2000年度 第2回UNIXゼミ

指導：川崎　　担当：佐野 吉田 奥田

2000年4月24日

目 次

第1章　シェルの活用

1.1　シェルとは

シェルとはユーザが入力したコマンドを解釈して，それをコンピュータに実行させる役割を持ったプログラムの総称です．UNIX システムでは，数種類のシェルが利用できますが，Linux ではbash(Bourne Again シェル) と呼ばれるシェルが標準的なシェルです．本ゼミではこのbash を用いて話を進めます．

1.2　コマンドラインの編集

通常，正しく設定されている bash の環境では，コマンドラインを編集するために，[←]や，[→]でのカーソル移動，[Backspace]による文字削除，[↑]，[↓]による履歴の参照などができます．このため，Windows での文字編集に慣れている方は特に何も覚えなくても簡単な編集は出来ると思います．
ここでは，それ以外に覚えておくと便利な bash のキー操作について説明します．1.2.1 節から 1.2.5 まではecho hello という入力に変更するための処理を示します．なお，Emacs や bash のマニュアルでは通常，キー操作の表示に特殊な記法を使います．また，ここでは，Windows にあわせたキーを表 1.2 書いておきます．

1.2.1　行頭・行末への移動

```
cho hello_ e が足りない
  ↓  C-a   行頭に戻る
cho hello
  ↓  e   e を入力
echo hello
```

逆に行末に移動するにはC-e を使用します．

表 1.1: キー操作の表示

コマンド	操作
M-p	[ESC]を押した後に，[p]を押す．あるいは，[Alt]を押しながら[p]を押す．
C-b	[Ctrl]を押しながら，[b]を押す．
M-C-e	[ESC]を押した後に，[Ctrl]を押しながら，[e]を押す．あるいは，[Alt]と[Ctrl]を押しながら[e]を押す．

1.2.2 単語単位の移動

```
echo ello_ h が足りない
  ↓  M-b  一つ前の単語の先頭に戻る
echo _ello
  ↓  h  h を入力
echo hello
```

一つ先の単語の先頭に移動するには`M-f` を使用します.

1.2.3 単語の削除

```
echo hello_ hello をbye に直す
  ↓  M-C-h  前の単語を削除
echo _
  ↓  bye  bye を入力
echo bye_
```

後の単語を削除するには`M-C-d` を使用します.

1.2.4 行単位の削除

```
echo hello, hello  カーソル以下をすべて削除したい
  ↓  C-k  カーソルから行末まで削除
echo _
```

行すべてを削除するには`C-u` を使用します.

1.2.5 削除した文字列の挿入

`M-d`,`M-C-h`,`C-k`,`C-u` で削除された文字列は，bash が一時的に記憶しています．`C-y` で記憶されている文字列をカーソル位置に挿入することが出来ます.

```
echo hello, hello
  ↓  C-k  カーソルから行末まで削除 (記憶される)
echo _
  ↓  C-y  記憶されていた文字列が挿入される
echo hello, hello_
```

1.3 履歴機能

bash はユーザーが入力したコマンドを記憶しています．履歴を表示するには`history` コマンドを実行します.

```
$ history Enter
1 ls
```

```
2 cd
3 mkdir temp
```

つまり最も最近に実行されたコマンドは`ls` で，次に`cd`, その次に`mkdir temp` が実行されたということが現れています.

1.3.1 直前のコマンドを実行

単に直前のコマンドを実行するだけならば`!!`でかまいません．次の例では，直前のコマンドの文字列の一部を置き換えて実行してみます．最初に次のように入力してみます.

```
$ (ここで!!と入力します．)
$ echo hello Enter
hello
```

この状態で，`hello` の部分 `bye` に変更したコマンドラインを入力してみる場合，に次のように入力する.

```
$ ^hello^bye^ Enter
echo bye
bye
```

`^hello^bye^`によって`echo bye` というコマンドが実行されたことが`history` コマンドを実行することによってわかります.

1.3.2 履歴番号での指定

`history` コマンドで表示した履歴番号を使用して対応したコマンドを実行できます.

```
$ history Enter
1 ls
2 cd
3 mkdir temp
$ !3 Enter
mkdir temp
```

1.3.3 履歴の検索

`!`の後に文字列を指定すると history リストの中からその文字列で始まるコマンドで最新のものを見つけて実行します．`$ !echo` なら最後に実行した echo コマンドを実行します.

1.4 エイリアス機能

エイリアスとは別名と言う意味ですが，bash にはコマンドに別の名前を付ける機能があります.
例えば`alias h="history"`とすれば以降は`h` だけで`history` を実行することが出来ます.
一度設定したエイリアスを解除するには`unalias` を使用します．先の例でいうと`unalias h` となります.

良く使われるものとしてはalias ls="ls -F"のように同じコマンド名にオプションを付けたものを設定することがあります．これをすると以降オプションを付けなくても標準でオプションが有効になります．この場合オプションの無いlsを実行したい場合はコマンド名に\を付けて実行します．
現在設定されているエイリアスをすべてみるには引数なしでaliasを実行します．

1.5 補完機能

bashの補完機能を使うと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出来ます．
例えばカレントディレクトリに"takeshi"というファイルがあるとします．cat[1] tとコマンドラインに入力するなら

```
$ cat t Tab
```

と入力すれば，先頭が"t"で始まるファイルがカレントディレクトリから検索されて，ファイル名の残り文字が補完されます．

```
$ cat takeshi
```

候補が複数ある場合は2回押すと候補一覧が表示されます．

1.6 ジョブとプロセス

UNIXはマルチタスクOSであることは前にも述べました．これは複数のジョブを同時に処理できることを意味しています．しかしジョブとはbashなどのシェルが管理しているコマンドの実行単位なのです．そしてUNIXシステムではプロセスという単位でプログラム一つ一つを管理しています．これはコマンドにも現れ，

```
ps [オプション]
```

で実行中のプロセス一覧を見ることができ，

```
jobs
```

で現在実行中のジョブ一覧を見ることができます．実はユーザがシステムにログインしたときに起動したシェルも，UNIXシステムが管理する一つのプロセスなのです．
UNIXでは[2]複数のジョブを同時に処理することができます．そのため，あるコマンドの実行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．そこでででてきたのがバックグランドジョブというものです．これはユーザが面している端末の裏側で実行されるジョブということです．あるコマンドの処理をバックグランドジョブとして扱いたい場合には，コマンドラインの末尾に”&”記号をつけます．例えばルートディレクトリ以下のtから名前が始まるファイルを調べ，その結果をtlist.txtというファイルに書き出す場合，

```
$ find / -name 't*' -print > tlist.txt &
```
[3]

とすることによってバックグランドジョブとして扱われ，ジョブ番号が表示された後，すぐにシェルプロンプト[4]が表示され，次のコマンドが実行できるようになります．もし”&”という記号をつけなかった場合は，そのジョブが終了するまで目の前の端末を占領します．このようなジョブの事をフォアグランドジョブと呼びます．
リモートログインでUNIXを用いる場合，ログアウト後にもUNIXにある処理をさせておきたいという場合があります．その場合，コマンドラインの頭にnohupをつけます．先ほどのコマンドを，このモードで実行する場合には

[1] cat コマンドは覚えていますよね？
[2] ユーザから見た場合はシェルの上において
[3] >という記号はリダイレクトといわれ，コマンドの実行結果を画面ではなく，記号の後に続くファイルに保存されるという働きをします．
[4] シェルプロンプトについてはあとで説明します

```
$ nohup find / -name 't*' -print > tlist.txt &
```

とします．この実行によって通信を切っても計算サーバ上では実行しつづけます．
あとここでは実行中のジョブに対する操作だけをまとめておきます．何かの参考にしてください．

- フォアグランドジョブを中断/一時停止する
  フォアグランドジョブを中断するには Ctrl + C
  フォアグランドジョブを一時停止するには Ctrl + Z
- バックグランドジョブを中断する
  kill %ジョブ番号
- フォアグランドジョブをバックグランドに移動する
  bg %ジョブ番号
- 一時停止中のジョブをバックグランドジョブとして再開する
  bg %ジョブ番号
- バックグランドジョブをフォアグランドに移動する
  fg %ジョブ番号
- 一時停止中のジョブをフォアグランドジョブとして再開する
  fg %ジョブ番号

1.7 シェルスクリプトを作る

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．簡単なシェルスクリプトプログラムを実行する前に変数の話を少しします．

1.7.1 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．例えばname という変数を定義し，値としてtakeshi と設定する場合には

```
$ name='takeshi'
```

とします．この値は

```
$ echo $name
```

と実行することによって確認できます．このようにシェル変数を参照する場合には"$"記号を変数名の前につけます．またシェル変数の一覧を表示するにはset コマンドを使い，シェル変数を削除するときにはunset コマンドを用います．
UNIX ではプロセスというプログラムの実行単位でシステムを管理しています．つまりシェルプロンプト[5]から実行したプログラムは，起動中のシェルの子プロセスとなります．その子プロセスではシェル変数を参照することはできません．ところが UNIX のコマンドやアプリケーションの中には特定のシェル変数を参照するものがあります．そこで必要になってくるのが環境変数です．
環境変数はシェル変数のうち子プロセスから参照できるものです．シェル変数を

```
$ export name
```

とすることで環境変数に変更できます．[6]このような操作のことをエクスポートといいます．

[5] シェルプロンプトとは UNIX システムにログインすると表示される"$"や"%"のような記号のことです．mikilab ではユーザ名@mikilab:~ $と設定されています．
[6] ここではname という変数をエクスポートしています

1.7.2 シェルスクリプトの書き方

シェルスクリプトとは響きは難しいですが，中身は UNIX のコマンドを 1 行ずつ順番に書いたテキストファイルを作れば，それだけでシェルスクリプトを作れます．例えばHello と表示させるシェルスクリプトを作るとします．

```
#! /bin/sh
# hello.sh
echo 'Hello'
```

というhello.sh というテキストファイルを作成したとします．これを実行すると

```
$ ./hello.sh
Hello
```

となり，echo 'Hello' というコマンドが実行されたのと同じ結果になります．
このようにシェルスクリプトでは，UNIX の既存のコマンドを組み合わせて，オリジナルの新しいコマンドを作ることができます．

1.8 初期化ファイル

bash ではホームディレクトリに.bashrc というファイル作り，ログインしたときに環境変数の設定やエイリアスの設定を自動的に行うことが出来ます．

```
# This is an sample .bashrc
alias ls="ls -F"
alias h="history"
```

第2章　viエディタの使い方

2.1　概要

viは Unix の標準的なエディタで Windows で使われるメモ帳などのエディタとはコマンドモード，テキストモード（挿入モードとも呼ぶ）と呼ばれる二つのモード（状態）が有ることが最も異なります．コマンドモード時にはその名の通り編集に関連するコマンドを入力し，テキストモード時には実際の文章などを入力します．vi はこの二つのモードを使いこなすことによって必要最低限のキーストロークで編集を行えるように作られているので，慣れれば編集の効率を上げることが出来ます．

2.2　起動方法

vi を起動するためには次のように入力します．

vi [ファイル名]Enter [2]

ファイルが存在する場合はそのファイルを開き，存在しない場合はファイルを作成します．
例えば sample.txt というファイルを開くには

vi sample.txtEnter

と入力します．図 2.1 は何も指定せずに起動した場合の初期画面です．

2.3　コマンドモードとテキストモード

2.3.1　コマンドモード

vi は起動時にコマンドモードになっており，このモードのときにキーボードに入力すると，入力した文字は画面に表示されず[3]，vi に対してのコマンド（命令）として解釈され，その入力した文字や記号によって様々な操作を行なうことが出来ます．大文字小文字は区別され，似た動作をします．また，

2.3.2　テキストモード

テキストモードではキーボードからの入力はそのまま画面に表示され，実際に文章など入力するのはこのモードで行います．

2.3.3　モードの切り替え

- **コマンドモードからテキストモードに切り替える**

[1] VIsual の略です．
[2] ここで使用されている表記法の [ファイル名] というのは BNF 記法と呼ばれ簡単に説明すると[...] は省略可能で<...>は省略不可ということです．
[3]「:」を入力すると，それ以後の入力は画面の最下行に表示されます．

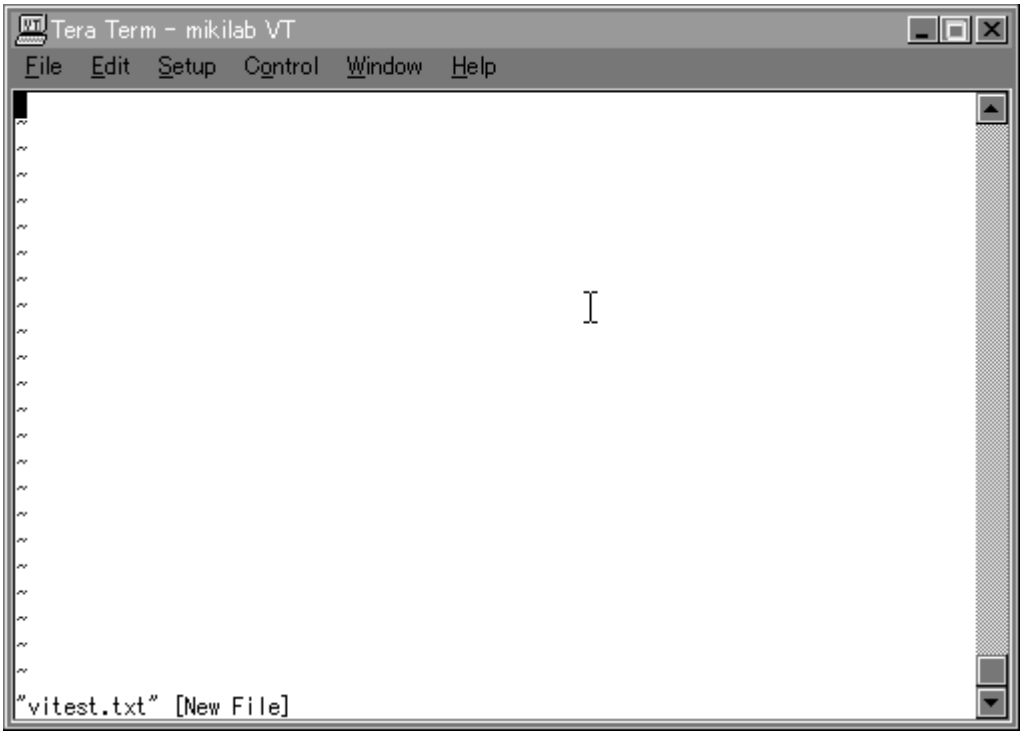

図 2.1: vi 起動時の画面

図 2.2: モードの表示

ESC を押します．既にコマンドモードのときはビープ音が鳴ります．

- **テキストモードからコマンドモードに切り替える**

  多くある中で表 2.1 に基本的なものを挙げます．

図 2.2 の様に画面下の方に-- INSERT --と出ているかどうかでモードを見分けることが出来る vi も有りますが基本的には画面上にはモードが表示されません．その場合 ESC を押してビープ音がなればコマンドモードです．
いずれにしても，ESC を押すとコマンドモードになるので必要に応じてテキストモードに切り替えてください．

2.4 文字を入力する

テキストモードなら直接入力を開始し，コマンドモードならa，i などを入力してテキストモードに移行してから入力します．

abc
↓ **a**
abc カーソル位置が右へ移動してテキストモードに移行
↓ **xyz**

表 2.1: モード切り替えコマンド

コマンド	動作
ESC	コマンドモードに切り替えます
a	カーソルの右からテキストを挿入します
A	行末からテキストを追加します
i	カーソルの左からテキストを挿入します
I	行頭からテキストを挿入します
o	下の行からテキストを追加します
O	上の行からテキストを追加します

表 2.2: 保存と終了コマンド

コマンド	動作
:w	上書き保存します
:w <ファイル名>	ファイル名をつけて保存します
:w! <ファイル名>[4]	既に存在するファイル名で保存します
:q	vi を終了します
:q!	編集したファイルを保存せずに終了します
:wq，ZZ	ファイルを保存して終了します

abxyzc　xyz が挿入される

2.5　ファイルの保存と終了方法

編集したファイルを保存するためにはコマンドモードで表 2.2 のようなコマンドを入力します．起動時にファイル名を指定しなかった場合に保存するには名前を付けて保存してください．また，ファイルはカレントディレクトリに保存されます．
例えば，新規作成したファイルに test.txt という名前を付けて保存する場合，

```
:w test.txt
```

とします．

2.6　カーソルを移動する

コマンドモードでカーソルを移動するためには通常カーソルを使いますが，カーソルの使えない環境も存在します．そのような環境では，表 2.3 のコマンドを使用します[5]．

[4] !は強制的に何かをするコマンドに使用されます．
[5] コマンドの前に数字を付けることにより複数回コマンドを実行したのと同じだけ移動することが出来ます．また，一般的に使用できます．

表 2.3: 移動系コマンド

コマンド	動作
h	左に移動します
j	下に移動します
k	上に移動します
l, space	右に移動します
0	行頭に移動します (Windows の Home)
$	行末に移動します (Windows の End)
G	ファイルの最後に移動します
:行番号	指定行に移動します

表 2.4: 削除系コマンド

コマンド	動作
x	カーソル位置の文字を削除します
X	カーソルの左の文字を削除します
dd	現在の行を削除します
dw	カーソル位置の単語を削除します
d<移動系コマンド>	カーソル位置から移動先までを削除します

2.7 文字を削除する

テキストモードで文字を削除するためには Backspace，もしくは Delete が使用できます[6]．
コマンドモードで文字を削除するためには表 2.4 のコマンドを使用してください．

次の三行があるとします
first line
second line
third line
↓ **dd**
first line　二行目が削除されます
third line
↓ **x**
first line
hird line　一文字削除されます
↓ **dw**
first line
line　一単語削除されます

2.8 テキストを置換する

置換には 1 文字を対象とするもの，文字列を対象とするものなどいくつかのタイプの置換があり，r 以外は入力後テキストモードになります．詳細は表 2.5 を参照してください．削除系コマンドと同様に3s によって 3 文字列を，5cc によって 5 行を置換することができます．

abc
↓ **r1**　b が1 に置換されます

[6]設定によって使えない場合もあります．

表 2.5: 置換系コマンド

コマンド	動作
r	1 文字を置換します
R	カーソル位置から ESC を押すまで置換します
s	カーソル位置の文字列を置換します
cc	行を置換します
C	カーソル位置から行末までを置換します

表 2.6: 検索系コマンド

コマンド	動作
/<文字列>	指定した文字列をファイルの終わりに向かって検索します (下方向検索)
?<文字列>	指定した文字列をファイルの先頭に向かって検索します (上方向検索)
n	次の文字列を検索します
Shift を押しながら n	検索方向と逆方向に文字列を検索します

a1c

2.9 文字列の検索

文字列を検索するには表 2.6 のコマンドを使用します．例えばカーソル位置より下の行にある section という文字列を検索するには/section と入力します．ｎを入力すると続けて次の候補を検索できます.. また，ファイルの終端まで検索するとファイルの先頭から検索を続けます.

2.10 コピー＆カット＆ペースト

Windows のクリップボードに相当するものとして vi には一時バッファというものが有り，ここにコピーコマンド (yy) や削除系のコマンド (dd など) で削除された文字 (列) が記憶されています．ペーストするためにはp を使います．これらのコマンドも数字を使って3yy で 3 行を，4yw で 4 つの単語をコピーすることが出来ます．表 2.7 に一覧をまとめます[7].

次の一行があるとします
left right
↓ **yw**　カーソル位置の単語 left をコピーします
left right
↓ **$**　行末に移動します
left right
↓ **p**　一次バッファ内の left を貼り付けます
left rightleft
↓ **yy → p**　行をコピーし，次の行に貼り付けます
left rightleft
left rightleft

[7] また，これらの操作をヤンクと呼びます.

表 2.7: ヤンク系コマンド

コマンド	動作
yy	現在の行をコピーします
yw	カーソル位置の単語をコピーします
dd	現在の行をカットします
dw	カーソル位置の単語をカットします
p	現在のカーソル位置に一時バッファの内容を挿入します.

表 2.8: その他便利なコマンド

コマンド	動作
e: <ファイル名>	続けて他のファイルを編集します
:r <ファイル名>	他のファイルをカーソルの下の行から追加します
:set number	行番号を表示します
Ctrl+g	現在の行番号を表示します
:set autoindent	オートインデントを有効にします. 無効にするには:set noautoindent

2.11 その他のコマンド

以上に出ていないコマンドで覚えていると便利なものを表 2.8 に挙げます.

第3章　正規表現とgrep

3.1　概要

本章ではまず，パターンを記述する**正規表現**について学びます．ファイルの整理やテキストの編集の際，あるファイルがディレクトリ構造上のどこにあるのかを調べたり，テキスト内の特定の文字列を探したりする必要が生じることがあります．このとき,「ある文字列が先頭にくる」といった指定ができれば，より高度な検索が可能となります．たとえば，C 言語のソースファイルは，ファイル名の最後に".c"をつけることになっているので，ファイル検索の際に「名前の最後が".c"であるファイル」という指定ができれば，膨大な数のファイルの中から C 言語のソースプログラムのみを探し出すことも可能になります．この,「名前の最後が".c"である」といったパターンを定義するのが正規表現です．

そして次に，UNIX の代表的な文字列検索コマンドである，grep について説明します．grep では，検索対象の文字列の指定に正規表現を用いることができます．これにより，上記のような検索が可能になります．

3.2　正規表現

3.2.1　正規表現とは

正規表現 (regular expression) は，パターンを定義する方法の一つです．パターンとは，簡単にいうと，ある「特徴」を持ったものの集まりです.「特徴」とは，前述の「名前の最後が".c"である」といった事柄を指します．正規表現は，文字列のパターンを定義します．ある正規表現の定義するパターンに一致する文字列は，その正規表現にマッチすると言います．

3.2.2　一文字にマッチする正規表現

特定の一文字にマッチする正規表現

正規表現の中で，最も単純でよく使われるのは，文字一個というものです．正規表現 x は，文字"x"とマッチします．例えば，正規表現 b は文字 (列)"b"とマッチし，正規表現 3 は文字"3"とマッチします．

任意の一文字にマッチする正規表現

正規表現**"."**(ドット) は，\n(改行文字) 以外の任意の一文字にマッチします．例えば，正規表現a. は，"aa"，"ab"，. . . といった文字列にマッチします．

文字クラス

文字クラスは，一対のブラケット ("["と"]") の間に文字のリストを並べたもので，その中の任意の一文字とマッチします．例えば，

```
[abcde]
```

は，アルファベット小文字の最初の 5 文字のうち，いずれか一つにマッチします．また，ダッシュ(`"-"`) を用いて文字の範囲を指定することによって，上と同じパターンを，

```
[a-e]
```

で表現することができます．

なお，文字クラス内の文字のリストに，右ブラケット (`"]"`) やダッシュ(`"-"`) を含めるには，その直前にバックスラッシュを置いて，それぞれ”`\]`”，”`\-`”，のように書きます．一般に正規表現では，特殊な意味を持つ文字をただの文字として扱う場合には，`"\"`をその文字の直前に置きます．

以下に，例を挙げておきます．

`[0123456789]`：数字一文字にマッチ．
`[a-zA-Z0-9]`：英文字，数字のいずれか一文字にマッチ．

また，**否定された文字クラス**というものがあります．文字クラス内の文字リストの先頭 (`"["`の直後) にキャレット (`"^"`) を置くと，リストに**含まれない**任意の一文字にマッチします．以下に，実例を示します．

`[^0-9]` 数字以外の一文字にマッチ．
`[^\^]` キャレット以外のいずれか一文字にマッチ．

3.2.3 複数の文字にマッチする正規表現

並び

複数個の文字にマッチする最も単純なパターンは，**並び**です．これは，`abc` という正規表現が最初に文字`"a"`，次に文字`"b"`，その次に文字`"c"` が来るような文字の並びとマッチすることを意味します．

繰り返し

ある正規表現の直後にアスタリスク (`"*"`) を置いたものは，直前のパターンが**0 個以上**並んだものとマッチします．同様に，プラス記号 (`"+"`) を置くとは**1 個以上**の繰り返しとなり，疑問符 (`"?"`) を置くと，直前のパターンが**0 個または 1 個**存在することを示します．例えば，`a*`は，

`""`(空列)，`"a"`，`"aa"`，`"aaa"`，...

とマッチします．また，`a+`は，上記のものから空列 (長さ 0 の文字列) を除いたものにマッチし，`a?`は空列と文字`"a"`にマッチします．

繰り返しの回数を指定したい場合には，**汎用繰り返し**を用います．汎用繰り返しは，`x{m,n}` のような書式になっています．この表記において，`x` は繰り返し対象の文字を，`m` は最小繰り返し回数，`n` は最大繰り返し回数を，それぞれ示しています．つまり，`c{3,5}` は，

`"ccc","cccc","ccccc"`

の三つのパターンとマッチします．また，最大繰り返し回数を省略して`x{m,}` のように書くと，「文字 x の m 個以上の並び」を意味し，コンマも省いて`x{m}` とすると，「文字 x の m 個の並び」を意味します．よって，`"*"`，`"+"`，`"?"`は，それぞれ`"{0,}"`，`"{1,}"`，`"{0,1}"` と等価です．

選択

ある文字列の選択肢から，そのうちの一つの文字列を指定したい場合，**選択**を用います．選択は，`expression1 | expression2` という書式になっています．例えば，`table|chair` は，文字列`"table"`か文字列`"chair"`の**どちらか**にマッチします．

3.2.4 位置指定

位置指定とは，パターン内のある部分を，文字列中の特定の位置に合致させることを言います．この目的で使用する正規表現は，文字にはマッチしません．位置指定には，"^"，"$"，"\b"，"\B"の 4 種類があります．

行頭と行末

本項で説明する位置指定は，パターン内の特定の部分を，行端 (**行頭**と**行末**) に一致させるものです．行頭の位置指定には^を，行末の位置指定には$を，それぞれ用います．例えば，^a は，行頭にくる"a"のみにマッチします．^が正規表現の先頭以外の場所にある場合は，単に文字"^"にマッチします．

先頭の文字"^"や，文字"$"を指定するには，その直前にバックスラッシュ(\) を置く必要があります．

単語境界

本項で説明する位置指定は，**単語境界**あるいは**単語境界以外**でマッチすることを指定します．\b は単語境界にマッチし，\B は単語境界ではない位置にマッチします．単語境界とは，

> 「[a-zA-Z] にマッチする文字と[^a-zA-Z] にマッチする文字との間，あるいは，[a-zA-Z] にマッチする文字と行端 (行頭と行末) との間」

と定義されます．例をいくつか挙げておきます．

fred\b : "fred"にマッチするが，"frederick"にはマッチしない．
\bwiz : "wiz"や"wizard"にマッチするが，"qwiz"にはマッチしない．
\bfred\b : "fred"にマッチするが，"frederick"や"alfred"にはマッチしない．
\b+\b : "x+y"にマッチするが，"++"や" + "にはマッチしない．
abc\bdef : いかなる文字列にもマッチしない (このような場所に単語境界はありえない)．
\bfred\B : "frederick"にマッチするが，"fred flintstone"にはマッチしない．

3.2.5 優先順位

これまで紹介してきた*，+，$，\，| などの記号には，**優先順位**が定められています．各記号の優先順位は，表 3.2.5 のように定められています．

表 3.1: 正規表現における記号の優先順位

優先順位	名前	表現
1	カッコ	()
2	繰り返し指定	+ * ? {m,n}
3	並び，位置指定	abc ^ $ \b \B
4	選択	|

これにより，たとえば a|b*は,「"a"が一個あるか，または"b"が任意個ある」と一意に解釈されます．また，優先順位が最も高い"("，")"を用いて (a|b)*とすると,「"a"と"b"からなる任意の長さの文字列」を示すことになります．このように，それぞれの記号の優先順位に逆らった使い方を望む場合には，カッコを用いると良いでしょう．

3.2.6 カッコによる記憶

3.2.5 項では，カッコを用いて記号間の結合を操作できることを述べました．しかし，カッコで囲まれたパターンにマッチした部分を，後で参照できるように**記憶**しておく働きがあります．参照には，**「バックスラッシュ + 整数」**という書式を用います.「整数」とは，その正規表現の先頭から数えて何番目のカッコかを示します．例えば，

```
a(.)b(.)c\2\1
```

という正規表現は，**adbeced** などにマッチします.

3.2.7 まとめ

本節で説明した記号の意味を，表 3.2 にまとめておきます.

表 3.2: 正規表現で使用する記号

優先順位	表現	意味
0	x	文字 x を示す
	.	任意の一文字（改行文字以外）
	[abc]	中のいずれかの一文字を示す文字クラス
	[^abc]	中に含まない一文字を示す否定された文字クラス
1	()	優先順位を変える．文字列を記憶する．
2	*	0 回以上の繰り返し
	+	1 回以上の繰り返し
	?	0 回以上 1 回以下の繰り返し
	{m,n}	m 回以上 n 回以下の繰り返し
3	abc	並び（特別な記号は無し）
	^	行頭の位置指定
	$	行末の位置指定
	\b	単語境界の位置指定
	\B	単語境界ではない位置の指定
4	|	選択（どちらかにマッチ）

3.3 grep

3.3.1 grep とは

grep はテキストファイル内の文字列を検索するコマンドです．grep では検索対象の文字列の指定に正規表現を用いることができる代表的なコマンドなので[1]，ここで紹介します.

3.3.2 書式

grep を用いて検索を行うには，プロンプトで次のように入力します.

```
grep [オプション] 検索パターン [ファイル名 1] [ファイル名 2] ...
```

[1] grep は "Global Regular Expression Print"(テキスト全体から，指定した正規表現に一致する文字列を検索し，出力する．) の略です.

引数には検索したいパターンをシングルクォーテーション(') で囲んで指定します．先にも述べたように，検索パターンには正規表現を用いることができます[2]．シングルクォーテーションは省略できます．検索文字列中にシェルが特別な意味として解釈する記号が含まれている場合，その記号がシェルによって解釈されないように，検索文字列をシングルクォーテーションで囲む必要があります．

検索パターンの後には検索対象とするファイルを指定します．ファイルは複数指定できます．次の例では，text1とtext2 という 2 つのファイルから，先頭が"d"である文字列を探しています．

```
grep 'd.*' text1 text2
```

grep の出力は一行ごとに表示されます．以下に出力例を示しておきます．例の各行において，最左端はファイル名であり，コロン(:) の右側は検索パターンと一致する文字列を含む行を抜き出したものです．

```
text1:a book on the desk.
text1:I am sad.
text2:it was suddenly happened.
```

ファイル名を指定しない場合，grep コマンドは標準入力からテキストデータを読み込み，検索を行います．デフォルトでは標準入力はキーボードに設定されていますが，パイプを用いて切り替えることができます．以下では，パイプを用いて上の操作と等価な検索を行っています．

```
cat text1 text2 | grep 'd.*'
```

3.3.3 オプション

grep の代表的なオプションを表 3.3 に示しておきます．

表 3.3: grep コマンドのオプション

オプション	機能
-c	検索パターンに一致する行の行数を表示
-i	検索パターンに指定した文字列の大文字/小文字を区別しない．
-l	検索パターンに一致する行を含むファイルのファイル名のみを表示する．
-n	検索パターンに一致する行の内容を行番号つきで表示する．
-v	検索パターンに一致しない行を表示する．

[2]正規表現には基本正規表現 (BRE) と拡張正規表現 (ERE) の 2 種類があり，"+"，"?"，"("，")"，""，""などは拡張正規表現に含まれます．拡張正規表現版grep として，UNIX にはegrep コマンドがあります．また，普通のgrep でも，-E オプションをつけると，拡張正規表現が利用できるようになります．