UNIX/Linux②基礎編
テキスト

2020 年 9 月版

明治大学
生田メディア支援事務室

はじめに

本テキストは、UNIX/Linux の操作の基礎を学ぶためのテキストです。本テキストで学ぶ内容は、コマンドによるファイル操作やテキスト編集、それに UNIX/Linux システム管理の基礎となる内容を含んでいます。UNIX/Linux 環境として、生田システムの「CentOS」（Linux）を利用することを前提としています。

各教室または生田情報メディア HP に『UNIX 利用の手引き』がありますので、必要に応じてそちらも参照してください。

https://www.meiji.ac.jp/isys/doc/UNIX2019.pdf

目次

図一覧

表一覧

1. UNIX/Linux①入門編の復習

本テキストでは、UNIX/Linux①入門編テキストで扱った以下の内容を前提にUNIX/Linuxの操作について説明します。そのため、本章ではUNIX/Linux①入門編で扱った内容について簡単に復習します。

1.1. UNIX/Linuxのファイルとディレクトリ

1.1.1. ファイルとディレクトリ

UNIX/Linuxは、プログラムや文章、画像など全てをファイルという単位で管理しています。また、WindowsやMacOSなどでフォルダとして扱われている概念は、UNIX/Linuxではディレクトリと呼ばれています。

1.1.2. ホームディレクトリとカレントディレクトリ

UNIX/Linux環境にログインした直後は、ある決まったディレクトリを参照している状態になります。この特別なディレクトリをホームディレクトリと呼び、ユーザ一人ひとりに専用のホームディレクトリが存在します。生田の情報処理教室の環境では、ホームディレクトリの名前はユーザ名(ログイン名)と同じです。

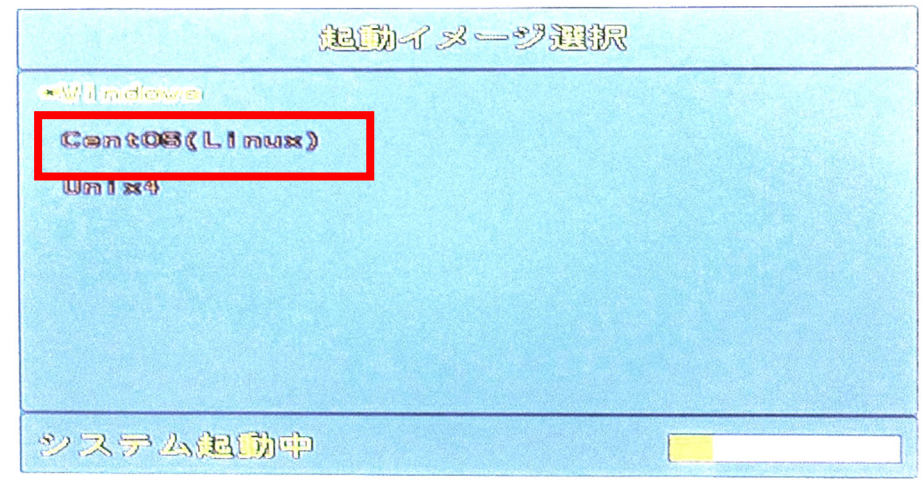
現在参照しているディレクトリのことをカレントディレクトリといいます。本テキストで特にコマンド操作をする場合は、カレントディレクトリがどこであるか意識するようにしてください。

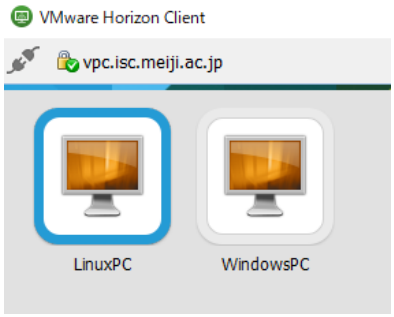
1.2. CentOSおよびテキストエディタ(メモ帳) geditの使い方

1.2.1. CentOSの起動

本テキストでは、CentOSを使ってUNIX/Linuxの使い方を説明します。CentOSを起動方法は、図 1　CentOSの起動 のようにPC起動時にWindows及びCentOSを選択する画面が表示されます。カーソルキーを用いてCentOSを選択してください。選択しない場合、Windowsが起動します。

Windowsを起動している状態からCentOSに移行したい場合は、Windowsから「再起動」した後、PC起動時と同様に進めてください。





仮想デスクトップ使用時は LinuxPC を選択

図 1　CentOS の起動

CentOS にログインするためにユーザ名とパスワードを入力します。ユーザ名とパスワードは Windows にログインするための基盤アカウントのユーザ名とパスワードと同一です。詳細については、UNIX/Linux①入門編の 11 ページを参照してください[1]。

[1] 以下、参照を次のように表現する。 UNIX/Linux①入門編テキストの 11 ページ「3.1. CentOS の起動」を参照　→　①p.11「3.1. CentOS の起動」

1.2.2. CentOS のテキストエディタの使い方

UNIX/Linux①入門編では、テキストエディタ gedit の使い方を紹介しました。「アプリケーション」メニューから「アクセサリ」、「テキストエディタ」を選択することで、gedit を起動することができます。

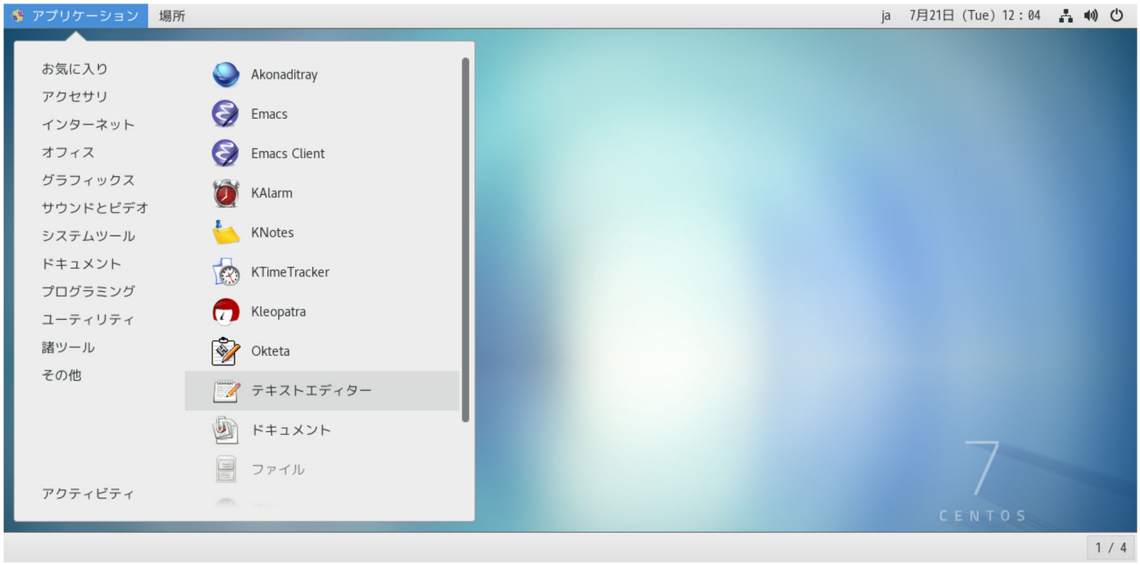


図 2　gedit の起動

このようなマウスを使った操作以外に、本テキストでは、コマンド操作で gedit を使用する方法や gedit 以外の主要なテキストエディタである Vim の使い方についても紹介します。

2. UNIX/Linux のコマンドについて

本章では、CentOS を用いて、UNIX/Linux におけるコマンドの入力方法、および代表的なコマンドについて説明します。シェルは基本的に sh 系のシェルである bash の使用を想定しています[2]。

2.1. コマンド操作の基礎

2.1.1. コマンドの基本形式

UNIX/Linux のコマンドは、基本的に以下のような形式になっています[3]。

$ コマンド名␣[オプション]␣[引数]

**$：**プロンプトと呼ばれます。UNIX/Linux システムがコマンドの入力を待っている状態です。一般に、csh 系のプロンプトでは%を、sh 系では$を、管理者では#を使います。本テキストでは、プロンプトを$で表しますが、適宜%や#に読み替えてください。

コマンド名：コマンド名の部分には実行したいコマンドの名前を入力します。コマンド名は省略することはできません。

オプション：オプションは、コマンドに対して細かい動作指示を与えるためのものです。利用できるオプションや指定方法は、コマンドによって違います。

引数（ひきすう）：引数は、コマンドに操作対象（ファイル名、ユーザ名、…など）を指定するためのものです。パラメータと呼ばれることもあります。

※オプションと引数は、[ ]で囲まれていますが、これは省略が可能であることを意味しています。ただし、コマンドによっては引数が必須のものもあります。

コマンドには以下の 3 つの入力ルールがあります。

1) コマンドはプロンプトの後に入力
2) コマンドは小文字で入力
3) オプションは、通常「-」(ハイフン)の後に 1 文字以上で入力

---

[2] UNIX/Linux①入門編テキストで紹介したように、シェルを C シェルから bash に変更するには、「端末」で bash と入力する

[3] 本テキストでは半角スペースを␣で表す

[例 1]

```
$ ls␣-a␣sample/
```

上記のコマンドの場合

- コマンド名は「ls」　：　ファイルやディレクトリの情報を表示するコマンド
- オプションは、「-a」　：　隠しファイルも表示するように指定
- 引数は、「sample/」　：　sample/ディレクトリ内の情報を表示するように指定

生田システムの UNIX/Linux 環境では、世の中の UNIX/Linux 環境のどこでも使える標準的なコマンドの他に、あらかじめインストールされているフリーソフト、教育研究用に購入したソフトなど、たくさんのコマンドが利用できます。ただし、そういったコマンドを利用したければ、そのコマンド名と使い方を一つひとつ確認する必要があります。

[例 2]

```
$ cat␣-n␣sample211.txt
```

上記のコマンド行は、次のように解釈されます。

- コマンド名は「cat」。ファイルや標準入力を標準出力に出力
- オプションは、「-n」。行番号を付加して出力
- 引数は、「*sample211.txt*」。標準出力へ出力するファイル名を指定

なお、複数のコマンドを「**;(セミコロン)**」で区切って、1 行で入力することもできます。この場合、一つ目のコマンドが処理された後、二つ目のコマンドが処理されます。

[例 3]

```
$ sleep␣5␣;␣echo␣"5 秒経過しました。"
```

上記のコマンド行は、二つのコマンド「sleep」と「echo」を順次実行します。
解釈は次の通りです。

- 一つ目のコマンド名は「sleep」。指定した時間だけシェルを停止
- オプションを選んでいません。引数は「5」です。(オプションを指定していないため、ここでは「5 秒」と解釈される)。「5m」や「5h」にすることで、遅延時間を「5 分」や「5 時間」に指定できます。
- 二つ目のコマンド名は「echo」。指定した文字列を出力
- オプションを選んでいません。引数は「**5 秒経過しました。**」[4]
- この例では、コマンド入力の 5 秒後に、 「5 秒経過しました。」と表示されます

[4] " "(ダブルクォーテーション)で囲んだ文字列は文字として扱われる

【補足 1】ショートカットキーと特殊文字の入力

➢ キー操作の表記について

今後本資料では、キーの操作について以下の表記をします。

表記	操作の内容
C-y	[Ctrl]キーを押しながら[y]キーを押す
C-x,u	[Ctrl]キーを押しながら[x]キーを押し、いったん両方のキーから指を離し、続けて[u]キーを押す
C-x,C-s	[Ctrl]キーを押しながら、[x]キーと[s]キーを順に押す
M-y	まず[Esc]キーを押して指を離し、次に[y]キーを押す

➢ 特殊な文字の入力について

UNIX の操作の中で特別な意味を持つ文字が幾つかあります。
その中には、キーボードのそれぞれのキーの上に印刷されている文字と、それを入力した時に画面の中に表示される文字とが一見すると同じ文字に見えないものがあります。

文字	説明
¥ と \	「エンマーク」と「バックスラッシュ」と呼ばれる文字。キーボードをよく見ると、最上段の右端の方に「¥」と書かれたキーがある。また、最下段の右端の方には「\」と書かれたキーがある。この 2 つはそれぞれ日本語環境、英語環境でのもので、表記は別々だが、実は同じ文字コードが割り当てられた同じ文字となっている。そのため、使っている環境や使用しているフォントなどによっては、[¥]キーを押しても画面には「\」が表示されることもあり、その逆もありえる。どちらの表示になっても UNIX の操作上は基本的に同じものなので、混乱しないように注意
~	「チルダ」と呼ばれる文字で、キーボード最上段、右端の[Back space]キーの 2 つ左にあるキーの上段に印刷されている。画面上は、「~」のような形で表示されることが多い。[Shift]キーを押しながらこのキーを押すと入力できる
|	「パイプ」と呼ばれる文字で、キーボード最上段、右端の[Back space]キーの 1 つ左にあるキーの上段に印刷されている。画面上は「¦」のように真ん中が少し切れているように表示されることもあるが、キーボード上の文字は縦棒 1 本になっているので注意

2.1.2. 代表的なコマンド

UNIX/Linux で使用できるコマンドは数多くあります。表 1 に代表的なコマンドを列挙します。この中のいくつかのコマンドについては、本テキストで詳しく解説します。

また、コマンド入力の途中で Tab キーを押すことで、コマンドの入力が補完できます。

表 1　代表的なコマンド

コマンド	機能
ls	ディレクトリの内容を一覧表示する
cat, more, less	ファイルの内容を表示する
find	ファイルやディレクトリを検索する
rm	ファイル、ディレクトリを削除する
mkdir	ディレクトリを作成する
rmdir	ディレクトリを削除する
cp	ファイル、ディレクトリをコピーする
mv	ファイル、ディレクトリを移動する、名前を変更する
cd	ディレクトリを移動する
pwd	現在のディレクトリ(カレントディレクトリ)を表示する
touch	新規に空のファイルを作成する ファイルのタイムスタンプ[5]を変更する
grep	ファイルの中から目的の文字パターンを持つ行を抜き出す
lpr	印刷する
man	オンラインマニュアルを表示する
echo	指定した文字列を表示する
clear	コンソールに表示されている文字を消去する
exit	シェルを終了する

[5] ファイルの作成日時や変更日時などの情報のこと

2.1.3. Ctrl(Control)キーの使用

コマンド操作の際には、[Ctrl]キーを用いることで、プログラムの挙動を制御できます。 表 2 に[Ctrl]キーを用いた入力による操作を列挙します。

表 2 Ctrl(Control)キーを伴う操作

入力	操作
C-c	プログラムや入力に割り込み、実行を中断
C-z	プログラムの実行を一時中断[6]
C-a	カーソルを行頭へ移動
C-e	カーソルを行末へ移動
C-u	カーソルよりも前の文字をすべて消去(行頭まで削除)
C-k	カーソルよりも後ろの文字をすべて消去(行末まで削除)
C-s	ファイルやプログラムの表示を一時的に中断(画面をロック)
C-q	C-s で一時停止した表示を再開(画面のロックを解除)
C-r	コマンド履歴を検索(詳細は 11 ページコマンド履歴の確認参照)

2.1.4. コマンド履歴

(1) コマンド履歴の確認

コマンド操作においては、同じコマンドをオプションや引数を変えて度々使用することがあります。その際には、コマンドを毎回最初から打ち直すことは非効率です。より効率的にコマンド操作を行うために、コマンド履歴を活用しましょう。プロンプトが表示されている状態で、「↑」キーまたは「C-p([Ctrl] + [p]キー)」を押すことで、コマンド履歴を表示させることができます。

例)

↑キー(C-p)を押す

```
$ ls␣-a␣sample/
```

直前に実行したコマンドが表示されます。

さらに↑キー(C-p)を押す

```
$ ls
```

さらに 1 つ前に実行したコマンドが表示されます。

(2) コマンド履歴の検索

コマンド履歴を検索することもできます。プロンプトが表示されている状態で、C-r([Ctrl] + [r]キー)とすることで以下のような表示に切り替わります。

[6] C-z で中断したプログラムは fg コマンドで再開できる

```
(reverse-i-search)`': 
```

この状態で検索したい文字列を入力すると、文字列を含むコマンド履歴が表示されます。

```
(reverse-i-search)`l': ls␣-a␣sample/
```

上記では l を入力しています。この状態で[Enter]キーを押すことでコマンド履歴に表示されたコマンドを実行することができます。

さらに C-r とすることで、古い履歴にさかのぼって文字列を含むコマンド履歴を表示することができます。

```
(reverse-i-search)`l': ls
```

コマンド履歴の検索を終了する場合には、[Esc]キーを押します。

(3) コマンド履歴の一覧表示

また、コマンド履歴の一覧を表示させることができます。履歴の一覧表示には history コマンドを使用します。

history␣[オプション]␣[表示するコマンドの数]

表 3 history コマンドのオプション

history コマンドの主なオプション	説明
-c	コマンド履歴を消去
-d␣番号	指定した番号のコマンド履歴を消去
-r	履歴ファイルに記載されているコマンド履歴を「端末」に読み込む
-w	現在「端末」に記録されているコマンドを履歴ファイルに書き込む

例)

```
$ history␣5
```

上記コマンドでは直近に実行した 5 つのコマンド履歴を表示します。オプション無しで実行すると番号付きのコマンド履歴が表示されます。

コマンド履歴からのコマンド実行

コマンド履歴から実行したいコマンドが見つかった場合は、以下の操作で実行できます。

!番号

例)

```
$ !32
```

2.2. コマンド操作の例

以下、コマンド操作の例を示します。

2.2.1. ファイルやディレクトリ操作の基本

1) ホームディレクトリに移動する

```
$ cd
```

2) test と書かれたテキストファイル *test221.txt* を作成する

```
$ gedit␣test221.txt
```

gedit テキストエディタが起動するので、中身に test と書き、保存して終了する

3) ファイルの内容を見る

```
$ cat␣test221.txt
```

4) ディレクトリを作成する

```
$ mkdir␣testdir
```

5) testdir ディレクトリが作成されたかどうか確認する

```
$ ls
```

6) ファイルを作成したディレクトリ内へ移動する

```
$ mv␣test221.txt␣testdir/
```

7) ディレクトリを移動する

```
$ cd␣testdir
```

8) ファイルが移動しているか確認する

```
$ ls
```

9) ファイルを削除する

```
$ rm␣test221.txt
```

10) ファイルが削除されたことを確認する

```
$ ls
```

11) ホームディレクトリに移動する

```
$ cd
```

12) 現在のディレクトリの場所を確認する

```
$ pwd
```

13) 作成したディレクトリを削除する

```
$ rm␣−r␣testdir
```

14) ディレクトリが削除されたことを確認する

```
$ ls
```

2.2.2. ワイルドカードを使ったファイルやディレクトリの操作

特定の文字を表すワイルドカードを使用することで、ファイルやディレクトリ検索などでコマンドの利便性が向上します。

表 4　ワイルドカード

主なワイルドカード	意味
*	0 文字以上の任意の長さの文字列
?	任意の 1 文字
[abc]	a，b，c のどれか 1 文字
[a-z]	a から z までのどれか 1 文字
[!0-9]	0 から 9 以外の文字（数字以外という意味）

例）

ホームディレクトリにサンプルファイル *test2.txt* および *today.txt* を作成。

カレントディレクトリ内に t ではじまるファイルをリスト表示する。

```
$ ls␣t*
```

実行結果

```
test2.txt␣today.txt
```

カレントディレクトリ内に test の後に、1,2,3 のいずれかが含まれているファイルを出力する。

```
$ ls␣test[1-3].txt
```

実行結果

```
test2.txt
```

2.2.3. ファイル内の文字列を検索する

(1) grep コマンド

grep コマンドを使用することで、特定のファイル内の指定されたパターンを検索、条件にマッチした行を出力することができます。検索パターンには、通常の文字列に加え、正規表現を使用することができます。grep コマンドの使い方は以下の通りです。

grep␣[オプション]␣文字パターン␣[ファイル名 1]␣[ファイル名 2]・・・・[7]

[7] ファイル名は、スペース区切りで複数指定することができる

表 5　grep のオプション

grep の主なオプション	説明
-n	検索してマッチした行を出力する際に、行頭に行番号を付けて出力する
-i	検索パターンで大文字と小文字を区別しない
-c	マッチした行数を出力する
-l	マッチした行があるファイル名のみ出力する
-v	マッチしなかった行を出力する
-N	マッチした行の前後 N(数値)行を合わせて出力する
-A␣N	マッチした行の後 N(数値)行を合わせて出力する
-B␣N	マッチした行の前 N(数値)行を合わせて出力する
-F	検索パターンを正規表現ではなく固定文字列として扱う

例)

test223-1.txt という名前のファイルを作成する。

```
$ gedit␣test223-1.txt
```

test223-1.txt

```
d
aadaa
aacaa
ddddd
aaDaa
```

test223-1.txt ファイル内の文字「d」を含む行を出力する。

```
$ grep␣d␣test223-1.txt
```

実行結果

```
d
aadaa
ddddd
```

test223-1.txt ファイル内の文字「d」および「D」を含む行を出力する。

```
$ grep␣-i␣d␣test223-1.txt
```

実行結果

```
d
aadaa
ddddd
aaDaa
```

(2) 正規表現

正規表現は、文字列パターンを、特定の文字と記号を使って表す表記法（表現）です。grep コマンドでは検索パターンを記述する際に、正規表現を使うことができます[8]。

表 6　正規表現

主な正規表現	意味
.	任意の 1 文字
*	直前の 1 文字または 1 パターンが 0 回以上出現する
[abc]	a または b または c のどれか 1 文字
[a-z]	a から z までのどれか 1 文字

例)

test223-2. txt という名前のファイルを作成する。

```
$ gedit␣test223-2.txt
```

test223-2. txt

```
te
tes
test
testt
```

test223-2. txt ファイル内の"tes"のあとに、*直前の"t"が 0 回以上出現する行を出力。

```
$ grep␣"test*"␣test223-2.txt
```

" "(ダブルクォーテーション)は引用符と呼ばれ、囲んだ文字列を文字として扱います。検索文字列に正規表現を使用する場合は、検索パターンを引用符で囲むことが推奨されています。grep で文字列を検索する場合*(アスタリスク)は正規表現のため、「tes」、「test」、「testt」全てが検索対象となります。

実行結果

```
tes
test
testt
```

正規表現を組み合わせるなど複雑な正規表現を使用する場合は、テストを繰り返して期待通り動作することを確認することが重要です。

[8] ファイルやディレクトリ操作の際に使用したワイルドカード(p. 14)との違いに注意

2.2.4. パイプによりコマンドを組み合わせる[9]

UNIX/Linux①入門編テキストで説明したリダイレクト機能は、プログラムの入出力先とファイルを結びつけるものでしたが、パイプ機能はプログラム同士を結びつける機能です。つまり、あるプログラムの標準出力を、そのまま別のプログラムの標準入力に結びつけます。パイプ機能を使って、コマンド 1 の出力を、コマンド 2 の入力として渡したい時には次のようにします。

$ コマンド 1␣|␣コマンド 2

例えば、ファイルの中身を cat コマンドで出力し、行数や単語数を表示する wc コマンドでその出力の行数を表示する処理は、以下のようになります。

test223-2. txt ファイルの中身を出力。

```
$ cat␣test223-2.txt
```

実行結果

```
te
tes
test
test
```

test223-2. txt ファイルの行数を出力。

```
$ cat␣test223-2.txt␣|␣wc␣-l
```

実行結果

```
4
```

cat コマンドが *test223-2. txt* の中身を出力し、そのまま出力を wc コマンドの入力に結び付けています。そのため、*test223-2. txt* の中身は表示されず、ファイル内に記載されているテキストの行数を wc コマンドが出力[10]しています。

パイプは 1 つだけでなく、次のように複数つなげることもできます。

$ コマンド 1␣|␣コマンド 2␣|␣コマンド 3

---

[9] p. 7 の ;（セミコロン）との違いについては、p. 18 の補足 2 を参照

[10] wc コマンドについては UNIX 利用の手引き p. 121 参照。「-l」は改行数だけを出力させる wc コマンドのオプション

【補足 2】コマンドラインでの複数コマンドの実行

コマンドラインにおいて複数のコマンドを組み合わせる方法は以下の通り複数あります。それぞれ処理が異なるため使用する際には注意が必要です。

表記	操作の内容
前 ； 後	前の実行結果に関わらず前が終了後、後を実行
前 & 後	前を実行しつつ後も実行。詳細は①p. 37「付録. プロセス、ジョブ、バックグラウンド処理について」を参照
前 && 後	前が正常終了(※)した場合には、後を実行する
前 ｜ 後	前の結果を後に渡して実行。詳細はp. 16 参照
前 ｜｜ 後	前が異常終了(※)した場合には、後を実行する

例)

&&(アンパサントアンパサント)について

$ cat␣test.txt␣&&␣echo␣"ファイルがあります、内容を要確認！"

test.txt ファイルが存在した場合、「ファイルがあります、内容を要確認！」と表示されます。ファイルが存在しない場合は、標準エラーが表示されます。

||(パイプパイプ)について

cd

ホームディレクトリに移動します。

$ cat␣sample2017␣||␣gedit␣sample2017

ホームディレクトリ直下に、sample2017 という名前のファイルが無い場合は、ホームディレクトリに sample2017 を作成します。sample2017 ファイルが存在しない状態で実際に上記のコマンドを実行すると、(gedit で新規ファイル sample2017 が作成します。gedit 画面で「保存」もしくは「保存せずに閉じる」を選択すると)以下の出力を確認できます。

cat: sample2017: そのようなファイルやディレクトリはありません

また、sample2017 ファイルを作成後、パイプパイプを用いた上記コマンドを実行すると、ファイルの中身が表示され、正常に cat コマンドが実行できた場合は、gedit が立ち上がらない(gedit コマンドが実行されない)ことを確認できます。

※コマンド終了時には、終了ステータスと呼ばれる情報が返されます。終了ステータスから、コマンドが正常に実行されたか(正常終了)、失敗したか(異常終了)を判断することができます。

2.3. コマンドの使い方が分からない時

コマンドの基本的な使用方法を把握した後に、それぞれのコマンドのオプションや引数の詳細を全て覚える必要はありません。コマンド操作に不明な点がある場合は、以下のような方法で調べることができます。

2.3.1. オンラインマニュアルを参照する(man コマンドの利用)

コマンドの詳細な説明を参照したい場合は、以下のようにすることで、オンラインマニュアルを表示することができます。

man␣[オプション]␣[セクション番号]␣コマンド名

表 7　man のオプション

man の主なオプション	説明
-a	該当する全てのセクションを出力
-s␣セクション番号	指定したセクションの中から、コマンドのマニュアルを検索
-f␣キーワード	完全一致のキーワード検索
-k␣キーワード	部分一致のキーワード検索

オンラインマニュアルページは表 8 のセクション(章)に分かれています。

表 8　マニュアルページのセクション

マニュアルページの主なセクション番号	セクション内容
1	一般ユーザコマンド
2	カーネルが提供する関数(システムコール)
3	システムライブラリに含まれる関数
4	デバイスファイル(/dev ディレクトリのスペシャルファイル)
5	ファイルの書式
6	ゲームとデモンストレーション
7	その他(マクロのパッケージや約束事)
8	システム管理コマンド

＜使い方＞

以下 man を使った cal のオンラインマニュアル参照方法です。

```
$ man␣cal
```

図 3 のようにオンラインマニュアルを参照できます。

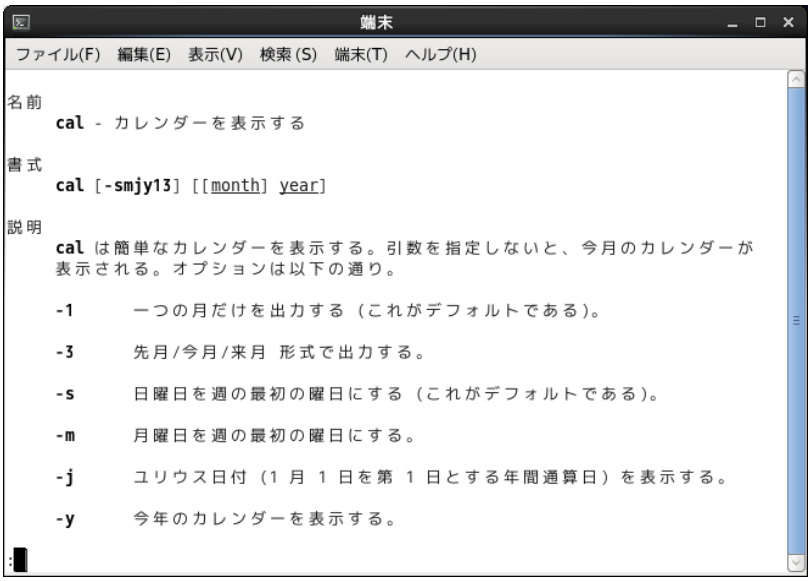


図 3　マニュアルのサンプル画面

man コマンドの画面操作

man コマンドを使うとターミナルにオンラインマニュアルの内容が表示されます。しかし、マニュアルの内容は大抵とても長いので、1 画面では収まらず画面の最下行に「**--続ける--**」や「**--継続--**」というメッセージを出力して止まります。man コマンドで表示される画面の制御は less コマンドで可能であり、主な操作は表 9　less コマンドの操作の通りです。

表 9　less コマンドの操作

less コマンドの主な操作	内容
[Space]キーもしくは[f]キー	次の 1 画面を表示する(forward)
[q]キー	less コマンドでの画面表示を終了する(quit)
[b]キー	前の 1 画面を表示する(before)
[↑]キー	1 行上にスクロールする
[↓]キーもしくは[Enter]キー	1 行下にスクロールする
/文字列	下方向に文字列を検索する
?文字列	上方向に文字列を検索する
[h]キー	操作のヘルプ（コマンド一覧）を表示する(help)

例)

man コマンドを用いてマニュアルを表示させた後、[Space]キーを押すことで、次の 1 画面分を表示させます。[q]キーを押すことで man コマンドを終了させることができます。

2.3.2. その他の方法

オンラインマニュアルは詳細に記述されており、最初はどこを参照して良いのか分からないことがあります。コマンドの使用方法を簡易的に知りたい場合や、コマンドの正確なスペルまたはセクションが不明な場合は、以下の方法を試すと良いでしょう。

コマンドの正確なスペルが分かっている場合

(1) --help オプションの利用

コマンドのオプションとして help を指定することで、そのコマンドの簡単なヘルプを参照することができます。利用できるオプションやパラメータを簡単に調べる場合には help オプションを使用できます[11]。

例)

```
$ man␣--help
```

(2) whatis コマンドの利用

オンラインマニュアルの最初の部分には、コマンド名と簡単な説明が書かれています。この部分[12]を表示させるのが whatis コマンドです。whatis コマンドを使用することで、コマンドのセクションを知ることができます。

whatis␣コマンド名

例)

```
$ whatis␣gedit
```

上記の検索結果から、gedit コマンドのマニュアルにはどのようなセクションがあるか知ることができます。

実行結果

```
gedit          (1)      - text editor for the GNOME Desktop
```

実行結果から gedit コマンドには、セクション 1（一般ユーザコマンド）にのみマニュアルがあることがわかります。複数のセクションにマニュアルがある exit コマンドや mkdir コマンドなどでは、セクションを指定して man コマンドを実行することでマニュアルの詳細を確認することができます。

例)

```
$ man␣3␣exit
```

[11] コマンドのオプションとして--help を指定できないコマンドもあるため注意が必要。また、-h と略記することができる。

[12] whatis データベースと呼ばれ、コマンド名やマニュアルの種類などマニュアルの目次とも言えるファイル

上記の例では、セクション 3 を指定して man のオンラインマニュアルを参照しています。man コマンドの使い方については、19 ページの「オンラインマニュアルを参照する」も参照してください。

コマンドの正確なスペルが不明な場合

(3) apropos コマンドの利用

マニュアルを参照したいコマンドのスペルが正確に分からない場合は、apropos コマンドを利用しましょう。apropos コマンドでは指定したキーワードをマニュアルの最初から部分一致検索し、その結果を表示するコマンドです。

apropos␣キーワード

例)

```
$ apropos␣whati
```

上記コマンドから、whati という文字列を含むコマンドは、makewhatis と whatis コマンドがあることが分かります。それぞれの詳細を確認したい場合は、(2)の whatis コマンドや man コマンドなどを使用します。

(4) シェルによる入力補完

コマンド名が不明な場合は、bash による入力補完を使うこともできます。例えば、what と入力後、Tab キーを押すと whatis と表示されます。また、wh と入力後、Tab キーを押すと何も変化はありませんが、続けて Tab キーを押すと whatis、whereis、which、while、whiptail、who、whoami と表示されます。これは wh で始まるコマンドが複数あり、補完する候補が一つではないためです。

コマンドの正確なスペルが不明な場合でも、(3)apropos コマンドや(4)入力補完を用いて正確なスペルを把握し、man コマンドや(1)--help オプション、(2)whatis コマンドでコマンドの詳細な情報を取得することができます。

これらを活用することは、コマンド操作に慣れる上で重要なポイントです。

3. テキスト編集について

本章では UNIX/Linux システムにおけるテキスト編集について説明します。

3.1. UNIX/Linux でのテキスト編集

UNIX/Linux システムでは、テキストデータが多く使用されており、システム設定の多くの場面でテキストファイルを編集する必要があります。UNIX/Linux で使用できるテキストエディタは、UNIX/Linux①入門編で紹介した **gedit** 以外にも多数あります。**Vim** と **Emacs** は UNIX/Linux のテキストエディタとして代表的なエディタであると言われています[13]。

ここでは、代表的なテキストエディタの一つである **Vim** の使い方について説明します。また、UNIX/Linux でテキスト処理に活用されることが多い **sed** コマンド、**awk** コマンドについて紹介します。

表 10　テキストエディタ

主なテキストエディタ	説明
Vim	UNIX/Linux 全般で用いられているエディタ
Emacs	高機能なエディタ
gedit	GNOME の標準エディタ
nano	CUI を用いて編集を行うテキストエディタ。画面下部にキー割り当てが表示されている。初心者が使用しやすいエディタである
Sublime Text	様々な OS で使用できるテキストエディタであり、プラグインが充実している

3.2. Vim エディタ

Vim エディタは UNIX/Linux で標準的に用意されているテキスト編集用のプログラム(エディタ)です。Vim は UNIX/Linux 系 OS では広く使用されているエディタです。Vim エディタを用いると、ファイルの作成、編集が行えます。

3.2.1. Vim の起動

Vim は、次のコマンドで起動します。

```
$ vim␣ファイル名
```

ファイル名に指定したファイルが存在している場合は、そのファイルのテキストがフル

[13] Emacs の使用方法については『UNIX 利用の手引き』の pp.60-65 を参照

スクリーンで画面に表示されます。

ファイルが存在していない場合は、図 4　Vim の起動 のように最下段に“New file”や“新規”と表示されます。

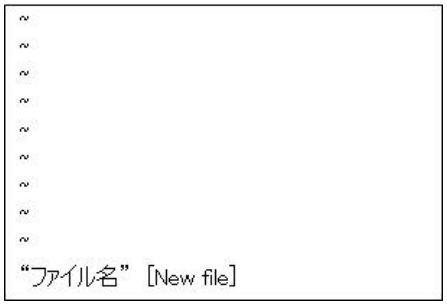


図 4　Vim の起動

もし誤ったファイル名を入力した場合は、「:q」を入力することで、Vim を終了させてもう一度起動し直します。

「:q」で終了できない場合は、「:q!」で終了させます[14]。

3.2.2. ノーマルモードと挿入モード

Vim エディタには大きく分けて 3 つの動作モードがあります。

1) 文字を入力する挿入モード
2) 文字列の編集作業をするノーマルモード
3) 画面の左下に:(コロン)や/(スラッシュ)を出して作業をするコマンドラインモード

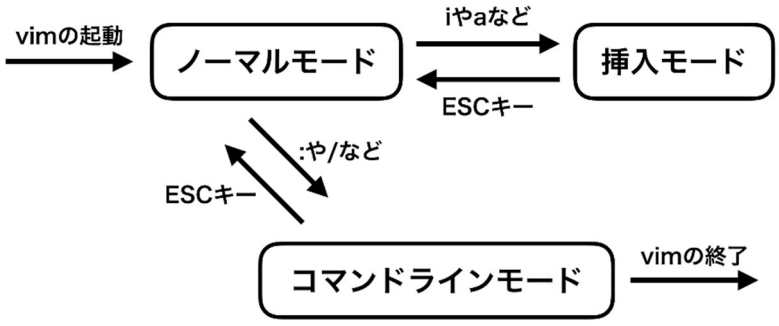


図 5　Vim のモード切り替え

[14] 「:q!」では変更を保存せずに Vim を終了する。詳しくは p.39 付録の表 15　Vim コマンド表(1/2)と p.39 付録の表 16　Vim コマンド表(2/2)を参照

図 5 の通り、Vim 上の作業はこれらのモードを行ったり来たりしながら行ないますが、このモードの切り替えを行うのが[Esc]キーと文字列挿入系コマンドなどです。

挿入モードにはノーマルモードで 「i」や「a」などの文字列挿入系コマンドを入力すると切り替わります。

図 6　Vim の挿入モード

挿入モードに切り替わると、画面の左下に“-- INSERT --”や--挿入--”と表示されます。

コマンドラインモードにはノーマルモードで「：(コロン)」あるいは「／(スラッシュ)」を入力すると切り替わります。

図 7　Vim のコマンドラインモード

コマンドラインモードに切り替わると、画面の左下に「：」あるいは「／」が表示され、コマンド待ち状態になります。

[Esc]キーは、今自分が Vim でどのモードを使っているのかわからなくなってしまった場合や、打ち込んだコマンドのキャンセルをする場合に押します。
[Esc]キーを入力した場合のアクションは次のようになります。

現在の状態が
ノーマルモード　→ ノーマルモードがリセットされる
挿入モード　→ ノーマルモードに切り替わる
コマンドラインモード　→ ターミナルの下に表示されていた「:」が消えてノーマルモードに切り替わる

まず何か作業をする前に必ず[Esc]キーを押す習慣をつけることで、入力ミスや誤った編集処理を回避できます。

何かを始めるときや、自分がどのモードにいるのかわからなくなったときなど、とりあえず[Esc]キーを入力する習慣をつけましょう。

1) ファイル作成

Vim を使ってテキストファイルを作成してみましょう。

```
$ vim␣test322.txt
```

Vim が起動しているので、[i]キーを押して挿入モードに移行

「meiji」と入力し[Enter]キーを押す

日本語入力を on(半角/全角キーを押す)にして、「明治大学」と入力

[Esc]キーでノーマルモードに移行、日本語入力を off(半角/全角キーを押す)にする

「:wq」コマンド<Enter>でファイルを保存して終了

[15]

cat コマンド等で *test322.txt* の内容を確認

2) コピー&ペースト

作成した *test322.txt* を Vim で開き、コピー&カット、ペーストをしてみましょう。

```
$ vim␣test322.txt
```

「 :set nu 」と入力（行の先頭に行番号を表示させるコマンド）

1 行目にカーソルを合わせて「yy」でコピー

2 行目に移動して「p」でペースト

---

15 ターミナルの設定で使用するテーマバリエーションを変更することができます。ターミナルを起動の上、メニューバーの「端末」→「設定(P)」→「全般」タブの「使用するテーマバリエーション(V):」から選択・変更できます。（本設定では「暗い」を選択）

さらに「p」を２回入力

２行目に移動して「yy」

５行目に移動して「p」を２回

[Esc]キーでノーマルモードに戻り、「:wq」〈Enter〉でファイルを保存して終了

cat コマンド等で.*txt* の内容を確認

3)　削除、変更を元に戻す

test322.txt を開き、以下の操作を行います。

```
$ vim␣test322.txt
```

「 :set nu 」を入力

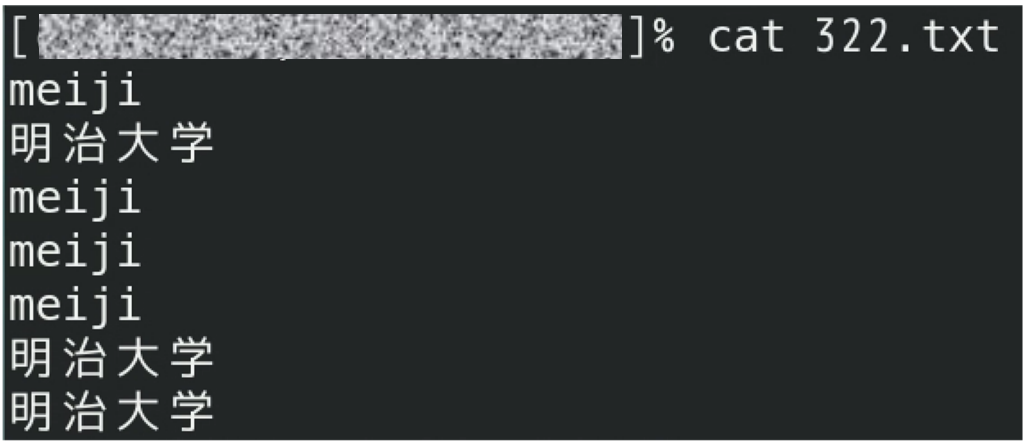
3行目の“meiji”の“e”で「x」を押し
「u」で元に戻す

5行目に移動して、「dd」を押し1行全て削除
「u」で元に戻す

1 行目に移動し、「dd」を押して全て削除

「:q!」〈Enter〉で変更を保存せずに終了

cat コマンド等で *test322. txt* の内容に変更がないか確認



【補足 3】Vimrc ファイルについて

Vimrc ファイルは、Vim 起動時の各種設定を記載するためのファイルです。Vim はデフォルトの状態では非常にシンプルなエディタですが、実際には様々な設定が可能であり、豊富なコマンドが用意されています。

それらを、Vim を起動するたびに設定するのは手間がかかります。Vimrc に設定項目を記述しておけば、Vim を起動するたびに「:set nu」とタイピングする必要はありません。

まず、ホーム以下に .vimrc ファイルを作ります。

```
$ vim ~/.vimrc
```

.vimrc

```
:set nu
```

上記ファイルを作成後、Vim を起動すると、「:set nu」を入力しなくても、行の先頭に番号が表示されていることが確認できます。このように頻繁に使う設定は .vimrc にまとめておくと便利です。

Linux には、「rc」と名前のつくファイルが他にも多数あります。たとえば、bash 起動時には ~/.bashrc が実行されるようになっています。このように、「なんらかのコマンドが実行・読み込まれるタイミングで対応する rc ファイルが実行・読み込まれるような仕組みが存在する」ということを覚えておいてください。

3.3. sed、awk コマンド

ここでは sed と awk コマンドの基本的な使い方について説明します。
sed コマンド(ストリームエディタ)はテキストエディタの一種です。sed コマンドを使用することで、入力されたテキストに対して様々な変換を行って出力することができます。このため UNIX/Linux でのテキストファイル処理にしばしば用いられます。また、awk は Unix/Linux で文字列処理をする際に用いられます。これらはスクリプト言語とみなすこともできます。スクリプト言語はコンパイル言語とは異なりコンパイルせずに処理ができるため、簡単なテキスト処理やテキスト処理の自動化に活用されています[16]。

3.3.1. sed

sed を使って文字列の変換を行うには以下のようにします。

sed␣[オプション]␣'スクリプト'␣[ファイル]

表 11　sed のオプション

sed の主なオプション	説明
-e	操作内容を指示するスクリプトを指定。スクリプトが 1 つの場合は省略可
-f	与えられたスクリプトファイルからスクリプトを読み込み、実行
-i	上書で置換をする。オプションの後に".bak"を入力することでバックアップ用ファイルを作成することができる
-n	各行を自動表示しない。p フラグと合わせて使用されることが多い。入力データに対する処理結果を標準出力で表示させる p フラグと合わせて使用することで、指定した行だけ表示することができる

sed では、文字列の置換に s コマンドと呼ばれる以下のようなスクリプトを用いることができます。

s/検索パターン/置換文字列/フラグ

検索パターンには、14 ページ「2.2.3.ファイル内の文字列を検索する」で扱った grep と同様に正規表現を使用することができます。また、フラグは置換方法を変更するものであり、省略することができます。
スクリプトには表 12　置換用スクリプトのようなものがあります。

[16] テキスト処理に用いられる主なスクリプト言語は、ここで扱う sed、awk 以外にも Perl、Ruby、Python など多数ある

表 12　置換用スクリプト

置換用スクリプト例	説明
s/検索パターン/置換文字列	検索パターンを置換文字列に置換。行ごとに最初の 1 つのみを置換
s/検索パターン/置換文字列/N	検索パターンを置換文字列に置換。行ごとに N(数値)番目のみを置換
s/検索パターン/置換文字列/ g	検索パターンを置換文字列に置換。すべて置換
s/検索パターン/置換文字列/i	大文字と小文字を区別せずに処理

例)

test331-1. txt という名前のファイルを作成

```
$ vim␣test331-1.txt
```

test331-1. txt

```
xxyyxx
zzxxzz
```

test331-1. txt 内の検索パターン“xx”をすべて置換文字列“XX”に置換

```
$ sed␣'s/xx/XX/g'␣test331-1.txt
```

実行結果

```
XXyyXX
zzXXzz
```

-i オプションを使用してバックアップファイルを作成

```
$ sed␣-i.bak␣'s/xx/XX/g'␣test331-1.txt
```

test331-1. txt 内の検索パターン xx をすべて置換文字列 XX に上書置換しますが、元のデータは *test331-1. txt. bak* というバックアップファイルに保存されます。ls コマンドなどで確認してみましょう。

-n オプション

ファイル内の特定の行を出力する際には、sed でオプション-n とフラグ p を組み合わせて使用することができます。

test331-1. txt の 2 行目のみを出力する

```
$ sed␣-n␣'2p'␣test331-1.txt
```

実行結果

```
zzXXzz
```

3.3.2. awk

コマンドラインによる awk プログラムの実行[17]

awk␣'awk プログラム'␣[ファイル]

awk では多様な制御構造や関数を使用することができます。ここでは特に文字列操作について紹介します。

awk での出力

awk プログラムでは print コマンドが頻繁に使われます。print コマンドの書式は以下の通りです。

print␣[引数(パラメータ)]

例)

```
$ vim␣test332-1.txt
```

test332-1.txt

```
わ␣ち␣い
か␣り␣ろ
よ␣ぬ␣は
```

ファイルの 1 列目[18]を表示させる

```
$ cat␣test332-1.txt␣|␣awk␣'{print␣$1}'
```

または、

```
$ awk␣'{print␣$1}'␣test332-1.txt
```

ここで$1 は 1 番目のフィールドを表しています。$2、$3・・・はそれぞれ 2 番目、3 番目・・・のフィールドを表し、$0 はフィールド全体を表します。

実行結果

```
わ
か
よ
```

複数列表示させる場合は、

```
$ awk␣'{print␣$1,$2}'␣test332-1.txt
```

のように表現します。

---

[17] プログラムが長くなる場合は、プログラムをファイルに書き込み、以下のようにして実行する（ファイルによるプログラムの実行）

awk␣–f␣awk プログラムファイル名␣編集したいファイル名

[18] awk プログラムでは通常、列や単語などの各要素を「フィールド」、行を「レコード」と呼ぶ

実行結果

```
わ␣ち
か␣り
よ␣ぬ
```

ファイルの 1 行目を表示させます。行を表示させる場合は、より簡単に以下のように書くことができます[19]。

```
$ awk␣'NR␣==␣1'␣test332-1.txt
```

実行結果

```
わ␣ち␣い
```

複数行表示させる場合は、

```
$ awk␣'NR␣==␣1;NR␣==␣3'␣test332-1.txt
```

のように表現します。上記の例では、*test332-1.txt* の 1 行目と 3 行目を表示します。

実行結果

```
わ␣ち␣い
よ␣ぬ␣は
```

```
$ awk␣'NR␣==␣1,NR==3'␣test332-1.txt
```

とした場合は、1 行目から 3 行目までを表示します。

文字列の抜き出し　substr 関数

substr(対象の文字列,m,n)

対象の文字列に対して、m 番目から始まる長さ n の文字列を返す(m は 1 から数える)。

例)

```
$ vim␣test332-2.txt
```

test332-2.txt

```
ABCDE
```

文字列「あいうえお」に対して、1 文字目から 1 文字表示[20]

```
$ echo␣'あいうえお'␣|␣awk␣'{print␣substr($0,1,1)}'
```

test332-2.txt ファイル内の文字列に対して、1 文字目から 1 文字表示

```
$ awk␣'{print␣substr($0,1,1)}'␣test332-2.txt
```

[19] awk において NR は読み込んでいる行の行数を表す組み込み変数

[20] ここで$0 は、現在のレコードを表す

実行結果

```
A
```

test332-2.txt ファイル内の文字列に対して、3 文字目から 2 文字表示

```
$ awk␣'{print␣substr($0,3,2)}'␣test332-2.txt
```

実行結果

```
CD
```

文字列の置換 gsub 関数

gsub("x","y",対象の文字列)

　対象の文字列に対して、x を y に置き換えることができます。

例)

```
$ vim␣test332-3.txt
```

test332-3.txt

```
xxXXXXxxXX
```

文字列「あいうえお」に対して、「い」を「か」に置き換えて表示する

```
$ echo␣'あいうえお'␣|␣awk␣'{gsub("い","か");print␣$0}'
```

test332-3.txt ファイル内のすべての文字列に対して、x を y に置き換えて表示する

```
$ awk␣'{gsub("x","y");print␣$0}'␣test332-3.txt
```

実行結果

```
yyXXXXyyXX
```

4. ファイルのパーミッション

本章ではパーミッション(権限)とその変更方法について説明します。

4.1. ファイルのパーミッションとは

UNIX/Linux では、全てのファイル、ディレクトリにおいて、所有者は誰かという情報を持っています。

また、ファイルの所有者は、そのファイルに対して、誰が書き込みや読み取り、実行などの操作を行っていいかの設定をすることができます。

これをファイルのパーミッションといいます。

パーミッションは

- そのファイルの所有者(user)に対してのもの
- ファイルの所有者と同じグループ(group)に所属している人に対してのもの
- それ以外の人(others)に対してのもの

という 3 種類の利用者単位で設定することができます[21]。

それぞれの利用者単位毎に設定できる許可内容は以下の 3 つです。

1) 読み取りの可/不可（r（read））
   ファイルに対しては、そのファイルの中身を読むことができるかどうかを設定できます。ディレクトリに対しては、そのディレクトリに含まれるファイルの一覧などを見ることができるかどうかを設定できます。
2) 書き込みの可/不可（w（write））
   ファイルに対しては、そのファイルへ書き込むことができるかどうかを設定できます。ディレクトリに対しては、そのディレクトリに新しくファイルを追加したり、ディレクトリの中のファイルを削除したりできるかどうかを設定します。
3) 実行の可/不可（x（execute））
   ファイルに対しては、そのファイルを実行することができるかどうかを設定します。ディレクトリに対しては、そのディレクトリの中へ cd コマンドなどで移動できるかどうかを設定します。

[21] 生田システムの現在の運用では、ユーザのグループ分けを行っていないので、利用単位設定の内、group に対するものは現時点で意識する必要はない。詳細は④p.12「5.1.ユーザとグループ管理」を参照

ファイルやディレクトリに設定されているパーミッションは、ls コマンドなどによって確認することができます。例えば、ls コマンドに-l というオプションを付けて実行すると次の図 8 のような表示が得られます。

```
iscadm09% ls -l
合計 16
-rwxr-xr-x   1 od01047  assist      4688 Mar 20 14:07 a.out*
-rw-r--r--   1 od01047  assist        83 Mar 19 20:01 report.txt
-rw-r--r--   1 od01047  assist       219 Mar 20 14:07 sample.c
drwxr-sr-x   2 od01047  assist       512 Mar 19 20:08 test/
iscadm09%
```

図 8　ls コマンドの実行結果

ここで、report.txt というファイルに注目してみましょう。左からスペースで区切られたフィールドが並んでいます。3 つ目のフィールドにある od01047 という部分がこのファイルの所有者を表しています。一番左のフィールドには記号のようなものが表示されていますが、これがパーミッションの状態を示しています。

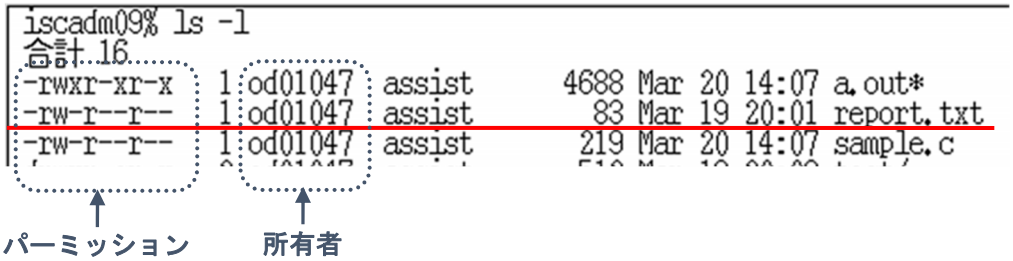


図 9　所有者とパーミッション

パーミッションは、rwx の 3 つが 1 セットで、これが u(user)g(group)o(others)について順番に表示されます。

前述の通り、「r」は読み出し許可、「w」は書き込み許可、「x」は実行許可を意味します。

図 9 のパーミッションのフィールドに表示されている「-(ハイフン)」は許可が与えられていないことを示します（一番左のフィールドはファイルの種別を表しており、図 9 のように、このフィールドに「-」が表示されていればファイルを意味し、「-」ではなく、「d」が表示されている場合はディレクトリを意味します)。

ls -l コマンドの結果から、ファイル report.txt は、user に対しては、rw-、group に対しては r--、others に対しては r--というパーミッションを持つことが分かります。

つまり、report.txt には以下のパーミッションが設定されていることになります。

- 所有者はこのファイルを読むことも書くこともできる
- 所有者と同一グループの人は、このファイルを読むことだけができる

➢　その他の人は、このファイルを読むことだけができる

ファイルを操作しようとして、「アクセス権がありません」などのエラーが発生した場合には、ファイルやディレクトリのパーミッションを確認してみてください。

自分が所有者となっているファイルやディレクトリについては、パーミッションを好みの状態に変更することができます。ただし、適切でないパーミッションを設定してしまうと、ログインすることができなくなったり、動作がおかしくなったりすることもあるので、充分に注意をした上で実行してください。

4.2. パーミッションの変更

パーミッションの変更は、chmod コマンドで行ないます。パーミッションを設定できるのは、ファイルの所有者と root ユーザ(管理者)だけです。

書式

chmod␣[-R]␣アクセス権指定␣ファイルまたはディレクトリ名

オプション

表 13　chmod のオプション

-R	ディレクトリに対して「chmod」コマンドを実行した場合に、指定したディレクトリだけではなく、そのディレクトリに含まれているファイルやディレクトリに対しても、指定したパーミッションに変更する。

＜使い方＞

「chmod」コマンドは、ファイルやディレクトリのパーミッションの変更に使用します。このコマンドを使うと、ファイルを読み取り専用(書き込みができない)状態にしたり、実行可能状態にできたりします。また、パーミッションは、そのファイルやディレクトリの所有者、同じグループに所属するユーザ、その他のユーザの 3 つのカテゴリーのユーザ毎に設定することができます。

アクセス権の指定には、記号を使う方法と数値指定があります。

1)　記号を使用

chmod␣[変更対象][変更方法][変更内容]␣ファイルまたはディレクトリ名

変更対象：u(ユーザ)、g(グループ)、o(その他のユーザ)、a(すべてのユーザ)

変更方法：=(指定した権限に移行)、+(指定した権限を付与)、－(指定した権限を除去)

変更内容：前述の通り、r(読み出し許可)、w(書き込み許可)、x(実行許可)

サンプルファイル *test42-1. txt* についてその他のユーザの書き込み許可を付与。

```
$ chmod␣o+w␣test42-1.txt
```

すべてのユーザの書き換え許可を設定。

```
$ chmod␣a+w␣test42-2.txt
```

u(ユーザ)のパーミッション状態（アクセス権）を a(すべてのユーザ)に移行（付与)。

```
$ chmod␣a=u␣test42-2.txt
```

2) 数値を指定

r(読み込み):4、w(書き換え):2、x(実行):1 で換算します。以下の例

```
$ chmod␣755␣test42-3.txt
```

において、図 10 のように変更内容が換算されるため、設定されるパーミッションは、rwx r-x r-x です。

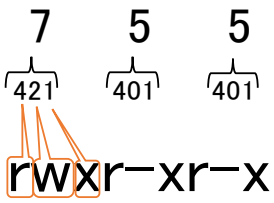


図 10 数値指定のアクセス権変更

例)

所有者にのみ読み込み、書き換え、実行を許可する

```
$ chmod␣700␣test42-4.txt
```

表 14 権限の記号表記と番号の対応

権限 / 番号	r (読み込み)	w (書き換え)	x (実行)	付与権限 (記号表記)
0	×	×	×	- - -
1	×	×	○	- - x
2	×	○	×	- w -
3	×	○	○	- w x
4	○	×	×	r - -
5	○	×	○	r - x
6	○	○	×	r w -
7	○	○	○	r w x

5. UNIX/Linux③中級編に向けて

本章では、UNIX/Linux③中級編で扱うシェルスクリプトの概要を紹介します。

UNIX/Linux で扱われるシェルスクリプトの基本的な内容を学びます。シェルスクリプトとは何かという概念から始まり、シェルスクリプトの文法や作成・実行方法・デバッグ技法について学んでいきます。そのために本テキストで扱ったコマンドやテキスト編集を活用していきます。

この観点から UNIX/Linux③中級編テキストでは主に、以下を扱います。

- シェルスクリプトの基礎

シェルスクリプトの基本的な知識について説明します。

- シェルスクリプトの文法

実際にシェルスクリプトを作成し、運用するために必要な文法や実行方法、デバッグ技法について具体例を交えながら紹介します。文法については主に以下を扱います。

- 変数
- 演算子
- 制御構造
- 関数

付録 Vim コマンド表

表 15　Vim コマンド表(1/2)

テキスト入力モードへのコマンド		画面の移動	
i	カーソルのある場所に挿入	C-b	1 ページ分上に画面を移動
a	カーソルの後ろに挿入	C-f	1 ページ分下に移動
I	行頭に挿入	C-g	行番号を表示
A	行末に挿入	H	画面の最上行に移動
O（大文字）	カーソルの上の行に挿入	L	画面の最下行に移動
o（小文字）	カーソルの下の行に挿入	M	画面の中央に移動
行内の移動		取り消しとやり直し	
w	次の単語の先頭へ移動	u	直前に行った変更の取り消し
b	前の単語の先頭へ移動	C-r	直前に行った変更のやり直し
0（ゼロ）	行頭へ移動	削除	
^	行頭へ移動	x（小文字）	カーソル位置の文字を削除
$	行末へ移動	X（大文字）	カーソル前の文字を削除
:行数	指定した行数へ移動	D	カーソル位置の文字から行末まで削除
Shift-左右矢印	単語単位で移動	dd	現在行を削除
行の移動		コピー・貼り付け	
gg	ファイルの先頭へ移動	yy	カーソル行をコピー
nG	ファイルの n 行目へ移動	yw	カーソル位置の単語をコピー
G	ファイル末へ移動	P（大文字）	カーソル行の上にペースト
保存と終了		p（小文字）	カーソル行の下にペースト
:w または zz	ファイルを保存		(x や dd で削除したものもペーストできます)
:w ファイル名	違うファイル名で保存	1 文字単位の移動	
:q	終了	k	上へ移動
:q!	変更を保存せずに終了	j	下へ移動
:wq または:x	ファイル保存して終了	h	左へ移動
		l（エル）	右へ移動

表 16　Vim コマンド表(2/2)

文字列検索		範囲選択	
*	カーソル位置の単語の後方検索	Ctrl-v	矩形選択
#	カーソル位置の単語の前方検索	Shift-v	行選択
/文字列	文字列検索	その他	
n	検索を行った後、次の検索結果に移動	:%s/文字列1/文字列2	文字列 1 を文字列 2 に置換
N	検索を行った後、前の検索結果に移動		

参考文献

(1) Cameron Newbam、Bill Rosenblatt 著、株式会社クイープ翻訳(2005)『入門 bash 第 3 版』オライリージャパン

(2) Lowell Jay Arthur、Ted Burns 著、坂本文訳(1999)『UNIX シェルプログラミング徹底解説』日経 BP 社

(3) 大角祐介(2014)『UNIX シェルスクリプトマスターピース 132』SB クリエイティブ

(4) 小林準(2011)『独習 Linux 第 2 版』翔泳社

(5) 中島能和、飛田伸一郎(2012)『CentOS 徹底入門第 3 版』翔泳社

(6) 橋本英勝(2010)『基礎からの Linux 改訂版』SB クリエイティブ

(7) 濱野賢一郎監修、中島能和著(2012)『Linux 標準資格教科書 LPIC レベル 1 対応』技術評論社