3. GNU と UNIX コマンド

3.1 コマンドライン操作

3.1.1 シェル

シェル (shell)

- ユーザからのコマンドを受け付け，必要なプログラムを実行するプログラム
- e.g. Bourne シェル (sh)，bash (Bourne Again Shell)，C シェル (csh)，tcsh，Korn シェル (ksh)，Z シェル (zsh)
  - **Bourne シェル**: UNIX の標準的なシェル
  - **bash**: Bourne を改良したもの．多くの Linux ディストリビューションの標準シェル．
  - **C シェル**: C 言語に似たスクリプトが利用できるシェル．
  - **tcsh**: C シェルを改良したもの．Linux で C シェルとして使われているもの．
  - **Korn シェル**: Bourne シェルを改良したもの．
  - **zsh**: ksh に bash や tcsh の機能を取り入れた高機能シェル．

参考:

- 利用可能なシェルは， `/etc/shells` ファイルで確認することができる．
- `chsh` コマンドを使ってデフォルトのシェルを変更することができる．

Ubuntu 20.04 の利用可能なシェル:

```
ai@ai-VirtualBox:~$ cat /etc/shells
# /etc/shells: valid login shells
/bin/sh
/bin/bash
/usr/bin/bash
/bin/rbash
/usr/bin/rbash
/bin/dash
/usr/bin/dash
```

CentOS 7 の利用可能なシェル:

```
[ai@localhost ~]$ cat /etc/shells
/bin/sh
/bin/bash
/usr/bin/sh
/usr/bin/bash
```

ログインシェル

- システムにログインした直後に起動されるシェル
- ユーザごとのログインシェルは，`/etc/passwd` ファイルに記述されている．

Ubuntu 20:

```
ai@ai-VirtualBox:~$ cat /etc/passwd | grep /home/ai
ai:x:1000:1000:Ai Tachibana,,,:/home/ai:/bin/bash
```

CentOS 7:

```
[ai@localhost ~]$ cat /etc/passwd | grep /home/ai
ai:x:1000:1000:Ai Tachibana:/home/ai:/bin/bash
```

- bash では，一般ユーザの場合 `$` が表示され，スーパーユーザ (root) の場合は `#` が表示される．
- 環境変数 `PS1` で，プロンプトの表示形式を自由に設定することができる．

3.1.2 シェルの基本操作と設定

bash には，コマンドラインの作業を効率よく行うためのさまざまな機能がある．

コマンドラインの基本操作

操作	説明
Tab	コマンドやディレクトリ名を補完する．
Ctrl + A	行の先頭へカーソルを移動する．
Ctrl + E	行の最後へカーソルを移動する．
Ctrl + D	カーソル部分を1文字削除する，ログアウトする．
Ctrl + H	カーソルの左を1文字削除する (Backspace と同じ)．
Ctrl + L	画面をクリアしてカレント行を再表示する．
Ctrl + C	処理を中断する．
Ctrl + S	画面への出力を停止する (キー入力を受け付けない)．
Ctrl + Q	画面への出力を再開する．
Ctrl + Z	処理を一時停止する．

ディレクトリの指定

bash では，ディレクトリを表す特殊記号 (**メタキャラクタ**) を使うことができる．

メタキャラクタ	説明
~	ホームディレクトリ
.	カレントディレクトリ
. .	1つ上のディレクトリ

3.1.3 シェル変数と環境変数

シェルはユーザと Linux システムとの対話をつかさどる．
⇒ ユーザに関する情報 (e.g. ユーザのホームディレクトリ，ログイン名等) を保持しなければならない．

Linux では，このような情報は変数に保存される．
この変数の有効範囲 (スコープ) によって，シェル変数と環境変数に分けられる．

シェル変数

- スコープがその変数を定義したシェル・プロセスのみ．
  - 1人のユーザが bash を複数起動していた場合，複数の bash プロセスが動作する．シェル変数の有効範囲は1つの bash プロセス上にとどまる．
  - 該当するシェル・プロセスを終了すると，シェル変数は失われる．
  - 別のシェルを新しく起動した場合，新しいシェルから元のシェルで定義した内容を参照することはできない．

環境変数

- その変数を定義したシェル上，およびそのシェルで実行されるプログラムにも引き継がれる変数．
- シェル変数を `export` コマンドでエクスポートすることで設定する．

```
(bash)
[ai@localhost ~]$ EV=world  <- 変数設定
[ai@localhost ~]$ SV=local  <- 変数設定
[ai@localhost ~]$ export EV <- エクスポート
[ai@localhost ~]$ bash      <- 新しいシェルを起動


(bash (子プロセス))
[ai@localhost ~]$ echo $EV
world   <- 変数 EV の内容が表示される．
[ai@localhost ~]$ echo $SV
        <- 変数 SV は子プロセスでは未定義．
[ai@localhost ~]$ EV='new world'    <- 変数 EV を再定義
[ai@localhost ~]$ exit              <- このシェルを終了
exit


(bash)
[ai@localhost ~]$ echo $EV
world   <- 子プロセスでの変更は繁栄されていない．
```

主な環境変数:

環境変数	説明
`EDITOR`	デフォルトのエディタのパス
`HISTFILE`	コマンド履歴を格納するファイル
`HISTFILESIZE`	`HISTFILE` に保存する履歴数
`HOME`	カレントユーザのホームディレクトリ
`HOSTNAME`	ホスト名
`LANG`	ロケール (言語処理方式)
`LOGNAME`	ログインシェルのユーザ名
`PATH`	コマンドやプログラムを検索するディレクトリリスト
`PS1`	プロンプトの表示文字列
`PS2`	複数行にわたる入力時のプロンプト
`PWD`	カレントディレクトリ
`TERM`	端末の種類
`USER`	現在のユーザ

Remark

- 端末
  - 入出力に特化した機器
    - キーボードから入力された文字をコンピュータに送る機能
    - コンピュータから送られてきた信号をディスプレイに表示する機能
  - 初期のコンピュータは，キーボードとディスプレイのセットを複数接続し，複数のユーザが同時に利用できるようになっていた．
- 端末エミュレータ
  - 端末をソフトウェアで実現したもの
    - e.g. GNOME-terminal, Konsole, TeraTerm

変数を定義する書式は以下．

```
変数名=値
```

- `=` の前後にスペースが入らないように注意．
- 変数名には，英字，数字，アンダーバーを使うことが可能．
  - 先頭の文字に数字を使うことはできない．
  - 大文字と小文字は区別される．
- 値にスペースが入る場合は，`"` (二重引用符) または `'` (単一引用符) で囲む．

定義された変数は `echo` コマンドを使って参照できる (`echo - display a line of text`)．

```
echo [文字列または$変数名]
```

e.g.

```
[ai@localhost ~]$ lpi='Linux Professional Institute'
[ai@localhost ~]$ echo $lpi
Linux Professional Institute
```

変数を削除するには， `unset` コマンドを使う．このとき，変数の先頭には `$` 記号をつけない．

```
unset 変数名
```

- 定義されている環境変数を一覧表示するには， `env` コマンドか， `printenv` コマンドを使う．
- 環境変数とシェル変数を両方表示したい場合は， `set` コマンドを使う．

シェル変数は，新たに起動したシェルから参照することはできない． `export` コマンドでエクスポートすることで参照できるようになる．

```
export 変数名[=値]
```

実行例:

```
[ai@localhost ~]$ VAR=lpic
[ai@localhost ~]$ echo $VAR
lpic    <- VAR 変数の内容の出力
[ai@localhost ~]$ bash  <- bash を新たに起動
[ai@localhost ~]$ echo $VAR
        <- 新しいシェルでは VAR 変数が定義されていないので何も表示されない．
[ai@localhost ~]$ exit
exit
[ai@localhost ~]$ export VAR    <- VAR 変数を export
[ai@localhost ~]$ bash  <- bash を新たに起動
[ai@localhost ~]$ echo $VAR
lpic    <- VAR が環境変数になったことで，表示される．
```

エクスポートと変数定義は1行で書くこともできる.

```
export VAR=lpic
```

Remark

- 定義には `$` は不要.
- 参照には `$` が必要.

3.1.4 環境変数 `PATH`

- プロンプトが表示されている状態でコマンドを入力すると，シェルはそのコマンド (プログラム) を実行する．
- コマンドには，**内部コマンド** と **外部コマンド** がある．
  - 内部コマンド: シェル自体に組み込まれているもの
  - 外部コマンド: 独立したプログラムとして存在するもの

e.g.

- `ls`： `/bin/ls` が実行される外部コマンド
- `cd`：シェルに組み込まれている内部コマンド

外部コマンドの場合，シェルはそのコマンドがどこに置かれているかを，環境変数 `PATH` に指定されたディレクトリを順に調べて見つけ出す．

- コマンドが置かれたディレクトリを環境変数 `PATH` に追加することを，「パスを通す」という．
- パスの通っていない場所に置かれているコマンドやプログラムを実行する場合，絶対パスまたは相対パスを指定する必要がある．
  - **絶対パス** 表記
    - 最上位のディレクトリ (`/`) から表記する方法．
    - システム内のファイル位置を一意に示す．
  - **相対パス** 表記
    - カレントディレクトリ (`.`) を基点とした相対位置で表す．

例えば，CentOS 7 の場合， `PATH` 変数に含まれているディレクトリは以下．

```
[ai@localhost ~]$ echo $PATH
/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/home/ai/.local/bin:/home/ai/bin
```

`PATH` 変数に含まれていないディレクトリにコマンドでも，絶対パスを指定すれば実行することができる (コマンドを実行する権限は必要)．

環境変数 `PATH` にパスを追加するには， `~/.bash_profile` などの環境設定ファイルの `PATH` 設定を修正するか，以下のコマンドを使用する．

```
PATH=$PATH:追加するディレクトリ名
```

e.g. `/opt/bin` ディレクトリを環境変数 `PATH` の末尾に追加:

```
[ai@localhost ~]$ PATH=$PATH:/opt/bin
[ai@localhost ~]$ echo $PATH
/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/home/ai/.local/bin:/home/ai/bin:/opt/bin
```

- シェルは，環境変数 `PATH` の先頭から順にディレクトリを検索する．
    - 同名のプログラムがあった場合は，環境変数 `PATH` の先頭に近い方のディレクトリに置かれているプログラムが実行される．

例えば，次のようにしてしまうとパスが通っているディレクトリが `/opt/bin` のみになり，外部コマンドが使えなくなる．

```
PATH=/opt/bin
```

通常，セキュリティ上の理由から，環境変数 `PATH` にはカレントディレクトリを含めない．カレントディレクトリにあるプログラムを実行するには，カレントディレクトリを意味する `./` を明示する．

3.1.5 コマンドの実行

コマンドラインは次のような要素から成り立っている．

```
コマンド オプション 引数
```

- コマンド
  - 実行可能なプログラムまたはスクリプト
- オプション
  - コマンドに対して動作を指示するスイッチ
  - ハイフン (- もしくは --) に続けて指定する．例外的にハイフンを必要としないコマンドもある．
- 引数
  - コマンドに渡す値．
  - 引数の有無で動作が変わるコマンド，引数を取らないコマンド，複数の引数が必要なコマンドなどがある．

コマンドは，1行に複数並べて実行できる．

複数のコマンドの実行制御:

コマンド	説明
`コマンド1; コマンド2`	コマンド1に続いてコマンド2を実行する
`コマンド1 && コマンド2`	コマンド1が正常に終了したときのみコマンド2を実行する
`コマンド1 || コマンド2`	コマンド1が正常に終了しなかった場合のみコマンド2を実行する
`(コマンド1; コマンド2)`	コマンド1とコマンド2を， ひとまとまりのコマンドグループとして実行する
`{ コマンド1; コマンド2; }`	現在のシェル内でコマンド1とコマンド2を実行する

実行例:

```
[ai@localhost ~]$ ls; ls -a
LPIC_workspace
.  ..  .bash_history  .bash_logout  .bash_profile  .bashrc  LPIC_workspace

[ai@localhost ~]$ cat .bash_profile | grep PATH && pwd
PATH=$PATH:$HOME/.local/bin:$HOME/bin
export PATH
/home/ai <- 正常に終了したので，pwd が実行される.

[ai@localhost ~]$ cat LPIC_workspace && echo $PATH
cat: LPIC_workspace: Is a directory
    <- 正常に終了しなかったので，echo $PATH は実行されない.

[ai@localhost ~]$ cat LPIC_workspace || echo 'This is not a file!'
cat: LPIC_workspace: Is a directory
This is not a file! <- 正常に終了しなかったので，echo ... が実行される.

[ai@localhost ~]$ cat .bash_profile | grep PATH || pwd
PATH=$PATH:$HOME/.local/bin:$HOME/bin
export PATH
    <- 正常に終了したので，pwd は実行されない.

[ai@localhost ~]$ (date; pwd; ls -a)
2021年 11月 20日 土曜日 15:48:20 JST
/home/ai
.  ..  .bash_history  .bash_logout  .bash_profile  .bashrc  LPIC_workspace

[ai@localhost ~]$ { date; pwd; ls -a; }
2021年 11月 20日 土曜日 15:48:47 JST
/home/ai
.  ..  .bash_history  .bash_logout  .bash_profile  .bashrc  LPIC_workspace
```

3.1.6 引用符

単一引用符 '

単一引用符の中は，全て文字列であると解釈される．

```
[ai@localhost ~]$ echo $LANG
en_US.UTF-8 <- 環境変数 LANG の内容が出力される.
[ai@localhost ~]$ echo '$LANG'
$LANG       <- 文字列 $LANG が出力される.
```

二重引用符 "

二重引用符内も文字列であるとみなされるが，二重引用符内に変数があれば，その変数の内容が展開される．
また，二重引用符内にバッククオーテーション (`) が使われていると，その中も展開される．

```
[ai@localhost ~]$ echo $LANG
en_US.UTF-8
[ai@localhost ~]$ echo "Locale: $LANG"
Locale: en_US.UTF-8
```

展開させたくない場合は，バックスラッシュ (\ ) を使う．バックスラッシュ直後の文字はすべて通常の文字とみなされる．バックスラッシュは **エスケープ文字** と呼ばれる．

```
[ai@localhost ~]$ echo "\$LANG: $LANG"
$LANG: en_US.UTF-8
```

バッククオーテーション `

- バッククオーテーション内にコマンドがあれば，コマンドを実行した結果が展開される．
- 変数の場合は，変数に格納されているコマンドを実行した結果が展開される．

```
[ai@localhost ~]$ echo "The current directory is `pwd`."
The current directory is /home/ai.
```

$(コマンド) を使ってもよい．バッククオーテーションはシングルクオーテーションと紛らわしいので，こちらの書き方の方が良い．

```
[ai@localhost ~]$ echo "The current directory is $(pwd)."
The current directory is /home/ai.
```

3.1.7 コマンド履歴

一度使ったコマンドをもう一度使ったり，一度だけ変更して使いたい場合は，bash の履歴機能を利用することができる．

history コマンド

- コマンド履歴が順に表示される．
- 古いものから順に番号がついているので，この履歴番号を直接指定して実行することもできる．
- 履歴番号を指定してコマンドを再度実行するには， `!履歴番号` のようにする．
- コマンド履歴は， `~/.bash_history` ファイルに保存されている．

```
[ai@localhost ~]$ history
    1  exit
    2  su -
    3  exit
    (略)
   57  echo "The current directory is $(pwd)."
   58  history
[ai@localhost ~]$ !57
echo "The current directory is $(pwd)." <- 実際に実行されるコマンドが表示される．
The current directory is /home/ai.
```

bash の履歴機能:

コマンド	内容
↑ (Ctrl + P)	1つ前のコマンドを表示する
↓ (Ctrl + N)	1つ次のコマンドを表示する
!文字列	実行したコマンドの中で，指定した文字列から始まるコマンドを実行する
!?文字列	実行したコマンドの中で，指定した文字列を含むコマンドを実行する
!!	直前に実行したコマンドを再実行する
!履歴番号	履歴番号のコマンドを実行する

3.1.8 マニュアルの参照

- Linux では，**オンラインマニュアルページ** (**man ページ**) が標準で用意されている．
  - man ページは， `man` コマンドで表示できる．
- マニュアルを構成するファイルは， `/usr/share/man` に置かれている．
  - man ページへの検索ディレクトリは，環境変数 `MANPATH` が参照される．
    - `MANPATH` に何も指定されていない場合は， `/etc/man.config` (`/etc/man.conf` ， `/etc/man_db.conf` ， `/etc/manpath.config` 等) に指定されたデフォルトのリストが使われる．
- `man` コマンドは，環境変数 `PAGER` で指定されたページやプログラム (通常は `less`) で表示を行うが，好みに応じて変更することができる．

```
man [オプション] [セクション] コマンド名あるいはキーワード
```

`man` コマンドの主なオプション:

オプション	説明
`-a`	すべてのセクションのマニュアルを表示する
`-f`	指定されたキーワード (完全一致) を含むドキュメントを表示する (`whatis` と同じ)
`-k`	指定されたキーワード (部分一致) を含むドキュメントを表示する (`apropos` と同じ)
`-w`	マニュアルの置かれているディレクトリを表示する

man ページの見出し:

見出し	説明
`NAME`	コマンドやファイルの名前と簡単な説明
`SYNOPSIS`	書式 (オプションや引数)
`DESCRIPTION`	詳細な説明
`OPTIONS`	指定できるオプションの説明
`FILES`	設定ファイルなど，関連するファイル
`ENVIRONMENT`	関連する環境変数

見出し	説明
NOTES	その他の注意事項
BUGS	既知の不具合
SEE ALSO	関連項目
AUTHOR	プログラムやドキュメントの著者

man ページの表示には， less コマンドというページャが使われている．

less の主なキー操作:

キー操作	説明
k，↑	上方向に1行スクロール
j，↓	下方向に1行スクロール
Space，f	下方向に1画面スクロール
b	上方向に1画面スクロール
q	終了
/検索文字列	下方向に文字列を検索
?検索文字列	上方向に文字列を検索
h	ヘルプを表示する

同一の名前で異なる内容を扱えるようにするため，セクション (章) が設定されている．**セクション**は，ドキュメントの内容による分類であり，Linux では次のようになっている．

セクション:

セクション	説明
1	ユーザコマンド
2	システムコール (カーネルの機能を使うための関数)
3	ライブラリ (C言語の関数)
4	デバイスファイル

セクション	説明
5	設定ファイル
6	ゲーム
7	その他
8	システム管理コマンド
9	Linux 独自のカーネル用コマンド

`whatis` コマンドを使うと，指定した検索キーワードと完全にマッチした一覧が表示される．

```
[ai@localhost ~]$ whatis crontab
crontab (1)          - maintains crontab files for individual users
crontab (5)          - files used to schedule the execution of programs
```

`apropos` コマンドを使うと，指定されたキーワードがマニュアルタイトルもしくは NAME 欄に含まれるマニュアルの項目一覧を表示する．

```
[ai@localhost ~]$ apropos crontab
anacrontab (5)       - configuration file for Anacron
crontab (1)          - maintains crontab files for individual users
crontab (5)          - files used to schedule the execution of programs
crontabs (4)         - configuration and scripts for running periodical jobs
```

参考:
シェルの内部コマンドの説明を表示するには， `man` コマンドではなく `help` コマンドを使う．

3.1.9 ファイル操作コマンド

`ls` コマンド

- ディレクトリを指定した場合，そのディレクトリ内のファイルを表示する．
- ファイル名を指定した場合，そのファイルの属性を表示する．
- 引数に何も指定しない場合，カレントディレクトリ内のファイルとサブディレクトリを表示する．

書式:

```
ls [option] [ファイル名またはディレクトリ名]
```

option:

オプション	説明
`-a`	．から始まるファイルも表示する．
`-A`	．から始まるファイルも表示するが，．と ．．は表示しない．
`-d`	ディレクトリ自身の情報も表示する．
`-F`	ファイルの種類も表示する．(`/` : ディレクトリ， `*` : 実行ファイル， `@` : シンボリックリンク)
`-i`	iノード番号を表示する．
`-l`	ファイルjの詳細な情報を表示する．
`-t`	日付順に表示する．
`-h`	単位付きで表示する．

```
[ai@localhost ~]$ ls -lA
total 20
-rw-------. 1 ai ai 1563 11月 20 22:43 .bash_history
-rw-r--r--. 1 ai ai   18  4月  1  2020 .bash_logout
-rw-r--r--. 1 ai ai  193  4月  1  2020 .bash_profile
-rw-r--r--. 1 ai ai  231  4月  1  2020 .bashrc
-rw-------. 1 ai ai   59 11月 20 16:45 .lesshst
drwxrwxr-x. 2 ai ai    6 11月 20 15:40 LPIC_workspace
```

`cp` コマンド

ファイルやディレクトリをコピーする．

書式:

```
cp [option] コピー元ファイル名 コピー先ファイル名

cp [option] コピー元ファイル名 コピー先ディレクトリ
```

option:

オプション	説明
`-f`	コピー先に同名のファイルがあれば上書きする．
`-i`	コピー先に同名のファイルがあれば，上書きするかどうか確認する．
`-p`	コピー元ファイルの属性を保持したままコピーする．
`-r, -R`	ディレクトリ内を再帰的にコピーする (ディレクトリをコピーする)．
`-d`	シンボリックリンクをシンボリックリンクとしてコピーする．
`-a`	できる限り元ファイルの構成と属性をコピー先でも保持する (`-dpR` と同じ)．

ディレクトリをコピーする場合，必ず `-r` または `-R` オプションをつける．

```
[ai@localhost LPIC_workspace]$ cp -p ~/.bashrc ./.bashrc.bk
[ai@localhost LPIC_workspace]$ ls -lA
total 4
-rw-r--r--. 1 ai ai 231  4月  1  2020 .bashrc.bk
```

`mv` コマンド

指定した場所にファイルやディレクトリを移動する．また，ファイル名の変更にも用いられる．

書式:

```
mv [option] 移動元ファイルまたはディレクトリ 移動先ファイルまたはディレクトリ
```

option:

オプション	説明
`-f`	移動先に同名のファイルがあれば上書きする．

オプション	説明
-i	移動先に同名のファイルがあれば上書きするするかどうか確認する.

```
[ai@localhost LPIC_workspace]$ ls -A
.bashrc.bk
[ai@localhost LPIC_workspace]$ mv .bashrc.bk .bashrc.bk_20211122
[ai@localhost LPIC_workspace]$ ls -A
.bashrc.bk_20211122
```

mkdir コマンド

空のディレクトリを作成する.

書式:

```
mkdir [option] ディレクトリ名
```

option:

オプション	説明
-m	指定したアクセス権でディレクトリを作成する.
-p	必要なら親ディレクトリも同時に作成する.

```
[ai@localhost LPIC_workspace]$ mkdir -m 700 3_dir
[ai@localhost LPIC_workspace]$ ls -l
total 0
drwx------. 2 ai ai 6 11月 22 15:53 3_dir


[ai@localhost LPIC_workspace]$ mkdir top/second/third
mkdir: cannot create directory ‘top/second/third’: No such file or directory
[ai@localhost LPIC_workspace]$ mkdir -p top/second/third
[ai@localhost LPIC_workspace]$ ls
3_dir  top
```

rm コマンド

ファイルやディレクトリを削除する.

書式:

```
rm [option] ファイル名
```

option:

オプション	説明
`-f`	ユーザへの確認なしに削除する.
`-i`	削除前にユーザに確認する.
`-r, -R`	サブディレクトリも含め，再帰的にディレクトリ全体を削除する.

```
[ai@localhost LPIC_workspace]$ ls
3_dir  top
[ai@localhost LPIC_workspace]$ rm -rf top/
[ai@localhost LPIC_workspace]$ ls
3_dir
```

`rmdir` コマンド

- 空のディレクトリを削除する.
    - ディレクトリ内にファイルやサブディレクトリが残っている場合は削除できない.

書式:

```
rmdir [option] ディレクトリ名
```

option:

オプション	説明
`-p`	複数階層の空ディレクトリを削除する.

```
[ai@localhost LPIC_workspace]$ ls
3_dir
[ai@localhost LPIC_workspace]$ rmdir 3_dir/
[ai@localhost LPIC_workspace]$ ls
```

`touch` コマンド

ファイルのタイムスタンプ (アクセス時刻と修正時刻) を現在時刻または指定した日時に変更する.

書式:

```
touch [option] ファイル名
```

option:

オプション	説明
`-t`	タイムスタンプを `[[CC]YY]MMDDhhmm[.SS]` に変更する (デフォルトは現在時刻). `CC`：西暦の上2桁 (省略可) `YY`：西暦の下2桁 (省略可) `MM`：月 `DD`：日 `hh`：時 (24時間表記) `mm`：分 `SS`：秒 (省略可，指定しない場合は `00`)
`-a`	アクセス時刻だけ変更する.
`-m`	修正時刻だけ変更する.

引数で指定したファイルが存在しない場合は，空のファイルを作成する.

```
[ai@localhost LPIC_workspace]$ touch -t 202111221111 touch_test
[ai@localhost LPIC_workspace]$ ls -l
total 0
-rw-rw-r--. 1 ai ai 0 11月 22 11:11 touch_test
```

`file` コマンド

ファイルの種類を表示する.
(プログラムなどのバイナリファイルを `cat` コマンドで開くと文字化けを起こしてしまう)

書式:

```
file ファイル名
```

```
[ai@localhost LPIC_workspace]$ file ~/.bashrc
/home/ai/.bashrc: ASCII text
```

3.1.10 メタキャラクタの利用

- メタキャラクタ
  - ファイル名のパターンを表す特殊な記号
  - メタキャラクタを使うと，パターンに一致する複数のファイルを一括して扱うことができる．

主なメタキャラクタ:

メタキャラクタ	説明
*	0文字以上の文字または文字列にマッチする．
?	任意の1文字にマッチする．
[]	[] 内に列挙されている文字のいずれか1文字にマッチする． 連続した文字列には - が使える．範囲指定の先頭に ! を使うと， マッチしない範囲を指定できる．
{}	, で区切られた文字列にマッチする．文字列の生成にも使える．

ここで，メタキャラクタを通常の文字として使いたい場合は，\ をメタキャラクタの直前に記述する．

```
[ai@localhost chapter3]$ ls
aaa  a.txt  bbb  b.txt  ccc  c.txt
[ai@localhost chapter3]$ ls *.txt
a.txt  b.txt  c.txt
[ai@localhost chapter3]$ ls a*
aaa  a.txt
```

3.2 パイプとリダイレクト

3.2.1 標準入出力

- Linux では，通常のファイルと同等にディスプレイへの出力やキーボードからの入力を扱うことができる．
  - キーボードからの入力もファイルの読み込みも同等に扱う．
  - ディスプレイへの出力もファイルへの書き出しも同等に扱う．
- データの入出力に伴うデータの流れを **ストリーム** と呼ぶ．
- Linux では，データをストリームとして扱うために，3つの基本的なインタフェースが定められている．

標準入出力:

番号	入出力名	デフォルト
0	標準入力	キーボード
1	標準出力	画面 (端末)
2	標準エラー出力	画面 (端末)

3.2.2 パイプ

- **パイプ** (パイプライン)
  - コマンドやプログラムの出力結果を，別のコマンドやプログラムの入力に渡すために使われる．
    - コマンドの標準出力を次のコマンドの標準入力に渡す．
  - `|` で表す．

```
[ai@localhost chapter3]$ ls | wc -l
6
```

`ls` コマンドの実行結果 (標準出力) を， `wc` コマンドの標準入力に渡す．

`tee` コマンド

コマンドの実行結果をファイルに保存するとともに，画面上にも表示したい場合，パイプだけでは解決できない．このような場合に `tee` コマンドを使う．

- `tee` コマンド
  - 標準入力から読み込み，それをファイルと標準出力へとT字型に分岐させる．
    - 実行結果をファイルに書き込みつつ，次のコマンドへと実行結果を渡すことができる．

書式:

```
tee [option] ファイル名
```

option:

オプション	説明
`-a`	ファイルに上書きするのではなく，追記する．

```
[ai@localhost chapter3]$ ls -l /usr/bin | tee ls_usr_bin | wc -l
735
```

3.2.3 リダイレクト

コマンドの実行結果を画面上に表示するのではなくファイルに保存したい場合や，コマンドへの入力にあらかじめ用意しておいたファイルを使う場合に役立つ．

リダイレクトとパイプ:

書式	説明
`コマンド > ファイル`	コマンドの標準出力 (実行結果) をファイルに書き込む
`コマンド < ファイル`	ファイルの内容をコマンドの標準入力へ送る
`コマンド >> ファイル`	コマンドの標準出力 (実行結果) をファイルに追記する
`コマンド 2> ファイル`	ファイルに標準エラー出力を書き込む
`コマンド 2>> ファイル`	ファイルに標準エラー出力を追記する
`コマンド > ファイル 2>&1`	ファイルに標準出力と標準エラー出力を書き込む
`コマンド >> ファイル 2>&1`	ファイルに標準出力と標準エラー出力を追記する
`コマンド << 終了文字`	終了文字が現れるまで標準入力へ送る
`コマンド1 | コマンド2`	コマンド1の標準出力をコマンド2の標準入力に渡す
`コマンド1 2>&1 | コマンド2`	コマンド1の標準出力と標準エラー出力を， コマンド2の標準入力に渡す
`コマンド1 | tee ファイル | コマンド2`	コマンド1の標準出力 (実行結果) を， コマンド2の標準入力に渡すとともにファイルに書き込む
`コマンド &> ファイル`	標準出力と標準エラー出力を同じファイルに書き込む

参考:
コマンドが出力するメッセージをいっさい画面上に出力したくない場合は， `コマンド > /dev/null 2>&1` とする． `/dev/null` は特殊なファイルで，入力されたすべてのデータを消し去る．

```
[ai@localhost chapter3]$ ls -l /etc/ > ls_etc
```

3.3 テキスト処理フィルタ

Linux では，テキストデータを処理するためのフィルタとなるコマンドが多数ある．シェル上でコマンドを組み合わせることで，強力なデータ処理が可能になる．

参考:
Linux システムでのフィルタとは，ファイルやテキストデータを読み込んで何らかの処理を行って出力するプログラムのこと．

3.3.1 テキストフィルタコマンド

`cat` コマンド

- ファイルの内容を標準出力に出力する．
  - シェルのリダイレクトを使って複数のファイルを結合するといった使い方もできる．

書式:

```
cat [option] [ファイル名]
```

option:

オプション	説明
`-n`	各行の左端に行番号を付加する

例えば， `file1` と `file2` を1つの `newfile` にまとめるには，以下のようにする．

```
cat file1 file2 > newfile
```

参考:
`cat` コマンドは引数を省略した場合，標準入力からの入力を受け付けるようになる．

`nl` コマンド

テキストファイルの一部または全部に行番号を付けて表示する．

書式:

```
nl [option] [ファイル名]
```

option:

オプション	説明
`-b 形式`	指定した形式で本文に行番号を付加する
`-h 形式`	指定した形式でヘッダに行番号を付加する
`-f 形式`	指定した形式でフッタに行番号を付加する

形式	説明
`a`	全ての行
`t`	空白以外の行
`n`	行以外の付加を中止

```
[ai@localhost chapter3]$ nl -b a ~/.bash_logout
     1  # ~/.bash_logout
     2
```

`od` コマンド

バイナリファイルの内容を8進数や16進数で表す.

書式:

```
od [option] [ファイル名]
```

option:

オプション	説明
`-t 出力タイプ`	出力するフォーマットを指定する

出力タイプ	説明
c	ASCII 文字
o	8進数 (デフォルト)
x	16進数

```
[ai@localhost chapter3]$ file /etc/localtime
/etc/localtime: symbolic link to `../usr/share/zoneinfo/Asia/Tokyo'
[ai@localhost chapter3]$ od -t x /etc/localtime
0000000 66695a54 00000032 00000000 00000000
0000020 00000000 03000000 03000000 00000000
0000040 08000000 03000000 08000000 70023ed7
0000060 f059edd7 70faf8d8 f03bcdd9 f00007db
0000100 f01daddb f0e2e6dc f0ff8cdd 01000100
0000120 01000100 a08c0000 00000001 0400907e
0000140 907e0000 444a0400 534a0054 00000054
0000160 01000001 66695a54 00000032 00000000
0000200 00000000 00000000 04000000 04000000
0000220 00000000 09000000 04000000 0c000000
0000240 ffffffff 70a4c265 ffffffff 70023ed7
0000260 ffffffff f059edd7 ffffffff 70faf8d8
0000300 ffffffff f03bcdd9 ffffffff f00007db
0000320 ffffffff f01daddb ffffffff f0e2e6dc
0000340 ffffffff f0ff8cdd 01020103 01020102
0000360 83000002 00000003 01a08c00 7e000004
0000400 00080090 00907e00 544d4c08 54444a00
0000420 54534a00 00000000 00000001 534a0a01
0000440 0a392d54
0000444
```

head コマンド

ファイルの先頭部分を表示する．オプションを指定しない場合，先頭から10行目までを表示する．

書式:

```
head [option] [ファイル名]
```

option:

オプション	説明
-n 行数	先頭から指定された行数分だけ表示する
-行数	先頭から指定された行数分だけ表示する (非推奨)
-c バイト数	出力するバイト数を指定する

```
[ai@localhost chapter3]$ head -n 5 /etc/services
# /etc/services:
# $Id: services,v 1.55 2013/04/14 ovasik Exp $
#
# Network services, Internet style
# IANA services version: last updated 2013-04-10
```

`tail` コマンド

- ファイルの末尾部分を表示する．
- デフォルトでは，最終行から10行が表示される．
- `-f` オプションを使うと，ファイルの末尾に行が追加されるとリアルタイムで表示する．

書式:

```
tail [option] [ファイル名]
```

option:

オプション	説明
`-n` 行数	末尾から指定された行数分だけ表示する
-行数	末尾から指定された行数分だけ表示する (非推奨)
`-c` バイト数	出力するバイト数を指定する
`-f`	ファイルの末尾に追加された行を表示し続ける

```
[root@localhost ~]# tail -n 5 -f /var/log/messages
Nov 22 17:33:03 localhost systemd: Started Network Manager Script Dispatcher Service.
Nov 22 17:33:03 localhost nm-dispatcher: req:1 'dhcp4-change' [eth0]: new request (2 scripts)
Nov 22 17:33:03 localhost nm-dispatcher: req:1 'dhcp4-change' [eth0]: start running ordered scripts...
Nov 22 17:45:56 localhost su: FAILED SU (to root) ai on pts/0
Nov 22 17:46:01 localhost su: (to root) ai on pts/0
```

`cut` コマンド

ファイルの各行から指定したフィールドを取り出す． `-c` オプションで，何文字目から取り出すかを指定する．

書式:

```
cut [option] [ファイル名]
```

option:

オプション	説明
`-c 文字数`	取り出す文字位置を指定する
`-d 区切り文字`	フィールドの区切り文字 (デリミタ) を指定する (デフォルトはタブ)
`-f フィールド`	取り出すフィールドを指定する

```
[ai@localhost chapter3]$ cat /etc/resolv.conf
# Generated by NetworkManager
nameserver 192.168.122.1
[ai@localhost chapter3]$ cut -c 5 /etc/resolv.conf
n
s
[ai@localhost chapter3]$ cut -c 1-7 /etc/resolv.conf
# Gener
nameser
```

デリミタ: フィールド間を区切る文字

`paste` コマンド

1つ以上のファイルを読み込んで，行ごとに水平方向に連結する．連結するときの区切り文字は，デフォルトではタブになっている．

書式:

```
paste [option] ファイル名1 ファイル名2 ...
```

option:

オプション	説明
`-d 区切り文字`	区切り文字 (デリミタ) を指定する

```
[ai@localhost chapter3]$ cat sample1.txt
aaaa
bbbb
[ai@localhost chapter3]$ cat sample2.txt
AAAA
BBBB
[ai@localhost chapter3]$ paste -d ";" sample1.txt sample2.txt
aaaa;AAAA
bbbb;BBBB
```

`tr` コマンド

標準入力から読み込まれた文字列を変換したり，削除したりする．

書式:

```
tr [option] [文字列1 [文字列2]]
```

option:

オプション	説明
`-d`	`[文字列1]` でマッチした文字列を削除する
`-s`	連続するパターン文字列を1文字として処理する

クラス	説明
`[:alpha:]`	英字
`[:lower:]`	英小文字
`[:upper:]`	英大文字
`[:digit:]`	数字
`[:alnum:]`	英数字
`[:space:]`	スペース

```
[ai@localhost chapter3]$ cat /etc/hosts
127.0.0.1   localhost localhost.localdomain localhost4 localhost4.localdomain4
::1         localhost localhost.localdomain localhost6 localhost6.localdomain6
[ai@localhost chapter3]$ cat /etc/hosts | tr 'a-z' 'A-Z'
127.0.0.1   LOCALHOST LOCALHOST.LOCALDOMAIN LOCALHOST4 LOCALHOST4.LOCALDOMAIN4
::1         LOCALHOST LOCALHOST.LOCALDOMAIN LOCALHOST6 LOCALHOST6.LOCALDOMAIN6
```

これは，次のようにクラスを用いて書くこともできる．

```
[ai@localhost chapter3]$ cat /etc/hosts | tr [:lower:] [:upper:]
127.0.0.1   LOCALHOST LOCALHOST.LOCALDOMAIN LOCALHOST4 LOCALHOST4.LOCALDOMAIN4
::1         LOCALHOST LOCALHOST.LOCALDOMAIN LOCALHOST6 LOCALHOST6.LOCALDOMAIN6
```

`sort` コマンド

行単位でファイルの内容をソートする．デフォルトでは昇順にソートする．

書式:

```
sort [option] [+開始位置 [-終了位置]] [ファイル名]
```

option:

オプション	説明
`-b`	行頭の空白は無視する
`-f`	大文字小文字の区別を無視する
`-r`	降順にソートする
`-n`	数字を文字としてではなく数値として処理する
`-k`	ソートの鍵の列の指定

```
[ai@localhost chapter3]$ cat score
yoshinori kawazu        85
keiichi oka     70
toru minemura   100
[ai@localhost chapter3]$ sort -n -k 3 score
keiichi oka     70
yoshinori kawazu        85
toru minemura   100
```

`split` コマンド

- 指定されたサイズでファイルを分割する．
- デフォルトでは，1000行ごとに複数ファイルに分割する．
- 分割されたファイルには，ファイルの末尾に `aa`，`ab`，`ac`，... が付加される．

書式:

```
split [option] [入力ファイル名 [出力ファイル名]]
```

```
[ai@localhost chapter3]$ split -5 sample.txt s_sample.
[ai@localhost chapter3]$ ls *sample.*
sample.txt  s_sample.aa  s_sample.ab  s_sample.ac
```

`uniq` コマンド

- 入力されたテキストストリームの中で重複している行を調べて，重複している行は1行にまとめて出力する．
  - 入力するテキストストリームはソートしておく必要がある．多くの場合は `sort` コマンドとパイプで組み合わせて使われる．
- テキストストリーム
  - ひとまとまりのテキストデータ．

書式:

```
uniq [option] [入力ファイル [出力ファイル]]
```

option:

オプション	説明
`-d`	重複している行のみ出力する
`-u`	重複していない行のみ出力する

```
[ai@localhost chapter3]$ cat file1
111
222
333
222
[ai@localhost chapter3]$ sort file1 | uniq
111
222
333
```

`wc` コマンド

- ファイルの行数，単語数，文字数を表示する．
  - オプションを省略すると，行数，単語数，文字数を表示する．

書式:

```
wc [option] [ファイル名]
```

option:

オプション	説明
`-c`	文字数 (バイト数) を表示する
`-l`	行数を表示する
`-w`	単語数を表示する

```
[ai@localhost chapter3]$ wc /etc/services
 11176  61033 670293 /etc/services
```

`xargs` コマンド

- 標準入力から受け取った文字列を引数に指定して，与えられたコマンドを実行する．

書式:

```
xargs コマンド
```

`xargs` コマンドの利点は，引数の数が多すぎた場合でも処理できること．

3.3.2 ファイルのチェックサム

ハッシュ関数 を使うと，ファイルを一方向に暗号化し，一定の長さの文字列にすることができる．

ハッシュ値を出力するコマンド:

コマンド	説明
`md5sum`	MD5 によるハッシュ値を出力する
`sha1sum`	SHA1 によるハッシュ値を出力する
`sha256sum`	SHA256 によるハッシュ値を出力する
`sha512sum`	SHA512 によるハッシュ値を出力する

```
[ai@localhost chapter3]$ md5sum sample.txt
1aa310e1cca0ffe92f5e3beef7888813  sample.txt
[ai@localhost chapter3]$ echo "a" >> sample.txt
[ai@localhost chapter3]$ md5sum sample.txt
bb71862c67a3cb0360cd1eddadad88a6  sample.txt
```

3.4 正規表現を使ったテキスト検索

3.4.1 正規表現

- **正規表現**
  - 特定の条件を表す文字列を抽象的に表現したもの

主な正規表現:

メタキャラクタ	説明
.	任意の1文字
*	直前の文字の0文字以上の繰り返し
[]	[] 内の文字のいずれか1文字 - : 範囲を指定 ^ : 先頭にあるときは「〜以外」
^	行頭
$	行末
\	次にくる文字をメタキャラクタではなく通常の文字として処理する
\s	スペース

3.4.2 `grep` コマンド

- ファイルやテキストストリームの中に，正規表現によって表される検索文字列があるかどうかを調べる．
- 引数にファイルを指定した場合，そのファイルの中で検索パターンにマッチした文字列が含まれる行を全て表示する．
- ファイルは複数指定することができる．

書式:

```
grep [option] 検索パターン [ファイル名]

grep [option] [-f ファイル名] [ファイル名]
```

option:

オプション	説明
`-c`	パターンがマッチした行の行数だけ表示する
`-f`	検索パターンをファイルから読み込む
`-i`	大文字小文字を区別せずに検索する
`-n`	検索結果と合わせて行番号も表示する
`-v`	パターンがマッチしない行を表示する
`-E`	拡張正規表現を使用する

```
[ai@localhost chapter3]$ grep -i s sample.txt
ls
ssh
csv
emacs
```

拡張正規表現 を使うには， `-E` オプションをつけるか， `egrep` コマンドを使う．

拡張正規表現:

メタキャラクタ	説明
`+`	直前の文字の1回以上の繰り返し

メタキャラクタ	説明
?	直前の文字の0回もしくは1回の繰り返し
|	左右いずれかの記述にマッチする
{n}	直前の文字のn回の繰り返し
{n,m}	直前の文字のn回からm回の繰り返し

例えば， 22/tcp もしくは 53/tcp が含まれる行を /etc/services から検索するには，以下のようにする．

```
[ai@localhost chapter3]$ egrep '\s(22|53)/tcp' /etc/services
ssh             22/tcp                          # The Secure Shell (SSH) Protocol
domain          53/tcp                          # name-domain server
```

検索パターンに正規表現を使わない場合， fgrep コマンドを使う．

```
fgrep '.*' sample.txt
grep '\.\*' sample.txt
```

これら2つのコマンドは同じ意味．

3.4.3 sed コマンド

- テキストストリームに対して処理を行う．
  - 編集する内容をコマンドやスクリプトとして sed に支持しておき， sed はその指示に基づいてストリームの編集を行い，標準出力に編集結果を書き出す．

書式:

```
sed [option] コマンド [ファイル]
指定したファイルに対してコマンドを適用する．
ファイルが指定されなかった場合は，標準入力から読み込まれる．

sed [option] -e コマンド1 [-e コマンド2 ...] [ファイル]
複数のコマンドを適用する．複数のコマンドを指定するときは，-e オプションをコマンドごとに使う．

sed [option] -f スクリプト [ファイル]
コマンドを記述したスクリプトファイルを指定する．
sed コマンドはスクリプトファイルを読み込んで，そこに記述されたコマンドを適用する．
```

option:

オプション	説明
-f ファイル	コマンドが書かれたスクリプトファイルを指定する
-i	処理した内容でファイルを上書きする

sed コマンド内で指定できる主なコマンド:

コマンド	説明
d	マッチした行を削除する
s	パターンに基づいて置換する． g スイッチを使うと，マッチ箇所全てを置換する
y	文字を変換する

d コマンド

file1.txt の1行目から5行目までを d コマンドで削除して file2.txt に保存する例:

```
[ai@localhost chapter3]$ cat file1.txt
Bifurcation
Transcritical bifurcation
Saddle-node bifurcation
Pitchfork bifurcation
Equilibrium points
Stable
Asymptotically stable
Unstable
Stable node
[ai@localhost chapter3]$ sed '1,5d' file1.txt > file2.txt
[ai@localhost chapter3]$ cat file2.txt
Stable
Asymptotically stable
Unstable
Stable node
```

`sed` に `-i` オプションを指定していないので，元のファイルを変更しない．

`s` コマンド

検索パターンにマッチする部分を置換パターンに置き換える:

```
s/検索パターン/置換パターン/
```

```
[ai@localhost chapter3]$ cat test_s.txt
red hat linux
vine linux
linux linux linux
[ai@localhost chapter3]$ sed s/linux/LINUX/ test_s.txt
red hat LINUX
vine LINUX
LINUX linux linux
```

- `s` コマンドでは，検索パターンにマッチする部分が1行に複数あっても，最初にマッチした部分だけを置換する．
- マッチする部分を全て置換するには， `g` スイッチを最後に記述する．

```
[ai@localhost chapter3]$ sed s/linux/LINUX/g test_s.txt
red hat LINUX
vine LINUX
LINUX LINUX LINUX
```

`y` コマンド

ストリーム中に検索文字にマッチする文字があった場合，その文字を置換文字の同じ位置の文字に置き換える:

```
y/検索文字/置換文字/
```

```
[ai@localhost chapter3]$ cat test_y.txt
AaBbCcDdEdFfGg
[ai@localhost chapter3]$ sed y/ABC/123/ test_y.txt
1a2b3cDdEdFfGg
```

3.5 ファイルの基本的な編集

テキストファイルを編集するには，テキストエディタを使う．
Linux では，vi, Emacs, nano などが使える．

3.5.1 エディタの基本

Linux でシステムやサービスの設定を変更する方法:

- 設定ファイル (テキストファイル) を編集する
- コマンドを実行する

vi エディタはほとんどの Linux/UNIX 環境にインストールされているため，vi エディタの扱いに慣れておくことが求められる．

環境変数 EDITOR にエディタのパスを設定しておくと，デフォルトのエディタを設定できる．

```
export EDITOR=/usr/bin/vim
```

とすると，デフォルトのエディタを vim にできる．

3.5.2 vi エディタの基本

vi エディタの特徴は，コマンドモード，入力モードという2つの動作モードを切り替えながら使う点である．

viの起動:

```
vi [-R] [ファイル名]
```

- ファイル名を指定しなければ，空の新規ファイルが開く．
- `-R` オプションを指定すると，読み取り専用モードでファイルを開く．

新しくテキストを入力するには，テキストを入力したい位置にカーソルを移動し，入力モードに切り替える．

vi の入力モード:

コマンド	説明
`i`	カーソルの前にテキストを入力する
`a`	カーソルの後にテキストを入力する
`I`	行頭の最初の文字にカーソルを移動し，その直前にテキストを入力する
`A`	行末にカーソルを移動し，その直後にテキストを入力する
`o`	カレント行の下に空白行を挿入し，その行でテキストを入力する
`O`	カレント行の上に空白行を挿入し，その行でテキストを入力する

入力モードで Esc キーを押すと，コマンドモードに切り替わる．

vi のカーソル操作:

コマンド	説明
`h`	1文字左へ移動する．左矢印キーと同じ．
`l`	1文字右へ移動する．右矢印キーと同じ．
`k`	1行上へ移動する．上矢印キーと同じ．
`j`	1行下へ移動する．下矢印キーと同じ．

コマンド	説明
`0`	行の先頭へ移動する.
`$`	行の末尾へ移動する.
`H`	画面の一番上の行頭へ移動する.
`L`	画面の一番下の行頭へ移動する.
`gg`	ファイルの先頭行へ移動する.
`G`	ファイルの最終行へ移動する.
`n G`	ファイルのn行目に移動する.
`: n`	ファイルのn行目に移動する.

これらのコマンドの前に数字を入力すれば，その回数分コマンドが繰り返される． `5h` とすると，左へ5文字移動する.

vi の終了，ファイル保存，シェルコマンドの実行:

コマンド	説明
`:q`	ファイルを保存せずに終了する (編集した場合は保存するかどうかを確認してくる)
`:q!`	編集中の内容を保存せずに終了する
`:wq`	編集中の内容を保存して終了する
`ZZ`	編集中の内容を保存して終了する (`:wq` と同じ)
`:w`	編集中の内容でファイルを上書き保存する
`:e!`	最後に保存した内容に復帰する
`:r ファイル名`	ファイルの内容をカレント行以降に書き込む
`:!コマンド`	viを終了せずにシェルコマンドを実行する
`:r!コマンド`	シェルコマンドの実行結果を挿入する

例えば，vi の実行中に `:!ls` とすると，カレントディレクトリのファイル一覧を確認できる.

```
[ai@localhost chapter3]$ vi

[No write since last change]
aaa    b.txt  file1      ls_etc       sample2.txt  s_sample.aa  test_s.txt
a.txt  ccc    file1.txt  ls_usr_bin   sample.txt   s_sample.ab  test_y.txt
bbb    c.txt  file2.txt  sample1.txt  score        s_sample.ac

Press ENTER or type command to continue
```

vi の編集コマンド:

コマンド	説明
x	カーソル位置の文字を削除する (Delete)
X	カーソル位置の手前の文字を削除する (Backspace)
dd	カレント行を削除する
dw	カーソル位置から次の単語までを削除する
yy	カレント行をバッファにコピーする
p	カレント行の下にバッファの内容を貼り付ける
P	カレント行の上にバッファの内容を貼り付ける
r	カーソル位置の1文字を置換する

これらのコマンドも数値による回数指定ができる.

vi の検索コマンド:

コマンド	説明
/パターン	カーソル位置から後方に向かって指定したパターンを検索する
?パターン	カーソル位置から前方に向かって指定したパターンを検索する
n	次を検索する
N	次を検索する (逆方向)
:noh	候補のハイライト表示を解除する
:%s/A/B/	最初に見つかった文字列Aを文字列Bに置換する

コマンド	説明
`:%s/A/B/g`	すべての文字列Aを文字列Bに置換する

`:set` を使って vi の設定を変更することができる．

vi の設定変更:

コマンド	説明
`:set nu`，`:set number`	行番号を表示する
`:set nonu`，`:set nonumber`	行番号を非表示にする
`:set ts=タブ幅`	タブ幅を数値で指定する

参考:
`:set` で行った変更は，vi を終了させると消えてしまう．設定を常に有効するには，ホームディレクトリの `.exrc` ファイル (vim を使っている場合は `.vimrc` ファイル) に設定を記述する．その場合，`:` は除く．