|  |
| --- |
| 博士課程教育リーディングプログラム  実世界データ循環学リーダー人材育成プログラム |
| データツールファースト |
| Linux入門 |

|  |
| --- |
| 名古屋大学  2014/03/03 |

* Linuxは、Linus Torvaldsの米国およびその他の国における登録商標あるいは商標です。
* UNIXは、The Open Groupの登録商標です。
* その他の、記載されている会社名、製品名は、各社の登録商標または商標です。

目次

[1. Linux 1](#_Toc381605471)

[1.1 Linuxとは 1](#_Toc381605472)

[2. シェル 3](#_Toc381605473)

[2.1 シェルとは 3](#_Toc381605474)

[2.1.1 標準入出力 4](#_Toc381605475)

[2.1.2 パイプ 5](#_Toc381605476)

[2.1.3 変数 6](#_Toc381605477)

[2.1.4 メタキャラクタ 8](#_Toc381605478)

[2.1.5 クォーティング 8](#_Toc381605479)

[2.1.6 コマンド展開 9](#_Toc381605480)

[2.1.7 構文 10](#_Toc381605481)

[2.1.8 コマンド行編集 10](#_Toc381605482)

[2.1.9 組み込みコマンド 12](#_Toc381605483)

[2.1.10 シェルスクリプト 13](#_Toc381605484)

[2.2 基本的なコマンド 14](#_Toc381605485)

[2.2.1 マニュアル 14](#_Toc381605486)

[2.2.2 ファイル操作 15](#_Toc381605487)

[2.2.3 プロセス管理 19](#_Toc381605488)

[2.2.4 システム 20](#_Toc381605489)

[3. エディタ 22](#_Toc381605490)

[3.1 エディタとは 22](#_Toc381605491)

[3.2 vi 23](#_Toc381605492)

[3.3 Emacs 25](#_Toc381605493)

[4. ファイル転送 27](#_Toc381605494)

[4.1 ファイル転送とは 27](#_Toc381605495)

[4.2 ftp 28](#_Toc381605496)

[4.3 sftp 28](#_Toc381605497)

[4.4 scp 29](#_Toc381605498)

[5. リモートログイン 31](#_Toc381605499)

[5.1 リモートログインとは 31](#_Toc381605500)

[5.2 telnet 32](#_Toc381605501)

[5.3 ssh 32](#_Toc381605502)

[6. 参考文献 35](#_Toc381605503)

[7. 変更履歴 36](#_Toc381605504)

# Linux

## Linuxとは

「Linux」とは、フィンランドのヘルシンキ大学生であったLinus Torvaldsが、1991年に開発を開始したUNIX互換のOSです。LinusとUNIXをあわせてLinuxというわけです。この「Linux」は、「リナックス」や「リヌクス」、「ライナックス」などと発音されます。

当時、Linus Torvaldsは、UNIX風の教育用OSとして開発されたMinixを使っており、変更や機能追加を行おうとしていました。しかし、それにはいくつかの制限があったため、UNIXと互換性を持ち、自由に改変できるフリーのOSを目指して、Linuxの開発を始めました。

Linuxは、インターネット上に公開して以来、多くの協力者を得て、より多くの機能を実現し、また多くのCPUに移植されてきました。なぜLinuxがこれほど注目を浴び、利用されるようになったのか、その理由には大きく次の2つの要素があります。

* UNIX互換

UNIXと高い互換性があったため、広く使われていたUNIXのユーザが、容易にLinuxへ移行できました。

* オープンソース

Linuxは、オープンソースとして公開されたため、フリーで利用できました。GNUプロジェクトのソフトウェアのようなフリーソフトウェアと、Linuxを組み合わせることで、フリーの完全なOSを、多くの人々が提供できるようになったのです。ただし、フリーといっても著作権まで全てを放棄したわけではありません。使う場合にはGPL、LGPLといった、ライセンス形態に従う必要があります。

この結果、Linuxは、企業のサーバ、特にインターネット関連のサーバとして普及が始まり、やがて基幹業務を担うOSとして、またデスクトップOSとしても、使われるようになったのです。

* UNIX、Linuxの略年表

1969 AT&T Bell Lab.でUNIXが誕生

1979 1BSD(1 Berkley Software Distribution)

1981 4.1BSD

1982 AT&T UNIX System III

1983 AT&T UNIX System V / 4.2BSD

1990 AT&T UNIX System V R4

1991 Linuxの開発開始

1995 PC Linuxの普及が始まる

# シェル

## シェルとは

　「シェル」とは、ユーザがLinuxを利用するときのフロントエンドです。ユーザが与えたコマンドを解析し、実行を制御するインタプリタです。貝殻(shell)のようにOSをつつみ、ユーザの指示を一手に引き受けることから、シェルと呼ばれています。

　多くのシェルがありますが、広く使われているものには、次のようなものがあります。

* sh
* bash
* csh
* tcsh

　sh(Bourne Shell)は最初のUNIXからあったシェルで、これに強力なコマンド行編集機能を付加したものが、bash(Bourne Again Shell)です。cshはBSD UNIXの標準シェルとして開発され、C言語風の構文が使えるようになっています。tcsh(Tenix like csh)はcshのコマンド行編集機能を強化したものです。

　Linuxでは、ユーザは好きなシェルを使うことができます。これらのシェルの基本的な機能は同じですが、その詳細は異なります。

　ここでは、bashを例にシェルを使ったコマンド実行のルールを説明します。

### 標準入出力

　Linuxのコマンドには、標準入力、標準出力、標準エラー出力といった入出力があります。これらの3つの入出力は、プロセスが生成された時点で自動的に作られ、通常その端末に割当てられています(下図参照)。

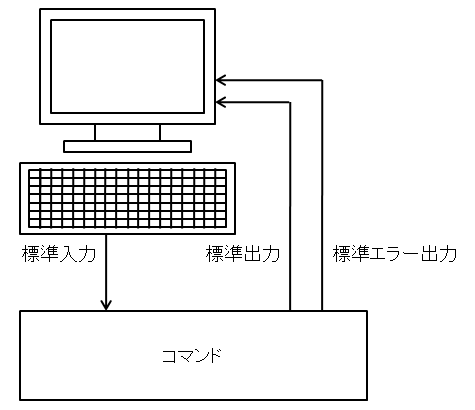


図 1標準入出力

たとえば、

|  |
| --- |
| $ ls mydir/  bar.txt baz.txt foo.txt |

は、lsコマンドの結果(mydirディレクトリの情報)を端末へ出力します。

また、

|  |
| --- |
| $ ls mydir/ > dlist  $ cat dlist  bar.txt  baz.txt  foo.txt |

は、lsコマンドの結果をdlistというファイルへ出力します。これは、「>」(大なり)に従って標準出力先がdlistに割り当てられているためです。このように、標準入出力の割り当てを変更することを、リダイレクションと呼びます(下図参照)。

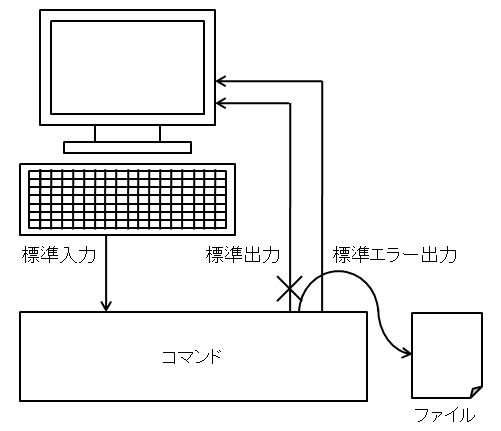


図 2リダイレクション

### パイプ

　パイプは、コマンドの標準出力を次のコマンドの標準入力とする役目をします(下図参照)。コマンドをパイプで繋げることで、単機能のコマンドを使って複雑な処理を実行できます。

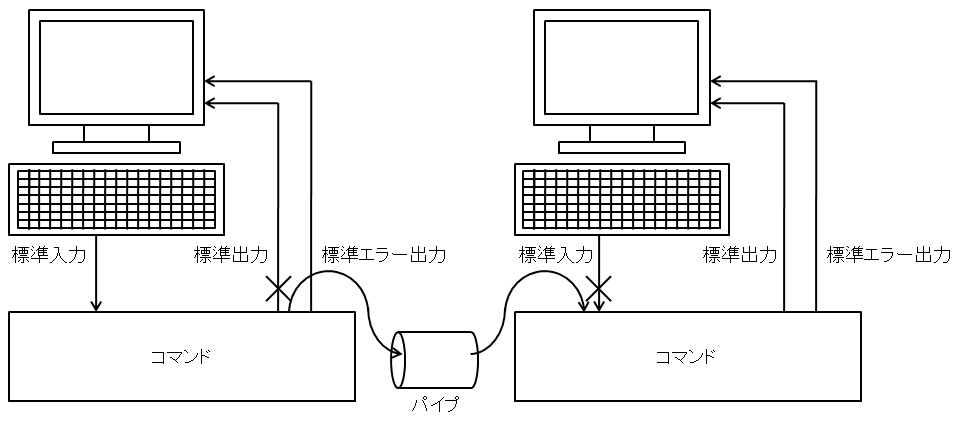


図 3パイプ

　たとえば、

|  |
| --- |
| $ ls mydir/ | wc –l  3 |

　は、lsコマンドの標準出力を、パイプを通してwcコマンドの標準入力へ渡します。wc -lはlsの結果から行数をカウントします。すなわち、出力結果はmydirディレクトリにあるファイルの数となります。

### 変数

　変数は、数値や文字列のような値を代入し記憶するものです。また、変数に付けた名前を使って、値を参照することができます。

　変数名に使用できる文字は、英数字と「\_」(アンダースコア)です。ただし、変数名の先頭に数字は使えません。

　変数に値を代入するには、「=」(イコール)を使います。このとき、=の前後に空白を入れることはできません。

|  |
| --- |
| $ VAR=123 |

　変数名を使って値を参照するには、「$」(ドル)を使います。

|  |
| --- |
| $ echo $VAR  123 |

　また、変数名を誤って解釈しないように、「{}」(波括弧)を使って範囲を指定できます。

|  |
| --- |
| $ echo $VAR456 # 変数名をVAR456と誤って解釈  $ echo ${VAR}456 # 変数名をVARと解釈  123456 |

　変数には、「シェル変数」と「環境変数」があります。シェル変数は、シェルが独自に保持しているもので、環境変数はそれぞれのプロセスが保持しています。これらは有効範囲が異なっており、シェル変数は、現在実行中のシェルでのみ有効です。環境変数は、現在実行中のシェルだけでなく、シェルから実行したプロセスにおいても有効です。

　bashでは、シェル変数に対してexportコマンドを実行することで、環境変数を設定できます。

|  |
| --- |
| $ VAR=123  $ export VAR |

　また、以下のように、値の代入と環境変数の設定を、1つにまとめて実行することもできます。

|  |
| --- |
| $ export VAR=123 |

　変数には、予約された機能を持った特殊なものがあります。以下に主なものを挙げます。

* $0   
  シェルのプログラム名
* $1～$9  
  引数の値(数値は引数の位置)
* $#  
  現在のプロセスに与えられた引数の数
* $@  
  現在のプロセスに与えられた引数  
  (「"」(ダブルクォート)で囲った場合、それぞれの値を正確に表すように展開)
* $\*  
  現在のプロセスに与えられた引数  
  (「"」(ダブルクォート)で囲った場合、連結されて1つの変数に展開)
* $-  
  シェルの起動時に指定したオプション
* $$  
  シェルのプロセスID
* $!  
  最後にバックグラウンドで実行されたコマンドのプロセスID
* $?  
  直前のコマンドの終了ステータス

### メタキャラクタ

　メタキャラクタは、ファイル名の展開に利用される特殊文字で、複数のファイルを効率よく指定することができます。

* \*

「\*」(アスタリスク)は任意個数の文字列(0個以上)を意味します。

* ?

「?」(クエスチョン)は任意の1文字を意味します。

* [ ]

「[ ]」(角括弧)内のどれかの1文字を意味します。

たとえば、[abc]は、abcのどれかの1文字を意味します。[1-5]は、[12345]と同じ意味になります。

　以下に、例を挙げます。

|  |
| --- |
| $ ls \*.c  test1.c test2.c test3.c test4.c test.c |

最後が.cで終わる全てのファイルを表示します。

|  |
| --- |
| $ ls test?.c  test1.c test2.c test3.c test4.c |

testで始まり任意の1文字があり.cで終わる全てのファイルを表示します。

|  |
| --- |
| $ ls test[1-3].c  test1.c test2.c test3.c |

test1.cからtest3.cまでのファイルを表示します。

### クォーティング

　クォーティングは、メタキャラクタのような特殊文字の意味を取り消して、通常文字として扱うように指示します。文字列検索などで、文字パターンを指定するときなどによく使います。

* "

「"」(ダブルクォート)で囲まれた文字のうち、「$」(ドル)と「`」(バッククォート)だけを特殊文字と見なします。

* '

「'」(シングルクォート)で囲まれた文字全てを通常の文字と見なします。

* \

「\」(バックスラッシュ)直後の文字を通常の文字と見なします。

　以下に、例を挙げます。

|  |
| --- |
| $ VAR=123  $ echo "$VAR"456  123456 |

変数VARの内容が表示されます。

|  |
| --- |
| $ echo '$VAR'456  $VAR456 |

文字列$VARが表示されます。

|  |
| --- |
| $ echo "\$VAR"456  $VAR456 |

文字列$VARが表示されます。

### コマンド展開

　コマンド展開は、指定した文字列をコマンドと見なし、その実行結果へ置き換えるように指示します。コマンド展開を指定するには、

|  |
| --- |
| $ echo current directory=`pwd`  current directory=/home/tux  $ echo current directory=$(pwd)  current directory=/home/tux |

　のようにして、実行結果へ置き換えたいコマンドを、「`」(バッククォート)か「$()」で囲みます。

　また、これらは以下のように入れ子にすることもできます。

|  |
| --- |
| $ curdir=`echo Current directory is \`pwd\``  $ echo $curdir  current directory=/home/tux  $ curdir=$(echo Current directory is $(pwd))  $ echo $curdir  current directory=/home/tux |

### 構文

　以下に、基本的な構文のいくつかを紹介します。

* cmd &

cmdをバックグラウンドで実行します。

* cmd1 ; cmd2

cmd1の実行に続けてcmd2を実行します。

* cmd1 | cmd2

cmd1の標準出力をcmd2の標準入力へ渡します。

* cmd1 && cmd2

cmd1の実行が正常終了した場合、cmd2を実行します。

* cmd1 || cmd2

cmd1の実行が正常終了しなかった場合、cmd2を実行します。

* cmd > file

cmdの標準出力をfileへ出力します。

* cmd >> file

cmdの標準出力をfileへ追加します。

* cmd < file

cmdの標準入力をfileから入力します。

### コマンド行編集

　bashやtcshなどのシェルでは、コマンド入力行を編集することができます。bashのコマンド行編集機能には、viモードとemacsモードがあり、デフォルトはemacsモードに設定されています。

　以下に、bashのemacsモードの基本的なキー操作を挙げます。

* Ctrl-b

カーソルを左に1文字移動します。

* Ctrl-f

カーソルを右に1文字移動します。

* Ctrl-a

カーソルを行の先頭に移動します。

* Ctrl-e

カーソルを行の末尾に移動します。

* DEL

カーソルの左1文字を削除します。

* Ctrl-d

カーソル位置の1文字を削除します。

* TAB

文脈に応じて補完します。

* Ctrl-p

1つ前のコマンドに移動します。

* Ctrl-n

1つ後のコマンドに移動します。

* !!

直前のコマンドに置き換えます。

* !*n*

履歴番号*n*のコマンドに置き換えます。

* !*string*

文字列*string*で始まる最近のコマンドに置き換えます。

* !$

直前のコマンドの最後の引数と同じものに置き換えます。

* !!*string*

直前のコマンドの最後に文字列*string*を追加したものに置き換えます。

### 組み込みコマンド

　コマンドには、「組み込みコマンド」と「外部コマンド」の2種類があります。

組み込みコマンドは、シェルの内部に用意されているコマンドです。主にシェル自身の機能の制御や、シェルでないと実現できないコマンドです。外部コマンドは、/usr/binディレクトリなどにあります。シェルが作成した別のプロセス上で、コマンドが実行されます。

以下に、組み込みコマンドのいくつかを紹介します。

* help [name]

組み込みコマンドに関するヘルプを表示します。nameには、詳細に表示したい機能名(コマンド名など)が指定できます。

* type name

コマンドの種類に関する情報を表示します。nameには、詳細に表示したい機能名(コマンド名など)が指定できます。

* echo [option] [string]

文字列stringを標準出力に書き出し改行します。

* pwd

現在のワーキングディレクトリを表示します。

* cd [dir]

dirディレクトリをワーキングディレクトリに設定します。dirを指定しない場合、ホームディレクトリを設定します。

* pushd [dir]

ディレクトリスタックの先頭にdirディレクトリを保存し、それをワーキングディレクトリに設定します。dirを指定しない場合、ディレクトリスタックの先頭と2番目を入れ換えます。

* popd

ディレクトリスタックの先頭のディレクトリを削除し、それをワーキングディレクトリに設定します。

* dirs

ディレクトリスタックの内容を表示します。

* alias [name[='cmd']]

cmdコマンドにnameという別名を定義します。引数を指定しない場合、別名の内容を表示します。

* unalias name

別名nameを削除します。

* history [lines]

コマンドの履歴を番号付きで表示します。linesには、表示する行数が指定できます。

* jobs

現在実行中のジョブの状態を表示します。

* fg [jobID]

jobIDで指定されたジョブをフォアグラウンドで実行します。jobIDを指定しない場合、カレントジョブが指定されます。

* bg [jobID]

jobIDで指定されたジョブをバックグラウンドで実行します。jobIDを指定しない場合、カレントジョブが指定されます。

* kill [option] ID

IDで指定されたプロセスまたはジョブを終了します。

* exit

シェルを終了します。停止中のジョブが残っている場合、警告が出力されます。再度exitを入力することで終了できます。

　その他、forやifなど、ループや条件分岐のための組み込みコマンドがあります。それらは、複雑なシェルスクリプトを作成するときなどによく利用されます。

### シェルスクリプト

　シェルスクリプトは、複数のコマンドを並べたテキストファイルです。シェルは、シェルスクリプトを解釈し、一連のコマンドを順番に実行できます。これは、Windowsのバッチファイルに似ています。

　ユーザは、シェルスクリプトを利用することで、決まりきったコマンドを何回も入力する手間を省くことができます。また、コマンドを組み合わせて独自のコマンドを作ることができます。

　以下に、単純な例sleep10.shを挙げます。

|  |
| --- |
| #!/bin/sh  echo start  sleep 10 # 10秒遅延  echo end |

　これは、標準出力へ文字列startを出力し、10秒後に文字列endを出力します。

先頭の「#!/bin/sh」は、このシェルスクリプトを実行するために使われるコマンドが、/bin/shであることを明示しています。「#」(シャープ)に続くテキストはコメントとして扱われ、行末まで無視されます。

　シェルスクリプトを実行するには、

|  |
| --- |
| $ sh sleep10.sh  $ sh < sleep10.sh |

　のようにして、実行するシェルにシェルスクリプトを与えることができます。

　また、

|  |
| --- |
| $ ./sleep10.sh |

　のようにして、直接シェルスクリプトを実行することもできます。この場合、シェルスクリプトには実行権限が必要です。

## 基本的なコマンド

　ここでは、マニュアル、ファイル操作、プロセス管理、システムに関する、基本的なコマンドを紹介します。

### マニュアル

　ここで紹介するコマンドの詳しい内容については、Linuxに含まれているマニュアルを見てください。マニュアルを参照するにはmanコマンドを使います。たとえば、lsコマンドのマニュアルを見る場合、次のようにします。

|  |
| --- |
| $ man ls  LS(1) LS(1)  名前  ls, dir, vdir - ディレクトリの中身をリスト表示する  書式  ls [options] [file...]  POSIX オプション: [-CFRacdilqrtu1]  GNU オプション (簡略形式): [-1abcdfghiklmnopqrstuvxABCDFGHLNQRSUX]  [-w cols] [-T cols] [-I pattern] [--block-size=SIZE] [--classify]  [--color[={none,always,auto}]] [--file-type] [--full-time]  ...... |

　manコマンドは次のように使います。

|  |
| --- |
| $ man [section] command |

　指定したcommandについて、その詳細な説明が表示されます。sectionはマニュアルの分類で、1がコマンド、2がシステムコール、3がライブラリ関数などとなっています。セクション番号を指定した場合、そのセクションにあるマニュアルを表示します。これは、同じ名前のマニュアルが複数のセクションにある場合に指定します。

　また、

|  |
| --- |
| $ man -k keyword |

　とすると、マニュアルの中から指定したキーワードkeywordに関する情報が表示されます。

### ファイル操作

* ls [option] [name]

　nameで指定されたファイルやディレクトリの情報を表示します。nameを指定しなかった場合、カレントディレクトリのファイル一覧を表示します。

|  |
| --- |
| $ ls –la |

　-lオプションは、パーミッション、所有者、サイズなどを含む、長い書式で表示します。-aオプションは、通常表示しない「.」(ドット)から始まる名前の、ファイルやディレクトリも表示します。

* cat [option] [file]

　複数のファイルを連結して内容を出力します。fileで指定された1つ以上のファイルを読み込み、標準出力に出力します。fileを指定しなかった場合、標準入力から読み込みます。

|  |
| --- |
| $ cat –n test.c |

　-nオプションは、行の先頭に行番号を付けて出力します。

* head [option] [file]

　fileで指定されたファイルの先頭から数行(デフォルトは10行)を表示します。fileを指定しなかった場合、標準入力から読み込みます。

* tail [option] [file]

　fileで指定されたファイルの末尾から数行(デフォルトは10行)を表示します。fileを指定しなかった場合、標準入力から読み込みます。

* less [option] [file]

　テキストファイルの内容をページ単位で表示します。よく使うキーバインドには、以下のようなものがあります。

f 次の画面に移動

スペースキー fと同じ

b 前の画面に移動

j 次の行に移動

k 前の行に移動

= 現在位置の行番号を表示

/*ptn* 文字列*ptn*を検索

n 次へ検索

N 前へ検索

g 先頭行に移動

G 最終行に移動

*n*G *n*行目に移動

h ヘルプを表示

q 終了

* cp [option] file1 file2  
  cp [option] file directory

ファイルやディレクトリをコピーします。file1をfile2の名前でコピーします。あるいは、fileで指定された1つ以上のファイルをdirectoryへコピーします。

|  |
| --- |
| $ cp –r dir1 dir2 |

dir1ディレクトリの全てのファイルとディレクトリを、dir2ディレクトリへコピーします。-rオプションは、再帰的にコピーします。

* mv [option] src-file or src-directory dst-file or dst-directory

　ファイルやディレクトリの移動または名前の変更をします。

|  |
| --- |
| $ mv test1 test2 |

　test1をtest2の名前に変更します。

* rm [option] file

　ファイルやディレクトリを削除します。

|  |
| --- |
| $ rm –ir mydir |

　mydirディレクトリの全てのファイルとディレクトリを削除します。-iオプションは、ファイルを削除して良いか確認を行います。-rオプションは、再帰的に削除します。

* mkdir [option] directory

　ディレクトリを作成します。

|  |
| --- |
| $ mkdir –p mydir/sampledir |

　-pオプションは、途中のディレクトリ(mydir)がない場合であっても、それを同時に作成します。

* rmdir [option] directory

　ディレクトリを削除します。

* chown [option] [owner] [file]

　ファイルやディレクトリの所有者を変更します。

|  |
| --- |
| $ chown –R user mydir |

　mydirディレクトリの全てのファイルとディレクトリの所有者を、userに変更します。-Rオプションは、再帰的に所有者を変更します。

* chgrp [option] [group] [file]

　ファイルやディレクトリのグループを変更します。

|  |
| --- |
| $ chgrp –R net-g proj |

　projディレクトリの全てのファイルとディレクトリのグループを、net-gに変更します。-Rオプションは、再帰的にグループを変更します。

* chmod [option] [mode] [file]

　ファイルやディレクトリのパーミッションを変更します。

|  |
| --- |
| $ chmod a+rw test |

testファイルの所有者、グループ、その他に、読み出しと書き込みの権限を与えます。

* touch [option] file

　ファイルのアクセス時刻と修正時刻を変更します。

|  |
| --- |
| $ touch –m prog |

　progファイルの修正時刻のみを、現在の時刻に変更します。-mオプションは、修正時刻のみを変更します。

* find [option] [directory] [exp]

　ファイルやディレクトリを検索します。

|  |
| --- |
| $ find –name '\*linux\*' –print |

　カレントディレクトリの、全てのファイルを検索し、名前に文字列linuxを含むファイルを表示します。-nameオプションは、ファイル名のパターン('\*linux\*')を指定します。-printオプションは、ファイル名を標準出力に表示します。

* grep [option] [file]

　ファイルから文字列を検索します。

|  |
| --- |
| $ grep "include" \*.c |

　カレントディレクトリの、ファイル名が.cで終わる全てのファイルから、文字列includeが含まれている行を検索し、その行を表示します。

* df [option] [file]

　ファイルシステムの使用量と容量を表示します。

* du [option] [file or directory]

　ファイルやディレクトリの容量を表示します。

|  |
| --- |
| $ du –a sampledir |

　sampledirディレクトリの全てのファイルとディレクトリの容量を、表示します。-aオプションは、ファイルの容量も表示します。

* tar [option] [tarfile] [otherfile]

　tar形式ファイルの作成と展開を行います。

|  |
| --- |
| $ tar cvf data.tar . |

　カレントディレクトリにある全てのファイルとディレクトリを集め、1つのtar形式ファイル(data.tar)を作成します。

* wc

　ファイルの改行、単語、容量(単位: Byte)をカウントします。

|  |
| --- |
| $ wc –l test |

　testファイルの改行をカウントします。

* diff

　先頭から順番に2つのファイルを行毎に比較します。

|  |
| --- |
| $ diff foo.c bar.c |

　foo.cファイルとbar.cファイルを比較し、異なる行があれば表示します。

* cmp

　先頭から順番に2つのファイルをByte毎に比較します。

|  |
| --- |
| $ cmp foo bar |

　fooファイルとbarファイルを比較し、2つのファイルが同じかを判定します。

### プロセス管理

* ps [option]

　実行中のプロセスを表示します。

|  |
| --- |
| $ ps aux |

　aオプションは、自分以外のユーザのプロセスも表示します。uオプションは、ユーザ名などを表示します。xオプションは、端末を持たないプロセスも表示します。

* kill [option] pid

　プロセスにシグナルを送ります。

|  |
| --- |
| $ kill -9 123 |

　プロセス123にシグナル9(強制終了)を送ります。-はシグナルの種類を指定します。

* top [option]

　現在実行中のプロセスで、CPUを多く使用しているものを、順番に表示します。

* nice [option] [cmd]

　優先度を調整してコマンドを実行します。優先度は、-20(高優先)から19(低優先)まで調整できます。負の値はスーパユーザのみ指定できます。

|  |
| --- |
| $ nice -n 3 grep "test" table |

　grepコマンドの実行を、通常より優先度を下げて実行します。-nオプションは、優先度を調整する値を指定します。

* sleep amount [unit]

　指定した時間だけ遅延します。unitにはs(秒)、m(分)、h(時間)、d(日)があり、デフォルトはsです。シェルスクリプトなどで、コマンドの実行を一定時間遅延させたい場合などによく使われます。

|  |
| --- |
| $ sleep 10 |

　10秒間遅延します。

### システム

* uname [option]

　システム情報を表示します。-rオプションはOSのリリース番号を表示、-sオプションはOSの名称を表示、-aオプションはホスト名など全てを表示します。

* date [option]

　日付と時間を表示、設定します。

* free [option]

　メモリの使用状況を表示します。システムの物理メモリとスワップメモリのそれぞれに対して、使用量と空き容量を表示します。-sオプションは、指定した秒毎に表示します。

* dmesg [option]

　OSのログを表示、制御します。

* ifconfig [inteface]  
  ifconfig interface [addr-family] [parameter|addr]

　ネットワークデバイスの情報を表示、設定します。

* ping [option] host

　リモートホストがネットワークに接続され、応答することを確認できます。

|  |
| --- |
| $ ping 192.168.10.123 |

　リモートホスト192.168.10.123が応答するか確認します。

* su [option] [user]

　指定したuserユーザに切り替えます。userの指定がない場合、スーパユーザ(root)に切り替えます。-オプションは、直接ログインした場合と同じ環境に初期化します。

* sudo [option]

他のユーザとしてコマンドを実行します。

|  |
| --- |
| $ sudo vi /etc/fstab |

スーパユーザ(root)としてviコマンドを実行します。オプションの指定がない場合、スーパユーザとなります。

* passwd [user]

ユーザのパスワードを変更します。スーパユーザのみ、他のユーザのパスワードを変更できます。

* shutdown [option]

システムをシャットダウン・再起動します。-hオプションはシャットダウン、-rオプションはリブート、-sオプションはシングルユーザモードで再起動します。

**演習問題**

* cdコマンドを使用して、Linuxのディレクトリを探検してみましょう。
* lessコマンドを使用して、テキストファイル/etc/servicesを閲覧、検索してみましょう。
* findコマンドを使用して、/usr/binディレクトリから、ファイル名にlinuxを含むファイルを検索してみましょう。
* grepコマンドを使用して、/etc/yum.repos.dディレクトリから、文字列httpを含む行を検索してみましょう。
* topコマンドを使用して、どのようなプロセスがCPUを使用しているか、確認してみましょう。

# エディタ

## エディタとは

　「エディタ」とは、メモを取る、プログラムのソースコードを書く、OSの設定ファイルを編集するなど、ユーザがLinuxでテキストを編集するときに、よく使用するアプリケーションです。

　今日のほとんどのエディタは、スクリーンエディタと呼ばれるものです。編集しているテキストが、画面に表示されるエディタは、スクリーンエディタと呼ばれています。

　Linuxで、特に人気のあるエディタと言えば、以下の2つがあります。

* vi
* Emacs

　viは、UNIXで初めてのスクリーンエディタです。単純でコンパクトなため、標準でインストールされていることが多く、システム管理者にとっては必須のツールと言えます。Emacsは、多機能なエディタです。特に特徴的なのは、独自のLisp実装を使ってプログラミングすることで、エディタの機能を容易に拡張できる点です。

　ここでは、vi、Emacsの基本的な使い方を説明します。

## vi

　viは、次のようにして起動します。

|  |
| --- |
| $ vi [file] |

　ファイルの内容がない部分には、行の先頭に「~」(チルダ)が表示されます。最下行は、モードやコマンドなどが表示されます。

　viには、モードという概念があり、コマンドモード、挿入モード、exモードがあります。

　コマンドモードでは、コマンドを使って、カーソルの移動、テキストの修正などを行います。viを起動した直後は、コマンドモードです。ユーザは、現在どのモードに入っているか分からなくなった場合、Escapeキーを押すことでコマンドモードへ戻ることができます。

　以下に、よく使うコマンドを挙げます。

k カーソルを上に移動

j カーソルを下に移動

h カーソルを左に移動

l カーソルを右に移動

b カーソルを前の単語に移動

w カーソルを次の単語に移動

0 カーソルを行頭に移動

$ カーソルを行末に移動

Ctrl + b カーソルを前の画面に移動

Ctrl + f カーソルを次の画面に移動

gg カーソルをファイルの先頭行に移動

G カーソルをファイルの最終行に移動

*n*G カーソルをファイルの*n*行目に移動

% カーソルを対応する括弧に移動

x カーソル位置の1文字を削除

X カーソル位置の1文字左を削除

yw カーソル位置の単語をバッファにコピー

yy カーソル行をバッファにコピー

*n*yy カーソル行を含む*n*行をバッファにコピー

y% カーソル位置から対応する括弧までをバッファにコピー

dw カーソル位置の単語を削除しバッファにコピー

dd カーソル行を削除しバッファにコピー

*n*dd カーソル行を含む*n*行を削除しバッファにコピー

d% カーソル位置から対応する括弧までを削除しバッファにコピー

p バッファからペースト

(単語:カーソル位置の1文字右に 行:カーソル行の下に)

P バッファからペースト

(単語:カーソル位置に 行:カーソル行の上に)

u 直前の操作の取り消し (アンドゥ)

/*string* *string*で指定した文字列をカーソル位置からファイルの末尾へ

向かって検索 (n:次の文字列へ移動 N:前の文字列へ移動)

?*string* *string*で指定した文字列をカーソル位置からファイルの先頭へ

向かって検索 (n:次の文字列へ移動 N:前の文字列へ移動)

i カーソル位置から挿入モードを開始

I カーソル行の行頭から挿入モードを開始

a カーソル位置の1文字右から挿入モードを開始

A カーソル行の行末から挿入モードを開始

o カーソル行の下に1行追加しその行頭から挿入モードを開始

O カーソル行の上に1行追加しその行頭から挿入モードを開始

cw カーソル位置の単語を削除し挿入モードを開始

cc カーソル行を削除し挿入モードを開始

*n*cc カーソル行を含む*n*行を削除し挿入モードを開始

s カーソル位置の1文字を削除し挿入モードを開始

S ccと同じ

J カーソル行と次の行を連結

　挿入モードでは、テキストの挿入、修正を行います。テキストは何行でも挿入でき、Enterキーで改行できます。間違いはBackspaceキーで修正できます。挿入モードへは、iコマンドなどで入ることができます。

　exコマンドモードでは、コマンドを使って、ファイルの操作などを行います。exコマンドモードへは、「:」キーで入ることができます。

　以下に、よく使うコマンドを挙げます。

:%s/*ptn1*/*ptn2*/g ファイル中の全ての文字列*ptn1*を文字列*ptn2*に置換

:%s/*ptn1*/*ptn2*/gc ファイル中の全ての文字列*ptn1*を文字列*ptn2*に対話的に置換

:w 編集中のファイルに上書き保存

:w [file] fileで指定したファイル名で保存

:q viを終了 (テキストに変更があった場合は警告)

:q! viを強制終了 (警告なし)

:wq 編集中のファイルに上書き保存してviを終了

## Emacs

　Emacsは、次のようにして起動します。

|  |
| --- |
| $ emacs [file] |

　最下行から2行は、編集中のファイルの情報やEmacsからのメッセージなどが表示されます。

　Emacsのコマンドは、Ctrlキーを多く使用します。C-xコマンドは、Ctrlキーを押しながらxキーを押すことを意味します。また、M-wコマンドは、Altキーを押しながらwキーを押すことを意味します。

　以下に、よく使うコマンドを挙げます。

C-p カーソルを上に移動

C-n カーソルを下に移動

C-b カーソルを左に移動

C-f カーソルを右に移動

M-b カーソルを前の単語に移動

M-f カーソルを次の単語に移動

C-a カーソルを行頭に移動

C-e カーソルを行末に移動

M-v カーソルを前の画面に移動

C-v カーソルを次の画面に移動

C-< カーソルをファイルの先頭行に移動

C-> カーソルをファイルの最終行に移動

M-x goto-line カーソルをファイルの指定した行に移動

C-d カーソル位置の1文字を削除

Backspace カーソル位置の1文字左を削除

C-Space カーソル位置にマークを設定

M-w マークからカーソル位置の一文字左までの範囲をバッファにコピー

C-w マークからカーソル位置の一文字左までの範囲を削除し

バッファにコピー

M-Backspace カーソル位置の一文字左から前の単語を削除しバッファにコピー

M-d カーソル位置から単語を削除しバッファにコピー

M-0 C-k カーソル位置の一文字左から行頭までを削除しバッファにコピー

C-k カーソル位置から行末までを削除しバッファにコピー

C-y カーソル位置にバッファからペースト

C-g コマンドの実行を中止

C-/ 直前の操作の取り消し (アンドゥ)

C-s 指定した文字列をカーソル位置からファイルの末尾へ向かって検索

C-r 指定した文字列をカーソル位置からファイルの先頭へ向かって検索

C-x 2 画面を上下に2分割

C-x 3 画面を左右に2分割

C-x 0 カーソルのある画面を非表示

C-x 1 カーソルのある画面以外を非表示

C-x o カーソルを他の画面へ移動

C-x C-s ファイルに保存

C-x C-c Emacsを終了

**演習問題**

* viを使用して、Hello, CentOS!と追記し、テキストファイルhello.txtといった名前で保存してみましょう。
* vimtutorを実行して、viの使い方を練習してみましょう。

# ファイル転送

## ファイル転送とは

　「ファイル転送」とは、ネットワークに接続されたマシン間で、ファイルを転送することです。たとえば、ソフトウェアを配布しているファイルサーバから、手元のマシンにファイルを転送するときなどに利用します。

　インターネットの初期から使われているファイル転送には、FTP(File Transfer Protocol)があります。FTPでファイル転送するには、ftpコマンドを使います。

　FTPは、通信を暗号化しません。これは、第三者から通信しているデータ、ログイン名やパスワードさえも盗み見される危険があり、セキュリティの観点から安全ではありません。このため、今日ではsftpコマンド、scpコマンドのような、通信が暗号化されたファイル転送がよく利用されています。

　sftpは、高機能で、ftpコマンドのような使い方ができます。scpは、軽量で、cpコマンドのような使い方ができます。sftp、scpは、どちらもSSH(Secure Shell)を使って、通信を暗号化しています。

　ここでは、ftp、sftp、scpの基本的な使い方を説明します。

## ftp

　ftpは、次のようにして起動します。

|  |
| --- |
| $ ftp [host] |

　hostには、リモートホストのホスト名かIPアドレスを指定します。

　たとえば、

|  |
| --- |
| $ ftp pc01 |

　は、リモートホストpc01に接続します。

リモートホストに接続すると、ログイン名とパスワードを尋ねられます。ログインに成功した場合、ftpのプロンプトが表示され、コマンドが実行できるようになります。

　以下に、よく使うコマンドを挙げます。

help [command] ヘルプを表示

pwd リモートホストの現在のワーキングディレクトリを表示

dir [remote-directory] リモートホストのディレクトリのファイル一覧を表示

cd remote-directory リモートホストのディレクトリを移動

lcd local-directory ローカルホストのディレクトリを移動

type [type-name] ファイル転送のタイプをtype-nameに設定

(type-nameを指定しない場合は現在のタイプを表示)

ascii ファイル転送のタイプをASCIIに変更 (デフォルト)

binary ファイル転送のタイプをバイナリに変更

get remote-file [local-file] リモートホストからローカルホストにファイルを転送

put local-file [remote-file] ローカルホストからリモートホストにファイルを転送

open host リモートホストと通信を開始

close リモートホストと通信を切断

quit ftpを終了

## sftp

　sftpは、次のようにして起動します。

|  |
| --- |
| $ sftp [user@]host |

　userには、ログイン名を指定します。省略した場合は、現在のユーザ名が使われます。hostには、リモートホストのホスト名かIPアドレスを指定します。

　たとえば、

|  |
| --- |
| $ sftp tux@pc01 |

　は、ユーザtuxとして、リモートホストpc01に接続します。

　リモートホストに接続すると、パスワードを尋ねられます。ログインに成功した場合、sftpのプロンプトが出力され、コマンドが実行できるようになります。

　以下に、よく使うコマンドを挙げます。

help ヘルプを表示

pwd リモートホストの現在のワーキングディレクトリを表示

lpwd ローカルホストの現在のワーキングディレクトリを表示

ls [remote-directory] リモートホストのディレクトリのファイル一覧を表示

lls [local-directory] ローカルホストのディレクトリのファイル一覧を表示

cd remote-directory リモートホストのディレクトリを移動

lcd local-directory ローカルホストのディレクトリを移動

get remote-file [local-file] リモートホストからローカルホストにファイルを転送

put local-file [remote-file] ローカルホストからリモートホストにファイルを転送

quit sftpを終了

## scp

　scpは、次のようにして起動します。

|  |
| --- |
| $ scp [[user@]host1:]file1 [[user@]host2:]file2 |

　userには、ログイン名を指定します。host*x*には、ホスト名かIPアドレスを指定します。file1は転送元のファイル名、file2は転送先のファイル名を指定します。[[user@]host*x*:]は、ローカルホストの場合は省略できます。

　たとえば、

|  |
| --- |
| $ scp TODO tux@pc01:TODO\_20140110 |

　は、ローカルホストのカレントディレクトリにあるTODOファイルを、リモートホストpc01のユーザtuxのホームディレクトリに、TODO\_20140110という名前で転送します。

　また、

|  |
| --- |
| $ scp -r tux@pc01:/etc etc\_pc01 |

　は、リモートホストpc01の/etcディレクトリにある全てのファイルを、再帰的にローカルホストの/etc\_pc01ディレクトリに転送します。-rオプションは、再帰的に転送します。

**演習問題**

* エディタの演習問題で作成したhello.txtを、db2.ertl.jpのホームディレクトリにコピーしてみましょう。
* db2.ertl.jpから、/home/courseware/LinuxCourseware\_text.pdfをローカルホストにコピーしてみましょう。

# リモートログイン

## リモートログインとは

　「リモートログイン」とは、ネットワークを介して遠隔地のマシンにログインすることです。たとえば、家のマシンから大学や会社のマシンにログインし、端末からシェルを実行してマシンを操作するときなどに利用します。

　単純なリモートログインの手段には、TELNET(Telecommunication Network)があります。TELNETでリモートログインするには、telnetコマンドを使います。

　TELNETは、FTPと同様に通信を暗号化しません。もしもログイン名やパスワードを盗み見された場合、リモートマシンを第三者に自由に操作され、大変危険です。このため、現在はリモートログインには、通信を暗号化するSSHがよく使われています。SSHでリモートログインするには、sshコマンドを使います。

　ここでは、telnet、sshの基本的な使い方を説明します。

## telnet

　telnetは、次のようにして起動します。

|  |
| --- |
| $ telnet [-l user] host |

　userには、ログイン名を指定します。hostには、リモートホストのホスト名かIPアドレスを指定します。

　たとえば、

|  |
| --- |
| $ telnet -l tux pc01 |

　は、ユーザtuxとして、リモートホストpc01に接続します。

　リモートホストに接続すると、パスワードを尋ねられます。userを省略した場合、ログイン名も尋ねられます。ログインに成功した場合、シェルのプロンプトが表示され、コマンドが実行できるようになります。

## ssh

　sshは、次のようにして起動します。

|  |
| --- |
| $ ssh [user@]host [command] |

　userには、ログイン名を指定します。省略した場合は、現在のユーザ名が使われます。hostには、リモートホストのホスト名かIPアドレスを指定します。commandはリモートホストで実行するシェルのコマンドです。

　たとえば、

|  |
| --- |
| $ ssh tux@pc01 |

　は、ユーザtuxとして、リモートホストpc01に接続します。

　リモートホストに接続すると、パスワードを尋ねられます。ログインに成功した場合、シェルのプロンプトが表示され、コマンドが実行できるようになります。

　また、sshは、直接シェルのコマンドを実行することもできます。

　たとえば、

|  |
| --- |
| $ ssh tux@pc01 uname –a |

　は、ユーザtuxとして、リモートホストpc01に接続し、uname –aを実行します。

　また、ssh-keygenを使えば、sshで公開鍵認証を行うための鍵を作ることができます。

ssh-keygenは、ユーザの秘密鍵と公開鍵を生成します。名前の通り、秘密鍵は、ローカルホストで流出しないように保管しなければなりません。公開鍵は、ログインするリモートホストへ登録します。

公開鍵認証は、ユーザが正しい秘密鍵を持たなければ、認証が成功しないため、パスワード認証よりも安全です。

　ssh-keygenは、次のように起動します。

|  |
| --- |
| $ ssh-keygen [-b bits] [-t type] |

　bitsには、生成する鍵のビット長を指定します。たとえばRSA鍵の場合は、最低で768Bit、デフォルトで2048Bitとなります。typeには、生成する鍵の種類を指定します。プロトコルバージョン1のRSA鍵はrsa1を指定します。プロトコルバージョン2のRSA鍵、DSA鍵は、それぞれrsa、dsaを指定します。

　たとえば、

|  |
| --- |
| $ ssh-keygen -b 2048 -t rsa  Generating public/private rsa key pair.  Enter file in which to save the key (/home/tux/.ssh/id\_rsa): Enterキーを押下  Enter passphrase (empty for no passphrase): パスフレーズを入力  Enter same passphrase again: パスフレースを再入力  Your identification has been saved in /home/tux/.ssh/id\_rsa.  Your public key has been saved in /home/tux/.ssh/id\_rsa.pub.  ... |

　は、ビット長2048のプロトコルバージョン2のRSA鍵を生成します。

　コマンド実行後、鍵の保管場所とパスフレーズを尋ねられます。ここでは、保管場所はデフォルト(ホームディレクトリの.sshディレクトリ)とします。パスフレーズとは、パスワードよりも長い文字列を指し、秘密鍵を使用するときに入力を求められます。

　リモートホストが公開鍵認証を許可している場合、以下の手順で公開鍵認証が利用可能となります。

それぞれの鍵を、ローカルホストとリモートホストの、決められたディレクトリに置きます。秘密鍵id\_rsaは、ローカルホストの~/.sshへ置くことで、sshコマンド実行時に自動的にその秘密鍵が使用されます。

リモートホストに~/.sshがない場合は、ディレクトリを作成します。

|  |
| --- |
| $ mkdir ~/.ssh  $ chmod 700 ~/.ssh |

リモートホストの設定によっては、~/.sshのパーミッションは「700」でないと、認証に失敗することがあります。

公開鍵id\_rsa.pubは、リモートホストの~/.ssh/authorized\_keyに追記します。

|  |
| --- |
| $ cat id\_rsa.pub >> ~/.ssh/authorized\_keys  $ chmod 600 ~/.ssh/authorized\_keys |

　このとき、上書きしないように注意しなければなりません。これは、既に登録済みの公開鍵が上書きされるためです。

　また、リモートホストの設定によっては、authorized\_keysのパーミッションは「600」または「400」でないと、認証に失敗することがあります。

**演習問題**

* db2.ertl.jpにログインし、slコマンドを実行して動作を確認してみましょう。
* db2.ertl.jpのhostnameコマンドを直接実行し、ローカルホストの結果と比較してみましょう。

# 参考文献

* Matt Welsh, Matthias Kalle Dalheimer, Terry Dawson, Lar Kaufman: 「Running Linux, 4th Edition」O’Reilly & Associates, Inc, 2002. 山崎 康宏, 山崎 邦子: 「Running Linux 第4版」オライリー・ジャパン, 2003.
* Arnold Robbins, Nelson H. F. Beebe: 「Classic Shell Scripting」O’Reilly Media, Inc, 2005. 日向 あおい: 「詳解 シェルスクリプト」オライリー・ジャパン, 2006.

# 変更履歴

|  |  |  |
| --- | --- | --- |
| **版数** | **変更内容** | **年月日** |
| 1.0 | 新規作成 | 2014/MAR/3 |