2025 年 5 月 10 日

非ジオ・エンジニアでもできる

マップタイル作成マニュアル

第 1.0 版 / 室木直樹@林野庁

はじめに

本マニュアルは、航空レーザ解析の各業務で整備されたデータを県域などの広域で統合し、マップタイルとして配信するためのデータ変換作業の手順をまとめたものである。

（１） PC 環境

使用した PC の諸元は、OS：Windows11、CPU：intel core i7 12th、RAM：64GB（DDR4-3200、32GB×２）、ROM：M.2 接続の SSD 1TB×２である。

使用したソフトウェアは、QGIS（ver.3.34）及びそれと併せてインストールされる OSGeo4W Shell (OSGeo for Windows command shell)である。なお、OSGeo4W は、GDAL や Python を一括で扱えるコマンドプロンプトである。

（２） 使用データ

基本的には、森林 GIS フォーラムが運用する「**森林資源データ解析・管理標準仕様書**」に沿って整備されたデータを、林野庁「**森林情報に関するオープンデータ 標準仕様書（案）【航空レーザ森林資源解析データ編】**」に沿ってマップタイルを作成する手順を解説している。

ただし、両標準仕様書において規定がないが、マップタイルを作成する上で追加的に規定されていることが望ましいと考えられた事項については、作業手順に追加している。追加した作業手順の概要は次の通りである。

① 航空レーザ解析業務において、業務範囲外のラスタ値に Nodata を割り当てることが多い[1]が、その**Nodata を RGB 値の場合は、白（255,255,255）に、ラスタ値の場合は-9999 に統一する手順を追加**した。Nodata の値を統一する理由は、マップタイルに変換する際に、透過処理や無効化処理を的確に行うためである。例えば、Nodata を意味する白を透過設定する際に、一部範囲で Nodata が黒になっていると、その黒は透過されず残ってしまうことを防止するためである。

② 航空レーザ解析業務において、業務範囲外のラスタ値について透過設定をし、それをアルファバンド（RGB に次ぐ４バンド目）に指定している場合がある。GDAL で変換作業を行う上で、３バンドまでしかないラスタデータと４バンドがあるラスタデータが混在するとエラーが発生するため、**アルファバンドを指定する作業を追加**した。

（３） 利用上の注意

筆者自身が非エンジニアであり、必ずしも、本稿で紹介する手順がもっとも効率的かつ合理的なものとは限らないことをご了承いただきたい。

---

[1] RGB 値は、白とすることが多いようであるが、黒（0,0,0）が設定されている場合もある。ラスタ値は、$\pm 3.4e^{18}$ や、-32,768 が設定されている場合もあり、航空レーザ解析業務では統一されていない。

内容

1　ラスタデータを扱う際の事前準備

（１） 貸与データの欠損･異常を確認する

航空レーザ解析業務においては、ラスタデータにあっては国土基本図図郭（2,500 やその４分の１）、ベクタデータにあっては、市町村等の行政区域を単位として作成されることが多い。この結果、特にラスタデータにあっては、大量のファイルを扱うことになるが、**貸与されたデータからいきなりマップタイルを作成すると、貸与データに不足があること、Nodata が適切に設定されていない等の異常に事前に気づくことができず、出戻り作業が発生する**ことも少なくない。

他方で、QGIS では、経験上、データファイル数が 100 を超えてくると動作が重くなることから、**ある程度取り回しの良いサイズにラスタデータをマージし、マージされたラスタでエラーがないか確認するほうが効率**がよい。このため、以下の手順により、貸与データを中間加工することを推奨する。

（２） tif を Geotiff に変換する

航空レーザ解析業務で作成されたラスタデータは、基本的には、tif と tfw の２つで構成されている。しかしながら、GDAL 等では位置情報付きの geotiff での作業が前提となることから、**gdalwarp**[2]のループ処理により、一括変換を行う。

▼コマンド例

```
for %f in (C:¥original¥*.tif) do gdalwarp -t_srs EPSG:6678 -r bilinear "%f" "C:¥Converted¥%~nf.tif"
```

このコマンドは、OSGeo4W Shell で実行するバッチ処理である。C:¥original フォルダ内の全.tif ファイルを「バイリニア補間」で補正しながら、EPSG:6678 の座標系に再投影して、C:¥ Converted フォルダに変換後のデータを保存するもの。

▼解説

for %f in (C:¥original¥*.tif)	C:¥original フォルダ内のすべての.tif ファイルを対象にループ処理を行うための前置き。***.tif とすることで、original フォルダ内の全ての tif を変換の対象とすることができる。**フォルダやファイル名に全角文字があっても支障[3]はなく、任意のディレクトリを使うことができる。
gdalwarp	GDAL の投影変換などに使う機能。

---

[2] 単なる tif の geotiff 変換は、gdal_translate で実行するものであるが、**経験上、貸与データには複数の座標系が混在している**ことが多々あり、（事実上、投影変換が空振りになる可能性はあるものの、）geotiff への変換とともに、投影変換も可能な gdalwarp を使用することにした。

[3] 括弧が禁則である。基本的には全角文字を混ぜないように習慣化してほしい。

-t_srs EPSG:6678	座標系を EPSG:6678 に変換することを指示するもの。**EPSG コードは任意の数値を与える**ことができる。
-r bilinear	投影変換の際のリサンプリング（補間）にバイリニア補間を使用することを指示するもの。（なめらかに変換）。**bilinear は、周囲のピクセル値を補間に使い、なめらかにするものであり、連続値データに向き、処理速度と品質のバランスに優れる**。このほかに、カテゴリカルデータに向く、最近傍補間（nearest）、連続値に向き、さらに品質のよいバイキュービック（cubic）などもある。
"%f"	変換前のファイルパス。今回はループ処理により一括変換を行うこととしており、前置きした%f がファイルパスとして“ ”で指定されている。
"C:¥Converted¥%~nf.tif"	変換後のファイルパス。%~nf は、変換前のファイル名である%f から拡張子.tif を除いたものを変換後のファイル名とすることを指示するもの。これにより、Converted フォルダに**同一名称のファイルを出力**することができる。

▼参考

このようにコマンドプロンプトが流れていけば成功である。

```
Creating output file that is 5000P x 3750L.
Processing C:¥original¥10LE7222.tif [1/1] : 0...10...20...30...40...50...60...70...80...90...100 - done.
```

変換前	変換後
10LD1924.tfw 2024/01/10 16:22 10LD1924.tif 2024/01/18 16:14 10LD1941.tfw 2024/01/10 16:23 10LD1941.tif 2024/01/18 16:15 10LD1942.tfw 2024/01/10 16:24 10LD1942.tif 2024/01/18 16:15	10LD1924.tif 2025/05/02 14:15 10LD1941.tif 2025/05/02 14:15 10LD1942.tif 2025/05/02 14:15

（３） 国土基本図図郭×業務範囲のベクタデータを作成する

（１）に記載したとおり、国土基本図図郭（2,500 やその４分の１）のままでは、その後の作業の取り回しが悪いことから、より大きい図郭単位にマージするほうが望ましい。また、航空レーザ解析業務は、１業務で１都道府県全域のデータが整備されていることは稀であり、市町村単位などにより、複数にデータファイルが分かれていることが多い。そのため、それぞれお互いに業務区域からはみ出たデータが残されていると、データの重複が発生し、どちらか一方のデータが隠れてしまうこともあり得る。

そこで、次の手順により、国土基本図図郭（2,500 やその４分の１）単位のデータをマージし、マップタイル変換を行うための中間データを作成する。なお、以下では、**国土基本図図郭 50,000 の４分の１単位を例**としているが、これは、前掲の林野庁「**森林情報に関するオープンデータ 標準仕様書（案）【航空**

レーザ森林資源解析データ編】」に沿うデータを併せて作成できるためである。この単位によれば、個々のデータファイルは概ね１～４GB の容量になり、QGIS の動作や G 空間情報センターへのデータ掲載においても支障がないため、推奨できる。

① 国土基本図図郭 50,000 ファイルを作成

QGIS において、MIERUNE 社が作成した **Japanese Grid Mesh プラグイン**を使うなどにより、国土基本図図郭 50,000 のファイルを作成する。

② 国土基本図図郭 50,000 の４分の１ファイルを作成

QGIS において、**ベクタ> 調査ツール> グリッドを作成**により、①で作成した図郭範囲に合致される形で水平 20km、垂直 15km のポリゴンを作成し、①のコードを基に②のコード名称を作成する。

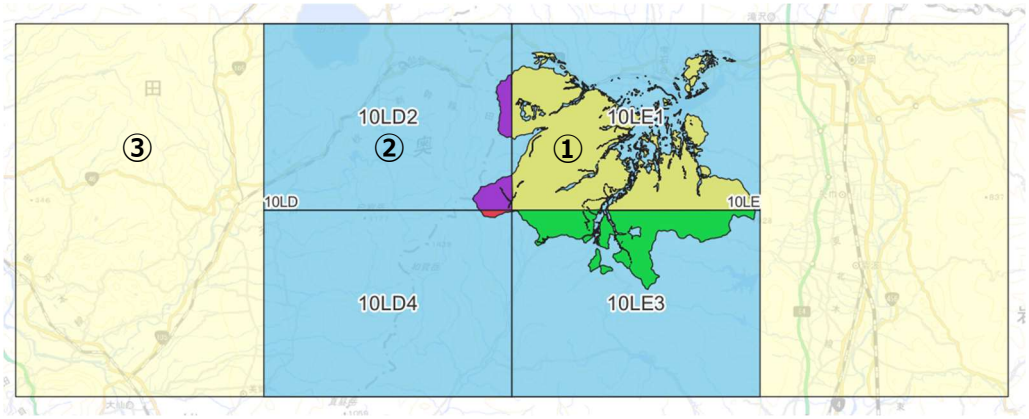
③ 業務範囲ポリゴンでクリップ

QGIS において、**ベクタ> 空間演算ツール> 切り抜く（Clip）**により、②で作成したポリゴンを業務範囲ポリゴンでクリップする。

④ クリップポリゴンを図郭ごとの分割ポリゴンとして出力

QGIS において、**ベクタ> データ管理ツール> 属性でレイヤを分割**により、③で作成したクリップポリゴンを図郭単位に切り分けたジオパッケージで保存する。（分割の基準に図郭のコードを指定する。）

▼参考｜分割ポリゴンのイメージ



（４） 図郭ごとの geotiff を結合し、分割ポリゴンでクリップする

（２）で変換した国土基本図図郭単位の geotiff をマージし、（３）で作成した分割ポリゴンでクリップする方法は次のとおりである。

① 仮想ラスタの作成

（３）④の分割ポリゴンでクリップするのに十分な範囲のラスタを特定し、あらかじめマージした geotiff を作成しておくことは、必要な図郭の選定に手間がかかる上、最終的に使いもしない大容量の geotiff を作り出すだけ（ストレージの容量をひっ迫するだけ）の無駄な行為である。そのため、在り様

の geotiff 全てを使った仮想ラスタ（.vrt）を作成し、最終的にクリップに用いる Geotiff を仮想ラスタとして用意しておく。GDAL の **gdalbuildvrt** を用いる。

▼コマンド例

```
gdalbuildvrt -vrtnodata "0 0 0" "C:¥Clipped¥merge.vrt"
"C:¥Converted¥*.tif"
```

▼解説

gdalbuildvrt	複数のラスタを仮想的に一つにまとめた.vrt ファイルを生成する GDAL の機能。
-vrtnodata "0 0 0"	入力ラスタのうち RGB(0, 0, 0)＝黒を **NoData（無効値）として扱うための指示**。あらかじめ QGIS で画像を展開し、Nodata として扱うべき RGB 値を把握し、必要なものを入力する。なお、1 の処理につき、1 の RGB 値しか指定できない。（同時に 2 色を Nodata とすることはできない。）
"C:¥Clipped¥merge.vrt"	vrt ファイルを保存するファイルを指定する。フォルダ名もファイル名（merge）も任意。
"C:¥Converted¥*.tif"	vrt ファイルを生成する基となる geotiff ファイルの保存先を指定する。*.tif とすることで、Converted フォルダ内の全ての tif がマージされる。

▼参考

このようにコマンドプロンプトが流れていけば成功である。

```
0...10...20...30...40...50...60...70...80...90...100 - done.
```

② 仮想ラスタの分割ポリゴンによる分割

上記で作成した仮想ラスタを（３）④で作成した分割ポリゴンでクリップした geotiff を作成する。GDAL の **gdalwarp** を用いる。

▼コマンド例

```
gdalwarp -cutline "C:¥code_10LD2.gpkg" -crop_to_cutline -co
COMPRESS=DEFLATE -co ZLEVEL=6 -co BIGTIFF=YES -t_srs EPSG:6678
"C:¥Clipped¥merge.vrt" "C:¥clipped¥10LD2.tif"
```

▼解説

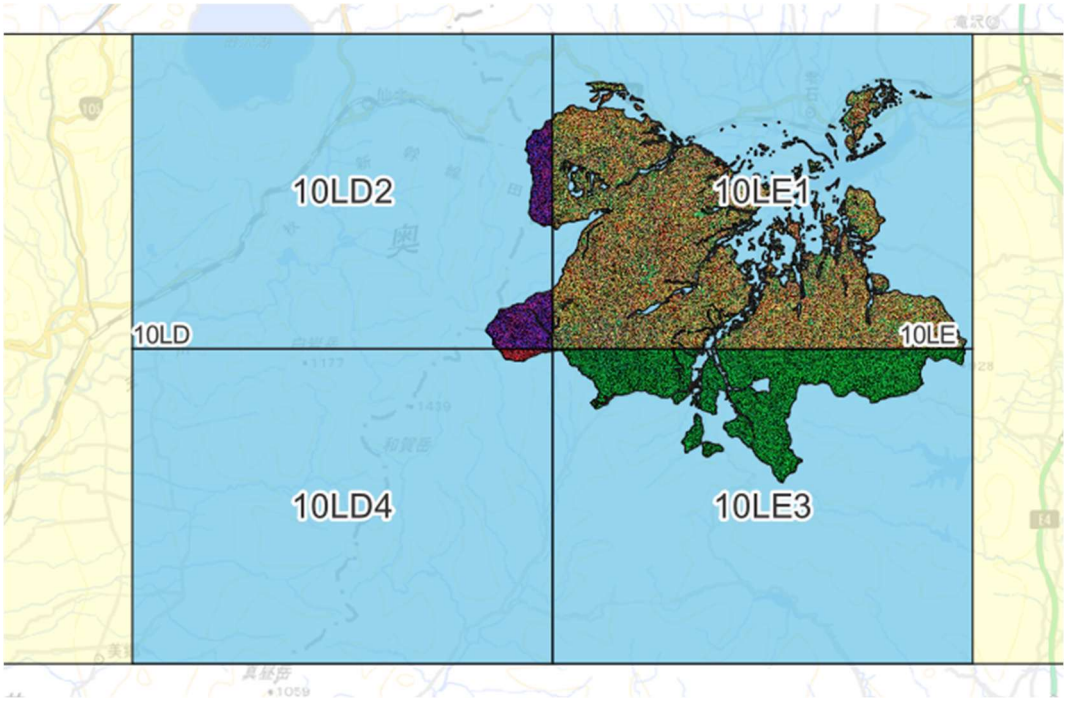
gdalwarp	GDAL の機能である。1 （２）の geotiff 変換に用いたものと同じである。
-cutline "C:¥code_10LD2.gpkg"	仮想ラスタを geotiff として切り出す境界を表わすポリゴンを指定する。

-crop_to_cutline	ポリゴンの外側を除去し、ラスタをトリミングする。
-co COMPRESS=DEFLATE	出力ラスタを可逆圧縮し、データ容量を小さくする。なお、DEFLATE のほか、LZW 圧縮（COMPRESS=LZW）も選択肢となる。
-co ZLEVEL=6	**圧縮率を 0～9 の数値で指定**する。6 は圧縮率と処理速度のバランスがとれた選択肢。数値を大きくするほど圧縮率は高くなるが、処理速度が遅くなる。作業ストレージに余裕があれば非圧縮（=NONE とする）でも構わない。
-co BIGTIFF=YES	出力ラスタのデータ容量が 4GB を超える可能性がある場合には指定しておく必要がある。国土基本図図郭 50,000 の 4 分の 1 を基本単位とする場合は、この指定が必要（たまに 4 GB を超える場合がある）。
-t_srs EPSG:6678	出力ラスタの座標系を EPSG:6678 に指定するもの。任意の座標系を選択できる。
"C:¥Clipped¥merge.vrt"	gdalbuildvrt で作成した仮想ラスタ（VRT）を指定する。
"C:¥clipped¥10LD2.tif"	クリップした出力ラスタのフォルダ名、ファイル名を指定する。

▼参考｜生成結果

このようにコマンドプロンプトが流れていけば成功である。

```
Creating output file that is 15854P × 54630L.
Processing C:¥Clipped¥merge.vrt [1/1] : 0Using internal nodata values (e.g. 0) for image C:¥Clipped¥merge.vrt.
Copying nodata values from source C:¥Clipped¥merge.vrt to destination C:¥clipped¥10LD2.tif.
...10...20...30...40...50...60...70...80...90...100 - done.
```



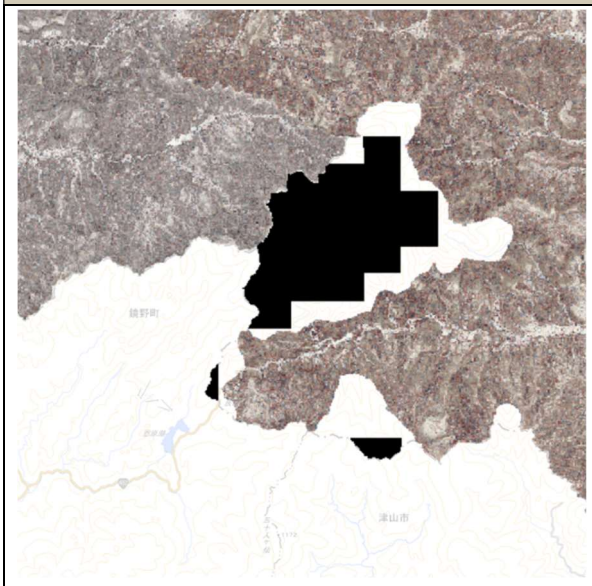
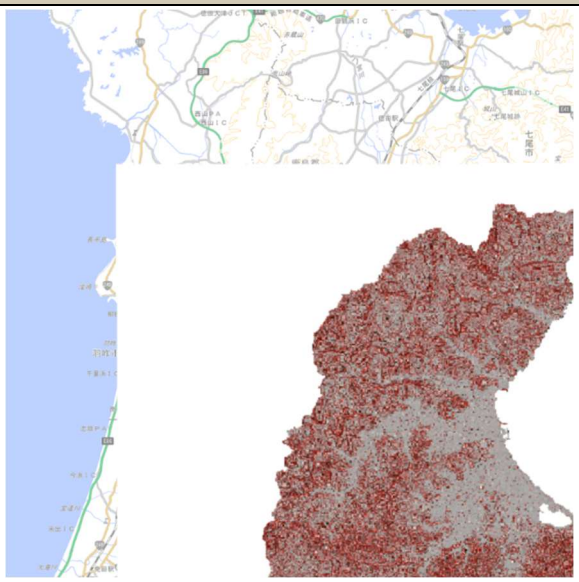
※図郭単位で切り出されていることを分かりやすくするため、着色して強調している。

2　ラスタ（RGB 値）データのマップタイル化

微地形表現図や林相識別図、簡易オルソ画像などの RGB 値で構成されるラスタデータをマップタイル（XYZ タイル）に変換する手順を記載する。

マップタイル化する前に処理すべきこととして、（１）Nodata の値を統一しておき、タイル生成時に当該 Nodata を透過設定できるようにしておくか、（２）透過設定したい RGB 値をアルファバンドに設定しておくか、の２パターンの処理がある。（１）と（２）は、必ずしも両方をする必要はなく、貸与データを踏まえながら任意に一方を選択すればよい。例えば、既にアルファバンドが指定されている貸与データがある、Nodata に割り当てられている RGB 値が白・黒以外もあるなどパターンが多い、という場合には、全てのデータにアルファバンドを指定するほうが作業量は少ない。他方で、ほとんどの貸与データにおいて、Nodata に白色が充てられており、一部のデータで Nodata を白色に置き換えるだけで済む場合は、（１）を選択するほうがよい。一方で、以下のような具体事例を踏まえると、あらかじめ、不用な部分は透明にしてある（２）のほうが後々のエラーを抑制でき、無難であると考えられる。

▼参考｜好ましくない公開データの例

Nodata の値を統一せず、白を透過設定してマップタイルを作成した（Nodata であるはずの黒が残った）例	Nodata の値は白に統一されていたが、白を透過設定せずにマップタイルを作成した例
	

（１） Nodata とする値を統一する手順

① Nodata に指定されている値の確認

GDAL の gdalinfo を用い、Nodata が設定されているか、設定されている場合は、どの RGB 値が Nodata になっているかを確認する。

▼コマンド例

```
gdalinfo "C:¥Converted¥10LD1924.tif"
```

▼参考

Nodata が設定されていない場合の出力結果

```
Band 1 Block=5000x1 Type=Byte, ColorInterp=Red
Band 2 Block=5000x1 Type=Byte, ColorInterp=Green
Band 3 Block=5000x1 Type=Byte, ColorInterp=Blue
```

Nodata に RGB(0,0,0)が指定されている場合の出力結果

```
Band 1 Block=15854x1 Type=Byte, ColorInterp=Red
  NoData Value=0
Band 2 Block=15854x1 Type=Byte, ColorInterp=Green
  NoData Value=0
Band 3 Block=15854x1 Type=Byte, ColorInterp=Blue
  NoData Value=0
```

② 特定の RGB 値を RGB（255,255,255）＝白に変換し、それを Nodata にする python コードを作成する

以下のコードをテキストエディタにコピーし、C:¥clipped に、RGB 値を変換したい geotiff が保存されているフォルダのパスを、C:¥clipped¥converted に RGB を変換した後の geotiff を保存したいフォルダのパスを入力し、mask = (r == 0) & (g == 0) & (b == 0) に変換したい RGB 値を入力し、python コードとして保存する[4]。テキストエディタがない場合は、メモ帳アプリにおいて、ファイル名称 replace_rgb とする場合は、replace_rgb.txt として保存した上で、その拡張子.py に書き換え、**replace_rgb.py** として保存することにより作成できる。

```
import os
from osgeo import gdal
import numpy as np

input_dir = r"C:¥clipped"
output_dir = r"C:¥clipped¥converted"
os.makedirs(output_dir, exist_ok=True)

for filename in os.listdir(input_dir):
    if filename.lower().endswith((".tif", ".tiff")):
        input_path = os.path.join(input_dir, filename)
        output_path = os.path.join(output_dir, filename)
```

[4] Python を使わずに、GDAL だけで処理することもできるが、手順が増えるため、一括処理できる Python コードを作成することとした。

```
        print(f"処理中: {filename}")

        ds = gdal.Open(input_path)
        cols = ds.RasterXSize
        rows = ds.RasterYSize

        r = ds.GetRasterBand(1).ReadAsArray()
        g = ds.GetRasterBand(2).ReadAsArray()
        b = ds.GetRasterBand(3).ReadAsArray()

        mask = (r == 0) & (g == 0) & (b == 0)
        r[mask] = 255
        g[mask] = 255
        b[mask] = 255

        driver = gdal.GetDriverByName("GTiff")
        out_ds = driver.Create(output_path, cols, rows, 3, gdal.GDT_Byte,
                               options=[ "COMPRESS=DEFLATE", "ZLEVEL=6" ,
"BIGTIFF=YES"])
        out_ds.SetGeoTransform(ds.GetGeoTransform())
        out_ds.SetProjection(ds.GetProjection())

        out_ds.GetRasterBand(1).WriteArray(r)
        out_ds.GetRasterBand(2).WriteArray(g)
        out_ds.GetRasterBand(3).WriteArray(b)

        for i in range(1, 4):
            out_ds.GetRasterBand(i).SetNoDataValue(255)

        out_ds.FlushCache()
        del out_ds
        del ds

print("処理完了")
```

③　OSGeo4W で作成した python コードを実行する

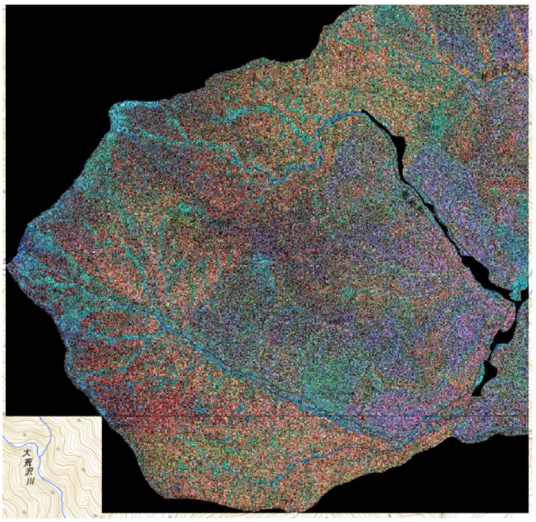
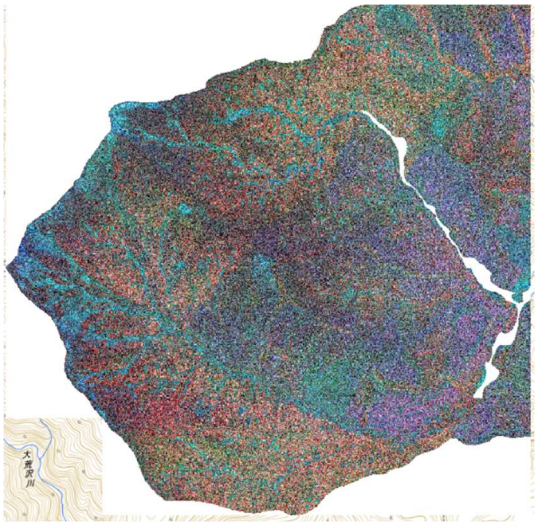
▼コマンド例

```
python "C:¥clipped¥replace_rgb.py"
```

▼解説

python	OSGeo4W において、python を使う場合は、最初に python と記述する。
"C:¥clipped¥replace_rgb.py"	②で作成した **python コードの保存先をフルパスで入力**する。ファイルパスは、当該ファイルを右クリックすることで調べることができる（Windows10 の場合は、SHIFT を押しながら当該ファイルを右クリックすることで調べることができる。）

▼参考｜変換結果

変換前	変換後
	
	

（２） 透過率をアルファバンドに設定する手順

gdalwarp のルート処理により、フォルダ内の tif を一括して、Nodata である RGB(255,255,255)をアルファバンドに指定し、透過率を記録させる手順は次のとおり。

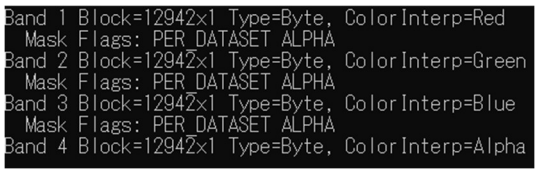
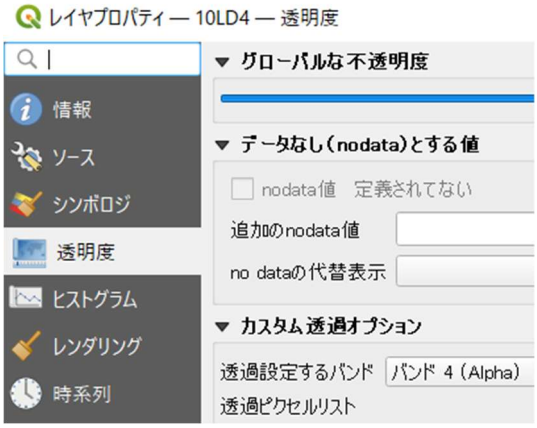
▼コマンド例

```
for %f in ("C:¥clipped¥*.tif") do gdalwarp -dstalpha -srcnodata "255 255 255" -co COMPRESS=DEFLATE -co ZLEVEL=6 -co BIGTIFF=YES "%f" "C:¥clipped¥alpha¥%~nxf"
```

▼解説

for %f in ("C:¥clipped¥*.tif")	C:¥clipped フォルダ内のすべての.tif ファイルに対してループ処理を行う前置き。
gdalwarp	GDAL の機能である。１（２）の geotiff 変換に用いたものと同じである。
-dstalpha	**アルファバンドに透明度を設定**する指示。
-srcnodata "255 255 255"	RGB(255,255,255)＝**白を透明として設定**する指示。
"%f"	変換前のファイルパス。今回はループ処理により一括変換を行うこととしており、前置きした%f がファイルパスとして“ ”で指定されている。
"C:¥clipped¥alpha¥%~nxf"	変換後のファイルパス。%~nf は、変換前のファイル名である%f から拡張子.tif を除いたものを変換後のファイル名とすることを指示するもの。これにより、clipped フォルダ内の alpha フォルダに**同一名称のファイルを出力**することができる。

▼参考

gdalinfo の結果	QGIS のレイヤプロパティ
	

（３） マップタイルの作成

① Python コード「gdal2tiles.py」の保存場所を確認

マップタイルの作成では、python コードである **gdal2tiles.py** を使用する。これは、QGIS でマップタイルを作成する際にバックグラウンドで使われているコードであるが、その保存場所が QGIS のバージョンによって異なる場合があるため、OSGeo4W で場所を確認する。（のちの作業でフルパスを指定する必要があるため、調べておく必要。）

▼コマンド例

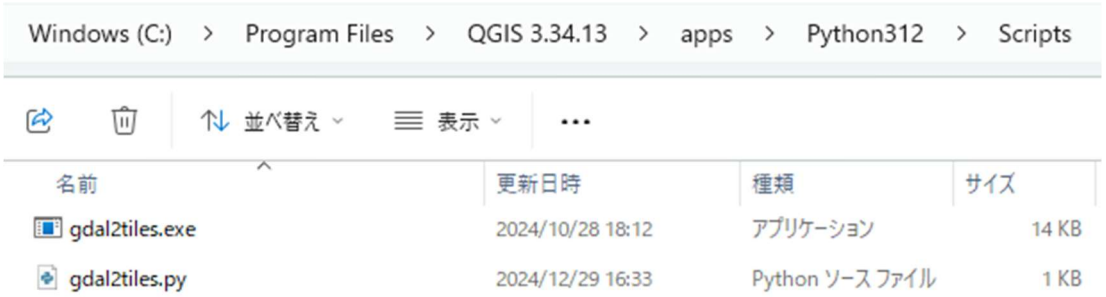
```
where gdal2tiles.py
```

▼解説

where	OSGeo4W において、最初に where と記述し、検索したいキーワードを入れると保存場所を教えてくれる。

▼参考｜検索結果

```
C:\Program Files\QGIS 3.34.13>where gdal2tiles.py
C:\Program Files\QGIS 3.34.13\apps\Python312\Scripts\gdal2tiles.py
```



② Nodata の値が統一され、アルファバンドが設定されていない場合

２（１）の作業を経て、Nodata の値を統一し、アルファバンドが設定されていない場合のデータからマップタイルを作成する手順は、２（１）で作成した geotiff 一式を仮想ラスタで便宜上つなぎ合わせた上で、その仮想ラスタの範囲に対して、マップタイルの作成を指示する流れとなる。

▼コマンド例

```
gdalbuildvrt -vrtnodata "255 255 255" "C:¥maptiles¥merge.vrt"
"C:¥clipped¥*.tif"
```

▼解説

gdalbuildvrt	複数のラスタを仮想的に一つにまとめた.vrt ファイルを生成する GDAL の機能。

-vrtnodata "255 255 255"	入力ラスタのうち RGB(255,255,255)＝白を NoData（無効値）として扱うための指示。ここで**指定した RGB 値が透過設定される**。
"C:¥maptiles¥merge.vrt"	vrt ファイルを保存するファイルを指定する。フォルダ名もファイル名（merge）も任意。
"C:¥clipped¥*.tif"	vrt ファイルを生成する基となる geotiff ファイルの保存先を指定する。*.tif とすることで、clipped フォルダ内の全ての tif がマージされる。

次に、以下のコマンドを入力する。

▼コマンド例

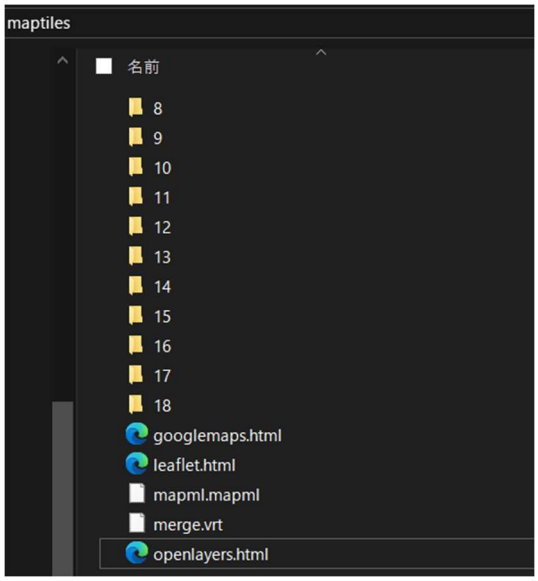
```
python "C:¥Program Files¥QGIS 3.22.16¥apps¥Python39¥Scripts¥gdal2tiles.py" --tiledriver=WEBP --webp-quality=75 --xyz -r antialias -z 8-18 --s_srs EPSG:6678 --exclude --resume --processes=12 "C:¥maptiles¥merge.vrt" "C:¥maptiles"
```

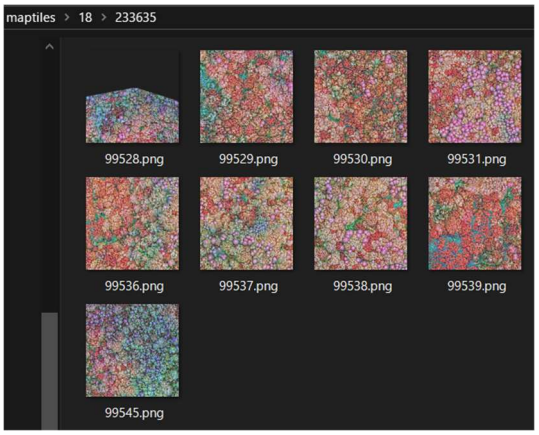
▼解説

python "C:¥...gdal2tiles.py"	gdal2tiles.py を Python 経由で実行する指示。①で where で検索した結果のとおり、フルパスで python コードの保存先を指定すること。
--tiledriver=WEBP	**タイルの拡張子**に webP を指定するもの。webp は、広く一般に使われている png と同様に透過設定もできる上、軽量かつ高圧縮が可能なため推奨する。
--webp-quality=75	webP の**圧縮品質を指定**するもの。0～100 を選択でき、一般に 75～85 程度がバランスがよい。なお、png を選択した場合は、これは記述する必要がない。
--xyz	**XYZ タイル形式を指定**するもの。これを指定しない場合は、WMTS（Web Map Tile Service）形式になってしまう。
-r antialias	**リサンプリング方法にアンチエイリアスを指定**するもの。微地形表現図や林相識別図などのいわゆる画像系は、アンチエイリアスを選択しておくと、画像の滑らかさを確保できる。
-z 8-18	**ズームレベルを指定**するもの。ZL-18 まであれば、基本的には十分な解像度を得られる。日本全国を対象とするタイルを生成する場合は、ZL-5 から作成することも検討。
--s_srs EPSG:6678	タイルの生成に用いる geotiff の座標系を指定するもの。指定がない場合は、EPSG:3857（WEB メルカトル）とされてしまう。
--exclude	保存先に指定されているフォルダに、既に存在する出力タイルの生成をスキップさせるための指示。

--resume	作業が中断してしまった場合に、前回の中断したところから処理を再開することを指示するもの。
--processes=12	並列処理に使用する CPU プロセス数を指示するもの（マルチコアで高速処理を図ることができる）。数字は個々の PC 環境に応じて選択すること。
"C:¥maptiles¥merge.vrt"	マップタイルの基となる仮想ラスタ（gdalbuildvrt で生成したもの）参照先を指定するもの。
"C:¥maptiles"	生成したマップタイルの保存先を指定するもの。

▼参考｜生成結果





③ Nodata の値がアルファバンドで透過設定されている場合

2（2）の作業を経て、アルファバンドが設定された geotiff に基づき、マップタイルを作成する場合は、既に透過設定が行われており、仮想ラスタを生成する過程で、特定の RGB 値について透過させなさいという指示を与える必要がないため、コマンドの一部を省略することができる。これ以外は、②と同様である。

アルファバンドの設定がない場合	アルファバンドの設定がある場合
gdalbuildvrt -vrtnodata "255 255 255" "C:¥maptiles¥merge.vrt" "C:¥clipped¥*.tif"	gdalbuildvrt "C:¥maptiles¥merge.vrt" "C:¥clipped¥*.tif"

3　ラスタ（ラスタ値）データのマップタイル化

DEM や DCHM などのラスタ値が格納されたラスタデータをデータ PNG 型のマップタイルに変換する手順を記載する。

マップタイル化する前に処理すべきこととして、第 1 章（p3～）で記載した事前準備のほか、**Nodata の値を統一しておき、タイル生成時に当該 Nodata を無効化処理できるようにしておくこと**が重要である。第 2 章（p8～）においても Nodata の統一について言及しているが、この第 3 章で紹介する手順（ラスタ値が格納されたラスタデータ）については、Nodata を-9999 に統一することが異なる。データ PNG を生成する過程において必ずしも Nodata を-9999 とすることが必須ではないものの、以下に記載する計算式（標高値を RGB 値に変換する過程）においてエラーが生じず、通常の標高値との混同が生じないものの代表例として-9999 を採用することを推奨するものである。

標高値の RGB 値への変換にあっては、いくつかの計算式があるが、本稿においては、①Terrain-RGB と②PNG 標高タイルの 2 種類の生成について手順を紹介する。

① Terrain-RGB

Terrain-RGB は、**Mapbox 社で考案された以下の変換式**が採用されている。QGIS や Maplibre GL JS などで表示でき、FOSS4G との相性がよい。記述できる標高値は、RGB（0,0,0）の-10,000m から RGB（255,255,255）の 1,667,722m である。

標高値 = -10000+(R×256×256+G×256+B)×0.1

② PNG 標高タイル

PNG 標高タイルは、**国土地理院の標高タイル**で採用されている変換式を用いるものである。日本国内のデフォルトとして採用されている変換式のため、他機関データと組み合わせて使用する場合に利点がある。記述できる標高値は、RGB（128,0,0）の-83,886m から RGB（127,255,255）の 83,886m である。

$X = 2^{16}×R+2^{8}×G+B$ として

$X < 2^{23}$ の場合 ： 標高値 = X×0.01

$X > 2^{23}$ の場合 ： 標高値 $=(X-2^{24})×0.01$

（1） Nodata とする値を統一する手順

① Nodata に指定されている値を確認する

GDAL の gdalinfo を用い、Nodata が設定されているか、設定されている場合は、どのラスタ値が Nodata になっているかを確認する。

▼コマンド例

```
gdalinfo "C:¥geotiff¥584146.tiff"
```

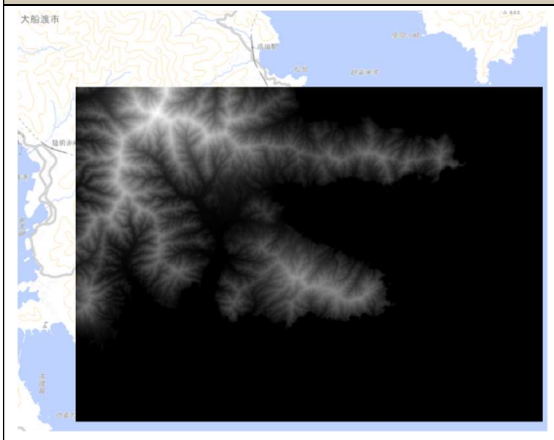
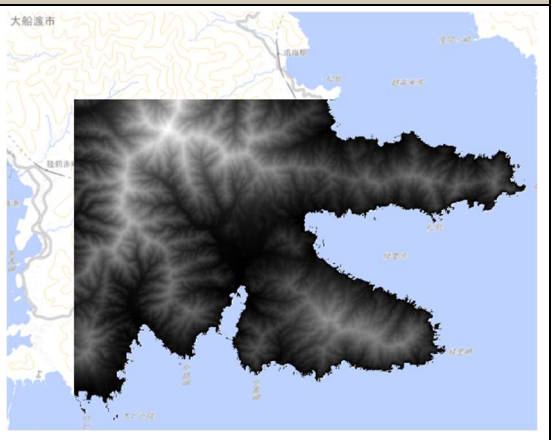
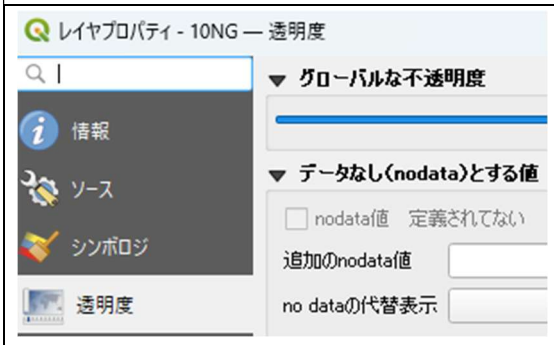
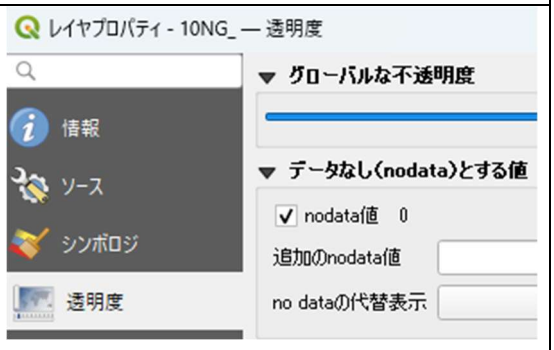
▼参考

Nodata が設定されていない場合の出力結果

```
Center      ( 141.8250000,  39.0416667) (141d49'30.00"E, 39d 2'30.00"N)
Band 1 Block=13500x1 Type=Float32, ColorInterp=Gray
```

Nodata に-9999 が指定されている場合の出力結果

```
Center      ( 141.8125000,  39.0416667) (141d48'45.00"E, 39d 2'30.00"N)
Band 1 Block=11250x1 Type=Float32, ColorInterp=Gray
  NoData Value=-9999
```

Nodata が指定されていない例	Nodata を指定されている例
	
	

② Python コード「gdal_calc.py」の保存場所を確認

ラスタ値の変換には、Python コードの **gdal_calc.py** を用いる。保存場所が QGIS のバージョンによって異なる場合があるため、OSGeo4W で場所を確認する。

▼コマンド例

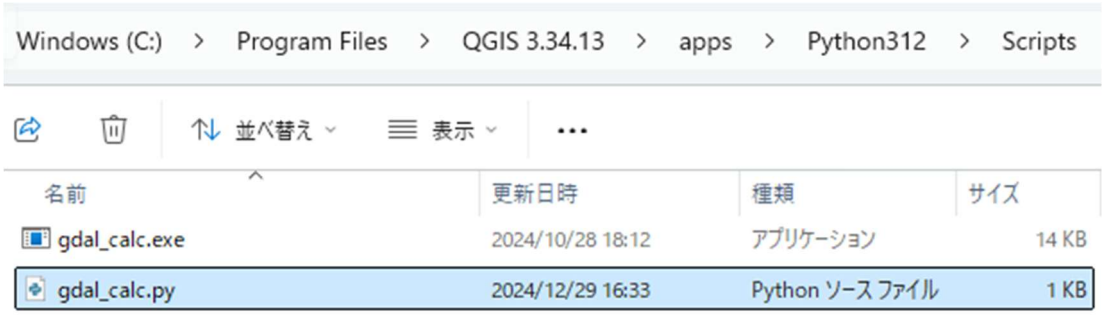
```
where gdal_calc.py
```

▼解説

where	OSGeo4W において、最初に where と記述し、検索したいキーワードを入れると保存場所を教えてくれる。

▼参考｜検索結果

```
C:\Program Files\QGIS 3.34.13>where gdal_calc.py
C:\Program Files\QGIS 3.34.13\apps\Python312\Scripts\gdal_calc.py
```



③ Nodata が指定されていない geotiff に新たに Nodata を指定する

gdalinfo で確認した結果、Nodata が指定されていない geotiff が処理データに含まれていることが確認された場合は、QGIS で展開の上、Nodata とするべき値を特定したうえで、**gdal_translate** のループ処理により、一括して特定の値を Nodata に位置付ける。これは、あくまで既存のラスタ値の一つを Nodata に位置付けるものであり、例えば、航空レーザ測量業務の対象範囲外について、あらかじめラスタ値 0 が与えられていた一方、その数値が Nodata に位置付けられていない（ただの 0m として扱われている）場合などに用いるものである。

一方、既に別の値（例えば 0）が Nodata に位置付けられていて、その数値を-9999 に置き換え、-9999 を Nodata として用いていくこととする場合は、以下の④で解説するので用いる方法を間違えないよう留意すること。

▼コマンド例

```
for %f in (C:¥ original¥*.tif) do gdal_translate -a_nodata 0 "%f"
"C:¥Converted¥%~nf.tif"
```

▼解説

for %f in (C:¥original¥*.tif)	C:¥original フォルダ内のすべての.tif ファイルを対象にループ処理を行うための前置き。***.tif とすることで、original フォルダ内の全ての tif を変換の対象とすることができる。**
gdal_translate	投影変換を伴わないラスタの変換などに使う GDAL の機能。
-a_nodata 0	ラスタ値＝0 を Nodata に指定する指示。
"%f"	変換前のファイルパス。今回はループ処理により一括変換を行うこととしており、前置きした%f がファイルパスとして“ ”で指定されている。
"C:¥Converted¥%~nf.tif"	変換後のファイルパス。%~nf は、変換前のファイル名である%f から拡張子.tif を除いたものを変換後のファイル名とすることを指示するもの。これにより、Converted フォルダに**同一名称のファイルを出力**することができる。

▼参考

```
Center      ( 141.8250000,  39.0416667) (141d49'30.00"E, 39d 2'30.00"N)
Band 1 Block=13500x1 Type=Float32, ColorInterp=Gray
```

↓

```
Center      ( 141.8250000,  39.0416667) (141d49'30.00"E, 39d 2'30.00"N)
Band 1 Block=13500x1 Type=Float32, ColorInterp=Gray
  NoData Value=0
```

※Nodata に 0 が指定された

④ Nodata に指定されている値を別の値に置き換えて Nodata として扱う

gdalinfo で確認した結果、別の値（例えば 0）が既に Nodata に位置付けられている場合において、その Nodata とされているラスタ値（例えば 0）を-9999 に置き換え、-9999 を Nodata として用いていくこととする場合は、Python コードの **gdal_calc.py** を用いる。③の手順で使った gdal_translate は、あくまで-9999 を新たに Nodata に設定することはできたとしても、すでに Nodata とされている 0 は 0 のままであり、置き換えができるわけではないことに留意する必要がある。

以下のコードをテキストエディタにコピーし、"C:¥original"にラスタ値を変換したい geotiff が保存されているフォルダのパスを、"C:¥Converted"にラスタ値の一部を-9999 に置き換えた geotiff を保存したいフォルダのパスを入力し、"C:¥Program Files¥QGIS 3.34.13¥bin¥python.exe"と"C:¥Program Files¥QGIS 3.34.13¥apps¥Python312¥Scripts¥gdal_calc.py"に②の手順で調べた python.exe と gdal_calc.py の保存先を、 0 に変換したいラスタ値を入力し、python コードとして保存する。テキストエディタがない場合は、メモ帳アプリにおいて、replace_to_nodata.txt として保存した上で、その拡張子を.py に置き換え、**replace_to_nodata.py** として保存することにより作成できる。なお、gdal_calc_path = r の後ろは、シングルクォート‘ ’で括られていることに注意すること。

```
import os
import glob
import subprocess

input_dir = r"C:¥original"
output_dir = r"C:¥Converted"
gdal_calc_path = r"C:¥Program Files¥QGIS 3.34.13¥bin¥python.exe"
"C:¥Program Files¥QGIS 3.34.13¥apps¥Python312¥Scripts¥gdal_calc.py"'

os.makedirs(output_dir, exist_ok=True)
tiff_files = glob.glob(os.path.join(input_dir, "*.tif"))

for tiff_path in tiff_files:
    filename = os.path.basename(tiff_path)
```

```
    output_path = os.path.join(output_dir, filename)

    cmd = f'{gdal_calc_path} -A "{tiff_path}" --outfile="{output_path}" --
calc="A*(A!=0)+(-9999)*(A==0)" --NoDataValue=-9999 --overwrite'

    print("Running:", cmd)
    subprocess.run(cmd, shell=True, check=True)

print("処理完了")
```

⑤ OSGeo4W で作成した python コードを実行する

▼コマンド例

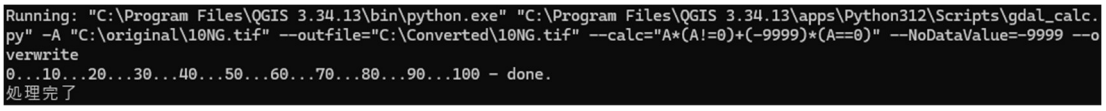
```
python "C:¥replace_to_nodata.py"
```

▼解説

python	OSGeo4W において、python を使う場合は、最初に python と記述する。
"C:¥replace_to_nodata.py"	④で作成した **python コードの保存先をフルパスで入力**する。ファイルパスは、当該ファイルを右クリックすることで調べることができる（Windows10 の場合は、SHIFT を押しながら当該ファイルを右クリックすることで調べることができる。）

▼参考

このようにコマンドプロンプトが流れていけば成功である。



（２） Terrain-RGB を生成する

① Nodata を無効化処理する（Nodata として扱っていた-9999 をラスタ値-9999 として扱う）

Terrain-RGB は、ラスタ値を RGB 値に変換した geotiff を生成し、当該 geotiff において、Nodata とすべき値（RGB 値）を透過設定しつつ、マップタイルを生成する、という二段階を経る。ただし、ラスタ値

を RGB 値に変換する上では、Nodata の設定がある geotiff を扱うことができず、一旦、Nodata とされている値を再びラスタ値として認識できるようにする（Nodata を無効化処理する）必要がある[5]。

Nodata の無効化処理には、GDAL の **gdal_translate** を用い、ループ処理によりフォルダ内の tif を一括して変換する。

▼コマンド例

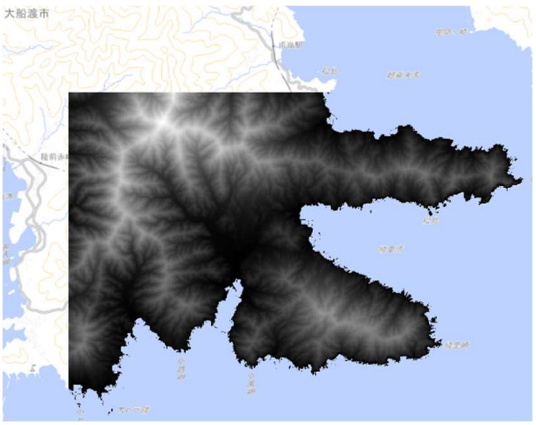
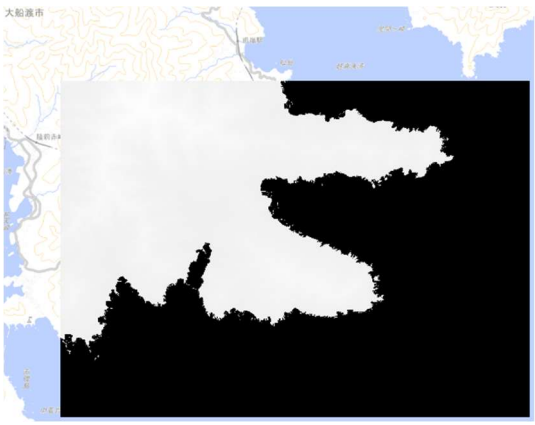
```
for %f in ("C:¥converted¥*.tif") do gdal_translate -a_nodata none "%~f" "C:¥Disabled¥%~nxf"
```

▼解説

for %f in ("C:¥converted¥*.tif")	C:¥converted フォルダ内のすべての.tif ファイルを対象にループ処理を行うための前置き。***.tif とすることで、converted フォルダ内の全ての tif を変換の対象とすることができる。**
gdal_translate	投影変換を伴わないラスタの変換などに使う GDAL の機能。
-a_nodata none	Nodata を無効化処理する指示。
"%~f"	変換前のファイルパス（フルパス）。今回はループ処理により一括変換を行うこととしており、前置きした%f がファイルパスとして“ ”で指定されている。
"C:¥Disabled¥%~nxf"	変換後のファイルパス。%~nxf は、変換前のファイル名を変換後のファイル名とすることを指示するもの。これにより、Converted フォルダに**同一名称のファイルを出力**することができる。

[5] そうであるならば、前処理において-9999 を Nodata としなければよいとの考え方もあるが、Nodata に何が設定されているかの確認の習慣をつけることと、後段で紹介する PNG 標高タイルの生成において重要となるため、このような手順の紹介としている。

▼参考

Nodata が指定されている例	Nodata が無効化された例
	

② Python パッケージツールの「rio-rgbify」をインストールする

ラスタ値な geotiff を Terrain-RGB の変換式に基づく RGB 値な geotiff への変換は、python パッケージツールの **rio-rgbify** を用いる。rio-rgbify は、QGIS をインストールしただけの初期環境ではインストールされていないため、OSGeo4W において以下のコマンドにより、インストールを実行する。

▼コマンド例

```
pip install rio-rgbify
```

次に、python パッケージツールが保存されている場所を確認するため、OSGeo4W において以下のコマンドにより、検索を行う。

▼コマンド例

```
where pip
```

▼参考

筆者の PC 環境では、「C:¥Program Files¥QGIS 3.34.13¥apps¥Python312¥Scripts」にあることが確認できる。この検索結果は、以下③において使用するので**メモしておく**こと。

```
C:\Program Files\QGIS 3.34.13>where pip
C:\Program Files\QGIS 3.34.13\apps\Python312\Scripts\pip.bat
```

③　ラスタ値を RGB 値に変換した geotiff を生成する

OSGeo4W のコマンドで python パッケージツールが起動するよう、以下のコマンドにより環境設定を行う[6]。C:¥Users…¥Scripts は、②で調べた結果を指定する（ただし、¥pip.bat は含めず、**¥Script 止め**とすること）。

▼コマンド例

```
set PATH=%PATH%;C:¥Users¥naoki¥AppData¥Roaming¥Python¥Python312¥Scripts
```

▼参考

以下のとおり何も表示されず、次のコマンド入力が行えるようになっていれば、正常に進んでいる。

```
C:\Program Files\QGIS 3.34.13>set PATH=%PATH%;C:\Users\naoki\AppData\Roaming\Python\Python312\Scripts

C:\Program Files\QGIS 3.34.13>
```

次に、rio-rgbify のループ処理により、フォルダ内の tif を一括して Terrain-RGB の変換式による RGB 値化された geotiff を生成する。

▼コマンド例

```
for %f in (C:¥Disabled¥*.tif) do rio rgbify -b -10000 -i 0.1 --co COMPRESS=DEFLATE --co ZLEVEL=6 "%f" "C:¥rgb¥%~nxf"
```

▼解説

for %f in (C:¥Disabled¥*.tif)	C:¥Disabled フォルダ内のすべての.tif ファイルを対象にループ処理を行うための前置き。***.tif とすることで、Disabled フォルダ内の全ての tif を変換の対象とすることができる**。
rio rgbify	次の記述の計算式に基づき、**ラスタ値を RGB 値に変換**する python パッケージツール。
-b -10000	**RGB（0,0,0）の時に-10,000m と規定するベース値**を表す。Terrain-RGB の変換式に基づき、-10,000 としている。
-i 0.1	**RGB 値が 1 変わるたびにラスタ値をどのように動かすかを規定するインターバル値**。Terrain-RGB の変換式に基づき、0.1 としている。

[6] PC のシステムプロパティから、この環境変数を経常指定することもできるようであるが、筆者の環境ではそれがうまくいかず、rio-rgbify を使用するたびに、このコマンド処理を行っている。

--co COMPRESS=DEFLATE	RGB 値化した geotiff を Deflate 圧縮するように指示するもの。（任意）
--co ZLEVEL=6	Deflate 圧縮のレベル（1-9）のうち、レベル 6（比較的高圧縮かつ処理が早め）を設定するもの。（任意）
"%f"	変換前のファイルパス（フルパス）。今回はループ処理により一括変換を行うこととしており、前置きした%f がファイルパスとして“ ”で指定されている。
"C:¥rgb¥%~nxf"	変換後のファイルパス。%~nxf は、変換前のファイル名を変換後のファイル名とすることを指示するもの。これにより、rgb フォルダに**同一名称のファイルを出力**することができる。

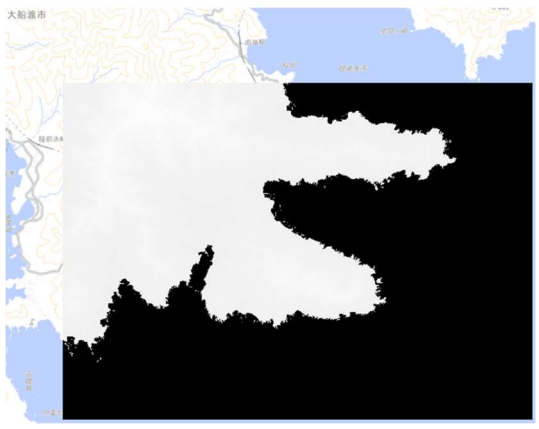
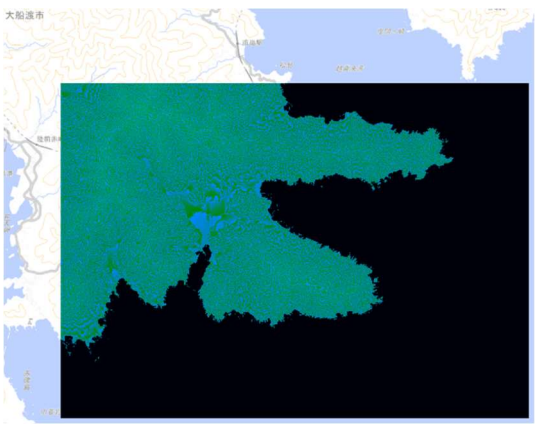
▼参考

以下のようにコマンドが流れていけば、正常に進んでいる。

```
C:\Program Files\QGIS 3.34.13>for %F in (C:\Disabled\*.tif) do rio rgbify -b -10000 -i 0.1 --co COMPRESS=DEFLATE --co ZL
EVEL=6 "%F" "C:\rgb\%~nxF"

C:\Program Files\QGIS 3.34.13>rio rgbify -b -10000 -i 0.1 --co COMPRESS=DEFLATE --co ZLEVEL=6 "C:\Disabled\10NG.tif" "C:
\rgb\10NG.tif"

C:\Program Files\QGIS 3.34.13>
```

ラスタ値の geotiff	RGB 値化された geotiff
	

④ マップタイルの作成

微地形表現図などのマップタイル化に用いたものと同様、python コードである **gdal2tiles.py** を使用する。ただし、以下のようにコマンド入力で数点異なるところがあるため、留意すること。

▼コマンド例

```
gdalbuildvrt -vrtnodata "0 0 10" "C:¥maptiles¥merge.vrt" "C:¥rgb¥*.tif"
```

▼解説

gdalbuildvrt	複数のラスタを仮想的に一つにまとめた.vrt ファイルを生成する GDAL の機能。
-vrtnodata "0 0 10"	入力ラスタのうち **RGB(0,0,10)＝NoData（無効値）**として扱うための指示。ここで指定した RGB 値が透過設定される。RGB（0,0,10）は、Terrain-RGB の変換式によればラスタ値＝-9999 であり、**もともと Nodata であるべきラスタ値を透過させることに相当**する。
"C:¥maptiles¥merge.vrt"	vrt ファイルを保存するファイルを指定する。フォルダ名もファイル名（merge）も任意。
"C:¥rgb¥*.tif"	vrt ファイルを生成する基となる geotiff ファイルの保存先を指定する。*.tif とすることで、rgb フォルダ内の全ての tif がマージされる。

▼コマンド例

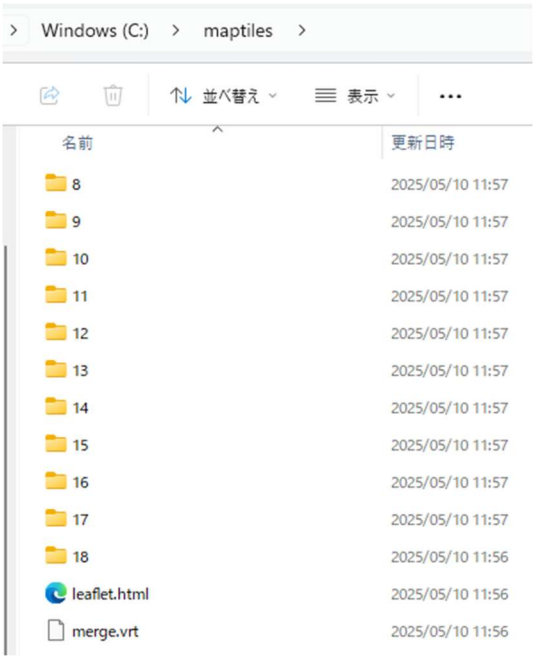
```
python "C:¥Program Files¥QGIS 3.34.13¥apps¥Python312¥Scripts¥gdal2tiles.py" --tiledriver=PNG -r near --xyz -z 8-18 --s_srs EPSG:6668 --exclude --resume --processes=12 "C:¥maptiles¥merge.vrt" "C:¥maptiles"
```

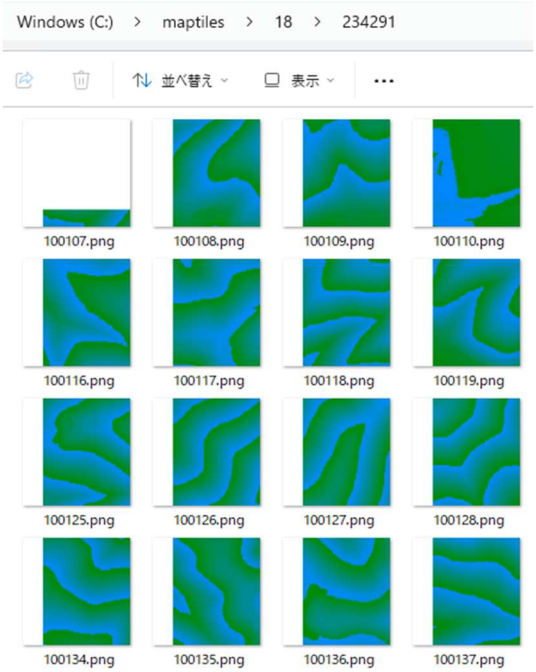
▼解説

python "C:¥...gdal2tiles.py"	gdal2tiles.py を Python 経由で実行する指示。フルパスで python コードの保存先を指定すること。
--tiledriver=PNG	**タイルの拡張子**に png を指定するもの。
-r near	**リサンプリング方法に最近傍を指定**するもの。アンチエイリアスなどの計算過程が含まれるもを選択することは、RGB 値上で計算が行われ、現実にそぐわない標高値が返されるおそれがあり、お勧めしない。
--xyz	**XYZ タイル形式を指定**するもの。これを指定しない場合は、WMTS（Web Map Tile Service）形式になってしまう。
-z 8-18	**ズームレベルを指定**するもの。ZL-18 まであれば、基本的には十分な解像度を得られる。日本全国を対象とするタイルを生成する場合は、ZL-5 から作成することも検討。
--s_srs EPSG:6668	タイルの生成に用いる geotiff の座標系を指定するもの。指定がない場合は、EPSG:3857（WEB メルカトル）とされてしまう。
--exclude	保存先に指定されているフォルダに、既に存在する出力タイルの生成をスキップさせるための指示。

--resume	作業が中断してしまった場合に、前回の中断したところから処理を再開することを指示するもの。
--processes=12	並列処理に使用する CPU プロセス数を指示するもの（マルチコアで高速処理を図ることができる）。数字は個々の PC 環境に応じて選択すること。
"C:¥maptiles¥merge.vrt"	マップタイルの基となる仮想ラスタ（gdalbuildvrt で生成したもの）参照先を指定するもの。
"C:¥maptiles"	生成したマップタイルの保存先を指定するもの。

▼参考｜生成結果





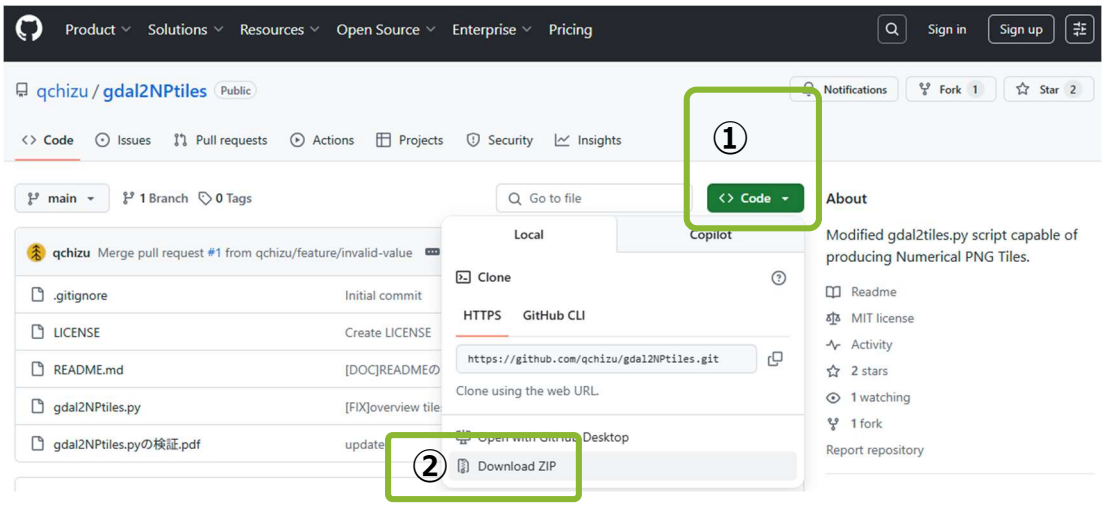
（３） PNG 標高タイルを生成する

PNG 標高タイルは、**全国 Q 地図の運営者さんが公開**している python コード **gdal2NPtiles.py** を使い生成する。（１）の手順に基づき、Nodata を-9999 に統一したラスタ値な geotiff が用意できていれば、そのまま、マップタイルの生成に移行できる。

① gdal2NPtiles.py のダウンロード

GitHub から ZIP をダウンロードし、その中にある gdal2NPtiles.py を任意の場所に保存する。

▼参考｜GitHub 画面



② gdal2NPtiles.py の実行

▼コマンド例

```
gdalbuildvrt -vrtnodata -9999 "C:¥maptiles¥merge.vrt"
"C:¥converted¥*.tif"
```

▼解説

gdalbuildvrt	複数のラスタを仮想的に一つにまとめた.vrt ファイルを生成する GDAL の機能。
-vrtnodata -9999	入力ラスタのうち**ラスタ値-9999 を NoData（無効値）**として扱うための指示。ここで指定した RGB 値が透過設定される。
"C:¥maptiles¥merge.vrt"	vrt ファイルを保存するファイルを指定する。フォルダ名もファイル名（merge）も任意。
"C:¥ converted ¥*.tif"	vrt ファイルを生成する基となる geotiff ファイルの保存先を指定する。*.tif とすることで、rgb フォルダ内の全ての tif がマージされる。

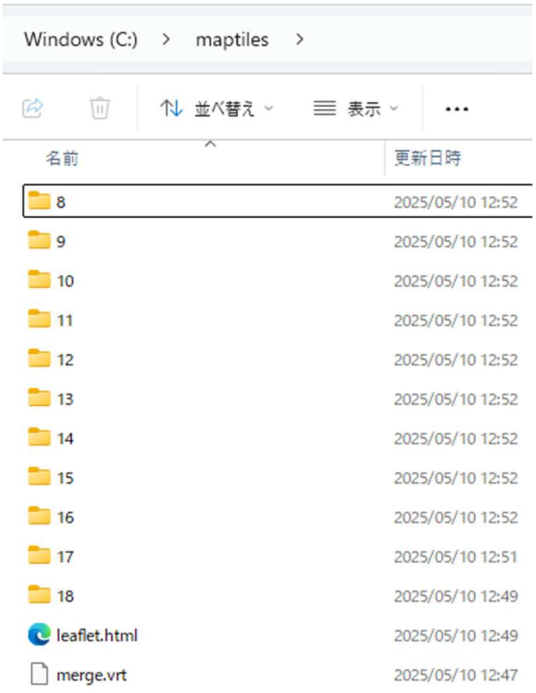
▼コマンド例

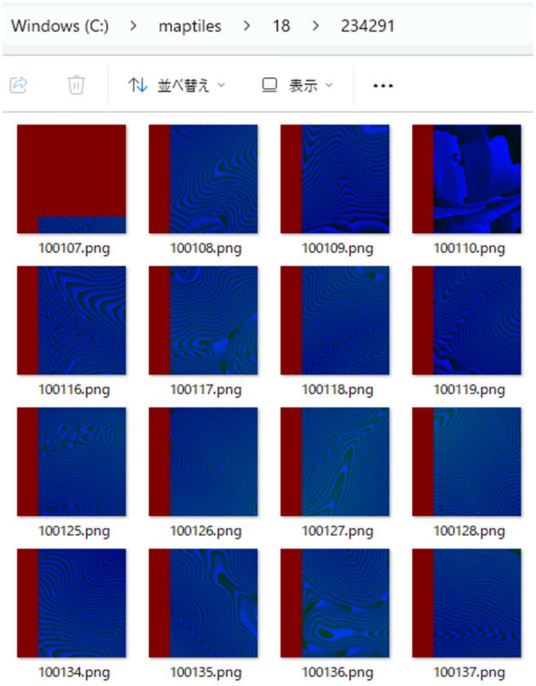
```
python "C:¥gdal2NPtiles.py" --numerical -z8-18 --xyz --srcnodata=-9999 --processes=12 "C:¥maptiles¥merge.vrt" "C:¥maptiles"
```

▼解説

python "C:¥gdal2NPtiles.py"	gdal2NPtiles.py を Python 経由で実行する指示。フルパスで python コードの保存先を指定すること。
--numerical	数値 PNG タイル生成モードを有効にする指示。
-z8-18	ズームレベルを指定するもの。ZL-18 まであれば、基本的には十分な解像度を得られる。日本全国を対象とするタイルを生成する場合は、ZL-5 から作成することも検討。
--xyz	**XYZ タイル形式を指定**するもの。これを指定しない場合は、WMTS（Web Map Tile Service）形式になってしまう。
--srcnodata=-9999	タイルの生成に用いる **geotiff の Nodata を-9999 と認識**させる指示。
--processes=12	並列処理に使用する CPU プロセス数を指示するもの（マルチコアで高速処理を図ることができる）。数字は個々の PC 環境に応じて選択すること。
"C:¥maptiles¥merge.vrt"	マップタイルの基となる仮想ラスタ（gdalbuildvrt で生成したもの）参照先を指定するもの。
"C:¥maptiles"	生成したマップタイルの保存先を指定するもの。

▼参考｜生成結果





4　ベクタデータのマップタイル化

森林計画図や樹種ポリゴン等のベクタデータから、ベクトルタイルの作成手順を紹介する。

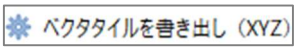
（１） 事前準備

ラスタタイルの場合は、第１章として事前準備を設けたが、ベクトルタイルの場合は、以下の点を GIS 等において、処理しておくだけである。具体の作業手順については、他誌が詳しいので省略する。

① １つのベクトルタイルとする範囲について、ベクタデータをクリップ、マージ[7]しておく。

② 属性フィールドの記載がそのまま使われることから、略称や記号表記について、視認性のよい表現に変換しておく。

（２） マップタイルの生成

QGIS のプロセッシングツールのうち、  を用いる。パラメータの入力は次のとおり入力する。

① ファイルのテンプレート

初期入力値 **{z}/{x}/{y}.pbf** のままでよい。

② 入力レイヤ

１つのベクトルタイルとしたいレイヤを全て選択する。（基本的には、上記のとおり、あらかじめ１つのレイヤにマージしておき、それのみを選択することを推奨。）

③ 最小ズーム値

ラスタタイルと異なり、ベクタデータの性質により１タイルあたりのデータ容量が変動する。特に、ズームレベルが小さい場合、１タイルあたりのデータ容量が大きくなり、WEB 利用に向かいない場合もあるので、注意が必要である。

④ 最大ズーム値

ラスタタイルと同様、ズームレベルを大きくすればするほど、タイルのサイズが小さくなる。元のベクタデータのポリゴン等のサイズがそれほど小さくない場合は、ズームレベルに大きな値を設定しなくても十分な場合もある。そのような場合に、ズームレベルを大きくすると、１のポリゴンが不必要に四角く切り分けられて、かえって見づらい場合もある。

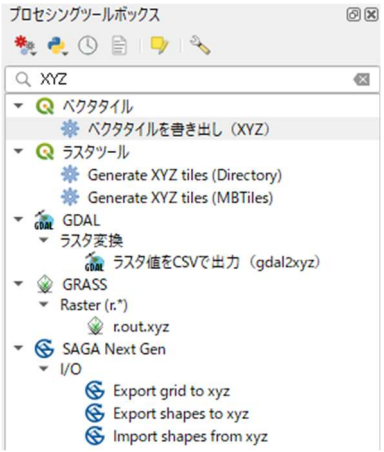
⑤ 領域、出力フォルダ

適宜指定する。

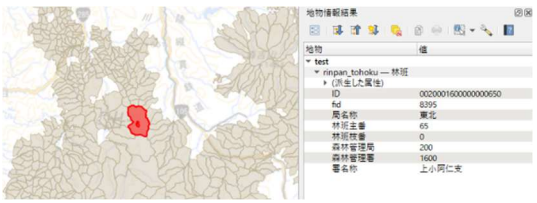
---

[7] 必ずしも１つのベクタデータとなっている必要はないが、第５章で紹介する style.json の記述が複雑となることから、マージしておくことを推奨する。

▼QGIS 画面



▼参考｜林班ポリゴン（１ポリゴンあたり 0.3km^2/属性 5 列）の例

ズームレベル小（8）の例	ズームレベル大（18）の例
種類 / サイズ PBF ファイル 324 KB PBF ファイル 603 KB PBF ファイル 1,155 KB PBF ファイル 1,469 KB	種類 / サイズ PBF ファイル 1 KB PBF ファイル 1 KB PBF ファイル 1 KB PBF ファイル 1 KB
	
１タイルあたり 1-2MB くらいであれば、通信上の支障も少なく、タイルでポリゴンより十分に大きい。	タイルがポリゴンに比べ小さすぎ、データ容量も小さすぎる。この例では、ZL-18 は不要。

5　ベクトルタイルのスタイル JSON の作成

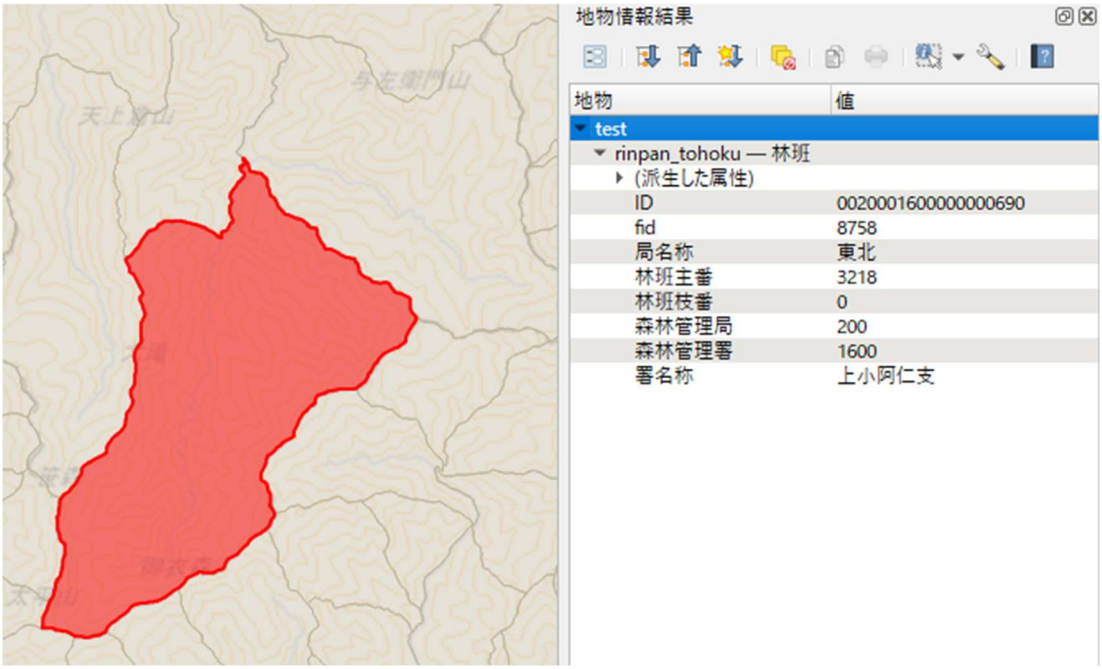
ベクトルタイルは、ラスタタイルと異なり、属性情報に基づきスタイル設定を任意で設定できることがメリットである一方、ベクトルタイルの利用者が QGIS 等において、一からスタイル設定を行うのはやや手間である。そこで、基本的なスタイル設定はベクトルタイルの配信と合わせ、style.json も配信しておくことが望まれる。style.json は、テキストエディタで記述する。

（１） ベクトルタイルの内容を確認する

QGIS でベクトルタイル 

を接続し、地物情報  を確認する。選択したポリゴンの名称を確認し、今回の事例の場合は、style.json の記述に必ず必要な**レイヤ名称**（rinpan_tohoku ― 林班）と、**ラベル表示の設定に使いたい属性名称**（林班主番）をメモしておく。

▼QGIS 画面



（２） json データを作成する

style.json の基本骨格は次のとおりである。以下の例は、

① レイヤ名（rinpan_tohoku ― 林班）とされていたベクトルデータから生成し、

② file:///C:/maptiles/{z}/{x}/{y}.pbf として保存されているベクトルタイルについて、

③ ズームレベル 8-18 は、透過率 50%の深緑でポリゴンを塗りつぶし、

④ ズームレベル 8-18 は、透過率 0%、1.0 ポイント幅の深緑の実線でポリゴンを括り、

⑤ ズームレベル 14-18 は、白で縁取りされた 16 ポイントの黒文字で林班主番をラベル表示する

という記述がされている。このほかにも、属性値によって塗りつぶしの色を変えるなどのアレンジができるが、林野庁が公開している**樹種ポリゴン**や**崩壊箇所等判読結果**の style.json なども参考に学習してほしい。

```
{
  "version": 8,
  "name": "林班ポリゴン初期設定",
  "sources": {
    "rinpan": {
      "type": "vector",
      "tiles": [
        "file:///C:/maptiles/{z}/{x}/{y}.pbf"                    ①
      ],
      "minzoom": 8,
      "maxzoom": 18
    }
  },
  "layers": [
    {
      "id": "rinpan-fill",
      "type": "fill",
      "source": "rinpan",
      "source-layer": "rinpan_tohoku ― 林班",                    ②
      "minzoom": 8,
      "maxzoom": 18,
      "paint": {                                                 ③
        "fill-color": "rgba(51, 160, 44, 0.5)",                  ④
        "fill-outline-color": "rgba(51, 160, 44, 1.0)"
      }
    },
    {
      "id": "rinpan-label",
      "type": "symbol",
      "source": "rinpan",
```

```
    "source-layer": "rinpan_tohoku ― 林班",                    ①
    "minzoom": 14,
    "maxzoom": 18,
    "layout": {
      "text-field": ["get", "林班主番"],
      "text-font": ["Noto Sans Regular", "Arial Unicode MS"],
      "text-size": 16,
      "text-anchor": "center",                                  ⑤
      "symbol-placement": "point",
      "text-allow-overlap": false
    },
    "paint": {
      "text-color": "#000000",
      "text-halo-color": "#ffffff",
      "text-halo-width": 1
    }
  }
 ]
}
```

▼参考｜上記 style.json の表示例

ズームレベル 8-13 まで	ズームレベル 14-18 まで