドは、ただ単に vi を終了させるだけのコマンドではなくて、テキストをファイルに保存してから vi を終了させるというコマンドなのです。

テキストをファイルに保存しないで vi を終了させたいときは、:q という ex コマンドを入力します。ただし、このコマンドを受け付けてもらえるのは、テキストに対して何も変更を加えていない場合だけです。テキストに対して何らかの変更を加えたにもかかわらず、テキストをファイルに保存しないで vi を終了させようとすると、vi は、

```
No write since last change
```

というメッセージを画面の一番下の行に表示します。これは、テキストに対する編集作業が失われることに対する警告です。もしも、警告を無視して強制的に vi を終了させた

いならば、:q のうしろに感嘆符を追加した、:q! というコマンドを入力する必要があります。

テキストを編集している最中に何らかのトラブルが発生して、vi の操作ができなくなった場合、最後に保存を実行した時点よりもあとの編集作業が失われることになります。ですから、テキストの保存は、エディターを終了させるときだけではなく、できるだけ頻繁に実行しないといけません。エディターを終了させないでテキストを保存したいときは、:w という ex コマンドを入力します。

4.9 コマンドの繰り返し

vi のコマンドの一部は、回数を指定することによって、その回数だけ実行を繰り返すことができるようになっています。すでに登場したコマンドの中では、i、a、j、k、l、h、w、b が、繰り返し可能なコマンドです。

コマンドの実行を繰り返したいときは、まず繰り返したい回数を入力して、それからコマンドを入力します。たとえば、6j というコマンドを入力することによって、カーソルを 6 行分だけ下へ移動させることができます。

i や a を繰り返すとどうなるか、ということも試してみましょう。繰り返しの回数を入力してから i または a を入力して、テキストを入力したのちに Esc を押してみてください。そうすると、最初に入力した回数だけテキストを増殖させたものが挿入されるはずです。

4.10 テキストの変更

テキストの一部分を別のテキストに変更したいときは、c というコマンドを使います。このコマンドは、変更する範囲を指定するために、カーソル移動コマンドと組み合わせて使う必要があります。たとえば、単語単位でカーソルを移動させる w というコマンドと組み合わせた、cw というコマンドを使うことによって、カーソル位置から、カーソルのある単語の末尾までのテキストを変更することができます。

c も、i や a と同じように、vi をコマンドモードから入力モードへ移行させます。c とカーソル移動コマンドを入力すると、指定された範囲のテキストが消去されます。この状態ですでに入力モードになっていますので、そこから先は i や a の場合と同じです。テキストを入力すると、そのテキストが挿入されます。それから Esc を押すと、コマンドモードに戻ります。

cw と同様に、c$ は、カーソル位置から行の末尾までのテキストを変更し、c0 は、行の先頭からカーソル位置までのテキストを変更します。

cc というコマンドを使うと、カーソルのある行の全体を別のテキストに変更することができます。

1 個の文字だけを変更したい場合は、r というコマンドを使います。r を入力してから 1 個の文字を入力すると、その文字とカーソル位置の文字とが置き換わります。

4.11 テキストの削除

テキストの一部分を削除したいときは、d というコマンドを使います。このコマンドも、c と同じように、削除する範囲を指定するために、カーソル移動コマンドと組み合わせて使います。たとえば、dw というコマンドは、カーソル位置から、カーソルのある単語の末尾までのテキストを削除します。同じように、d$ は、カーソル位置から行の末尾までのテキストを削除します。

テキストを行単位で削除したいときは、dd というコマンドを使います。dd を入力すると、カーソルのある行だけが削除されます。削除したい行数を入力してから dd を入力すると、カーソルのある行から下に向かって、指定された個数の行が削除されます。

テキストを文字単位で削除したいときは、x というコマンドを使います。x を入力すると、カーソルのある文字だけが削除されます。削除したい文字数を入力してから x を入力すると、カーソルのある行から右に向かって、指定された個数の文字が削除されます。

行と行とのあいだにある改行を削除したいという場合、つまり、二つの行を連結してひとつにしたいという場合は、J というコマンドを使います。J を入力すると、カーソルのある行とその次の行とが連結されて、ひとつの行になります。

ex コマンドを使うことによって、行番号で範囲を指定して、その範囲のテキストを削除する、ということもできます。そのための ex コマンドは、

: 行番号 1 , 行番号 2 d

と入力します。そうすると、行番号 1 で指定された行から行番号 2 で指定された行までが削除されます。たとえば、

```
:38,57d
```

という ex コマンドを入力すると、38 行目から 57 行目までが削除されることになります。

4.12 テキストの移動とコピー

d や dd などでテキストを削除した場合、削除されたテキストは、「バッファー」と呼ばれる場所に一時的に保管されます。バッファーに保管されているテキストは、p または P というコマンドを使うことによって、カーソルの位置に挿入することができます。ですから、テキストを削除するコマンドと p または P とを組み合わせることによって、テキストを別の場所に移動させる、ということができます。

p と P との違いは、p の場合はカーソル位置の文字の右側にテキストを挿入するのに対して、P の場合はカーソル位置の文字の左側にテキストを挿入する、という点です。なお、バッファーの中に、行単位で削除されたテキストがある場合、そのテキストは、カーソルのある行の下または上に挿入されます。p が下で、P が上です。

d または dd の代わりに y または yy というコマンドを使うことによって、テキストを削除しないでバッファーにコピーする、ということができます。ですから、y または yy と p または P を組み合わせることによって、テキストのコピーを作って、それを別の場所に挿入する、ということができます。

y は、c や d と同様に、カーソル移動コマンドと組み合わせて使うコマンドです。たとえば、yw というコマンドを入力すると、カーソル位置から単語の末尾までがバッファーにコピーされます。

yy は、テキストを行単位でバッファーにコピーするコマンドです。行数を入力してから yy を入力することによって、複数行を一度にコピーすることもできます。

ex コマンドを使うことによって、行番号で範囲を指定して、その範囲のテキストを別の場所へ移動させたり、その範囲のコピーを別の場所に挿入する、ということもできます。

テキストを移動させる ex コマンドは、

: 行番号 1 , 行番号 2 m 行番号 3

と入力します。そうすると、行番号 1 で指定された行から行番号 2 で指定された行までの範囲を、行番号 3 で指定された行の下へ移動させます。たとえば、

```
:42,51m14
```

という ex コマンドを入力すると、42 行目から 51 行目までが 14 行目の下に移動することになります。

テキストのコピーを別の場所に挿入する ex コマンドは、

: 行番号 1 , 行番号 2 co 行番号 3

と入力します。そうすると、行番号 1 で指定された行から行番号 2 で指定された行までのコピーを作って、行番号 3 で指定された行の下へそれを挿入します。たとえば、

```
:31,46co215
```

という ex コマンドを入力すると、31 行目から 46 行目までのコピーが 215 行目の下に挿入されることになります。

A UNIX リファレンスマニュアル

`cal` 月 年

指定された年と月のカレンダーを出力します。月が省略された場合は、1 年間のすべてのカレンダーを出力します。月も年も省略された場合は、現在の月のカレンダーを出力します。

`cat` ファイル名

指定されたファイルの内容を読み込んで、それを標準出力に出力します。引数が省略された場合は、標準入力からデータを読み込みます。

`cd` ディレクトリ名

指定されたディレクトリをカレントディレクトリにします。引数が省略された場合は、ホームディレクトリをカレントディレクトリにします。

`cp` パス名 1 パス名 2

パス名 1 で指定されたファイルのコピーを、パス名 2 で指定されたディレクトリに作ります。パス名 2 がファイル名の場合は、それを新しいファイルの名前にします。

`echo` 引数 ・・・

すべての引数を標準出力に出力します。

`exit`

シェルを終了させます。

`ls` ディレクトリ名

指定されたディレクトリの下にあるものの一覧を標準出力に出力します。引数が省略された場合は、カレントディレクトリの一覧を出力します。`-l` というオプションを付けると、最終更新日時やデータの大きさなどの詳細が出力されます。`-F` というオプションを付けると、ディレクトリ名の右側にスラッシュ(`/`)が出力されます。

`mkdir` ディレクトリ名

指定された名前を持つディレクトリを作ります。

`mv` パス名 1 パス名 2

パス名 1 で指定されたファイルまたはディレクトリを、パス名 2 で指定されたディレクトリへ移動させます。パス名 2 がファイル名の場合は、ファイル名をそれに変更します。

`pwd`

カレントディレクトリの絶対パス名を標準出力に出力します。

`rm` パス名
　指定されたファイルを削除します。

`rmdir` ディレクトリ名
　指定されたディレクトリを削除します。ただし、空ではないディレクトリは削除できません。

`vi` ファイル名
　vi を起動して、指定されたファイルの中のテキストを編集の対象にします。

B　vi リファレンスマニュアル

B.1　テキストの保存と vi の終了

`:w`　テキストを保存します（なるべく頻繁に使ってください）。

`:q`　vi を終了します。

`:wq`　テキストを保存して終了します。

`:q!`　修正を破棄して終了します。

B.2　行番号

`:set nu`　行番号を表示します。

`:set nonu`　行番号の表示を消します。

`Ctrl-G`　カーソルのある行の番号などを表示します。

B.3　カーソルの移動

`j`　1 行下へ。

`k`　1 行上へ。

`l`　1 文字右へ。

`h`　1 文字左へ。

`w`　次の単語の先頭へ。

`b`　前の単語の先頭へ。

`0`　行の先頭へ。

`$`　行の末尾へ。

`G`　指定された番号の行へ（番号を省略すると、テキストの最後の行へ移動します）。

B.4　検索によるカーソルの移動

`f`　指定された文字を検索します（カーソルのある行の中だけで有効です）。

`/`　指定されたテキストを下へ向かって検索します。

`?`　指定されたテキストを上へ向かって検索します。

n　同じテキストを同じ方向で検索します。

N　同じテキストを逆の方向で検索します。

B.5　テキストの挿入

i　カーソル位置の前にテキストを挿入します。

a　カーソル位置のうしろにテキストを挿入します。

B.6　テキストの変更

c　テキストを変更します（カーソル移動コマンドと組み合わせる必要があります)。

cc　カーソルのある行を変更します。

r　カーソル位置の文字を変更します。

B.7　テキストの削除

d　テキストを削除します（カーソル移動コマンドと組み合わせる必要があります)。

dd　カーソルのある行を削除します。

x　カーソル位置の文字を削除します。

J　カーソルのある行と、その次の行とを連結します。

:行番号1,行番号2d
　行番号 1 の行から行番号 2 の行までを削除します。

B.8　テキストの移動とコピー

p　バッファーの中のテキストを、カーソルのうしろ（下）に挿入します。

P　バッファーの中のテキストを、カーソルの前（上）に挿入します。

y　テキストをバッファーにコピーします（カーソル移動コマンドと組み合わせる必要があります)。

yy　カーソルのある行をバッファーにコピーします。

:行番号1,行番号2m行番号3
　行番号 1 の行から行番号 2 の行までを、行番号 3 の行の下へ移動させます。

:行番号1,行番号2co行番号3
　行番号 1 の行から行番号 2 の行までのコピーを作って、行番号 3 の行の下へそれを挿入します。