R_GIS 03

Les packages qgisprocess et whitebox

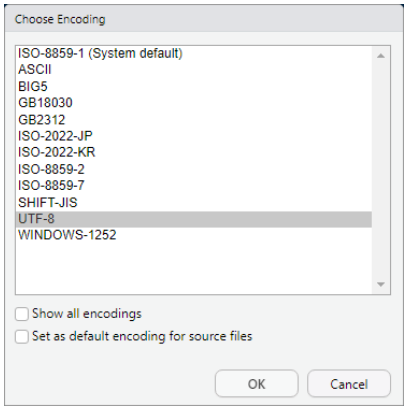
Septembre 2026

LIÈGE université
Gembloux
Agro-Bio Tech

TABLE DES MATIERES

1. Introduction

- L'objectif de ce tutoriel est de présenter les nombreuses possibilités offertes par le package qgisprocess qui permet d'exécuter les algorithmes présents dans l'environnement QGIS au sein d'un script R.
- Même si la plupart des géotraitements de base sont réalisables à l'aide des fonctions de packages tels que sf, terra ou encore dplyr, on peut trouver certains outils dans l'environnement QGIS qui n'ont pas d'équivalents dans ces librairies. Il est dès lors intéressant de pouvoir les exploiter en mode script dans R : c'est là que réside l'intérêt du package qgisprocess.
- Une attention est également portée à la mise en œuvre des composantes de la boîte à outils WhiteBox qui comporte notamment une série de fonctions hydrologiques.
- L'ensemble des exemples présentés dans ce tutoriel sont rassemblés au sein d'un script **R_GIS_03.r** disponible dans le jeu de données qui accompagne le présent document. Les numéros des paragraphes de ces notes d'exercice peuvent être utilisés comme point de repère pour retrouver les lignes de code dans le script.
- Les manipulations présentées dans ce tutoriel ont été testées dans l'environnement de R Studio, avec la version 4.6.1 de R.
- **Remarque** : le script **R_GIS_03.r** a été créé avec l'encodage « UTF-8 ». Si la version de R Studio dans laquelle le script est affiché utilise un autre encodage, certains caractères accentués ne s'afficheront pas correctement. Pour résoudre ce problème, il suffit de rouvrir le script avec la commande **[File] → [Reopen with encoding …]** et de sélectionner l'encodage « UTF-8 ».



2. Utilisation du package qgisprocess

2.1 Installation et configuration des packages qgisprocess

- L'installation du package qgisprocess se fait classiquement depuis le dépôt CRAN.
- La version utilisée dans ce tutoriel est 0.4.2.

```
# 1. Installation et chargement des librairies --
library(terra)
library(dplyr)
library(sf)
library(qgisprocess)
library(whitebox)

# version de R et de qgisprocess
R.version$version.string
packageVersion("qgisprocess")
```

```
> # version de R et de qgisprocess
> R.version$version.string
[1] "R version 4.6.1 (2026-06-24 ucrt)"
> packageVersion("qgisprocess")
[1] '0.4.2'
```

- La fonction ***qgis_configure()*** permet de s'assurer que qgisprocess a bien détecté une version de QGIS avec laquelle elle peut fonctionner correctement. Dans le cas présent, il s'agit de la version 3.40.5-Bratislava.

```
# vérifier que R trouve une version de qgisprocess correctement installée
qgis_configure()
```

```
QGIS version is now set to: 3.40.5-Bratislava
Using JSON for output serialization.
Using JSON for input serialization.
4 out of 4 available processing provider plugins are enabled.

Standard error message from 'qgis_process':
Problem with GRASS installation: GRASS was not found or is not correctly
installed


You now have access to 1515 algorithms from 7 QGIS processing providers.
```

- En cas de difficulté à trouver une version correcte de QGIS, il est nécessaire de préciser explicitement quelle version utiliser. L'exécution de la commande présentée ci-dessous doit être suivie d'un redémarrage du RStudio.

```
# forcer l'utilisation d'une version QGIS (nécessite de relancer RStudio)
#Sys.setenv(R_QGISPROCESS_PATH =
#           "C:/OSGeo4W/bin/qgis_process-qgis-ltr.bat")
```

2.2 Définition des chemins vers les répertoires

- Les données utilisées pour cet exercice sont rangées dans le répertoire /data_in. Créer une variable qui contient l'adresse de ce répertoire, ainsi qu'une autre variable qui contient celle du répertoire destiné à recevoir les résultats. Ces variables seront utilisées par la suite pour définir les noms des fichiers d'entrée et de sortie.

```
# 2. Chemin vers les données ---------
path0="C:/tmp/R_GIS_03"
path_in=paste0(path0,"/data_in")
path_out=paste0(path0,"/output")
```

2.3 Algorithmes disponibles dans qgisprocess

- La fonction ***qgis_algorithms()*** dresse la liste des algorithmes présents dans l'environnement QGIS de votre ordinateur.

```
# 3. liste des algorithmes reconnus par qgisprocess -------
df=qgis_algorithms()
names(df)
```

```
> names(df)
 [1] "provider"
 [2] "provider_title"
 [3] "algorithm"
 [4] "algorithm_id"
 [5] "algorithm_title"
 [6] "provider_id"
 [7] "can_cancel"
 [8] "deprecated"
 [9] "group"
[10] "has_known_issues"
[11] "help_url"
[12] "name"
[13] "requires_matching_crs"
[14] "short_description"
[15] "tags"
[16] "provider_can_be_activated"
[17] "default_raster_file_extension"
[18] "default_vector_file_extension"
[19] "provider_is_active"
[20] "provider_long_name"
[21] "provider_name"
[22] "supported_output_raster_extensions"
[23] "supported_output_table_extensions"
[24] "supported_output_vector_extensions"
[25] "supports_non_file_based_output"
[26] "provider_version"
[27] "provider_warning"
```

- Parmi les informations générées par cette fonction, on retrouve les noms des fournisseurs d'algorithmes. En fonction des extensions installées sur l'ordinateur utilisé, la liste proposée peut être différente de celle affichée dans la figure qui suit.

```
 3d   gdal native   pdal   qgis sagang    wbt
  1     57    266     17     41    587    546
```

- Les algorithmes du fournisseur « qgis » correspondent à ceux qui sont utilisés dans les fonctionnalités de base de QGIS.

```
# liste des algo QGIS
df_qgis=df[df$provider=="qgis",]
df_qgis$algorithm
```

```
> df_qgis$algorithm
 [1] "qgis:advancedpythonfieldcalculator"
 [2] "qgis:barplot"
 [3] "qgis:boxplot"
 [4] "qgis:checkvalidity"
 [5] "qgis:climbalongline"
 [6] "qgis:convertgeometrytype"
 [7] "qgis:definecurrentprojection"
 [8] "qgis:distancematrix"
 [9] "qgis:distancetonearesthublinetohub"
[10] "qgis:distancetonearesthubpoints"
```

- Les algorithmes du fournisseur « native » correspondent quant à eux au contenu de l'extension « Processing ».

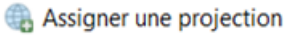
```
# liste des algo natifs (extension processing)
df_native=df[df$provider=="native",]
df_native$algorithm
```

```
> df_native$algorithm
  [1] "native:addautoincrementalfield"
  [2] "native:addfieldtoattributestable"
  [3] "native:adduniquevalueindexfield"
  [4] "native:addxyfields"
  [5] "native:affinetransform"
  [6] "native:aggregate"
  [7] "native:angletonearest"
  [8] "native:antimeridiansplit"
  [9] "native:arrayoffsetlines"
 [10] "native:arraytranslatedfeatures"
 [11] "native:aspect"
```

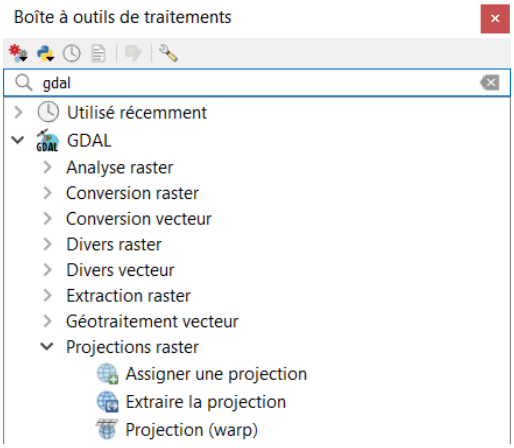
- Le fournisseur « gdal » propose pour sa part, l'ensemble des fonctionnalités de la librairie GDAL.

```
# liste des algo liés à GDAL
df_gdal=df[df$provider=="gdal",]
df_gdal$algorithm
```

```
> df_gdal=df[df$provider=="gdal",]
> df_gdal$algorithm
 [1] "gdal:aspect"
 [2] "gdal:assignprojection"
 [3] "gdal:buffervectors"
 [4] "gdal:buildvirtualraster"
 [5] "gdal:buildvirtualvector"
 [6] "gdal:cliprasterbyextent"
 [7] "gdal:cliprasterbymasklayer"
 [8] "gdal:clipvectorbyextent"
 [9] "gdal:clipvectorbypolygon"
[10] "gdal:colorrelief"
[11] "gdal:contour"
[12] "gdal:contour_polygon"
```

- La correspondance avec les outils présents dans la boîte à outils de traitements de QGIS peut être établie assez facilement. Ainsi l'algorithme «**gdal :assignprojection** » correspond à l'outil 

.



- Etant donné le nombre très important d'algorithmes disponibles, il peut être utile de rechercher parmi ceux-ci la présence de mots clés, comme dans l'exemple qui suit où les algorithmes dont le nom contient le terme « buffer » sont extraits de la liste complète.

```
# recherche d'algorithme sur base d'un mot clé
df$algorithm[grepl(pattern = "buffer", df$algorithm)]
```

```
> df$algorithm[grepl(pattern = "buffer", df$algorithm)]
 [1] "gdal:buffervectors"              "gdal:onesidebuffer"
 [3] "grass7:r.buffer"                 "grass7:r.buffer.lowmem"
 [5] "grass7:v.buffer"                 "native:buffer"
 [7] "native:bufferbym"                "native:multiringconstantbuffer"
 [9] "native:singlesidedbuffer"        "native:taperedbuffer"
[11] "native:wedgebuffers"             "qgis:variabledistancebuffer"
[13] "saga:featuresbuffer"             "saga:rasterbuffer"
[15] "saga:rasterproximitybuffer"      "saga:thresholdbuffer"
```

2.4 Exemple 1 – Créer des buffers multiples (anneaux)

- Dans ce premier exemple, nous utilisons l'algorithme « native:multiringconstantbuffer » qui permet de générer des buffers multiples à distance constante.
- La fonction ***qgis_descriptions()*** fourni une description de l'algorithme.

```
# Exemple 1 - commande "Tampons multi-anneaux" ---

algo="native:multiringconstantbuffer"
qgis_get_description(algo)
qgis_show_help(algo)
```

```
> qgis_description(algo)


          native:multiringconstantbuffer
"This algorithm computes multi-ring ('donuts') buffer
 for all the features in an input layer, using a fixe
d or dynamic distance and rings number."
```

- La description fournie est analogue à celle qui apparaît dans la boîte de dialogue de la commande dans l'interface de QGIS.

- La fonction ***qgis_show_help()*** décrit les différents paramètres à définir pour faire fonctionner correctement l'algorithme.

```
> qgis_show_help(algo)
Multi-ring buffer (constant distance) (native:multiringconstantbuffer)

----------------
Description
----------------
This algorithm computes multi-ring ('donuts') buffer for all the features in
 an input layer, using a fixed or dynamic distance and rings number.

----------------
Arguments
----------------

INPUT: Input layer
        Argument type:  source
        Acceptable values:
                - Path to a vector layer
RINGS: Number of rings
        Argument type:  number
        Acceptable values:
                - A numeric value
DISTANCE: Distance between rings
        Argument type:  distance
        Acceptable values:
                - A numeric value
OUTPUT: Multi-ring buffer (constant distance)
        Argument type:  sink
        Acceptable values:
                - Path for new vector layer

----------------
Outputs
----------------

OUTPUT: <outputVector>
        Multi-ring buffer (constant distance)
```

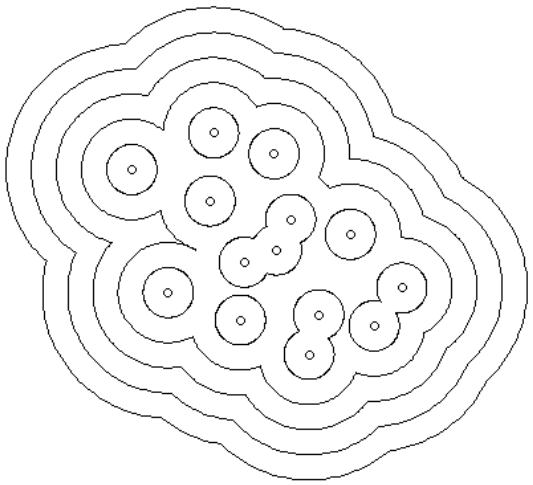
- L'exécution d'un algorithme est réalisée avec la fonction ***qgis_run_algorithm()***. Dans l'exemple qui suit, 5 anneaux de 1000 m sont générés autour des points du shapefile **localites.shp**. Les arguments utilisés sont INPUT, RINGS, DISTANCE et OUTPUT. Il est important de respecter la casse des noms d'arguments. Dans le cas présent ils sont écrits en majuscule, mais ce n'est pas toujours le cas.

```
f_in=paste0(path_in,"/localites.shp")
f_out=paste0(path_out,"/buffers_5x1km.shp")
result = qgis_run_algorithm(
  algorithm=algo,
  INPUT = f_in,
  RINGS = 5,
  DISTANCE = 1000,
  OUTPUT = f_out,
  .quiet = TRUE)
```

- L'option « .quiet = TRUE » est utilisée pour ne pas afficher les messages envoyés par le système vers la console R Studio.

```
buff=st_read(f_out)
loc=st_read(f_in)
loc
plot(buff$geometry)
plot(loc$geometry,add=T)
```

```
> loc
Simple feature collection with 1 feature and 2 fields
Geometry type: MULTIPOINT
Dimension:     XY
Bounding box:  xmin: 679120.8 ymin: 611222.5 xmax: 689863.5 ymax: 620044.5
Projected CRS: ETRS89 / Belgian Lambert 2008
  fid ID                       geometry
1   1  1 MULTIPOINT ((679120.8 61858...
```



- Remarque : le résultat se présente sous la forme de buffers « fusionnés » car la couche de départ (**loc**) est constituée d'un seul multipoint.

2.5 Exemple 2 – Créer un polygone minimum concave

- Ce second exemple montre comment créer un polygone minimum concave autour d'un groupe de points.
- Le nom de l'algorithme peut être retrouvé dans la liste à l'aide du mot clé « concave ».

```
# Exemple 2 - Créer un polygone minimum concave --------
df$algorithm[grepl(pattern = "concave", df$algorithm)]

> df$algorithm[grepl(pattern = "concave", df$algorithm)]
[1] "native:concavehull"
```

- L'algorithme utilisé est **« qgis:knearestconcavehull »**.

```
algo="native:concavehull"
qgis_show_help(algo)
> qgis_show_help(algo)
Concave hull (native:concavehull)

----------------
Description
----------------
This algorithm computes the concave hull of the features from an input layer.

----------------
Arguments
----------------

INPUT: Input layer
        Argument type:  source
        Acceptable values:
                - Path to a vector layer
ALPHA: Threshold (0-1, where 1 is equivalent with Convex Hull)
        Default value:  0.3
        Argument type:  number
```

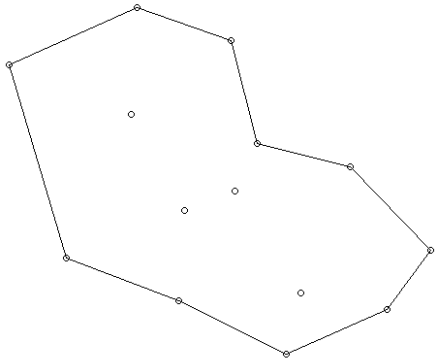
- L'affichage de l'aide permet d'identifier les arguments à utiliser pour que l'algorithme fonctionne correctement.

```
f_in=paste0(path_in,"/localites.shp")
f_out=paste0(path_out,"/concave_hull.gpkg")

result = qgis_run_algorithm(
  algorithm=algo,
  INPUT = f_in,
  OUTPUT = f_out,
  .quiet = T)

# Affichage du résultat
hull=st_read(f_out)
plot(hull$geom)
loc=st_read(f_in)
plot(loc$geometry,add=T)
```

- Lorsque des arguments ne sont pas précisés dans la commande, ils reçoivent les valeurs par défaut prévues par QGIS.

```
> result = qgis_run_algorithm(
+   algorithm=algo,
+   INPUT = f_in,
+   OUTPUT = f_out,
+   .quiet = T)
Argument `ALPHA` is unspecified (using QGIS default value).
Argument `HOLES` is unspecified (using QGIS default value).
Argument `NO_MULTIGEOMETRY` is unspecified (using QGIS default value).
```

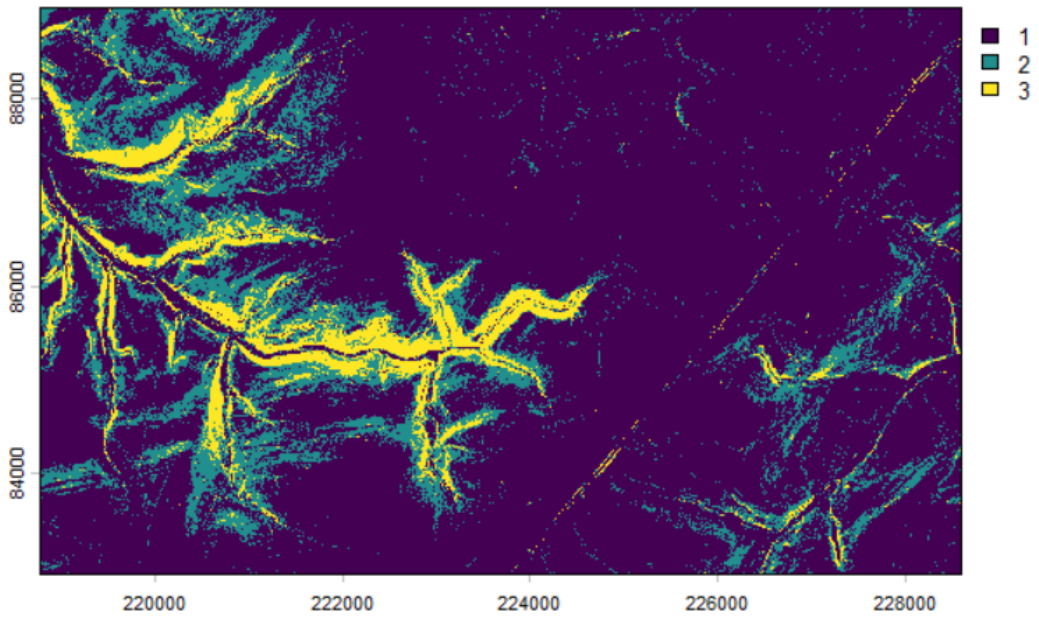
```
# Affichage du résultat
hull=st_read(f_out)
plot(hull$geom)
loc=st_read(f_in)
plot(loc$geometry,add=T)
```



2.6 Exemple 3 – Calculer la pente du terrain

- Dans ce troisième exemple, une couche MNT est convertie en couche de pente, exprimée en pourcent, à l'aide de l'algorithme « **gdal:slope** ».

```
# Exemple 3 - Calculer la pente du terrain (GDAL) ---

algo="gdal:slope"
qgis_show_help(algo)

f_in=paste0(path_in,"/mnt_2m.tif")
f_out=paste0(path_out,"/pente_pct.tif")
result = qgis_run_algorithm(
  algorithm=algo,
  INPUT = f_in,
  BAND = 1,
  AS_PERCENT = TRUE,
  OUTPUT = f_out,
  .quiet = TRUE)

# classes de pentes
slope=rast(f_out)
rcl = matrix(c(-Inf, 15, 1, 15, 30, 2, 30, Inf,3),
             ncol = 3, byrow = TRUE)
rcl
cl_slope = classify(slope, rcl = rcl)
plot(cl_slope)
f_cl_slope=paste0(path_out,"/classe_pente.tif")
writeRaster(cl_slope,f_cl_slope)
```



2.7 Exemple 4 : tamisage d'une couche raster (GDAL sieve)

- La couche de classe de pente qui a été générée au paragraphe précédent peut être « nettoyée » à l'aide d'un tamisage qui va supprimer les petits groupes de pixels (taille < 20).

```
# Exemple 4 - Tamiser un raster (GDAL Sieve) ----

# appliquer 1 tamisage
algo="gdal:sieve"
qgis_show_help(algo)

f_out=paste0(path_out,"/classe_pente_tam20.tif")
result = qgis_run_algorithm(
  algorithm=algo,
  INPUT = f_cl_slope,
  THRESHOLD = 20,
  OUTPUT = f_out,
  .quiet = TRUE)

cl_pente_sieve=rast(f_out)
plot(cl_pente_sieve)
```

3. Utilisation du package whitebox

3.1 Introduction

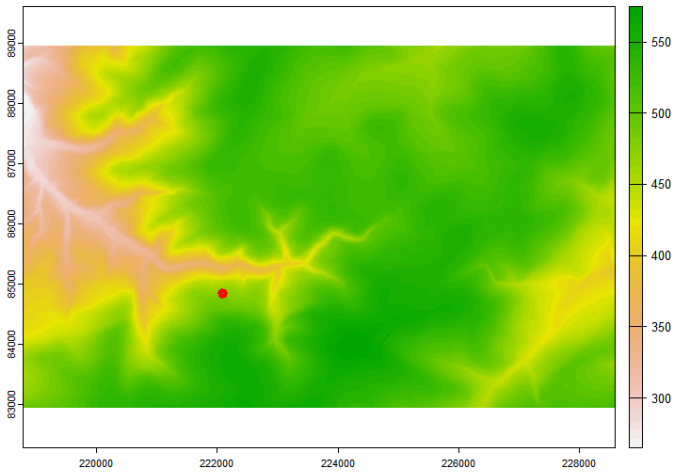
- Le package WhiteBox intègre de nombreuses fonctions d'analyse spatiale. Il est en quelque sorte le pendant de l'extension WhiteboxTools de QGIS.
- Whitebox s'installe également depuis le depuis le dépôt CRAN.
- La version utilisée dans ce tutoriel est la version 2.4.0

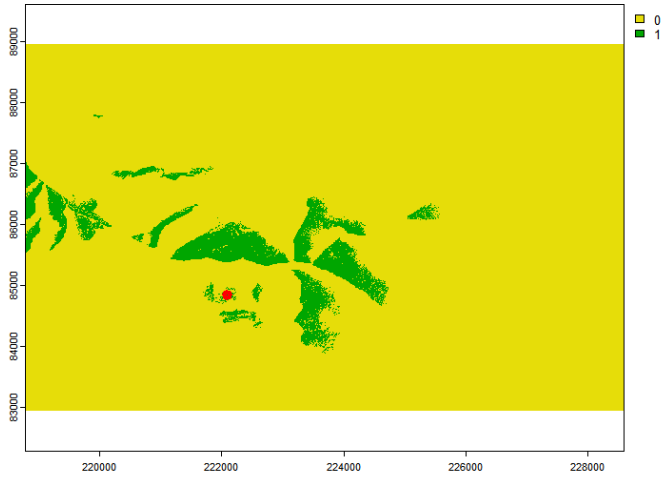
```
# version de whitebox
wbt_version()

> wbt_version()
WhiteboxTools v2.4.0 (c) Dr. John Lindsay 2017-2023

WhiteboxTools is an advanced geospatial data analysis platform dev
eloped at
the University of Guelph's Geomorphometry and Hydrogeomatics Resea
rch
Group (GHRG). See www.whiteboxgeo.com for more details.
```

3.2 Exemple 1 : Viewshed

- L'exemple qui suit utilise l'algorithme **viewshed** pour générer 1 couche raster représentant les zones visibles depuis 1 (des) point(s) de vue. Le relief est décrit par un MNT.
- La fonction wbt_init() est utilisée pour vérifier si l'exécutable Check if a suitable 'WhiteboxTools' executable is present.

```
# Exemple 1 Viewshed ------
point=st_read(f_pdv)
f_mnt=paste0(path_in,"/mnt_2m.tif")
f_viewshed=paste0(path_out,"/viewshed.tif")
f_pdv="D:/tmp/R_GIS_03/data_in/point_de_vue.shp"

wbt_init()
result <- wbt_viewshed(dem = f_mnt,
                       stations = f_pdv,
                       height = 1.6,
                       output = f_viewshed,
                       verbose_mode = FALSE)
mnt=rast(f_mnt)
plot(mnt)
plot(point$geometry,add=T,col="red",lwd=5)

vsh=rast(f_viewshed)
plot(vsh)
plot(point$geometry,add=T,col="red",lwd=5)
```





3.3 Exemple 2 - fonctions hydrologiques

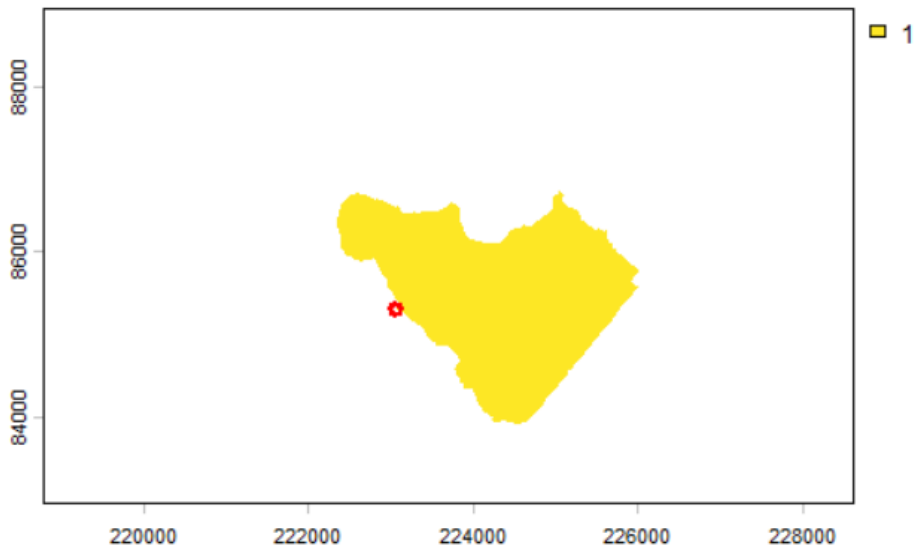
- Whitebox comporte une série de fonctionnalités utilisables pour la création de couches hydrologiques dérivées de MNT.
- Dans l'exemple qui suit, nous utilisons 1 fichier **mnt_2m.tif** (MNT) et **exutoire.shp** (localisation d'un exutoire) pour délimiter le bassin versant situé en amont de l'exutoire.

```
# Exemple 2 - Fonctions hydrologiques ----------

# Input and output data
f_in=paste0(path_in,"/mnt_2m.tif") # MNT
f_exutoire=paste0(path_in,"/exutoire.shp") # exutoire du BV
f_fill=paste0(path_out,"/mnt_fill.tif") # MNT sans dépression
f_fl_dir=paste0(path_out,"/fl_dir.tif") # Direction d'écoulement
f_fl_acc=paste0(path_out,"/fl_acc.tif") # Accumulation d'écoulement
f_watershed=paste0(path_out,"/watershed.tif") # bassin versant
f_str=paste0(path_out,"/stream.tif") # axes d'écoulement
f_str_vect=paste0(path_out,"/stream.shp") # axes d'écoulement vectoriels
```

- La délimitation du bassin versant s'opère en 3 étapes :
  - production d'un MNT sans dépression
  - calcul des directions d'écoulement
  - délimitation du bassin versant situé en amont de l'exutoire.

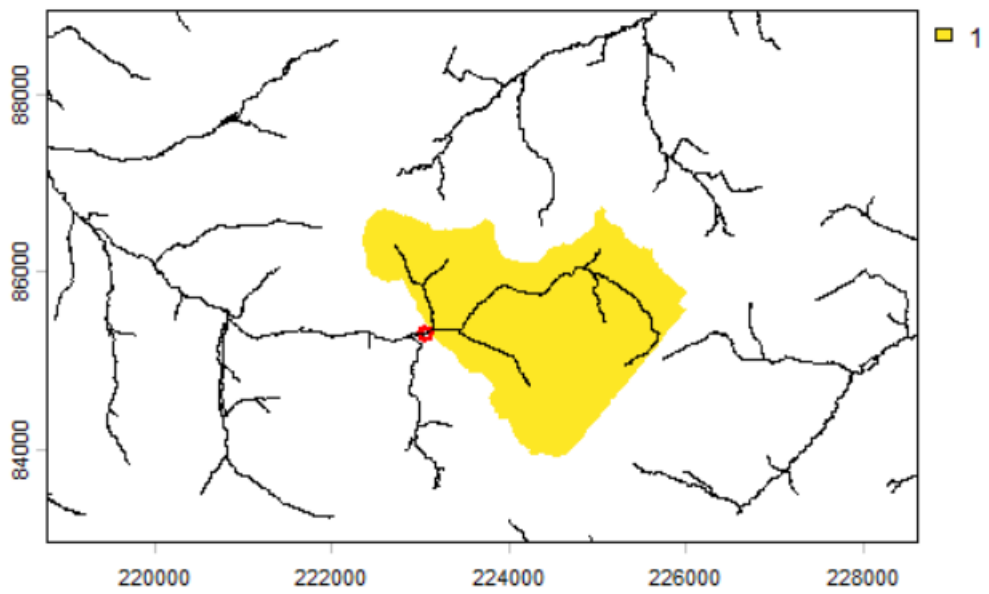
```
# Délimitation d'un bassin versant --------------

# Step 1 : Fill depressions
wbt_fill_depressions_planchon_and_darboux(
  dem = f_in,
  output = f_fill,
  fix_flats = TRUE
)
> wbt_fill_depressions_planchon_and_darboux(
+   dem = f_in,
+   output = f_fill,
+   fix_flats = TRUE
+ )
fill_depressions_planchon_and_darboux - Elapsed Time (including I/O): 9.353s
# Step 2 : D8 pointer - Calcul de la direction d'écoulement
wbt_d8_pointer(
  dem = f_fill,
  output = f_fl_dir
)
> # Step 2 : D8 pointer - Calcul de la direction d'écoulement
> wbt_d8_pointer(
+   dem = f_fill,
+   output = f_fl_dir
+ )
d8_pointer - Elapsed Time (excluding I/O): 0.189s
```

```
# Step 3 : Watershed - Délimitation du bassin versant à partir
# de l'exutoire
wbt_watershed(
  d8_pntr = f_fl_dir,
  pour_pts = f_exutoire,
  output = f_watershed
)

watershed - Elapsed Time (excluding I/O): 1.268s

# affichage du résultat
r <- rast(f_watershed)
plot(r)
exut <- st_read(f_exutoire, quiet =T)
plot(st_geometry(exut), add = TRUE, col = "red",lwd = 3)
```



- Avec les mêmes données de base, il est possible de tracer les axes d'écoulement dont la surface contributive (accumulation d'écoulement) est supérieure à une valeur seuil donnée (ici ssLa)
    - Calcul des accumulations d'écoulement.
    - Seuillage des accumulations d'écoulement pour identifier les axes d'écoulement.
    - Vectorisation des axes d'écoulement.

```r
# Extraction des axes d'écoulement ---------------

# Step 1 : Accumulation d'écoulement
wbt_d8_flow_accumulation(
  input = f_fill,
  output = f_fl_acc
)

# Step 2 : Extraction des axes d'écoulement
wbt_extract_streams(
  flow_accum = f_fl_acc,
  threshold = 50000,
  output = f_str
)

# Step 3 : Vectorisation des axes d'écoulement
wbt_raster_streams_to_vector(
  streams = f_str,
  d8_pntr = f_fl_dir,
  output = f_str_vect
)

str <- st_read(f_str_vect, quiet = TRUE)
plot(st_geometry(str), add = TRUE )
```



- **Remarque importante** : les algorithmes wbt ne supportent pas le format gpkg. Les couches vectorielles doivent donc être sauvegardées en format shapefile.