【共通】Linux 基礎研修

講義の目的と目標

目的

みなさんが普段目にする Web サイトや Web アプリケーションはどこで動いているか、ご存じでしょうか。

研修や検証・試験等の目的であれば自分の PC 内で動かせばよいですが、多くの人にサービスを提供しようとすると普通の PC では足りません。

よりパワーと安定性のある「サーバー」という機械の上でアプリケーションを動かすことで、世界中の人にサービスを提供することができるようになります。

サーバー上にアプリケーションを置く、ということは

・開発したアプリケーションを動かすため、サーバーにデプロイする。
・何かエラーが起きたときにサーバーに入って原因を探る。
・アプリケーション開発に今や必要不可欠なツールである「Git」の操作を行う。

といったことを行わなければならないため、Linux の操作技術は必須で身につけなければいけない技術となります。

逆にこれができないと、どれだけコーディングができても業務に支障をきたす可能性があります。

目標

本研修では、演習を通し Linux に触れることで、基本的なコマンドの使い方を学びます。

Linux を mac や windows と同じように、ディレクトリ移動、ファイル作成、読み書きなどが当たり前にできるようになることが目標です。

Linux とは

Linux とは

前項でも説明しましたが、開発したアプリケーションや Web サイトは「サーバー」に乗せてサービスを提供します。この「サーバー」の中で最も多くのシェアを獲得しているのが Linux と呼ばれる OS です。

この Linux のコードはオープンソースとしてリリースされ無償で配布されています。そのため、システム導入時のコストを圧倒的に低く抑えることができます。

また、世界中の数万人規模の技術者がバグの発見などをしていて、その修正はインターネット上のコミュニティにより商用 OS を上回る速度で行われています。

一般に Linux をはじめとするオープンソースのソフトウェアはセキュリティ対策が非常に迅速です。

Linux 基礎

ログイン方法

ターミナルから EC2 インスタンスに SSH 接続する

Mac なら**ターミナル**から、Windows なら **WSL2** から AWS 上の EC2 インスタンスにアクセスする

1．事前に配布された「learnWEB_key.pem」をデスクトップに配置する。

2．Mac なら「ターミナル」を、Windows なら「WSL2」を起動する。

3．下記コマンドを実行する

```
cd

chmod 600 ~/Desktop/learnWEB_key.pem

ssh -i ~/Desktop/learnWEB_key.pem ec2-user@研修用 EC2 の IP アドレス
```

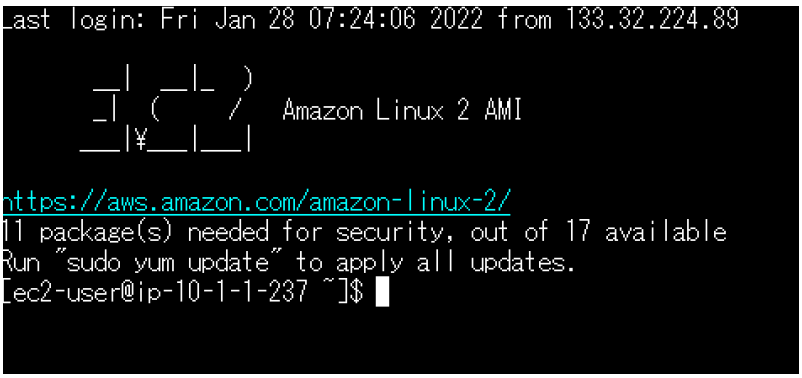
WSL2 の場合は↓

```
cd

chmod 600 /mnt/c/Users/ユーザー名/Desktop/learnWEB_key.pem

ssh -i /mnt/c/Users/ユーザー名/Desktop/learnWEB_key.pem ec2-user@研修用 EC2 の IP アドレス
```

4．上記コマンドの結果、下記が表示されたら接続 OK

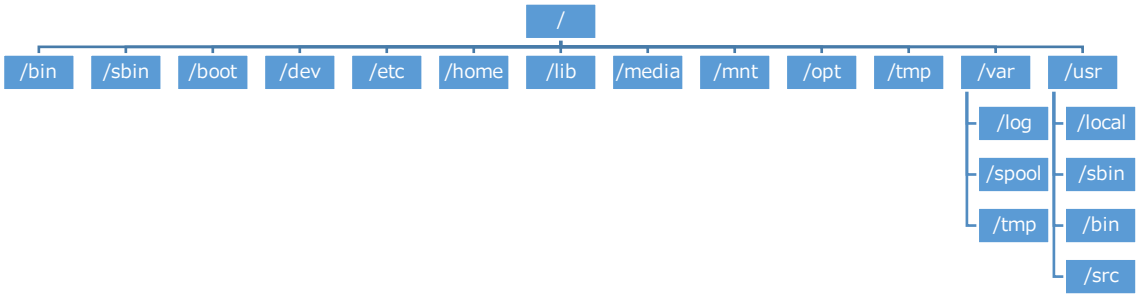


ファイル構造

Linux では Windows でいう『フォルダ』にあたるものを『ディレクトリ』といいます。また、Linux は /（ルート） ディレクトリを頂点とするツリー構造になっており、全てのファイルやサブディレクトリは / ディレクトリの中に収められています。

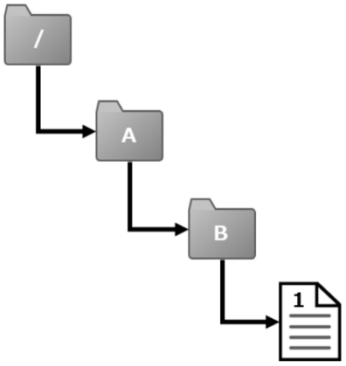
代表的なものとして以下があります。

/home	ユーザのホームディレクトリ
/etc	システム全体に関わる設定ファイルが格納されている
/bin	一般ユーザが扱うようのコマンドが格納
/sbin	システム管理系のコマンドが格納
/usr	ユーティリティやアプリケーションが格納される
/var	内容が常に可変するデータが格納される。例えばログファイルとか
/tmp	一時的なファイル置き場



絶対パスと相対パスについて

あるファイルを指定するときに、ファイル構造の頂点＝「/(ルート)ディレクトリ」からそのファイルまでの道筋(パス)を完全に記述する形式を絶対パス（あるいはフルパス）といいます。また、カレントディレクトリ(現在の場所)からそのファイルまでのパスを記述する形式を相対パスといいます。



例えば、カレントディレクトリが /A であるとき、
右図の「1」の表し方は 2 通りあり、
絶対パス：/A/B/1
相対パス：B/1
のようになります。

パスを指定する場合に注意する事は、先頭が“/”で始まると『絶対パス』になるということです。

また、相対パスを指定する際、“.”は『カレントディレクトリ（自分が今いるディレクトリ）』を表し、“..”は『カレントディレクトリより 1 階層上のディレクトリ』を表します。
/A/B/ ディレクトリから見た場合、それぞれ以下のようになります。

. : /A/B/
.. : /A/

Tips

便利なキー

Ctrl + a	入力中の先頭にカーソルを移動
Ctrl + e	入力中の最後にカーソルを移動
Ctrl + u	カーソルから前の文字をすべて削除
Ctrl + k	カーソルから後ろの文字をすべて削除
Ctrl + w	カーソルから前の文字を単語単位で削除
Ctrl + l	ターミナルの画面をクリアする
Ctrl + r	過去のコマンドから指定の文字列で検索

コマンド実践

コマンドの構文

コマンドは下記構文で構成されており、オプションや引数はそれぞれのコマンドごとに様々なものが用意されています。

そのため、そのコマンドのオプションや引数はどういう意味があるのか、しっかり理解する必要があります。

```
$ コマンド名 -オプション 第１引数 第２引数

例）
$ cp -r /home/yamada /home/suzuki
上記の場合は、コマンド「cp」オプション「-r」
第１引数「/home/yamada」第２引数「/home/suzuki」となります。
```

ファイル・ディレクトリの一覧を表示（**ls**）

ls コマンドを使うことでファイル・ディレクトリの一覧を表示させることができます。

```
$ ls              引数無しで実行すると現在の場所の一覧を表示してくれます
$ ls -l           -l オプションは詳細な情報を表示してくれます
$ ls -a           隠しファイル・ディレクトリも含めて表示してくれます
$ ls /home/yamada          引数に与えたディレクトリの一覧を表示してくれます
$ ls -la /home/yamada      オプション・引数を組み合わせる事が可能です
```

ディレクトリの移動（**cd, pwd**）

ディレクトリを移動するには「**cd**」コマンドを使います。また現在のディレクトリ位置を確認するには「**pwd**」を使います。

※初めの頃は cd コマンドで移動を繰り返すと、自分が今どこにいるかが分からなくなり迷子になりがちです。必ず pwd コマンドを使う癖をつけて自分が今どこにいるのか意識するようにしましょう。

```
$ cd /var/tmp          /var/tmp ディレクトリに移動します
$ cd ../               １個上の階層に移動します。左記例だと/var へ移動
$ cd -                 直前にいたディレクトリに戻ります
$ cd                   引数無しで移動するとユーザのホームディレクトリに移動します
$ pwd                  現在のディレクトリ位置を表示してくれます
```

ファイル等の操作（**touch, mkdir, rm, mv, cp**）

ファイル・ディレクトリを作成・削除したり、移動、コピーといったコマンドとして **touch rm mv cp** といったコマンドがあります。

$ touch file1	file1 というファイルを作成します
$ mkdir dir1	dir1 ディレクトリを作成します

$ rm file1
$ rm -r dir1
※ディレクトリを削除するときは-r オプションをつける必要があります
	file1 を削除します
	dir1 ディレクトリを削除

$ mv file1 ./dir1/file1 $ mv file1 file2	file1 を./dir1 ディレクトリに移動します。 第 2 引数を同じディレクトした場合はリネームができます

$ cp file1 ./dir1/file2
$ cp -r dir1 dir2
※ディレクトリをコピーするときは-r オプションをつける必要があります
	file1 を dir1 ディレクトリ配下に file2 としてコピーします
	dir1 を dir2 としてディレクトリコピーします

ファイルの閲覧（**cat, less**）

コマンド：cat, less

ファイルの閲覧として **cat** と **less** コマンドがあります。

```
$ cat file1
$ less file1
```

上記共に、引数として閲覧したいファイル名を指定します。

違いとして、cat はファイルの中身をそのまま表示して終了しますが less ではビューワーが開き、ページをめくるようにファイルを閲覧できる事ができます。

less 実行時の操作コマンドは以下の通りです。

q	less コマンドを終了します
↑↓	上と下矢印キーで1行進んだりも戻ったりします
スペース	次のページに移動します
b	前のページに戻ります
g	ファイルの先頭に移動します。Shift+g でファイル末尾に移動します
/検索文字	文字列を検索します
n	次の検索結果に移動します。Shift+n で前の検索結果に移動します

文字の入力（**vi**）

Windows や Mac で文字を書くときはメモ帳や word といったツールを使うかと思ますが、linux でも vi というツールが用意されています。(vim という拡張版も用意されています。)

コマンドベースでの入力となるので最初はとっつきにくいかもしれないですが、Linux で文書の作成や修正が必要な際にこのコマンドが使えないと何もできないので必ずマスターしてください。

$ vi file1	引数に編集したいファイル名を指定します

vi はコマンドモードと挿入モードに分かれます。

コマンドモード	このモードではファイルの保存や vi の終了、検索といったことができます。 このモードでは文字の入力できないです。 「ESC キー」を押すことでコマンドモードになります。
挿入モード	このモードで実際に文字の入力や削除が可能となります。 「i」「a」「o」を押すことで挿入モードになります。 キーの違いは挿入の位置の違いとなります。

vi コマンド実行直後は、コマンドモードとなっています。

コマンドモード時は下記のようなコマンドが使えます。

(現時点で覚えるべき最低限のコマンドを紹介します)

i	挿入モードに切り替え(カーソルの左から入力を開始)
a	挿入モードに切り替え(カーソルの右から入力を開始)
o	挿入モードに切り替え(カーソルの 1 行下から入力を開始)
O(Shift+o)	挿入モードに切り替え(カーソルの 1 行上から入力を開始)
←↑↓→	上下左右に移動
/	文字列の検索
n	次の検索結果へ移動
u	直前の操作に戻す(アンドゥ)
Ctrl+r	アンドゥで取り消した操作を元に戻す(リドゥ)
yy	現在の行をコピー
p	現在の行下に張り付け
P	現在の行上に張り付け
dd	現在の行を切り取り(削除)
:行番号	例えば:10 と入力すると 10 行目に移動
:w	保存
:q	vi を終了
:q!	vi を強制終了(保存せずに終了したいときに使う)
:wq	w と q は同時に使うことができる。この場合は保存して終了

文字を入力する場合は、まず「i」を押して挿入モードになってみましょう。

このモードになると、メモ帳のような感覚で文字が打てるかと思います。

Tips

コマンド入力の省力化

コマンドラインで操作しているときには非常に便利な機能として、『ヒストリ機能』と『補完機能』があります。この二つを使いこなす事で、素早く正確にコマンドを入力できるようになります。

↑	以前入力したコマンドを呼び出す事ができます。↓で戻る事もできます
Tab キー	例えば、less を les まで入力してから tab を押すことでコマンドを保管してくれます。複数の補完候補がある場合は tab を2回押すことで補完する候補を表示してくれます

PATH 変数

Linux では、動作環境を設定するために、変数が利用されています。 変数は箱によく例えられますが、その箱の中（値）が何かによって、システムの動作が変わります。

変数にはいろいろあるのですが、TAB 補完に関係するものとして、PATH 変数があります。この変数の中にはいくつかのディレクトリが登録されています。 登録されているディレクトリは以下のコマンドで確認ができます。

```
$ echo $PATH
出力例）
/usr/kerberos/bin:/usr/local/bin:/bin:/usr/bin:/usr/X11R6/bin:/home/suzuki/bin
```

この実行結果から現在 PATH 変数に登録されているディレクトリは

[/usr/kerberos/bin]、[/usr/local/bin]、[/bin]、[/usr/bin]、[/usr/X11R6/bin]、[/home/suzuki/bin]

の 6 ディレクトリであることが確認できます。

これらのディレクトリの中に含まれるコマンドは、「パスが通っている」という状態で TAB 補完の際の候補として表示がされます。逆に言うとこれ以外のディレクトリにあるコマンドは「パスが通っていない」という状態で、TAB 補完で表示されません 。

また「パスの通っている」コマンドはコマンド名だけで実行することができますが、「パスの通っていない」コマンドは、絶対パスなどでそのコマンドの存在するディレクトリも含めて指定をする必要があります。

[/bin]ディレクトリにパスが通っているときは “touch testfile”で実行できますが、[/bin]ディレクトリにパスが通っていないときには“/bin/touch testfile”と実行しなければなりません。

コマンド実行結果の書き出し（>、>>）

これまでの基本操作では、コマンドを実行すると画面上に結果が表示されたかと思います。これを**標準出力**と呼びます。

例えば、結果を保存したくなった場合、画面上の結果をマウスでコピーして vi で開いて書き込んで保存、といった事も出来ますがとても非効率です。

それを解決する手段として、「リダイレクト」という機能があります。

```
$ echo “明日は晴れです”
※echo というコマンドは引数に文字列を渡すことで、その文字列を画面に表示します。

$ echo “明日は晴れです” > hoge.txt
上記コマンドを実行すると、結果が画面に何も表示されません。その代わり hoge.txt に画面に表示される内容が記載されています。
```

“>”は出力をファイルにリダイレクトして上書きしますが、“>>”を使用すると既存ファイルへの出力を上書きするのではなく、追記で行えます。

```
$ echo “今日の晩御飯は何にしよう” > hoge.txt
$ echo “焼肉にしよう” >> hoge.txt
2 つ目の echo の結果は追記となるので、hoge.txt には両方の結果が記載されています。
追記するべきファイルに>とやってしまうとこれまでの内容は全て消えてしまうので注意が必要です。
```

コマンド結果を別のコマンドへ渡す（|）

パイプと呼ばれる機能を使うことで、コマンドの結果を別のコマンドへ渡すことができます。 例えば“ls”コマンドを実行したときにその出力結果が画面の範囲を超えるくらいに膨大な量であるとき、最初の方で出力された内容を見ることができません。パイプを使って出力結果を“less”といった別のコマンドの入力データとして取り込ませると解決できます。

更に、後続で出てくる grep や awk といったコマンドを使うことによって、ls の結果から欲しい情報を抜き出し、更に指定した列のみを出力する事が可能となります。

パイプを使いこなすと、単一のコマンド同士を組み合わせる事で、様々な使い方が出来るようになります。アイデア次第であっと驚くことも可能です。

気になる人は、google 検索で「linux ワンライナー」と検索してみてください。コマンド同士を組み合わせた実用的な使い方がたくさん紹介されています。

```
＊ls -l の結果を less で見る
$ ls -l /home/dir1 | less
```

```
＊CPU 使用率が高いプロセスを見つける
$ vmstat 1 | awk '{print strftime("%y/%m/%d %H:%M:%S"), $0}'

＊物理メモリを多く利用しているプロセスを表示
$ ps aux | sort -n -k 6 | tail -n 10
```

ファイルを探す（**find**）

ファイルの検索には“**find**”コマンドを使用します。

```
$ find /etc -name resolv.conf
```

一つ目の引数は探したい場所、-name オプションに探したい文字列を記載します。
上記例だと、/etc 配下で resolv.conf というファイル名を検索する意味となります。
注意として、検索したい場所に”/”とした場合、全てのファイルを検索する事になるので、ファイルが膨大にあるサーバで実行すると負荷がかなり上がるので注意してください。

指定した文字列を行で抽出する（**grep**）

ファイルから文字列に一致する行を抽出するコマンドとして”**grep**”があります。

```
$ grep ftp /etc/passwd
```

上記例だと、/etc/passwd ファイルから ftp という文字列を含む行を抜き出してくれます。

この“grep”はパイプと共によく使用されます。 例えば、以下のように実行すれば、/etc ディレクトリ直下にあるものの中から、conf という文字列を含むものだけを出力することができます。

```
$ ls -l /etc/ | grep conf
```

列を抽出する（**awk**）

ファイルから列を抽出するコマンドとして”awk”があります。
イメージとしては下記のような文字列から例えば 2 列目を抽出したい場合です。

```
Apple,Banana,raspberry
kiwi,grapefruit,coconut
cherry,pineapple,papaya

上記が記載されたファイル hoge.txt に対して下記コマンドを実行します。
$ awk -F, '{print $2}' hoge.txt
そうすると結果としては下記出力となります。
Banana
grapefruit
pineapple
```

awk コマンドで列を抽出する場合は、列と判断する区切り文字の指定が必要です。
apple banana coconut
の場合は、スペースが区切り文字
apple:banana:coconut
の場合は、:が区切り文字となります。
-F オプションでどれを区切り文字とするかを指定しないと思った通りの列が抽出できないので注意してください。デフォルトではスペースが区切り文字なので、スペースの場合は-F オプションは不要です。

文字列の置換をする（**sed**）

例えば、ringo を apple という文字列に置換したい場合は”sed”があります。

```
$ sed “s@ringo@apple@g” hoge.txt
上記コマンドを実行すると、hoge.txt から ringo の部分を apple として標準出力のみしてくれます。
ここで注意なのは、あくまでも標準出力だけなので実際の中身は書き換わりません。
書き換えたい場合は、下記のように-i オプションをつける必要があります。
$ sed -i “s@ringo@apple@g” hoge.txt
```

文字数を数える（**wc**）

wc はファイルの文字数・単語数・行数を表示するコマンドです。例えば以下のようにアクセスログの行数を数えることで、WEB サーバへのアクセス数をカウントできます。

```
$ cat /var/log/httpd/access_log | wc -l
24570
```

オプション　説明
-c　データに含まれる文字数を数える
-w　データに含まれる単語数を数える
-l(小文字の”エル”)　データに含まれる行数を数える

並び替えをする（**sort**）

sort は並び替えを行うコマンドです。
以下のように実行すれば、行先頭の文字を文字コード順に並び替えて出力します。

```
# cat /etc/passwd | sort
```

オプション
-r　逆順として並び替える
-n　文字列を数値として並び替える
-k　指定した列で並び替えをする
　　-k 5 なら 5 列目 -k3 なら 3 列目を数値として並び替える
-t　フィールドの区切り文字を指定する。-t , なら,で区切る。デフォルトはスペース

重複する行を削除する（**uniq**）

重複する行を削除するコマンドとして **uniq** があります。ただし、重複する行が連続で続いていないと重複であるかどうか判断できないので、上記の sort とともに利用されることが多くなっています。

```
# cat datafile | sort | uniq
```

オプション
-c　各行の前に出現回数を出力する。
　これを使うと例えば apple は 5 回、sabamiso は 10 回というぐらいに出力されてくれるので、例えば apache のアクセスログから最もアクセスが多いパスはなにで、かつアクセス回数を出すことができて便利です。

演習

Linux のコマンド操作では、実行後に何も出力されないことがよくあります。

正しく実行できていることを確認するために、実行後に必ず確認する癖をつけましょう。

演習 1

/tmp に移動し、**file_Name** というファイルと **dir_Name** というディレクトリを作ってください。（Name には自分の名前を入れる）

演習 2

dir_Name に移動し、演習 1 で作った **file_Name** を **dir_Name** の中に移してください。

演習 3

file_Name を **file_Name2** にリネームしてください。

演習 4

file_Name2 を削除し、**dir_Name** の中に **file_Name3** と **file_Name4** を作成してください。

演習 5

/tmp に移動し、**dir_Name** を削除してください。

演習 6

/tmp の中に再度 **file_Name** というファイルを作成してください。

作成したら、

1:Suzuki:AB

2:Aoki:A

3:Fukada:O

4:Okumura:B

と記述してください。

演習 7

file_Name を下記の通り修正してください。

・3 行目をコピーして 1 番上の行に貼り付ける

・1～4 行目をコピーして 2 行目の下に張り付ける

・2 行目を削除する

・1～3 行目を切り取り一番後ろの行に貼り付ける

演習 8

file_Name から「Fukada」という文字列がふくまれる行が何行あるのか集計してください。

演習 9

file_Name から 3 列目が「A」の行は何行あるか集計してください。

演習 10

vi コマンドを使用せずに、**file_Name** を最初に記述したとおりに戻して **file_Name2** という名前で保存してください。

付録

ローカル環境構築

WSL2 インストール(Windows)

※自分の PC 内に Linux 環境を構築する方法

1. Windows ボタン ＞ 設定 ＞ アプリ ＞ プログラムと機能
＞ Windows の機能の有効化または無効化
＞ 仮想マシンプラットフォームにチェック ＞ OK
2. OK を押すと再起動されるのでしばらく待つ。
3. コマンドプロンプトを管理者権限で開き、下記コマンドを入力

```
> wsl –install
```

4. インストール後、PC を再起動する。
5. コマンドプロンプトに下記のコマンドを入力

```
> Invoke-WebRequest -Uri https://aka.ms/wslubuntu2004 -OutFile Ubuntu.appx -
UseBasicParsing
```

6. ダウンロード後、appx ファイルを実行する

環境設定

下記の設定を行うことでさらに便利に使えます。

LANG 設定（Windows[WSL2]）

```
# sudo su -
# apt -y install language-pack-ja
# update-locale LANG=ja_JP.UTF8
# exit
# sudo su –
# ls --help
日本語のヘルプが出れば成功
```

LANG 設定（Mac）

1．ターミナル > 環境設定 > プロファイル > 詳細
「Control+V で非 ASCII 入力をエスケープ」のチェックを外し、
「Unicode 東アジア A（曖昧）の文字幅を W（広）にする」にチェックを入れる
2．ターミナル > 環境設定 > エンコーディング
Unicode(UTF-8)にチェックを入れる
3．ターミナルから下記コマンドを実行(ファイルに追記されるので複数回実行しないこと)

```
$ cd
$ echo ‘export LANG=ja_JP.UTF-8’ >> ./.zshenv
$ exec $SHELL -l
```

プロンプト設定・alias 設定・PATH 追加(共通)

```
$ vi .bashrc
最終行に下記追記
--------------------------
export PS1="[¥u@¥h ¥w]$"
alias rm='rm -i'
alias cp='cp -i'
alias mv='mv -i'
alias lla=’ls -al’
export PATH=$PATH:/home/ユーザー名/bin
--------------------------
```

.bashrc ファイル・・・ログイン時に自動的に読み込まれる設定ファイル
alias コマンド・・・別名を付けることができるコマンド
export コマンド・・・環境変数を設定できるコマンド

.vimrc 修正

vi の表示や挙動をカスタマイズし、使いやすくする設定。

```
# cd
# vi .vimrc
下記追記
----------------------------
# 文字コードを UFT-8 に設定
set fenc=utf-8
#  バックアップファイルを作らない
set nobackup
#  スワップファイルを作らない
set noswapfile
#  編集中のファイルが変更されたら自動で読み直す
set autoread
#  バッファが編集中でもその他のファイルを開けるように
set hidden
#  入力中のコマンドをステータスに表示する
set showcmd

#  行番号を表示
set number
#  現在の行を強調表示
set cursorline
#  現在の行を強調表示（縦）
set cursorcolumn
#  行末の 1 文字先までカーソルを移動できるように
set virtualedit=onemore
#  インデントはスマートインデント
set smartindent
#  ビープ音を可視化
set visualbell
#  括弧入力時の対応する括弧を表示
set showmatch
```

```
#  ステータスラインを常に表示
set laststatus=2
#  コマンドラインの補完
set wildmode=list:longest
#  折り返し時に表示行単位での移動できるようにする
nnoremap j gj
nnoremap k gk
#  シンタックスハイライトの有効化
syntax enable
#  不可視文字を可視化(タブが「▸-」と表示される)
set list listchars=tab:¥?¥-
#  Tab 文字を半角スペースにする
set expandtab
#  行頭以外の Tab 文字の表示幅（スペースいくつ分）
set tabstop=2
#  行頭での Tab 文字の表示幅
set shiftwidth=2
#  検索文字列が小文字の場合は大文字小文字を区別なく検索する
set ignorecase
#  検索文字列に大文字が含まれている場合は区別して検索する
set smartcase
#  検索文字列入力時に順次対象文字列にヒットさせる
set incsearch
#  検索時に最後まで行ったら最初に戻る
set wrapscan
#  検索語をハイライト表示
set hlsearch
#  ESC 連打でハイライト解除
nmap <Esc><Esc> :nohlsearch<CR><Esc>
----------------------------
```