Linux 一般ユーザ 基礎、コマンドとシェル

目次

Linux 一般ユーザ 基礎、コマンドとシェル

はじめに

世界の Web サイトに占める Unix 系 OS のシェアは 2022 年 2 月の集計で 79.9%[1]、このうちの 47%が Linux（その他 52.7%が不詳 Unix 系）という調査結果をオーストリアの Q-Success 社が発表しています[2]。

有料ディストリビューションでは Red Hat Enterprise Linux(通称 RHEL)が 2018 年調べでシェア 33.9%、Microsoft 社が 47.8%となっています[3]。RHEL の無料クローンである CentOS は無料ディストリビューションを含む Linux 市場全体で 20.6%を占めていました[4]が、Red Hat 社が 2019 年に IBM 社に完全に買収されてからは CentOS 開発停止のアナウンスがあり、どんどんシェアが落ちています。サーバ AP の開発では RHEL か CentOS を知っていれば十分という状況は変わってきているようです。CentOS を代替えするディストリビューション候補がいくつか出ていますが、以降の説明では極力ディストリビューションに影響されない範囲で、一般ユーザが最低限 知っておいた方がよさそうな事柄を紹介します。Linux はカーネル、ドライバ、サービスその他ソフトの殆どがオープンソースで構成されており纏まった情報は見つけ辛く、初心者はどこから手を付ければよいか分からないと思いますので、この資料では一般ユーザが触れることが多いターミナル、シェル、汎用的なソフトウェア／コマンドに絞って紹介します。対象者は Windows は使いこなせているけれど Linux は触ったことが無いというレベルの人です。また、最新、詳細な情報はネット上に沢山でているのでそれらの情報を探す手掛かりになるようにキーワードになりそうな単語を極力混ぜていきます。

1. Linux ディストリビューションとは

ディストリビューションは配布者（ディストリビュータ）がカーネル及びパッケージ群の組み合わせをインストール可能な状態で媒体にしたもののことです。多くの組織がオープンソースでサービスやライブラリ、アプリケーションを開発しており、定番といえるソフトウェアでも競合するものが複数存在する場合があります。利用者が自分の好きなソフトウェアを集めて Linux 環境を作ることも可能ですが、依存ライブラリと組み合わせて動作確認をしていくのに手間が掛かるため一般的には RHEL や Ubuntu、Debian 等のディストリビューションを使います。因みに Linux のカーネルはディストリビューションによる違いはありません（バージョンが異なることはあります）。

ディストリビューションによって異なる点の一つはパッケージャでソフトウェアのインストールコマンド等が異なります。また、同じディストリビューションでもバージョンが上がってサービスの起動・終了プロセス（SysVinit,Upstart,systemd）のデフォルトが変わり設定ファイルの書き方と配置先、操作コマンドが変わっています。

---

[1] Usage statistics of operating systems for websites https://w3techs.com/technologies/overview/operating_system

[2] Usage statistics of Unix for websites https://w3techs.com/technologies/details/os-unix

[3] Red Hat、エンタープライズ Linux サーバー市場をリード https://www.redhat.com/ja/blog/red-hat-leading-enterprise-linux-server-market

[4] Historical yearly trends in the usage statistics of Linux subcategories for websites https://w3techs.com/technologies/history_details/os-linux/all/y

2. Linux サーバへの接続

一般的な Linux サーバは専用の筐体がなく、ローカル接続するためのコンソールを持ちません（Unix 専用マシンはコンソール接続用のシリアルポートを持っていました)。よほど古いサーバでなければネットワーク接続のプロトコルは ssh です。Windows であれば ssh 接続するツールとして Tera Term や PuTTY が使われてきましたが、Windows10 ではそのまま ssh が使えます。コマンドプロンプトから where ssh を実行して C:¥Windows¥System32¥OpenSSH¥ssh.exe のように表示さることを確認してください。

2.1. シェルで接続（ssh）

sshd(Secure Shell のサービス)が稼働しているサーバには、Windows10 等のコマンドプロンプトから ssh コマンドにサーバに登録済のユーザとサーバを指定して接続することができます。

```
C:¥Users¥User>ssh ユーザ@サーバ    …サーバ：ip アドレスかサーバ名で指定します
ユーザ@サーバ's password:          …サーバに登録されているパスワードを入力し [Enter]
```

ユーザ名 adminuser でサーバ ubuntu2004 に接続するとプロンプトに adminuser@ubuntu2004:~$と表示され（デフォルト設定の場合)、Linux の**ターミナル**（仮想端末）の上でシェルが起動した状態になります。

※サーバ側 sshd の設定によりパスワードによる認証を許可していない場合もあります

複数の接続を同時に行うことができ、サーバ側は w コマンドにより接続元が確認できます。

--サーバ側--

```
adminuser@ubuntu2004:~$ w
 08:59:10 up 11:10,  2 users,  load average: 0.10, 0.44, 0.40
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
adminuse pts/0    172.31.48.1      08:57    1:26   0.04s  0.04s -bash
adminuse pts/1    172.31.48.1      08:58    1.00s  0.05s  0.00s w
```

※TTY の pts は仮想端末、0,1 は端末の識別で、行右端の bash、w が実行中のコマンドです

※コマンドプロンプトで**カーソルが消える**現象がでたら、**IME 予測入力**の**オフ**を試してください

2.2. 資材のコピー（scp）

sshd が稼働しているサーバに対しては、scp(Secure Copy)コマンドを使って相互にファイル／フォルダの転送ができます。Linux 側のパス区切文字は"/"で、ドライブ指定はありません。

```
C:¥Users¥User>scp (-r) 送信元 送信先
```

-r：再帰的な処理で、フォルダごと転送します。転送元にフォルダを指定する場合に指定します。

送信元／送信先：資材のパスを指定します。サーバ側は*ユーザ@サーバ：資材のパス*で指定します。

＜実行例＞

```
C:¥Users¥User>scp "C:¥Users¥User¥Desktop¥新しいテキスト.txt" adminuser@172.31.53.18:/tmp
adminuser@172.31.53.18's password:<ユーザのパスワードを入力>
新しいテキスト.txt                                    100%    0     0.0KB/s   00:00
```


--サーバ側--

送信先ディレクトリ（この例の場合は/tmp）を ls コマンドのパラメータにして実行すると正常に送られていることが確認できます。

```
adminuser@ubuntu2004:~$ ls /tmp
新しいテキスト.txt
```

※Wndows のファイル名は UTF-16 になっているため、サーバ側のバージョンによっては文字化けする可能性があります。英数字だけを使ったファイル名を使った方が無難です。

2.3. 接続先のサーバ名（hosts 他）

サーバに接続するためには ip アドレスを知っている必要があります。
クライアントが Windows の場合は以下の方法を使ってサーバ名から ip アドレスに変換できます。

① 以下のフォルダに格納した hosts ファイルに書いておく
〔環境変数 SystemRoot が指すフォルダ：通常は C:¥WINDOWS〕¥System32¥drivers¥etc

※hosts ファイルを開く場合は**メモ帳(notepad.exe)を使い**管理者権限で上書きします。
🤦**他のエディターはファイルをロック**し一時的に hosts ファイルが使われなくなる場合があります。

② ネットワーク内の DNS に登録し、ネットワーク設定から当該 DNS を参照する

③ Samba(nmbd)が Linux サーバで起動している場合
サーバで hostname コマンドを実行し、その出力がサーバ名として使うことができる名前です。これは Windows で使っている NetBIOS による名前解決で、Linux サーバで nmbd というサービスが起動していることが前提になります。
開発環境等で DHCP で ip アドレスを割り当てている環境では便利です。

名前で ip アドレスを解決できるか否かは、以下のコマンドで分かります。
＜実行例＞

```
C:¥Users¥User>ping -a 172.31.53.18
```

-a：ip アドレスからホスト名に変換

--実行結果--

```
ubuntu2004.mshome.net [172.31.53.18]に ping を送信しています 32 バイトのデータ:
172.31.53.18 からの応答: バイト数 =32 時間 <1ms TTL=64
172.31.53.18 からの応答: バイト数 =32 時間 <1ms TTL=64
（中略）
172.31.53.18 の ping 統計:
    パケット数: 送信 = 4、受信 = 4、損失 = 0 (0% の損失)、
ラウンド トリップの概算時間（ミリ秒）:
    最小 = 0ms、最大 = 0ms、平均 = 0ms
```

※ip アドレスからホスト名を逆引きできた場合は、実行結果の 1 行目に出てきます

2.4. 接続に使うユーザの識別（ユーザ名、UID、GID）

Linux サーバの利用者は当該サーバの/etc/passwd ファイルか、サーバが属するドメインを管理している LDAP(Lightweight Directory Access Protocol)等に利用者登録されている必要があります。

/etc/passwd は cat や less 等のコマンドで誰でも内容を確認できます。

＜実行例＞

```
adminuser@ubuntu2004:~$ cat /etc/passwd
```

--実行結果--

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
（途中略）
tss:x:106:111:TPM software stack,,,:/var/lib/tpm:/bin/false
sshd:x:112:65534::/run/sshd:/usr/sbin/nologin
adminuser:x:1000:1000:admin user:/home/adminuser:/bin/bash
genuser:x:1001:1001:General User,,,:/home/genuser:/bin/bash
```

1 レコードは ":" で区切った以下の 7 列があります。

1 列目：ユーザ名　　…ログインで利用者の識別に使う名前

2 列目：パスワード　　…”x”になっているのは/etc/shadow でパスワードを管理している

3 列目：ユーザ ID(UID)　…システム内でユーザを特定する番号。

この UID や次列の GID を使って資源（ファイル等）の所有権やアクセス権の設定を行っている。若い番号はシステムで使い（root は 0)、一般利用者に振られる UID の開始番号はディストリビューションにより異なる(500～、1000～等)

4 列目：グループ ID(GID)　…システム内でユーザが属するグループを特定する番号。

UID と併せて資源（ファイル等）の所有権やアクセス権の設定に使われる。

ユーザは複数のグループに属することができ、/etc/group に登録・管理される

5 列目：コメント欄　　…一般的にはユーザのフルネーム等を登録する

6 列目：ホームディレクトリ…ユーザ固有の資材を格納しておくディレクトリ。

通常のディレクトリパスは /home/*ユーザ名*

7 列目：ログインシェル　…ユーザがログインしたときに起動する使われるシェル

nologin、false と書かれているユーザはログインできない。サーバ AP も UID,GID を持ち、この UID,GID に許された権限の範囲（パーミッション）でリクエストの処理を行う。

サーバ AP 用のユーザ名はセキュリティ上の観点からログインできなくしておく

3. デフォルト設定

Linux の用途（デスクトップ、サーバ、etc.）によって初期インストール時にソフトウェアや各種の設定を選択・変更できますが、基本的な設定は以下のようになります。

3.1. ディレクトリ構成

Linux のディレクトリ構成は FHS(Filesystem Hierarchy Standard)[5]として規定されていますが全てのディストリビューションが完全に一致しているわけではありません。

Ubuntu20.04 の場合は以下のようになっています。一般ユーザが使うものだけ説明を付けました。

```
/           …ルートディレクトリ（管理者アカウント<スーパーユーザ>の root とは関係なく、ディレクトリの根本）
  |--bin    …一般ユーザも使える実行形式のファイル、コマンド
  |--boot
  |--dev    …デバイスファイル、一般ユーザが使うのは出力不要(SYSOUT=DUMMY)の/dev/null 他
  |--etc    …各種の設定ファイル、シェルの初期化ファイルやサービスの起動パラメータ他（一般ユーザは参照のみ）
  |--home   …ユーザ登録時に各ユーザのホームディレクトリがユーザ名で作られる
  |--lib
  |--lib32
  |--lib64
  |--libx32
  |--lost+found
  |--media  …DVD 等の取り外し媒体をマウントするポイント
  |--mnt    …ディスクのマウントポイント
  |--opt
  |--proc
  |--root   …スーパーユーザ root のホームディレクトリ（一般利用者は接触禁止）
  |--run
  |--sbin
  |--snap
  |--srv
  |--sys
  |--tmp    …作業用に一時的なファイル等を格納する。使い終わったら削除しておく
  |--usr
  |--var    …サービス AP のデータやログ等、可変データを保存する。システムログは/var/log/syslog に保存
```

※ディレクトリにはディスクや外部媒体を繋ぐ（mount）ことができ、 /（ルート）を頂点とするツリー構造の一部として見えます。ファイル形式が異なって（別のファイルシステム）もマウント可能です。またディレクトリの階層は /home/adminuser/dummy.txt のように / で表現します。

---

[5] Linux Foundation (FHS Specifications Archive) https://refspecs.linuxfoundation.org/fhs.shtml

3.2. 文字コード（locale）

使用言語に「日本語」を選択した場合、主な Linux のディストリビューションは以下のように設定されます。

●文字集合　　：Unicode
●符号化方式　：UTF-8

使用する言語や通貨、時刻表示等はロケール（言語／国／文字コードで記号化）で規定され、locale コマンドで確認できます。

```
adminuser@ubuntu2004:~$ locale
LANG=ja_JP.UTF-8
```

※locale コマンドで表示される“LANG”がロケールを表す環境変数です。

海外のソフトウェアの中には日本語を考慮しておらず実行結果が文字化けするものがあります。このようなときは、ロケールの変換処理をしない（ローカライズなしのデフォルト）に設定します。

以下のようにして、“C”を LANG 環境変数に設定します。

```
adminuser@ubuntu2004:~$ LANG=C
```

※LANG が“C”に設定されていると、en_US で文字コードは ASCII になります。この場合は日本語を表示すると文字化けします

3.3. 時刻（timedatectl、UTC、JST、Time zone）

時刻はローカル時刻と世界協定時(UTC[6])、それにハードウェアの時刻があります。

ローカル時刻は UTC を Time zone に合わせて表示したもので、日本の場合は日本標準時(JST)として UTC+9 時間を使います。

また、Linux は起動時にハードウェアから時刻(hwclock)を取得し OS のメモリ内で時刻・時間を管理しますが、ハードウェアの時刻にはタイムゾーンの情報がないためローカルとして扱うか UTC として扱うかは設定次第になります。システム稼働中は ntp(Network Time Protocol)サーバから時刻を受け取り[7]OS 内の時刻を補正します。

---

[6] UTC(Universal time coordinated:協定世界時): 子午線を基準に〔グリニッジ平均時(GMT)〕世界で協定した調整時刻…詳細はネットで！

[7] NICT 情報通信研究機構-日本標準時 https://www.nict.go.jp/JST/JST5.html

タイムゾーン等は timedatectl コマンドで確認できます。

```
adminuser@ubuntu2004:~$ timedatectl
               Local time: 金 2022-03-18 12:15:40 JST
           Universal time: 金 2022-03-18 03:15:40 UTC
                 RTC time: 金 2022-03-18 03:15:39
                Time zone: Asia/Tokyo (JST, +0900)
System clock synchronized: yes
              NTP service: active
          RTC in local TZ: no
```

※時刻を表示している 3 つ目の RTC time(Real Time Clock time)がハードウェア時刻です。
上記の例ではハードウェアは UTC で設定され（Universal time とほぼ一致）ており、最後の行に“RTC in local TZ: no”と出力されています⇒実際は Windows 上(UTC 変換済)の VM なので、そのままハードウェアから UTC を受け取ったように見せています

※コマンドで表示する時刻やログはローカル時刻を使い“JST”や”UTC+0900“をつけて表示してくれますが、アプリケーションによっては Time zone つけずにデータやログを出力するものがあります。この場合は想定する時間との差（9h or 0h）から判定する以外に方法がありません。

3.4. シェル（sh、bash、dash、環境変数）

シェルの起動方法はログイン時に自動で起動するログインシェル、コマンド（対話的）起動、シェルスクリプトファイルの実行の 3 つあり、Linux ディストリビューションによりそれぞれで使われるシェルと初期化（環境変数の設定等）の内容が異なります。

（1）ログインシェル

ssh 等で接続したときに起動するのがログインシェルです。設定は/etc/passwd （項番 2.4 接続に使うユーザの識別（ユーザ名、UID、GID）参照）で管理し、デフォルトは bash です。

ログインシェルが起動すると、以下の初期化ファイルにより環境設定が行われます。

ア．最初に以下の初期化ファイルが実行されます。

・/etc/profile

・*ホームディレクトリ*/.profile（.bash_profile と .bash_login が存在しない場合だけ実行する）

イ．上記の初期化ファイルは以下のファイルが存在したらそれを実行します（bash 以外でも）。

・/etc/profile.d/*.sh

ウ．更に、ログインシェルが bash だった場合は以下の初期化ファイルも実行します。

・/etc/bash.bashrc

・*ホームディレクトリ*/.bashrc

※bash が行うログイン時の初期化ファイルはもっとあります（ネーミングは決まっている。存在しなくてもよい）。名前と実行条件の詳細は bash のマニュアルページ[8]を参照してください。

[8] bash オンラインマニュアル https://linuxjm.osdn.jp/html/GNU_bash/man1/bash.1.html

（２）sh コマンド（対話型シェル）

Linux の場合 sh（Bourne シェル）は別の POSIX 準拠のシェルへのリンクになっています。ターミナルから sh コマンドを実行するとログインシェルの子（または孫、ひ孫…）プロセスとしてリンク先のシェルが起動します。どのシェルが起動するかは sh のリンク先で確認できます（コマンド例：`which sh|xargs ls -l`）。

＜実行例＞ディストリビューション=Ubuntu20.04

```
adminuser@ubuntu2004:~$ which sh|xargs ls -l
lrwxrwxrwx 1 root root 4  2月 23 17:50 /usr/bin/sh -> dash
```

シェル起動時の初期化ファイルはシェルの種類により異なり、以下のように行われます。

ア．dash

環境変数 ENV に設定されたファイルが実行されます。

イ．bash

・*ホームディレクトリ*/.bashrc が実行されます※。

※sh という名前で bash を実行した場合（sh のリンク先が bash）は Bourne シェルの動作に近づけるために .bashrc は実行されず、環境変数 ENV に設定されているファイルを実行します

（３）シェルスクリプト

連続した文やコマンド、変数宣言を書いたスクリプトファイルは実行が要求されると 1 行目の記述（シバン）を参照してスクリプトを実行するプログラムが決まります。

シェルスクリプトはシバンに起動シェルを書きます。慣用的に/bin/sh と書かれたものが多いですが、使われるシェルはディストリビューションにより異なります。

＜シバン＞

```
#!/bin/sh
```

Ubuntu20.04 では以下のように sh は dash へのリンクになっていて、dash が起動します。

```
lrwxrwxrwx 1 root root 4  2月 23 17:50 /bin/sh -> dash
```

※/bin は/usr/bin へのリンクになっていて/bin/dash と/usr/bin/dash は同じものです

＜非対話型で実行した場合の初期化ファイル＞

bash の場合は非対話型（cron 等で起動し端末と非接続）で起動された場合、環境変数の BASH_ENV に設定されているファイルを実行します（bash マニュアル参照）。しかし、bash 以外のシェルや運用ツールで起動する場合があるのなら BASH_ENV を使うよりも source(または”.”)コマンドで初期化ファイルをシェルの内部に展開する方が簡単です。

また、非対話型のシェル実行の場合/etc/profile 等の初期化ファイルを読み込まないため、PATH 環境変数は初期値（一般的な値は/usr/gnu/bin:/usr/local/bin:/usr/ucb:/bin:/usr/bin でシステム依存…bash オンラインマニュアル参照）になるため、後付けでインストールしたコマンドやライブラリが見つからない場合はフルパスで指定するか、シバンに#!/bin/bash -l のように-l オプションをつけて初期化ファイルが読み込まれるようにします。

（4）環境変数／シェル変数／エイリアス

シェルやコマンドから参照が必要な環境に関する情報は環境変数やシェル変数として保存しておきます。環境変数やシェル変数はログイン時に初期化ファイル（/etc/profille、ホームディレクトリの.profile 等）で設定され、環境変数については export コマンドで追加された変数を含めて子プロセスやコマンドに引き継がれていきます。また、エイリアスはオプション付きのコマンドで本来のコマンド名を置き換えて利便性を上げるために使います。これらの設定値は以下のコマンドで確認できます。

＜コマンド＞＜表示内容＞

env	環境変数
set	環境変数、シェル変数、シェル関数
alias	エイリアス（別名）の登録状態

① ssh でログイン直後に env コマンドを実行した例を以下に示します（重要なものだけ抜粋）。

```
adminuser@ubuntu2004:~$ env
SHELL=/bin/bash        …ログインシェル（実行中のシェルではない）
PWD=/home/adminuser    …カレントディレクトリ（作業ディレクトリ。相対指定でパスを指定するときの起点になる）
LOGNAME=adminuser      …ログイン名（ログインしている利用者のユーザ名）
HOME=/home/adminuser   …ログインしている利用者のホームディレクトリ
LANG=ja_JP.UTF-8       …現在のロケール
USER=adminuser         …ログイン名（LOGNAME と同一の値が入る）
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
                       …実行コマンドを探すディレクトリ（複数のパスを指定する場合は、コロン[:]で連結）
```

※表示形式は *変数名*=値

② bash で set コマンドを実行した例を以下に示します（重要なものだけ抜粋）。

```
adminuser@ubuntu2004:~$ set
EUID=1000              …実効ユーザ ID（だれとして＜権限で＞実行しているか。実行ファイルの suid 属性）
HISTFILE=/home/adminuser/.bash_history…コマンド履歴（過去のコマンドの再利用に使える）の保存場所
HISTSIZE=1000          …コマンド履歴に記憶するコマンド数
HOSTNAME=ubuntu2004    …現在のホスト名
IFS=$' ¥t¥n'           …フィールドの区切り文字（bash や read コマンドが単語の切り出しに使用）
PS1='¥[¥e]0;¥u@¥h: ＜途中略＞ ¥$ '…プロンプトとして表示する文字列
PS2='> '               …PS1 に対する入力でコマンドが完成していないときに出すプロンプト
PS4='+ '               …source コマンド等でスクリプトを展開したときに、階層を示すために表示する文字
UID=1000               …現在のユーザ ID
__expand_tilde_by_ref () { ＜以下略＞ …シェル関数
```


③ alias コマンドを実行した例を以下に示します。

```
adminuser@ubuntu2004:~$ alias
alias alert='notify-send --urgency=low -i "$([ $? = 0 ] && echo terminal || echo error)"
"$(history|tail -n1|sed -e '¥''s/^¥s*[0-9]¥+¥s*//;s/[;&|]¥s*alert$//'¥'')"'
alias egrep='egrep --color=auto'
alias fgrep='fgrep --color=auto'
alias grep='grep --color=auto'
alias l='ls -CF'
alias la='ls -A'
alias ll='ls -alF'
alias ls='ls --color=auto'
```

※*名前='実行内容'* 上記の例では grep と ls の実行結果に色が付ける等の登録をしています

4. ユーザ固有の環境設定（ホームディレクトリ）

ユーザ毎に動作を変えたり前回の状態を保存するための場所がホームディレクトリです。設定は /etc/passwd （項番 2.4 接続に使うユーザの識別（ユーザ名、UID、GID）参照）で管理し、デフォルトは /home/*ユーザ名* です。cd コマンドでパラメータを指定しなかった場合、このディレクトリにカレントディレクトリが移ります。また、パス指定では~(チルダ)でホームディレクトリを表すことができます。

ls -a または la（エイリアス）で保存されている全てのディレクトリやファイルを見ることができます。

```
adminuser@ubuntu2004:~$ ls -a ~   …パラメータのパスに ~ を指定するとホームディレクトリに"チルダ展開"されます
.  ..  .bash_history  .bash_logout  .bashrc  .cache  .profile  .ssh  .sudo_as_admin_successful
```

※「.」はカレントディレクトリ（環境変数 PWD と一致)、「..」は親ディレクトリを表します
cd .. でカレントディレクトリが親ディレクトリに移ります

※ファイル名の先頭が . のファイルは隠しファイルで ls コマンドに -a オプションを付けた時だけ表示されます。隠しフォルダや隠しファイルにはアプリケーションが実行時に参照するユーザ毎のパラメータや初期値等が書き込まれます

利用者が自分用の環境変数やエイリアス、シェル関数を作りたい時は、ホームディレクトリの.profile か.bashrc を編集することができます。

（改変前に cp -a .profile .profile.origin のようにバックアップを作ってください）

5. 権限(パーミッション)

Linux／Unix のセキュリティの核はパーミッションとユーザ／グループによる権限管理です。

5.1. ファイルのアクセス権

ディレクトリやファイルには read、write、execue の 3 つのアクセス種類を、所有者(user)、グループ(group)、その他(other)の 3 レベルに区分して許可が設定されています（パーミッション)。

/home/*adminuser* (ユーザ名 adminuser のホームディレクトリ)で ls -la コマンドを実行すると以下が表示されます。(ls コマンドに -l オプションをつけて long フォーマットで表示する)

```
adminuser@ubuntu2004:~$ ls -la
合計 36
drwxr-xr-x 4 adminuser adminuser 4096  3 月 24 15:33 .    …自ディレクトリを表す
drwxr-xr-x 4 root      root      4096  3 月 16 14:16 ..   … 1 階層上のディレクトリを表す
-rw------- 1 adminuser adminuser 5934  3 月 25 15:38 .bash_history
-rw-r--r-- 1 adminuser adminuser  220  2 月 25  2020 .bash_logout
-rw-r--r-- 1 adminuser adminuser 3771  2 月 25  2020 .bashrc
drwx------ 2 adminuser adminuser 4096  3 月 16 13:46 .cache
-rw-r--r-- 1 adminuser adminuser  807  2 月 25  2020 .profile
drwx------ 2 adminuser adminuser 4096  3 月 16 10:33 .ssh
-rw-r--r-- 1 adminuser adminuser    0  3 月 24 15:33 .sudo_as_admin_successful
①②--③--④-- ⑤ ⑥所有者名 ⑦グループ ⑧バイト ⑨作更新時刻  ⑩ファイル名／フォルダ名
```

表示している内容は以下の通り。

① ファイルタイプ（-:通常ファイル、d:ディレクトリ、l:シンボリックリンク...他特殊ファイル）
② 所有者のアクセス権
③ グループのアクセス権
④ その他（②、③以外）アクセス権
⑤ ハードリンクの数（ファイルタイプがディレクトリの場合は、配下のサブディレクトリの数）

＜アクセス権＞

アクセス権は各レベル毎に r,w,x で許可されている行為を表し、-は許可がない状態です。

r=4、w=2、x=1 の 8 進数で表現する場合もあります。全てのレベルで全てのアクセスを許可している場合は 777 に、所有者だけが read 権だけを持っている場合は 400 になります。

ディレクトリの場合は、実行権 x を持たないユーザ／グループは当該ディレクトリを開くことができません。通常ファイルのパーミッションは初期値として実行権は付きません。それ以外は作成者の umask で決まります。umask は隠す（マスクする）パーミッションを指定します。umask コマンドで設定したり現設定値を確認することができ、以下のように解釈されます。

```
adminuser@ubuntu2004:~$ umask
0002   …666（実行権除外）－ 2 = 664 ⇒ 所有者=6:rw-、グループ=6:rw-、その他=4:r--
```

umask は 4 桁ありますが、Linux では 1 桁目は使われません。

5.2. 利用者権限（root、管理者グループ）

一般ユーザはファイルの参照や更新、コマンドの実行にパーミッションによるアクセス制御が掛かりますが、管理・監視で使われる root アカウントは特権ユーザ、スーパーユーザと呼ばれ扱いが異なります。

（1）資源へのアクセス

root は、他ユーザが所有者になっているファイルでも chown コマンドで所有者を書き換えたり、su コマンドでパスワードなしに所有者に成り代わることができるため、殆ど無制限に全ての資源に更新・削除を含むアクセスが可能です（参照だけなら chown や su も不要です）。

（2）root だけがアクセス可能な資源

ユーザのパスワード（のハッシュ）が書かれている/etc/shadow は root だけがアクセスできるため、ユーザの登録等は root だけができる作業です。その他にも/dev や/proc 等にも root だけがアクセスできる資源があり、これらの参照を必要とするコマンドは一般ユーザには実行できません。

（3）root（管理者）権限の抑止

root のアカウントは強力過ぎて危険です。侵入者に root 権限を奪われると LAN ケーブルを抜くか場合によっては電源を落とす等の物理的な対処以外に手がありません。

そのため、デフォルトでは root を使えなくしているディストリビューションもあります。この場合、管理者権限が必要な作業を行うには以下の手順が必要になります[9]。

① root 権限が必要な時だけ su コマンドで root になる

su コマンドにパラメータ（ユーザ名）を指定しないと root の対話型シェルが始まります。

この場合、root のパスワードの入力を求められます。更に － オプションを付ける(su -)とログインし直したように環境変数も初期化されます。

su コマンドを抜けるには exit で root の対話型シェルを終了します。

② /etc/sudoers でコマンド実行を許可しているグループに入り、sudo でコマンドを実行する

/etc/sudoers には初期値が書かれていますが、ディストリビューションによってグループ名が異なります（慣例的には wheel か adm、sudo）。

自分のユーザ名がどのグループに属していかは id コマンドで分かります。

```
adminuser@ubuntu2004:~$ id
uid=1000(adminuser) gid=1000(adminuser) groups=1000(adminuser),4(adm),24(cdrom),
27(sudo),30(dip),46(plugdev),116(lxd)
```

sudo コマンドのパラメータとして管理者用コマンドを渡すと root 権限で実行されます。

この場合、root ではなく実行ユーザのパスワード入力を求められます。

```
adminuser@ubuntu2004:~$ sudo cat /etc/sudoers
[sudo] adminuser のパスワード:
```

※sudo や su コマンドの実行は監査用のログに記録されます

[9] その他にも SELinux を有効にして root の権限を制限する方法もあります

6. 対話型シェル（bash）の操作

bash を使って開発や運用を行うときに便利な機能を纏めます。bash には bash の中だけで使える組み込みコマンドがありますが、外部コマンドとの違いはオンラインマニュアルを見る方法ぐらいです。

6.1. オンラインマニュアル（man）

Linux に入っているソフトウェアは“オンラインマニュアル”が同時にインストールされます。この文書は規格化されていて man コマンドにより定型様式で表示することができます。

man コマンドの結果は環境変数 PAGER に設定されているアプリか、当該環境変数の設定がない場合は/usr/bin/pager により表示しますが、Ubuntu20.04 の場合は/usr/bin/pager が less へのリンクになっており、less が起動してターミナルにオンラインマニュアルを表示します。

（1）bash のオンラインマニュアル

```
man bash
```

（2）bash の組み込みコマンドのオンラインマニュアル

以下の man コマンドで bash の組み込みコマンドのマニュアルを見ることができます[10]。

```
man 7 bash-builtins
```

組み込みコマンドの場合は、help コマンドや info コマンドで使い方を調べることもできます。

＜help コマンドの実行例＞

```
adminuser@ubuntu2004:~$ help [
[: [ arg... ]
    条件式を評価します。

    これは test 組み込み関数と同義語です。ただし、最後の引数に開始の`[' と一致
    するように文字`]' を与えなければいけません。
```

＜info コマンドの実行例＞

```
adminuser@ubuntu2004:~$ info [
info: No menu item '[' in node '(dir)Top'
TEST(1)                          User Commands                          TEST(1)
NAME
       test - check file types and compare values
SYNOPSIS
       test EXPRESSION
       test
       [ EXPRESSION ]
       [ ]
       [ OPTION
DESCRIPTION
       Exit with the status determined by EXPRESSION.
(以下略)
```

[10]日本語化されていない場合があります。ネット上に日本語オンラインマニュアルが公開されています。
https://linuxjm.osdn.jp/html/GNU_bash/man1/builtins.1.html

6.2. マニュアル／テキストビューア（less）

Windows や古い Unix には more というテキストビューアがありましたが、more はページを戻ることができなかったためページを戻す機能を付けた less というアプリが作られました。

less のパラメータにテキストファイルを指定するとファイルの内容を表示します。デフォルトの環境変数 LESSOPEN/LOSSCLOSE が設定されていれば gz 等圧縮ファイルの中身も閲覧できます。

less *テキストファイル*

less の主なキー操作は以下のようになっています。

スペース、f：1 ページ進める、b ：1 ページ戻る
↑ ：1 行戻る、↓：1 行進める
G ：ファイルの最終行に移動、g ：ファイルの先頭行に移動
/ ：文字列の検索 検索（Enter 押下）後は、n：次へ、N：一つ前
-N ：行番号を表示（-, N, [Enter]確認メッセージ表示, [Enter]のキーストローク）
-n ：行番号を非表示
q ：終了

6.3. その他のビューア（tail、head）

大きなファイルの一部（例えば、ログファイルの最新部分やファイルの確認等）を表示するのに便利なのが tail コマンドと head コマンドです。

（1）tail

tail コマンドはファイルの最後-n 行分を表示します。-f オプションを付けるとコマンドは終了せず最新情報を n 行分更新し続けます（-f を使ったら ctrl＋c で終了させてください）。

```
adminuser@ubuntu2004:~$ tail -n5 -f /var/log/syslog
Mar 31 10:47:15 ubuntu2004 systemd[1]: ua-timer.service: Succeeded.
Mar 31 10:47:15 ubuntu2004 systemd[1]: Finished Ubuntu Advantage Timer for running ...
Mar 31 10:50:21 ubuntu2004 systemd[1]: fwupd.service: Succeeded.
Mar 31 11:17:01 ubuntu2004 CRON[14288]: (root) CMD (   cd / && run-parts --report /etc/...
Mar 31 12:17:01 ubuntu2004 CRON[14485]: (root) CMD (   cd / && run-parts --report /etc/...
```

※注意：tail の help には以下の注意が記載されています
--follow (-f)を使用すると追跡しているファイルの名前が変更されたとしても、tail は元のファイルの終端を追跡し続けます。ファイルの名前が変わる場合(例: ログのローテーションなど)には --follow=name を使用してください。これにより名前の変更、削除、作成などにあわせて名前のついたファイルの末尾を追跡するようになります。

（2）head

ファイルの先頭-n 行分出力します。tail コマンドに似ていますが-f オプションはありません。-n オプションのデフォルトは 10 行です。

```
adminuser@ubuntu2004:~$ head -1 /var/log/dmesg
[    0.000000] kernel: Linux version 5.4.0-105-generic (buildd@lcy02-amd64-066) (gcc
version 9.4.0 (Ubuntu 9.4.0-1ubuntu1~20.04)) #119-Ubuntu SMP Mon Mar 7 18:49:24 UTC 2022
(Ubuntu 5.4.0-105.119-generic 5.4.174)
```


6.4. コマンドの履歴・再実行（history）

~/.bash_history に過去に実行したコマンドが書き込まれています（最大でシェル変数 HISTSIZE に設定されている数）。この履歴は以下のコマンドで表示・再利用できます。

・history コマンド：コマンド実行履歴を通し番号付きで表示します
・ ↑ ：一つ前に実行したコマンドを表示します
・ ↓ ：一つ後に実行したコマンドを表示します
・!nnn ：nnn（history で表示した番号）番目を再実行します
・!! ：直前に実行したコマンドを再実行します
・!nnn + [esc] + ^ ：コマンド行に nnn 番目のコマンドを編集できる状態に履歴展開します

6.5. コマンド行の編集（コマンド／パス補完、カーソル移動他）

bash のコマンドを入力できる行では、以下のキー操作を受け付けます（デフォルト設定の場合）。

・*xx* [tab] ：コマンド、パスが入力できるタイミングで 1 文字以上を入力後 tab キーを押下すると候補を表示します
・[ctrl] + a ：行の先頭にカーソルが移動します
・[ctrl] + e ：行の終端にカーソルが移動します
・[ctrl] + k ：カーソル以降の文字列を削除します

6.6. Linux ターミナル上のコピー＆ペースト（ctrl-c はダメ）

Windows でコピーのショートカット[ctrl] + c は Linux では前面で実行中のコマンドに対して打ち切りのシグナルを送ります。Linux のショートカットはターミナルソフト及びキー設定により変わります（コピーは[ctrl] + [shift] + c という設定が多いようですが）。

Unix 機ではターミナル上で選択するとコピーが行われ（バッファに送られる）、3 つボタンマウスの中ボタンや 2 つボタンの同時押しで入力行にペーペストされるのが一般的でした。Linux 向けのターミナルでも踏襲しているものがあるので試す価値はあります。

Windows10 のコマンドプロンプトで ssh を起動した場合はマウスを使った以下手順が有効です。

・左クリックで範囲選択
・右クリックで選択範囲のコピー…選択が解消されます
・もう一度右クリックでペースト

 ctrl + c は処理を打ち切りたいときにだけ実行します

7. シェルスクリプト

開発や運用の現場では定型の作業や、自前で開発したアプリケーションの実行の前準備や後処理をスクリプト化して利用します。スクリプト、特に定型作業の書き方を実践に即して説明します。

7.1. 予備知識

（1）パイプ

コマンドの実行結果は｜(パイプ）で繋ぐことで別のコマンドの入力に渡すことができ、その受け渡しには「標準出力」と「標準入力」が使われます。

例えば、find コマンドで条件に合うファイルを見つけて ls -l コマンドで情報を見たい場合...

＜単純にパイプで連結した場合＞

```
adminuser@ubuntu2004:~$ find /etc -name "*.sh" | ls -la
合計 44
drwxr-xr-x 4 adminuser adminuser  4096  4月  1 12:29 .
drwxr-xr-x 4 root      root       4096  3月 16 14:16 ..
-rw------- 1 adminuser adminuser 10708  4月  1 10:22 .bash_history
（以下省略）
```

※ls は標準入力を使わないため、表示された結果は adminuser のホームディレクトリの内容です

パイプを繋ぐために xargs コマンドを使います。

```
adminuser@ubuntu2004:~$ find /etc -name "*.sh" | xargs ls -la
find: ‘/etc/polkit-1/localauthority’: 許可がありません
find: ‘/etc/multipath’: 許可がありません
find: ‘/etc/ssl/private’: 許可がありません
-rwxr-xr-x 1 root root  467  3月 16 10:32 /etc/console-setup/cached_setup_font.sh
-rwxr-xr-x 1 root root  358  3月 16 10:32 /etc/console-setup/cached_setup_keyboard.sh
（以下省略）
```

※但し、このままでは名前に空白を含むファイルはうまくいきません。空白を含む場合の対処方法については man xargs で確認してください

（2）標準エラー出力

プロセスには起動時に標準入力、標準出力、標準エラーの３つのファイル（標準ストリーム）が割り当てられます。パイプがないときは入出力はターミナルに接続され、前項のケースでは find コマンドで発生したエラー（標準エラー）が欲しい情報（標準出力）に混ざってしまいます。標準エラーは以下のようにして振り分けることができます。

```
adminuser@ubuntu2004:~$ find /etc -name "*.sh" 2>/dev/null | xargs ls -la
-rwxr-xr-x 1 root root  467  3月 16 10:32 /etc/console-setup/cached_setup_font.sh
-rwxr-xr-x 1 root root  358  3月 16 10:32 /etc/console-setup/cached_setup_keyboard.sh
（以下省略）
```

※“2>/dev/null”の 2 は標準エラーを表すファイル識別で、>はリダイレクト（出力先の変更)、/dev/null はヌルデバイスと呼ばれる疑似デバイスでここから先にはどこにも渡されません

（３）変数の扱い

bash は変数の値やコマンドの結果（標準出力）をコマンドラインに埋め込んだり、ファイルとして扱うことができます。以下に例を挙げます（詳細は bash オンラインマニュアル「コマンド置換」、「クォート」、配列については「配列」と「パラメータの展開」を参照してください）。

`コマンド`　：`[バッククォート]で囲まれた部分は実行されて結果の値に置き換わります

$(コマンド)　：バッククォートと同様にコマンドの標準出力に置き換わります

<(コマンド)　：プロセス置換（Process Substitution）。コマンドの実行結果を（入力）ファイルとして扱います

$((*算術式*))　：算術式が計算されて置き換わります（算術式展開）

”$xxxx”　：'[ダブルクォート]で囲まれた文字列に含まれる変数はコマンドの実行前に変数の値に置き換わります

'$xxxx'　：'[シングルクォート]で囲まれた文字列に含まれる変数名と一致する文字列が、変数の値に置き換わることなくそのままの文字列として評価されます

${xxxx}yy　：変数とリテラルを連続した文字列にする場合、変数名を識別するために${*変数*}とします

${*配列変数*[*ix*]}：bash は declare -a *配列変数*、declare -A *連想配列* で配列が宣言できます。また、*配列変数*[*ix*]=value、配列変数=(value1 value2...)で自動生成します

（４）文字列／ファイルパスの置換

bash は**コマンドを呼び出す前**にコマンドライン上のパス・ファイル名を置換（展開）します。

チルダ展開：ls -a ~ の〜はログインユーザのホームディレクトリ(/home/ユーザ名)に置換

パス名展開：以下の特殊パターン文字を使って検索し、結果（ファイルリスト）で置換します。

*　空白を含む任意の文字列

?　任意の１文字

[ ] [と]の間に列挙した文字のいずれか一文字、または :alnum: のように指定した文字クラス

例えば...

```
adminuser@ubuntu2004:~$ ls -l
 (中略)
-rw-rw-r-- 1 adminuser adminuser  142  4月  7 11:03 請求書1.txt
-rw-rw-r-- 1 adminuser adminuser  139  4月  7 11:03 請求書2.txt
```

のとき...

```
adminuser@ubuntu2004:~$ find . -name *.txt
find: paths must precede expression: `請求書2.txt'
find: possible unquoted pattern after predicate `-name'?
```

※find コマンドがパラメータを受け取る前にパス名展開が行われ、期待通りに動作しません

```
adminuser@ubuntu2004:~$ find . -name "*.txt"
./請求書1.txt
./請求書2.txt
```

※ダブルクォートで文字列を囲むことによりパス名展開が抑止され、find コマンドが *.txt の文字列を処理したことで期待通りの動作になっています

（５）複合コマンド

複数のコマンドを現環境に影響がないように子プロセスとして実行したり、セミコロンやパイプで繋いで実行して結果を取り出すときに複合コマンドを使います。

シェル関数は位置パラメータを持った複合コマンドです。

(*コマンド*)：子プロセスを生成してコマンドが実行され、最後のコマンドの結果が得られます

{ *コマンド*;}：現在のシェルで実行され（子プロセスを作らない）最後のコマンドはセミコロン;か改行で終わることと、{ }は予約後のため前後に空白かメタ文字[11]が必要です。最後のコマンドの結果が得られます。

7.2. コマンドの返り値判定／ジョブの実行状況確認（grep）

コマンドの実行結果は$?に設定され、正常結果（grep の場合は見つかったとき）であれば 0 です。簡易的に先行ジョブが終わったか否かや重複して実行中が無いか等の確認は以下のようにするとシェルスクリプト内で判定できます。

```
ps xa | grep -v grep | grep ジョブ名
ret=$?
if [ "$ret" -eq 0 ]; then
    echo OK
else
    echo NG
fi
```

※grep コマンド自身がパラメータに指定したジョブ名で拾われてしまうため、オプション-v を付けて grep という文字列を含む情報を除外します。判定する場合は、返り値が 0 の時に実行中です

7.3. 正規表現と複数文字列の検索（egrep）

grep はディスクに保存されているファイルや標準入力（他のコマンドの実行結果）から文字列を探すのによく使われます。正規表現による検索は egrep または grep -E で行います。

以降で説明する awk や sed はシェルの中でデータ加工のためによく使われます。/bin、/etc フォルダの中を探すとシステムで実際に組み合わせて使っている例が多数あり、参考になります。

```
egrep -rwIi "#¥!.+sh|awk|sed" /bin /sbin 2>/dev/null
```

＜この例で使っているオプション＞

-r:ディレクトリ内を再帰的に検索します。-w:対象が単語単位で一致させます。

-I:バイナリファイルを対象外にします。-i:大文字、小文字を区別しません。

＜正規表現の内容＞

複数の検索対象を or 連結するのが | です。この例では#!*<任意の文字列>*sh、awk、sed のいずれかに一致した文字列が書かれている行がマッチします（-w オプションがあると単語単位でマッチ）。また検索文字列（PATTERNS）に記号やスペースを含む場合は""で囲みます。

---

[11] メタ文字（metacharacter)クォートされていない場合に、単語区切りとなる文字。次の文字のうちのいずれかです: | & ; ( ) < > space tab

7.4. ディスクの空き容量確認／コマンド出力から項目取り出し（df、awk）

ディスクに空きがなくなるとシステムの動作が遅くなったり正常に動作しなくなります。大きなファイルや大量のファイルを持ち込む場合はディスクの空きを確認する必要があります。

例えば、/home ディレクトリに 1 Gbyte の資材をコピーし、1 Gbyte の空きを残したい場合...

```
var1=`df /home | awk 'NR>1{print $4}'`
if [ "$var1" -lt 2000000000 ]; then
  echo 容量不足
else
  echo ok
fi
```

＜df コマンドの出力＞

```
adminuser@ubuntu2004:~$ df /home
Filesystem                         1K-blocks    Used Available Use% Mounted on
/dev/mapper/ubuntu--vg-ubuntu--lv  63958684 5400600  55279444   9% /
①------------------------------  ②------ ③-----  ④空容量
```

2 行目の、スペースで区切られた 4 番目の項目で目的のディスクの空容量が分かります。

＜awk コマンド＞

```
awk 'NR>1{print $4}'
```

awk は標準入力から 1 行ずつ取り出し、パラメータで渡された処理を行います。

パラメータの文字列は条件と処理になっていて、NR は処理中の行番号を表す予約語です。NR>1 は処理中の行が 2 行目以降だったら {} の中の処理を行います。

$4 は、1 行を空白で分割した（awk が自動で行う）4 番目の項目を表し、pring で出力します。

＜シェル変数への代入＞

```
var1=`df /home | awk 'NR>1{print $4}'`
```

``（バッククォート）で一連の処理により得られた「df の出力した 2 行目の 4 番目の項目」が変数の$var1 に代入されます。

7.5. テキストファイルの一括更新処理（sed）

シェルでテキストファイルを取り扱うこともできます。単純な文字列置換は sed(Stream Editor)で行えます。同一の修正を複数のファイルに対して一括で行う場合は、以下のようにします。

----------<更新前のファイル>---------- ①請求書 1.txt、②請求書 2.txt を処理

```
adminuser@ubuntu2004:~$ cat 請求書*
        請求書①

A 工務店 様       発行日  2022 年 4 月 1 日

請求合計額 ￥100,000,000-
但し、2022 年 3 月分として

        請求書②

B 運送 様       発行日  2022 年 4 月 1 日

請求合計額 ￥100,000,000-
但し、2022 年 3 月分として
```

----------<更新後イメージ>----------

発行日と但し書きを一括して変えてファイルに書き戻す場合は、以下のようにします。

```
adminuser@ubuntu2004:~$ sed -i "s/2022 年 4 月/2022 年 5 月/g; s/2022 年 3 月/2022 年 4 月/g " 請求書*
adminuser@ubuntu2004:~$ cat 請求書*
        請求書①

A 工務店 様       発行日  2022 年 5 月 1 日

請求合計額 ￥100,000,000-

但し、2022 年 4 月分として


        請求書②

B 運送 様       発行日  2022 年 5 月 1 日

請求合計額 ￥100,000,000-

但し、2022 年 4 月分として
```

※最初は-i オプションを付けず標準出力にでる更新後イメージを確認します。また、複数個所を更新する場合は、sed “s/aaa/bbb/g; s/xxx/yyy/g”のように;で繋ぎます

7.6. テキストファイルの読み取り／パーミッションの変更（sed、chmod）

sed はテキストファイルを範囲（アドレス）指定で処理することができます。また編集後の文字列をコマンドとして実行することもできます。

以下、指示ファイルからファイル名と変更後パーミッションを読み込んで実行する例です。指示ファイルで使う区切り文字はカンマ(,)かタブ(¥t)のどちらかが使われることを想定しています。

----------<パーミッション変更前の状態>----------

```
adminuser@ubuntu2004:~$ ls -la test*
-rw-rw-r-- 1 adminuser adminuser 0  4月  5 13:51 testfile1
-rw-rw-r-- 1 adminuser adminuser 0  4月  5 13:51 testfile2
-rw-rw-r-- 1 adminuser adminuser 0  4月  5 13:51 testfile3
```

----------<指示ファイル（sedtest)>----------

```
adminuser@ubuntu2004:~$ cat sedtest
@update1
testfile1                   777
/home/adminuser/testfile2   666
./testfile3                 555
@update2
testfile1,444
testfile2,444
testfile3,444
@end
```

----------<sed コマンド …chmod 実行>----------

sedtest ファイルの@update2～@end の行から、chmod 444 testfile1 ...の文字列を作り実行します

```
sed  -n -r "/^@update2/,/^@.*/{/^@.*/! {s/(^.*)[,¥t](.*)$/chmod ¥2 ¥1/e}}" chmodF
          ①------範囲----- ②-除外-- ③-置換前文字列---④-置換-----⑤     入力ファイル
```

＜オプション＞

-n：読み込んだファイルの内容を標準出力にプリントするのを抑止します

-r：編集に拡張正規表現を使います

①：範囲指定(/*from*/,/*to*/)。行の先頭(^)から@update2 と書かれている行から、行の先頭(^)が@の行までを処理の対象にする（@の後の .* は「任意の文字(.)が任意の数(*)」に合致）

②：①の条件に合う入力行に対する {処理}

③：/^@.*/で行頭(^)が@の行以外(!)に対して{処理}を行う

④：置換前文字列を㋐行頭から任意文字列 ^.*、㋑カンマかタブ[¥t]、㋒その後の行末迄 .*$ に分けて㋐と㋑を保存し、利用して文字列を置き換えます。…(　) で囲んだ部分が保存されて後で¥1、¥2（1,2 は出現順）で参照できます

⑤：s/*置換前*/*置換後*/*フラグ* のフラグに e を指定すると置換後の文字列がコマンドとして実行されます。フラグが e ではなく p だと実行ではなく標準出力への出力になります。

----------<コマンド実行後のパーミッション>----------

```
adminuser@ubuntu2004:~$ ls -la test*
-r--r--r-- 1 adminuser adminuser 0  4月  5 13:51 testfile1
-r--r--r-- 1 adminuser adminuser 0  4月  5 13:51 testfile2
-r--r--r-- 1 adminuser adminuser 0  4月  5 13:51 testfile3
```


7.7. パーミッションの変更／シェル関数による実装（sed、chmod）

フラグを e から p に変えて前項の例を実行すると、

```
adminuser@ubuntu2004:~$ sed -n -r "/^@update2/,/^@.*/{/^@.*/! {s/(^.*)[,¥t](.*)$/chmod ¥2 ¥1/p}}" chmodF
chmod 444 testfile1
chmod 444 testfile2
chmod 444 testfile3
```

上記のコマンドが実行されたのが確認できますが、sed から直接実行したコマンドが確認できないことや対象のファイルが本当に存在するか、処理が正常に終わったかが確認できません。

シェル関数を作ってファイルの存在チェックを入れると以下のようになります。以下の例は、対話型シェルに展開（./source コマンド）して使うことを想定し、以下の機能も追加してあります。

・実行結果をターミナルとログファイルの両方に出力(tee コマンド)する

・実行前のパーミッションに戻せるように、chmod 前の状態をファイル(undo_)に保存する

① シェル関数(ファイル名：chmodproc.sh)

```
#
# sedtest ファイルを読み、引数で指定された範囲の chmod を実行します。
# 実行結果は、指定されたログファイルに出力します。
#
# 引数１　：(必須) モード設定<file 名 パーミッション>ファイルを指定
# 引数２　：(必須) 実行範囲 モード設定ファイル内の@xxxxx を指定
# 引数３　：(任意) ログファイル出力パス。指定がない場合はカレントディレクトリの do_chmod.log
#
# 戻り値　：実行範囲の全てが正常に更新できた場合に 0、パラメタ誤り 1、以外 2
# 環境変数：更新しません
#
do_chmod() {
#set -x
  #ログファイル仮設定
  local logf=./do_chmod_$$.log
  #ログファイルが指定され、存在しなかったら正常に作れるか確認
  if [ $# -eq 3 ]; then
    if [ -f "$file" ]; then
      :
    else
      touch "$3"
      local ret=$?
      if [ "$ret" -ne 0 ]; then
        echo "ログファイル$3 が作れませんでした。" |tee -a $logf
        return 1
      fi
      logf=$3
    fi
  fi

  if [ $# -gt 3 ] || [ $# -lt 2 ]; then
    echo "指定された引数は$#個です。モード設定ファイル、実行範囲、ログファイルを指定してください。"
|& tee -a $logf
    return 1
  fi
```



```
  local modf="$1"
  if [ ! -f "$modf" ]; then
    echo "モード設定ファイル$1 が見つかりません。" |& tee -a $logf
    return 1
  fi

  #モード設定／実行範囲の指定を配列に取り出し
  local farr=()
  mapfile -t farr< <(sed -n -r "/^$2/,/^@.*/{/^@.*/! {s/^([^¥t]+)[,¥t]+(.*)$/¥2,¥1/p}}" "$modf")

  #状態復元のファイル作成
  local undof=undo_${modf}.$$
  echo "$2" >> "$undof"
  #配列の数だけ繰返し
  for para in "${farr[@]}"; do
    local perm=$(echo "$para" | cut -d ',' -f 1)
    local file=$(echo "$para" | cut -d ',' -f 2)

    if [ ! -e "$file" ]; then
      echo "指定されたファイルが存在しません。$file" |& tee -a $logf
      return 2
    fi

    if [[ "$perm" =~ [0-7][0-7][0-7] ]]; then
      :
    else
      echo "パーミッションは 0-7 の 3 桁で指定してください。$file,$perm" |& tee -a $logf
      return 2
    fi

    local befmod=`stat -c "%a" "$file"`
    echo "$file" 変更前パーミッション "$befmod" |& tee -a $logf
    printf "%s¥t%s¥n"  "$file" "$befmod" >> "$undof"
    echo chmod "$perm" "$file" |& tee -a $logf
    chmod "$perm" "$file"  |& tee -a $logf
    ret=${PIPESTATUS[0]}
    if [ "$ret" -ne 0 ]; then return 2; fi

  done

  return 0
}
```

② chmod 対象ファイルの初期状態

```
adminuser@ubuntu2004:~$ ls -l test*
-rwxrwxrwx 1 adminuser adminuser 0  4 月  5 13:51 testfile1
-rwxrwxrwx 1 adminuser adminuser 0  4 月  5 13:51 testfile2
-rwxrwxrwx 1 adminuser adminuser 0  4 月  5 13:51 testfile3
```

③ モード設定ファイル

```
adminuser@ubuntu2004:~$ cat chmodF
@update1
testfile1,444
/home/adminuser/testfile2,550
./testfile3,660
```


④ 実行（関数の呼び出し）

```
adminuser@ubuntu2004:~$ . chmodproc.sh              ….(ピリオド)でスクリプトファイル展開
adminuser@ubuntu2004:~$ do_chmod chmodF @update1    …関数（do_chmod）呼び出し
```

```
----------<ターミナル出力>-----------
testfile1 変更前パーミッション 777
chmod 444 testfile1
/home/adminuser/testfile2 変更前パーミッション 777
chmod 550 /home/adminuser/testfile2
./testfile3 変更前パーミッション 777
chmod 660 ./testfile3
```

⑤ 変更前保存ファイル（undo_chmodF.*nnnn*）

```
adminuser@ubuntu2004:~$ cat undo_chmodF.28961
@update1
testfile1  777
/home/adminuser/testfile2  777
./testfile3   777
```

以上

