第 1 回 Linux ゼミ

ゼミ担当者　：　横田 山都，山下 尊也
開催日　：　2008 年 7 月 9 日

1　はじめに

本ゼミでは，サーバなどでよく使われる UNIX 系 OS（Linux，FreeBSD など）を操作する際に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．Windows や MacOS では GUI 操作が一般的であるがサーバの操作では CUI 操作が一般的である．そこで，本ゼミでは Linux 上での CUI 操作で必要なコマンドや Shell について学ぶ．

2　Linux とは

2.1　Linux の概要

Linux とは，1991 年にヘルシンキ大学の Linus Torvalds 氏により開発された UNIX 互換の OS である．Linux は既存の UNIX のソースを流用せず，何もないところから書きおこされたものであり，オープンソースソフトウェアとして，自由に改変，再配布ができるようになっている．Linux と呼んでいるのはカーネルであったり，ディストリビューションであったりする．カーネルとは，OS の基本機能を実装したソフトウェアであり，ディストリビューションは Linux（カーネル）を一般利用者が導入（インストール）したり，利用できる形にまとめ上げた配布形態である．

2.2　ディストリビューションの種類

Linux ディストリビューションごとの違いは，日本語化の度合いや，インストーラの完成度などである．現在，かなりの数のディストリビューションが存在しているが，主に Table1 に示すように「RPM 系」「Debian 系」「Slackware 系」の三つに分けることが出来る．我々の研究室で主に利用されているのは，Debian GNU/Linux というデストリビューションである．

Table 1　Linux ディストリビューションの大別

系統	ディストリビューション
RPM 系	Red Hat、Fedora、CentOS、Turbolinux
Debian 系	Debian、KNOPPIX、Ubuntu
Slackware 系	Slackware、Plamo Linux

2.3　オープンソースとは

オープンソースとは，あるソフトウェアのソースコードを無償で公開し，世界中のプログラマの誰もが自由にそのソフトウェアを改良して再配布することを許すソフトウェア開発方式のことである．そのため，公開されているソフトウェアの類似品や技術の転用が可能であり，またそれを基盤に自分の環境に合わせたり，新たな機能を追加するといったことも可能である．

3　Linux のコマンド操作

Linux の基礎的なコマンドを以下に紹介する．

3.1　コマンドの基本文法

Linux におけるコマンドの基本的な書式を以下に示す．ここでオプションとは，コマンドの付加的な機能を実行させるもので,「-(ハイフン) アルファベット」で表現されるものが多く，また引数とは，コマンドやオプションに与える値やファイル名などのことを示す．

1. コマンド

   例)pwd

2. コマンド 引数

   例)mkdir dir

3. コマンド 引数 1 引数 2

   例)cp file1 file2

4. コマンド [オプション]

   例)ls -l

3.2　man コマンド

man コマンドでは引数にマニュアルを読みたいコマンド名を指定する．man コマンドでは，一画面分の表示をすると，そこで表示を停止し、画面の一番下に”-more-”と表示される．この後はユーザーの入力した文字 (この場合はリターンキーを押さない) によって下のように動作が変わる．

- スペースキー　→　次の一画面を表示
- リターンキー　→　次の一行を表示

- q → man コマンド終了
- h → ヘルプ機能

3.3 ディレクトリ操作

コマンド操作では，カレントディレクトリ (ユーザが現在作業を行っているディレクトリ) やディレクトリ内の情報を把握するため，次のようなコマンドが必要である．

- pwd

  カレントディレクトリを絶対パスで表示する．

```
 $ pwd
/home/andy
```

  この表示は現在「home」ディレクトリの中の「andy」がカレントディレクトリであることを示している．

- ls

  ファイルやディレクトリの情報を表示する．

```
 $ ls
dir1 dir2
```

  「ls」の入力に対して「dir1」と「dir2」が表示されている．これは,「andy」ディレクトリには「dir1」と「dir2」が存在することを示している．また，このコマンドと共によく用いるオプションを Table2 に示す．

Table 2 ls の主なオプション

オプション	機能
-a	ドットで始まる隠しファイルも表示
-l	ファイルやディレクトリの詳細を表示
-lh	-l の出力を読みやすい形で表示
-F	名前の後ろにファイルの型を表示

- mkdir

  ディレクトリを作成する．

```
 $ mkdir dir3
 $ ls
dir3
```

  「mkdir dir3」と入力することでカレントディレクトリに「dir3」というディレクトリを作成している．

- cd

  カレントディレクトリを移動する．ディレクトリの変更には絶対パスと相対パスを用いた方法があるが，相対パスを用いる場合は，現在のカレントディレクトリを基準として、指定したいディレクトリ名を用い「cd [ディレクトリ名] 」と入力することで移動できる．なお，相対パス入力の場合は,「.」でカレントディレクトリ,「..」で 1 つ上のディレクトリへ移動できる．絶対パスを用いる場合は，ルートディレクトリを基準として，指定したいディレクトリまでを「／」区切りで示した絶対パスを用いて「cd [絶対パス]」と入力することで移動できる．

相対パス

```
 $ ls
dir1
 $ cd dir1
 $ pwd
/home/andy/dir1
```

絶対パス

```
 $ pwd
/home/andy
 $ cd /home/andy/dir1/
 $ pwd
/home/andy/dir1
```

3.4 ファイル操作

次にファイル名の変更，移動，コピー，削除といったファイル操作を行うコマンドを紹介する．

- mv

  ファイルの移動やファイル名の変更に使用する.「mv [移動するファイル名] [移動先ディレクトリ名]」と入力することでファイルをディレクトリに移動できる．また，ファイル名の変更は「mv [変更前ファイル名] [変更後ファイル名]」と入力することでファイル名を変更できる．

```
 $ ls
dir1 sample1
 $ mv sample1 dir1
 $ ls
dir1
 $ cd dir1
 $ ls
sample1
```

ここでは， sample1 というファイルを dir1 というディレクトリに移動している．

- cp
  ファイルやディレクトリをコピーする．「cp [コピー元ファイル名] [コピー先ファイル名]」と入力することでファイルをコピー できる．また，”-r”のオプションをつけることで，ディレクトリをコピーすることもできる．

```
 $ ls
sample1
 $ cp sample1 sample2
 $ ls
sample1 sample2
```

- rm
  ファイルやディレクトリを削除する．「rm [削除するファイル名]」と入力することでファイルを削除できる．ディレクトリの削除は「rm -r [削除するディレクトリ名]」で削除できるが，ディレクトリを削除する場合は，そのディレクトリに存在するファイルを全て削除しておく必要がある．

```
 $ ls
sample1 sample2
 $ rm sample1
 $ ls
sample2
```

3.5 ファイル・ディレクトリに関するコマンド

ファイル・ディレクトリに関する基本的コマンドを以下に紹介する．

- find
  ファイルを検索する．「find [開始ディレクトリ] [検索条件] [コマンド]」でファイルを検索できる．検索条件にはオプションを用いる．

```
 $ ls
dir2 sample2
 $ cd ../../
/home
 $ find /home -name sample2
/home/andy/dir1/sample2
```

- du
  ディレクトリ以下のファイル・ディレクトリのサイズを表示する．このコマンドと共によく用いられるオプションを Table3 に示す．

Table 3　du の主なオプション

オプション	機能
-k	1 ブロック 1KB 単位で表示
-h	読みやすい形式で表示
-s	指定ディレクトリについて表示

4 パーミッションの設定

本章では，パーミッションの設定を行う．パーミッションとは，ファイルやディレクトリに対するユーザのアクセス権のことである．

4.1 パーミッションの設定対象

Linux におけるパーミッションではファイル，ディレクトリの所有者である Owner，同じマシンを利用できるユーザ全体を意味する Group，そしてその他の Other に対して，それぞれ「読み込み (r)」「書き込み (w)」「実行 (x)」の権限の設定ができる．以下にパーミッションの確認方法と変更方法を紹介する．

4.2 パーミッションの確認方法

ファイル・ディレクトリの情報を表示する「ls」コマンドに，ファイルの詳細を表示するオプション「-l」をつけて実行することで確認する．以下の例ではファイル「sample1.txt」のパーミッションを確認している．

```
 $ ls -l   sample1.txt
-rw-r--r-- 1 andy None 0 Apr 11 13:51
sample1.txt
```

「ls -l」コマンドによる出力結果を空白で区切るとすると，左端「-rwxr-xr-x 」において，一番左の文字が「-」ならファイル,「d」ならディレクトリという意味である．一番左の文字を除いた残りの文字は左から 3 文字の区切りに分けられ，Owner，Group，Other に与えられた権限を示している．例えば,「-rw-r–r–」という表示では，Owner には「読み込み (r)」「書き込み (w)」，Group には「読み込み (r)」，Other には「読み込み (r)」の権限が与えられているということを示している．以降順に「1」がリンク数,「andy」が所有者,「None」がグループ所有者でグループ所有者がいない事を示している.「0」がファイル・ディレクトリサイズ,「Apr 11 13:51」がファイルの修正時刻を表している．

4.3 数値を用いたパーミッションの設定

- chmod 【モード (数値)】 【ファイル名】
  パーミッションを設定する際は 3 桁の数字が用いられる．左から順に Owner 権限，Group 権限，Other 権限を意味し，それぞれの対象に与えたい権限を，Table4 に示す数字を足し合わせて指定する．例えば「chmod 755 sample1」であれば，sample1 の Owner に全ての権限，Group と Other に読み込み，実行の権限が与えられる．

Table 4　数字，文字と権限の対応

記号	数字	権限
r	4	読み込み
w	2	書き込み
x	1	実行
-	0	なし

```
 $ ls -l sample1.txt
-r-xr--r-- 1 andy ...  sample1.txt
 $ chmod 755 sample1.txt
 $ ls -l sample1.txt
-rwxr-xr-x 1 andy ...  sample1.txt
```

4.4 文字を用いたパーミッションの設定

- chmod モード (文字) ファイル名
  この方法は，どのユーザに対して，どのパーミッションを追加・削除するかを決める方法である．Table4 に示すように，権限を与える対象を Owner(u)，Group(g)，Other (o)，全て (a) とし，行う操作は+（追加），-（削除），=（設定）のいずれかである．オプションには,「対象・操作・権限」の順に続けて入力する．例えば「chmod go+rw sample1」と入力した場合は，sample1 に対して，Group と Other に「読み込み (r)」と「書き込み (w)」の権限を追加したことになる．

Table 5　文字による権限設定

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)
権限	r(読み込み)，w(書き込み)，x(実行)

```
 $ ls -l sample1.txt
---------- 1 andy ...  sample1.txt
 $ chmod go+rw sample1,txt
 $ ls -l dir1
----rw-rw- 1 andy ...  sample1.txt
```

5 プロセスに関するコマンド

プロセスに関する基本的なコマンドを以下に紹介する．

- ps

  現在動作中の自分のプロセスを確認する．Table6 に主なオプションと機能の対応を示す．

Table 6　ps のオプション

オプション	機能
aux	全てのプロセスを表示
a	他のユーザが権限を持つプロセスも表示
u	ユーザ名と開始時刻を表示
x	制御端末のないプロセスについても表示

```
 $ ps
PID PPID PGID WINPID TTY UID STIME
COMMAND
S 4212 1 4212 4212 con 1000 21:12:46
/usr/bin/bash
S 3904 4212 3904 1528 con 1000
21:16:53 /usr/bin/top
S 6136 4212 6136 3108 con 1000
21:18:02 /usr/bin/ps
```

このように「ps」を使うことで，3 つのプロセスが動いていることが確認できる．

- top

  実行中のプロセス情報を表示する．メモリの消費量，スワップ消費量，各プロセスのプロセス ID，メモリ消費量などがわかり，現在動作しているプロセスをリアルタイムに表示して確認することができる．

- kill

  ジョブ・プロセスを終了させる.「kill プロセス番号」と入力することによって，指定したプロセスを停止することができる．強制終了する際は，オプションに「-KILL」もしくは「-9」を用いる．オプションにはシグナルを指定することができ，それはシグ

ナル名でもシグナル番号でもよい．シグナル番号は「kill -l」で確認できる．

6　テキストエディタ

Linux では，ユーザの文書やプログラムソースだけでなく，システム設定ファイルまでもがテキスト形式となっている．そのため，テキストエディタが必須となる．Linux 上で使われる代表的なテキストエディタに vi と Emacs がある．本章では，その vi と Emacs の特徴と基本的な操作方法を紹介する．

6.1　vi とは

vi は Linux システム標準のテキストエディタであり，emacs に比べてプログラムが小さく起動が速く負荷も少ない．しかし，キーバインドが独自であり，操作を行うにはコマンドを覚える必要がある．これは，次節で説明する Emacs も同様である．

vi にはコマンド入力モードとテキスト入力モードがあり，それらを切り替えながらテキストを作成する．コマンド入力モードではファイルを編集したい位置にカーソルを移動したり，ファイルを保存したり，テキストを検索するといった操作を行う．それに対して，テキスト入力モードでは，実際に入力したい文字を入力する．

6.1.1　vi ファイル編集

まず，vi 起動をするために,「vi」とコマンド入力する．ファイル名を指定する場合は「vi ファイル名」と入力する．エディタ起動後にファイルを開くためには,「:e ファイル名」と入力する．ファイルを保存するためには,「:w」と入力する．また,「:w ファイル名」と入力することで，別ファイルに保存することができる．vi で用いるコマンドを Table7 に示す．

入力された文字を消去したいときは，その文字の上にカーソルを移動して「x」と入力する．また,「dd」と入力すると，カーソルのある一行が全て消去される．vi を終了するときは「:q」と入力する．

6.2　Emacs とは

Emacs とは，vi と並び，Linux でよく使われるエディタである．一般的に GUI で表示されるエディタで，Emacs Lisp というプログラミング言語を用いることでエディタの機能をさらに拡張することができる．本節では Emacs の基本的な使い方についてのみ説明する．

6.2.1　Emacs ファイル編集

Emacs をコマンドライン上で起動するためには，「emacs」と入力することで起動する．ファイル名を指定する場合「emacs ファイル名」と入力し，エディタを開く．以前に保存したファイルを呼び出すには「C-x C-f」と入力し，画面下部で呼び出すファイル名を入力する．

Table 7　vi コマンド表

コマンド	意味
i	カーソルの直前にテキストを入力
a	カーソルの直後にテキストを入力
I	行の末尾にテキストを入力
A	行の先頭にテキストを入力
/ *keyword*	*keyword* を下方向に検索
? *keyword*	*keyword* を上方向に検索
o	改行してテキストを入力
ESC	コマンドモードへ戻る
:e	ファイルの読み込み
:w	ファイルの保存
:w *filename*	*filename* に保存
:q	vi の終了
:q!	保存せずに強制終了
:wq	書込みと同時に vi を終了
h	カーソルを 1 文字左へ移動
l	カーソルを 1 文字右へ移動
j	カーソルを 1 文字下へ移動
k	カーソルを 1 文字上へ移動
x	カーソル位置の文字を削除
$	カーソルを行の末尾に移動
0	カーソルを行の先頭に移動

編集したファイルを保存するには「C-x C-s」と入力する．新規ファイルの場合，画面の下部で保存ファイル名を入力することによって保存できる．また，そのファイルが以前に作成されたファイルならば，上書きされる．Emacs でのコマンドを Table8 に示す．なお，Table8 の「C-x」は Ctrl キーを押しながら'x' を押すという意味であり,「M-x」は Meta キーを押しながら'x' を押すという意味である．Meta キーについては，近年のパソコンでは，Alt キーに相当する事が多く，また，Eac キーを押し，その後に，'x' を押す事で，同様の動作をする事が出来る．

Table 8　Emacs コマンド表

キー入力	意味
C-x C-f	ファイルを開く
C-x C-s	上書き保存
C-x s	すべてのファイルを保存
C-x C-c	ファイルを閉じる
C-x C-v	別のファイルを開く
C-x i	カーソル位置に別ファイルを挿入
C-x C-w	ファイルを別名で保存
C-s *keyword*	*keyword* を下方向に検索
C-r *keyword*	*keyword* を上方向に検索
C-f	カーソルを 1 文字前へ移動
C-b	カーソルを 1 文字後ろへ移動
C-n	カーソルを 1 文字下へ移動
C-p	カーソルを 1 文字上へ移動
C-a	カーソルを行の先頭に移動
C-e	カーソルを行の末尾に移動
C-v	次ページへ移動
M-v	前ページへ移動
M-<	ファイルの先頭に移動
M->	ファイルの末尾に移動

6.3　vi と Emacs の違い

vi と Emacs の重要な違いは次の通りである．

- vi はモードのあるエディタであるが，Emacs にはモードがない．vi ではテキストの変更やカーソルの移動を非常に単純なキー入力のコマンドで行うため，それらコマンドとテキスト内容としてのキー入力の区別が必要となる．結果として、ユーザーはテキスト入力モードとコマンド入力モードを切り替えながら編集を行うことになる．
- vi は小型で高速だが，(少なくとも本来は）カスタマイズがあまりできない．Emacs は低速（特に立ち上げ時）だが，カスタマイズは無制限である．
- vi は通常，テキストモードのコンソール上で使用されるが，GNU Emacs は一般にグラフィカルユーザインタフェース (GUI) を備えたアプリケーションとして動作する．ただし，vi の派生版である vim には，GUI を備えた gvim というアプリケーションが存在する．一般的に vi はテキストモードのコンソール上で，Emacs は GUI アプリケーションとして動作する．一般ユーザからすれば GUI で動作する Emacs の方が扱い易いが，コマンド操作に慣れている人にとっては，キー操作のみで処理でき，小型で高速な vi の方が使い勝手が良いとされている．

7　シェル (Shell)

7.1　シェルの概要

Linux において，ユーザは OS の核となるカーネルに直接アクセスして操作することができない．このため，ユーザーが入力したコマンドを解釈して，ユーザとカーネルの橋渡しをするシェルが必要となる．シェルはユーザの操作を受付けて与えられた指示をカーネルに伝える役割を持つ．シェルにより，ユーザはカーネルと対話しているように OS を操作できる．

シェルには主にコマンドインタプリタとスクリプト言語の 2 つの機能がある．Linux を利用するにあたって，コマンド入力をシェルが解釈しカーネルに伝えるというこの一連の動作は，インタプリタとしてシェルが動作している状態である．

このように対話的に操作を行う一方で，シェルにはバッチ処理を行わせるプログラミングの機能がある．シェルの記述法に基づいてスクリプトを書くことによって，複数のコマンドを 1 つに組み合わせることができる．また，繰り返しや分岐などの制御構造を持つことで，単なる順次処理だけではなく，より複雑な処理を記述できる．これら 2 つの機能から，シェルは Linux を利用する上で必要不可欠なものである．

7.2　代表的なシェル

シェルにはいくつか種類があるが，ここでは代表的なものを 4 つ紹介する．

- sh(Bourne Shell)
  現在利用できる最も古いシェルで，B シェルとも呼ばれている．このシェルは AT&T のベル研究所で開発され，開発者の 1 人である Steven Bourne 氏に因んで名付けられた．様々なシェルの中で共通項的な位置にあり，ほとんど全ての UNIX で利用できる標準的なシェルである．
- bash(Bourne Again Shell)
  B シェルを拡張したシェルである．ユーザーインターフェイスとしての機能を強化するため，ヒスト

リー機能，エイリアスなどが追加されている．Linux では，デフォルトのシェルとしてこの bash を使えるようになっている．本ゼミでは，bash を基に進めていく．

- C シェル (csh)
  シェルプログログラミングの方法が C 言語の構造に似ているため,「C Shell」と呼ばれている．B シェルにはない便利な機能が多くあるが，B シェルと C シェルではかなり非互換部分がある．

- Korn シェル (ksh)
  このシェルは，David G. Korn によって開発され，B シェルと互換性がある．C シェルのよう，組み込み演算機能，配列，文字列操作などのプログラミング機能がある．

- Z Shell(zsh)
  このシェルは，bourne shell 系のコマンドと csh 系のコマンドの両方が使える上に，ksh 系のコマンドライン編集機能も実装された究極のシェル．

8 シェルコマンド

シェルコマンドはシェルのコマンドインタプリタの機能である．本ゼミで扱った「mv」,「cd」などのコマンドもこれに当たる．本節以降で説明するスクリプト言語に関して，出力やファイル操作などを行う際に便利となるコマンドを紹介する．

8.1 基本コマンド

下記に各コマンドとその特徴，実行例を示す．

- echo
  文字列をそのまま出力する．

```
$ echo 文字列
```

ex:echo

```
$ echo aiueo
aiueo
```

- cat
  ファイルの内容を出力する．

```
$ cat ファイル名
```

ex:cat

```
$ cat example1.txt
kakikukeko
sasisuseso
```

- grep
  ファイル内の文字列を検索する．

```
$ grep 検索文字列 ファイル名
```

ex:grep

```
$ grep kaki example1.txt
kakikukeko
```

- sed
  ファイル内の文字列を置換して出力する．

```
$ sed s/置換前文字列/置換後文字列/ ファイル名
```

ex:sed

```
$ sed s/kakikukeko/aiueo/ example1.txt
aiueo
sasisuseso
```

- expr
  数値演算を行う．
  基本の演算子は和「+」差「-」積「*」商「/」である．

```
$ expr 数式
```

ex:expr

```
$ expr 5 + 3
8
$ expr 8 - 2
6
```

- cut
  ファイルの一部を出力する．下記のように x と y に数値を入れることにより，ファイル内の各行のファイルの x 文字目から y 文字目までを表示する．

```
$ cut -c x-y ファイル名
```

ex:expr

```
$ cut -c 1-3 sample1.txt
kak
sas
```

これらはファイルを開かずに目的に応じて，ターミナルなどに結果を出力することができる．これらのコマンドを次節に述べるシェルの機能と組み合わせることで，効率的な作業が行える．

9　シェルの便利な機能

9.1　リダイレクト機能

リダイレクトとは「入出力の切り替え」という意味で，標準入力をキーボード以外のファイルに指定したり，標準出力をディスプレイ以外のファイルに指定することである．Table 9 にリダイレクト機能を用いるための記号を示し，リダイレクト機能の簡単な例を示す．

Table 9　リダイレクト機能

コマンド	機能
<	標準入力のリダイレクト
>	標準出力のリダイレクト
>>	標準出力をファイルへ追加する
>&	エラー情報もファイルへ出力する
>>&	上記の追記版
2>	エラー情報のみをファイルへ出力

ex:redirect

```
$ echo Hello! > out1.txt
$ cat out1.txt
Hello!
```

この例では，標準入力の内容を指定したファイルに出力した．指定したファイルに出力すると同時に，画面出力も行いたい場合は，tee コマンドを用いる．

```
$ コマンド | tee 出力ファイル名
```

ex:tee

```
$ echo hello! | tee out2.txt
hello!
$ cat out2.txt
hello!
```

この例では，標準出力とファイル出力を同時に行っている．

9.2　コマンドの連続実行

9.2.1　条件なし連続実行

基本的に複数のコマンドを連続実行するには，次の記述を用いることで行える．

```
$ コマンド 1; コマンド 2; …
```

ex:連続実行

```
$ pwd;ls
/home/mitsuharu
example1.txt sennryaku smap.sh test
```

この例では，pwd コマンドと ls コマンドを連続して実行している．

9.2.2　条件あり連続実行

条件ありの連続実行では，前の実行コマンドが正常終了，異常終了かを判断して次の処理を進める．

コマンドを下記のように「||」で区切ることにより，前のコマンドが異常終了した場合に次のコマンドを実行する．

```
$ コマンド 1||コマンド 2|| …
```

また「&&」で区切ることにより，前のコマンドが正常終了した場合に次のコマンドを実行する．

```
$ コマンド 1&&コマンド 2&& …
```

連続実行の利点として，出力を一括してファイル出力できるため，リダイレクトを用いる場合に逐次追記する必要がないなどが考えられる．

9.3　出力の利用

シェルの機能としてコマンドの出力を利用できる．出力を利用するためのコマンドを$() で囲むことにより，その出力がコマンドや引数として動作する．

```
$  $(コマンド)
```

ex:出力の利用

```
$ cat example2.txt
ls
$ $(cat example2.txt)
example1.txt sample2.txt sennryaku
smap.sh test
```

9.4　パイプ

シェルにはパイプという機能があり，あるコマンドの標準出力を別のコマンドの標準入力に連結することが可能である．

```
$  コマンド 1|コマンド 2|コマンド 3…
```

上記のように入力することによりコマンド 1 の出力をコマンド 2 の引数として用いられる．

ex:パイプ

```
$ cat example3.txt
c
b
a
$ cat example3.txt|sort
a
b
c
```

この例では，cat コマンドで「c，b，a」を出力し，それをパイプ機能を用いて，sort コマンドでソートし「a，b，c」を出力している．

9.5 正規表現

正規表現とは，文字列パターンを指定できる表現方法である．任意の文字を「.」，直前文字の繰返しを「*」などの記号で表現できるため，あらゆる文字列のパターンの指定ができる．そのため，コマンドによる検索や置換など，複数のファイルやディレクトリを同時に指定し操作することが可能である．Table10 に代表的な正規表現の記号を示す．

Table 10　正規表現

記号	意味
.	任意の 1 文字または 0 文字
*	直前文字の 0 回以上の繰返し
+	直前文字の 1 回以上の繰返し
[ ]	列挙された任意の 1 文字
^	行頭を表す
$	行末を表す

ex:正規表現

```
$ cat a.txt
bbbb
99dlakj
$ sed s/^[0-9][0-9]*// a.txt
bbbb
dlakj
```

この例では，正規表現を用いて，文字列の先頭が数字の場合は，その数字が続く限り，それらの数字を消去している．

10 シェル変数と環境変数

10.1 シェル変数

シェル変数とは，動作中のシェル内でのみ有効な変数である．ユーザはシェル変数を利用することによって，シェルやコマンドの動作を操作できる．シェル変数は以下のように入力することにより設定できる．シェル変数名は，英文字あるいはアンダースコアで始まり，その後ろに 0 個以上の英数字，あるいはアンダースコアの並びで定義する．

```
$ set シェル変数名
```

また，下記のように「=」を使用することによりシェル変数に値を格納することもできる．

```
$ シェル変数名=値
```

設定したシェル変数の一覧は「set」と入力することで確認できる．一覧の表示では，新規に設定した変数が降順に表示される．また，既存の変数名を入力することにより，その変数の値を確認できる．シェル変数には，ユーザが設定したシェル変数以外にも，あらかじめ用意されている定義済みシェル変数がある．定義済みシェル変数を変更することによって，ユーザのシェル環境を自由に変更することができる．

一度設定したシェル変数は次のように入力することで削除することができる．

```
$ unset シェル変数名
```

シェル変数は，宣言をしたシェルのみで有効であり，使用しているシェルを終了させると，値は消えてしまう．

10.2 環境変数

環境変数とは，そのシェルから起動されたプロセスに受け継がれる変数である．上述したシェル変数は，設定したシェルの中でのみ有効である．そのため，他のシェルやアプリケーションからは利用できない．設定したシェル変数を別に起動したシェル内で使用するには，シェル変数を環境変数にする必要がある．環境変数を定義するコマンドは次の通りである．

```
$ export 環境変数名
```

シェル変数と同様に，ユーザが設定した環境変数以外にも，あらかじめ用意されている定義済み環境変数がある．これらの環境変数の一覧は次のように入力することで確認できる．

```
$ env
```

10.3 エイリアス

エイリアス機能を使うと，コマンドの別名 (エイリアス) を登録できる．ユーザは頻繁に用いるコマンドや，長いコマンドを簡単な名前のコマンドで登録すること

で，効率的な作業が行える．エイリアスを登録するには，alias コマンドを用いる．

```
$ alias 新コマンド=コマンド名
```

エイリアス機能を使って，初めからコマンドとして存在する文字列をエイリアスに指定してしまうと，初めから存在するコマンドを使用できなくなってしまう．そこで，alias コマンドを用いる前に，登録する文字列がコマンドとして存在しないことを確認しておく必要がある．コマンドの存在を確認するには which コマンドを用いて，「which コマンド名」とすることで確認できる．which コマンドを実行して，「Command not found」と表示されたら，その文字列はコマンドとして存在しないので，エイリアスとして使用できる．

ex:alias

```
$ alias lsl="ls -l"
$ lsl
total 1
-rw-r--r-- 1 andy None 2 May 9 17:10
sample1.txt
```

この例では，"ls -l"をエイリアス機能を使って"lsl"で実行できるようにしている．

11　シェルスクリプト

シェルは，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解釈し実行する機能を備えている．これにより，バッチ処理のような一連の仕事を，一つのコマンドでできるようにし，処理の自動化が可能となる．本章では，シェルスクリプトの基本文法を中心に説明する．

11.1　シェルスクリプトの基本

本節では，シェルスクリプトの記述方法，実行方法について説明する．

11.1.1　シェルスクリプトの記述方法

以下にシェルスクリプトの記述方法を例に基づいて説明する．

ex:sample1.sh

```
#!/bin/bash
date
echo "I am $USER"
echo "Hello!"
```

1 行目は，スクリプトを実行するのに使用するプログラムを指定している．ここでは，bash シェルを起動している．2 行目からは，スクリプトで実行したいコマンドを記述している．例では，日付，ユーザ，メッセージを順に表示するようになっている．

11.1.2　シェルスクリプトの実行方法

作成したシェルスクリプトを実行するには，いくつか方法がある．

```
$ .  スクリプトファイル
```

```
$ source スクリプトファイル
```

```
$ bash  スクリプトファイル
```

いずれのコマンドも，同様にスクリプトを実行できる．

sample1.sh の実行結果

```
$ .  sample1.sh
Fri May 11 23:07:09 2007
I am andy
Hello!
```

この例では，date コマンドを用いて日付を表示し，環境変数に格納されたユーザ名を表示している．また，スクリプトをシェルコマンドとして使用したい場合は，alias コマンドを用いて，「alias sample1.sh=". sample1.sh"」のようにエイリアスを定義すればよい．

ex:スクリプトのコマンド化

```
$ alias ex1=".  sample1.sh"
$ ex1
Fri May 11 23:12:42 2007
I am andy
Hello!
```

11.1.3　特別なシェル変数

シェル変数には，あらかじめ用意された特別な変数があり，その変数を使用，参照することができる．Table 11 に代表的な変数の一覧を示す．

Table 11　シェル変数

変数	説明
$n	スクリプトに渡された n 番目の引数
$#	与えられた引数の個数
$@	$0 以外の全引数

これら特殊な変数を用いたシェルスクリプトの例を以下に示す．

ex:sample2.sh

```
#!/bin/bash
echo "$0"       #スクリプト名
echo "$1,$2"    #1 番目と 2 番目の引数
echo "$#"       #引数の個数
echo "$@"       #全引数
```

ex:実行結果 sample2.sh

```
$ ./sample2.sh shogo mitsuharu shibutani
sample2.sh
shogo,mitsuharu
3
shogo mitsuharu shibutani
```

この例では，スクリプト実行と同時に引数を渡し，それらを表示している．

11.2 シェルスクリプトの制御構文

シェルスクリプトは，他のプログラミング言語のように制御構文がある．本節ではシェルスクリプトで使用する制御構文について説明する．

11.2.1 条件式

制御構文では，条件判定がよく用いられる．制御構文で用いる条件判定で，文字列の比較や整数値の大小比較を行う場合，以下のように記述する．

```
$ test 条件式
```

```
$ [ 条件式 ]
```

前者は記述方法は，test コマンドを用いた条件判定であり，条件が真の時は 0 を返し，偽の時は 1 を返す．また，後者の記述方法は条件式の前後にスペースを空けなければエラーとなるので注意する．

条件判定には次のような演算子が用いられる．文字列比較演算子を Table 12，ファイル属性演算子を Table 13，論理演算子を Table 14，数値評価演算子を Table 15 にそれぞれ示す．

Table 12　文字列比較演算子

記号	意味
文字列 1 = 文字列 2	2 つの文字列が一致する
文字列 1 != 文字列 2	2 つの文字列が一致しない
-n 文字列	文字列が null でない
-z 文字列	文字列が null である

Table 13　ファイル属性演算子

記号	意味
-d file	指定ファイルがディレクトリである
-e file	指定ファイルが存在する
-f file	指定ファイルが普通のファイルである

Table 14　論理演算子

記号	意味
!条件	条件が偽である
条件 1 -a 条件 2	2 つの条件がともに真である
条件 1 -o 条件 2	2 つの条件のどちらかが真である

Table 15　数値評価演算子

記号	意味
数値 1 -lt 数値 2	数値 1 が数値 2 より小さい
数値 1 -le 数値 2	数値 1 が数値 2 以下
数値 1 -eq 数値 2	数値 1 と数値 2 が等しい
数値 1 -ge 数値 2	数値 1 が数値 2 以上
数値 1 -gt 数値 2	数値 1 が数値 2 より大きい
数値 1 -ne 数値 2	数値 1 と数値 2 が等しくない

test コマンドを用いて,「/home/andy/sample1.sh」がディレクトリであるか調べた結果を以下に示す．

ex:ファイル属性演算子

```
$ test -d /home/andy/sample1.sh
$ echo $?
1
```

”$?”は直前のコマンドの終了ステータスを意味する．「/home/andy/sample1.sh」はファイルであるため，1 を出力する．

11.2.2 if 文

条件分岐である if 文は，if，elif，else を上から順に判定し条件が成立となれば，そのコマンドを実行する．test を用いて真偽の判定も可能である．

if 文の構造

```
if [条件]
then
      コマンド 1
elif [条件]
then
      コマンド 2
else
      コマンド 3
fi
```

ex:if.sh

```
#!/bin/bash
tmp-10
if[ $tmp -lt $1]
then
echo "10 < "  $1
elif[ $tmp -ge $1 ]
then
echo "10 >" $1
fi
```

ex:if.sh 実行結果

```
$ . if.sh 5
10> 5

$ . if.sh 13
10< 13
```

この例では，引数とシェル変数 tmp との大小を判別している．ここで，シェル変数 tmp は 10 であり，引数を 5 とすると「10>5」となり，13 とすると「10<13」と不等号で表示される．

11.2.3 for 文

for 文は下記のように記述することで do と done で挟まれたコマンドを繰り返す．リストには「引数 1 引数 2 引数 3 …」といったように引数が複数用いられ，繰り返し毎に 1 つ 1 つ変数に代入され引数がなくなるまで処理を繰り返す．シェルスクリプトによる繰り返し文は，ファイルを引数として用いることもでき，C 言語などの for 文とは異なり，複数のファイルを処理することによく使われる．

for 文の構造

```
for 変数 in  リスト (引数...)
do
      コマンド
done
```

ex:for.sh

```
#!/bin/bash
for x  in   1 2 3
do
      echo $x
done
```

ex:for.sh 実行結果

```
\$ .for.sh
1
2
3
```

この例では，引数である「1」「2」「3」が順に変数 tmp に格納される．引数がなくなった時点で，繰り返しは終了する．実行結果では，echo コマンドの繰り返しにより,「1，2，3」が順に表示される．また，変数は「$」を前につけることにより，値を参照できる．

11.2.4 while 文

while 文は条件の真偽により処理を繰り返すことに用いられる．条件が真の間 do-done 間を繰り返す．偽ならば処理終了となる．

while 文の構造

```
while [条件式]
do
      コマンド
done
```

ex:while.sh

```
#!/bin/bash
tmp=50
while [ $tmp -gt $1]
do
echo $tmp
tmp=$(expr $tmp - 5)
done
```

ex:while.sh 実行結果

```
$ . while.sh 20
50
45
40
35
30
25
```

この例では，シェル変数 tmp と比較して「tmp>引数」となる間，処理を繰り返すというシェルスクリプトである．while 文の中では，シェル変数 tmp の値を表示し，

その後で，シェル変数 tmp の値を 5 減算するという操作をしている．

11.3 シェル関数

シェルスクリプトでも関数の作成が可能である．シェル変数に関数を定義することでコマンドのようにも使用することが可能である．シェルスクリプト実行時に引数を記入することで，関数内部にも引数を渡すことが可能である．

シェル関数

```
関数名 (){
        コマンド
```

ex:file_func

```
#!/bin/bash
func(){
echo"$1+$2=$(expr $1 + $2)"
echo"$1-$2=$(expr $1 - $2)"
echo"$1*$2=$(expr $1 \* $2)"
echo"$1/$2=$(expr $1 / $2)"
echo"$1\%$2=$(expr $1 \% $2)"
}
```

ex:file_func 実行結果

```
$source file_func
$ func 8 2
8+2=10
8-2=6
8*2=16
8/2=4
8\%2=0
```

この例では，四則演算を file_func ファイルの func 関数に処理を行わせている．func 関数はシェル変数を 2 つ用いているので，func の後に 2 つ引数をつけている．

12 screen

screen は，1 つの端末上で複数の仮想的な端末を実現する．お互いの端末同士で連携したり，不意の切断時にも復旧できたりという機能を持っており，リモートログインをしての作業や，ウインドウを複数表示しきれないノート PC などで重宝されている．

12.1 screen の使い方

Table16 に示すコマンドを使うと，1 つのターミナルで複数のスクリーン（ウィンドウ）を開き，ウィンドウを切り替えることができる．Table16 に screen のキー割り当ての一部を示すが、これ以外にも多数の割り当てがある．「C-a」は,「Control」を押しながら「a」のキーを押す．

Table 16　screen コマンド表

コマンド	意味
C-a	次のウィンドウに切り替える
C-a	前のウィンドウに切り替える
C-a 数字	指定した数字のウィンドウに切り替える
C-a c	新しいウィンドウを作成して bash を起動する
C-a w	開いているウィンドウの一覧を表示する
C-a A	ウィンドウのタイトルを変更する
C-a?	ヘルプを表示する
C-a C-a	直前にアクティブだったウィンドウに切り替える

12.2 デタッチ

端末が何らかの理由で終了すると，その上で動いていた screen は「デタッチ」という状態になる．この様なときに「screen -r」というオプションをつけて screen を起動させると，デタッチ状態の screen を現在の端末上に復旧（アタッチ）させることができる．

12.3 コピーモード

コピーモードでは，screen に表示された文字列をコピーすることができる．コピーを行う際のコマンドを Table17 に示す．

Table 17　コピーモードコマンド表

コマンド	意味
C-a [	コピーモードに入る
C-a Enter	コピーする始点・終点を指定する
C-a]	コピーされた文字列をペーストする