VSCode使い方の手引き・ヒント・小技

ショートカット（キーバインド）

システム系キーバインド

- Ctrl-Shift-P コマンドパレット（操作リスト）
- Ctrl-Shift-X 拡張機能のページ
- Ctrl- 設定項目検索窓
- Ctrl-PgUp, Ctrl-PfDn タブの切り替え
- 目次はCtrl + shift + P を押して検索窓で「table of Contents」で作れる

重要キーバインド

- Ctrl-n 次の行
- Ctrl-p 前の行
- Ctrl-b 前の文字に移動
- Ctrl-f 次の文字に移動
- Ctrl-a 行頭に移動
- Ctrl-e 行末に移動
- Ctrl-k カーソルの前の行全部を削除
- Ctrl-d 前方一文字削除
- Ctrl-h 後方一文字削除
- Ctrl-s ページ内検索
- Ctrl-X Ctrl-S 保存
- Ctrl-space マーク付ける
- Ctrl-w マーク地点からカーソルまでを切り取り
- Ctrl-y 貼り付け
- Ctrl-x Ctrl-c 終了

他にも便利なキーは存在するため、調べてみるのも良い。

マークダウン記法

- --- 水平線 （ハイフン3つ以上）
- [リンク名] (URL) URLの挿入
- 箇条書きの際は、**タブキー**で字下げ。**Shift-Tab**で字下げを元に戻せる。
- 表の作成
    - 列名は | で囲む
    - その下にハイフン2つを | で囲む
    - 要素も列名と同様に | で囲む
- プログラミングコードの挿入
    - バッククォーテーション3つ(```)で囲んだなかにコードを書く。その際、上のバッククォーテーションの直後に用いる言語を書く。

VS Codeエディターの機能

PDFへの出力

拡張機能Markdown PDFをインストールしておく。

- Ctrl-Shift-Pでコマンドパレットを起動し、exportと入力
- Markdown PDF: Export (pdf)を選択
- 同じディレクトリ内にPDF形式で出力

図の作成

Draw.io Integrationという拡張機能をインストールしておく

- .drowio.png という拡張子付きのファイルを作成し、そこで図を作る。
- ![図の名前](ファイル名)で挿入

スライドの作成

拡張機能のMarp for VS Codeをインストールしておく。

- ファイル名は ~~ slide.md にしておくと良い。
- 詳しくは、以下の画像どおり。

slide.mdを開いて、ファイルの先頭に、次のように記述してください。

```
---
marp: true
---
<!--
headingDivider: 2
-->
```

marp:とtrueの間に一つスペースを入れるのを忘れないようにしてください。これで、レベル2の見出しがスライド頁の区切りになります。

たとえば、以下のように入力してみてください。プレビューがスライドになるはずです。

```
---
marp: true
---
<!--
headingDivider: 2
-->

# Marpで楽しくスライド作り

1234567 工科太郎

## これは頁タイトルになる

ここは2ページ目である。

1. 番号付き箇条書き項目である
2. 番号付き箇条書き項目である
3. 番号付き箇条書き項目である
```

PDFに出力することも簡単にできますので試してみてください(編集画面の上のほうにある三角マークからできます)。

Anaconda仮想環境

Anacondaにはbaceというデフォルトの仮想環境がある。

仮想環境の構築

ここでは myenv という仮想環境を作る

```
conda create -n myenv
```

何か聞かれたら y を入力。

環境ができているかチェック

```
conda info -n
```

作成した環境を起動し、必要なパッケージを入れる。Python3.8を入れてみる。

```
conda install python=3.8
```

ついでにコード補完のやつも入れてみる

```
conda install ipykernel
```

仮想環境のコマンド操作

- conda deacitivate 現環境の停止
- conda activate bace環境の起動
- conda env list 現在存在している仮想環境を表示。起動している環境に*
- codna list 現環境にインストールされているパッケージの表示
- conda remove package名 指定したpackageを現環境からアンインストール
- conda remove -n 環境名 --all 指定した環境を削除

バージョン管理

用語まとめ

用語	解説
Git	バージョン管理のためのツール。リモートリポジトリとローカルリポジトリを同期させることができる。
リポジトリ	1つのプロジェクトに関する全てのファイルを管理する装置。作業ディレクトリに格納されており、.git という隠しフォルダがその正体。
Git Hub	インターネット上にリポジトリを置くためのサービス。複数の開発者が共通のリモートリポジトリを介して共同作業を行うことができる。

用語	解説
リモートリポジトリ	インターネット上のリポジトリ
ローカルリポジトリ	開発者のパソコン上に存在するリポジトリ
プッシュ	リモートの状態をローカルな状態に同期
プル	ローカルな状態をリモートの状態に同期
コミット	作業をリポジトリの履歴に記録する

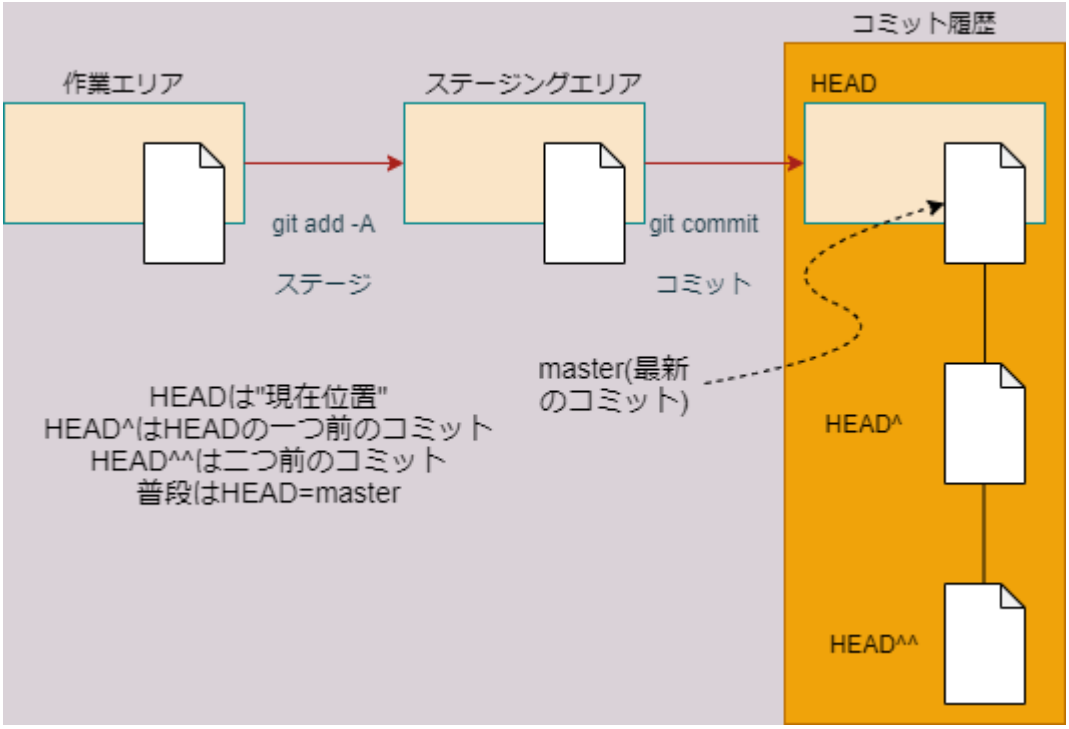
git bash のコマンド操作

- git clone リポジトリのURL : リモートリポジトリをローカルにコピーする操作（クローン）
- git reset : ステージングエリアに入ったがまだコミットしていない変更をリセットし、アンステージできる。
- git reset : ステージングエリアをコミットせずに破棄
- git reset --hard HEAD : 作業エリアとステージングエリアをコミットせずに共に破棄
- git reset --soft HEAD^ : コミットを取り消してステージングした状態に戻す。（HEAD^に注意。別のとこに記述)
- git --mixed HEAD^ : コミット履歴を一つ前に戻してステージングエリアもそれに合わせる。
- git reset --hard HEAD^ : コミット履歴を一つ前に戻して作業エリアとステージングエリアもそれに合わせる。
- git commit -a : ステージングとコミットをまとめて行う
- git commit -a -m "message" : メッセージも付けれる
- touch @@@ : @@@という名前の空のファイルを作成
- git ls-files : リポジトリ内のファイルをリストアップ
- ls : 作業エリア内のファイルをリストアップ

git reset系のコマンドは十分に注意して実行する！色々複雑！

git bashとVSCodeの操作

ステージングとコミット



つまり、流れとしては

1. 作業エリアで開発し、保存
    1. VSCode内の左サイドバーの上から3つ目のボタンにマークがつく（変更箇所が存在することを教えてくれてる）
    2. このボタンを押すと変更箇所を比較表示してくれる。
    3.
2. git bushを開き、ステージング
    1. ```
       git status
       ```

       と入力すると　modified:変更があったファイル名　と赤色で表示
    2. ```
       git  add -A
       ```

       でステージング。(git statusを入力すると緑色でコミットは未完であることを教えてくれる)
3. 続いて、コミット
    1. ```
       git commit -m "メッセージ（一行だけ）"
       ```

       もしくは、

       ```
       git commit
       ```
       ```

と入力すると、VSCodeが起動しコミットメッセージの入力を求められる。こうす ることで一行以上のメッセージをかける。**最初の一行はシンプルなメッセージを書き、一行あけてより詳しいメッセージを書くこと**に注意。その後、保存してVSCodeを終了する。

プッシュ

プッシュとは、リモートの状態をローカルに同期させることをいう。 プッシュの前に以下のコマンドを実行しておくとよい。

```
git branch --set-upstream-to=origin/master
```

大雑把に言うと、「このリポジトリは、リモートリポジトリoriginのmasterと繋げる」という意味。

上流ブランチが正しく設定されているか確かめる。

```
git status -sb
# master...origin/master となれば成功
```

pushは以下のコマンドで行う。

```
git push
```

-set-upstream-toの設定を行わなかった場合は、

```
git push origin master(もしかしたらmain)
```

もしくは

```
git push -u origin master
```

と入力する。originはリモートリポジトリを表し、masterはリモートリポジトリの最新のコミットを表す。つまり、"git push origin master"は「リモートリポジトリ(origin)に保存されている最新のコミット(master)に、ローカルリポジトリの最新版をプッシュして上乗せ更新せよ」よいう意味になる。

ローカルをリモートに同期(プル)

2台以上のパソコンやリモートリポジトリがローカルリポジトリよりも進んでいるときに、この操作を行う必要がある。

- 初めてローカルリポジトリを作るとき

    - 単にクローンするだけ
- クローン後
    - 以下のコマンドを実行

```
git fetch
git pull
```

fetchはリモートの状態を確かめるコマンド。git fetchのあとにgit statusを入力すると、ローカルが何コミット分遅れているか表示される。リモートの方が新しいとき、git pullによってリモートに同期させることができる。

過去のバージョンを閲覧する

過去にコミットされたバージョンを閲覧するとき

```
git checkout 35d6
```

ここでの35d6は**SHA1-ID**というもの。最初のある程度を渡せばよい。

最新のコミットに戻ってくるには、以下の通り。

```
git checkout master(もしかしたらmain)
```

ブランチを作る

ここの作業は慎重に行う

newという新規ブランチを作る。

```
git branch new
```

ブランチの切り替え(newにしたいとき)

```
git checkout new
```

現在のブランチを確かめるとき

```
git branch
```

アスタリスクがついてるのが今いるブランチ

ちなみに、ブランチの作成とチェックアウトを同時に行うには、「git checkout -b new」を実行

ブランチのマージ

新規ブランチ内で作業・変更・修正を行い、メインバージョンにしたい場合は**マージ**する。

まずは本流のブランチにいく。

```
git checkout mastar(もしかしたらmain)
```

git branchも実行して、確認も怠らない！

newブランチをmasterブランチにマージ

```
git merge new
```

これでnewの変更がmasterに反映された。

なお、これは**fast-forward merge**と呼ばれるマージで、newの分岐以降masterに全く変更が無かった場合に起こるマージ。一般的には両方に変更が起こりえるが、個人で行う場合はfast-forward mergeで事足りる。

masterにマージしたらnewは必要ないので消しとく

```
git branch -b new
```

過去のコミットを打ち消す(git revert)

過去のコミットを打ち消すコマンド

```
git revert *****
```

*****はSHA1-ID　最新のコミットをrevertしたいなら、SAH1-IDの代わりにHEADと指定。あくまでも打ち消しコミットなのでメッセージの入力を求められるが、デフォルトのままでよい。

revertを行った形跡はlogで確認できる。（これが特長）

過去のコミットを無かったことにする (git reset)

resetは過去のコミットを無かったことにする（つまり履歴にも残らない）resetには以下の3種類ある。**これを理解できるとGitも理解しているということ**

resetのオプション	**機能**

resetのオプション	機能
--hard	作業ディレクトリもステージングエリアも両方戻す
--soft	履歴だけ残す
--mixex	履歴とステージングエリアだけ戻す

このオプションを指定しないgit resetはステージングエリアの取り消しを意味する。

例えば、最新のコミットだけ取り消すとき

```
git reset --soft HEAD^
```

最新とそこから一つ前のコミットを取り消すとき

```
git reset --soft HEAD^^
```

--softオプションのとき、変更はステージング前の状態になる。コミット履歴と共にステージングエリアももとに戻したいとき

```
git reset --mixed HEAD^
```

作業ディレクトリもステージングエリアも元に戻す（つまり何もなかったことにする）には、

```
git reset --hard HEAD^
```

一連のコマンドを実行したら、git log, git statusで確認すること。

コマンドエイリアスを作る

後で。

gitに追跡されない作業ファイル (.gitignoreファイ)

作業ディレクトリの中に .gitignore というファイルを作成し、その中に無視するファイル名の一覧を記入する。

例えば、.vscodeというディレクトリ、codeというディレクトリを丸ごと無視、sample-で始める名前のファイルを全て無視してほしいときは以下のように.gitignoreファイル内に記入する。

```
.vscode/
code/
sample-*
```

このままでは.gitignoreファイル自体は追跡されるので、それも記入しておくとよい。

ファイル削除のステージング (git rm test)

作業エリアからファイルを削除しても、リポジトリからは削除されない。両方から削除するには、(***はファイル名)

```
git rm ***
```

「***の削除」という変更がステージングされるので、コミットしてリポジトリからも削除される。

ファイル名のステージング

変更のスタッシュ

リモートリポジトリの作成

新規ブランチのプッシュ

リモートブランチの削除

ホームページを作る