Table des matières

1 Préambule **2**

2 Pour commencer... **2**
2.1 Comment lancer un script . . . 2
2.2 Script minimal . . . 2

3 Importation et exportation d'images **3**
3.1 Ouvrir un fichier .jpg . . . 3
3.2 Ouvrir un fichier de manière générale . . . 3
3.3 Exporter une image en .xcf . . . 3
3.4 Exporter une image en bitmap .jpg ou .png . . . 3

4 Traitements de l'image **3**
4.1 Convertir en niveaux de gris . . . 4

5 Ajouter un calque **4**
5.1 Copier un calque . . . 4
5.2 Courbe de niveaux . . . 4
5.3 Définir le calque actif . . . 4
5.4 Sélection par couleur . . . 4
5.5 Diminuer la sélection . . . 5
5.6 Augmenter la sélection . . . 5
5.7 Définir la couleur de premier plan (utile pour colorier) . . . 5
5.8 Remplissage avec une couleur . . . 5
5.9 Utiliser le plugin G'MIC/Greycstoration . . . 5

6 Astuces et bouts de code **5**
6.1 Sélectionner un calque à partir de son nom . . . 5
6.2 Trouver les couleurs extrémales d'une région . . . 6

1 Préambule

Ceci est un tutoriel pour montrer comment utiliser Gimp en ligne de commande via des scripts python, c'est à dire python-fu. Je ne parlerai pas des scripts python-fu qu'on peut faire pour pouvoir les utiliser *dans* Gimp, mais des scripts à utiliser directement en ligne de commande, sans jamais lancer l'interface Gimp, pour faire tout le traitement uniquement avec un script.

Ce tutoriel a la prétention de présenter les bases pour pouvoir réaliser des scripts et donner les moyens de complexifier par la suite car, ligne de commande oblige, certaines choses sont moins intuitives que via l'interface classique.

2 Pour commencer...

2.1 Comment lancer un script

La ligne de commande pour lancer un script en ligne de commande est :

```
gimp -idf --batch-interpreter python-fu-eval -b
"import sys; sys.path=['.']+sys.path;import script;script.run('.')"
 -b "pdb.gimp_quit(1)"
```

où la commande est lancée dans le dossier contenant le script que l'on souhaite lancer, et qui s'appelle **script.py**. Je ne connais pas les détails mais la première partie lance gimp en ligne de commande sans l'interface graphique, puis on choisit l'interpréteur pour lancer le script. Et on fait évaluer une expression python qui importe dans un premier temps **script.py** via `import script` après avoir ajouté le répertoire courant dans le path de recherche des modules (`import sys; sys.path=['.']+sys.path`).

Ensuite, la commande qui est lancée est `script.run('.')` et ça, c'est propre au script, ça veut dire que **script.py** contient une fonction nommée **run()** à qui on passe un argument `'.'`.

2.2 Script minimal

Le script minimal ressemble à quelque chose de ce genre :

```
#!/usr/bin/env python

import os,glob,sys,time
from gimpfu import *

def run(directory):
  start=time.time()
  print "Running on directory \"%s\"" % directory
  if not(os.path.exists('processed')):
          os.mkdir('processed')
  for infile in glob.glob(os.path.join(directory, '*.jpg')):
          process(infile)
  end=time.time()
  print "Finished, total processing time: %.2f seconds" % (end-start)
```

process() est ici une fonction qui contient toutes les opérations qui seront réalisées sur les images (des **.jpg** ici). Je n'en parle pas pour l'instant, je détaillerai dans la suite du tutoriel toutes les commandes de bases (libre à vous de faire des choses plus complexes après).

> Vous devez avoir installé python pour gimp. Si vous ne savez pas si c'est fait, ouvrez gimp, dans le menu **Filtre**, un sous-menu **Python-Fu** doit apparaître à coté de **script-fu**

En important **gimpfu**, vous accédez aussi à toute la base de données des fonctions gimp via le module **pdb** (qui n'est pas le module pdb de python, rien à voir avec un débugueur)

3 Importation et exportation d'images

Le principe est que chaque image sera associée à un objet où certaines méthodes sont définies. Cet objet sera alors passé en paramètre de fonctions dans le script, pour qu'on sache à quelle image s'appliquent les opérations.

3.1 Ouvrir un fichier .jpg

```
image = pdb.file_jpeg_load(infile, infile)
```

image est l'instance d'objet qui sera créée tandis que **infile** est simplement le nom du fichier (par exemple **image.jpg**).

3.2 Ouvrir un fichier de manière générale

J'utilise cette commande pour des fichiers .xcf par exemple :

```
image = pdb.gimp_file_load(infile, infile)
```

image est l'instance d'objet qui sera créée tandis que **infile** est simplement le nom du fichier (par exemple **image.xcf**).

3.3 Exporter une image en .xcf

C'est pratique si on souhaite conserver les calques et pouvoir retoucher *a posteriori* l'image qu'on a traité avec le script.

```
pdb.gimp_xcf_save(0, image, drawable, outfile_xcf, outfile_xcf)
```

où **outfile_xcf** est le nom du fichier de sortie, **image** est l'instance d'objet contenant l'image que l'on souhaite enregistrer. **drawable** est le calque actif je crois, mais il suffit, il me semble, que ce soit un objet de type calque pour que ça fonctionne.

> **Remarque** : Si jamais vous avez un soucis, il est possible de récupérer le calque actif avec la commande :
>
> ```
> drawable = pdb.gimp_image_get_active_layer(image)
> ```

3.4 Exporter une image en bitmap .jpg ou .png

Ici les choses sont un peu plus compliquées (en tout cas pour moi qui n'ai peut-être pas tout compris). Je n'ai pas réussi à enregistrer simplement l'image, il ne m'enregistre que le calque actif. Pour contourner le problème, je fusionne tous les calques avant d'enregistrer.

Pour enregistrer en .jpg, on a alors les lignes suivantes :

```
drawable = pdb.gimp_image_merge_visible_layers(image, EXPAND_AS_NECESSARY)
pdb.file_jpeg_save(image, drawable, outfile,
                   outfile, "0.5",0,1,0,"",0,1,0,0)
```

Pour enregistrer en .png, on a :

```
drawable = pdb.gimp_image_merge_visible_layers(image, EXPAND_AS_NECESSARY)
pdb.file_png_save(image, drawable, outfile_png,
                  outfile_png, 0, 9, 0, 0, 0, 0, 0)
```

> **Remarque** : Je n'explique pas les options de ces commandes, parce qu'il n'y en a pas besoin pour une utilisation classique. Il sera toujours temps de regarder le manuel de la commande pour plus de précisions. Je donne simplement des boites noires pour arriver à faire le minimum.

4 Traitements de l'image

La liste des calques d'un objet image s'obtient en faisant :

```
image.layers
```

4.1 Convertir en niveaux de gris

```
pdb.gimp_convert_grayscale(image)
```

5 Ajouter un calque

```
new_layer = pdb.gimp_layer_new(image,image.width,
                               image.height,3,"blanc",100,0)
pdb.gimp_image_add_layer(image,new_layer,-1)
```

La première ligne déclare un calque avec les mêmes dimensions que l'image. Le numéro, 3, donne le type de calques (plus de détails dans le manuel). De mémoire, c'est un calque en niveau de gris avec une couche alpha (c'est à dire qu'on autorise la transparence. Le nom associé à 3 doit être quelque chose du style **AGRAYSCALE**, le "A" symbolisant la présence de la transparence (ce que j'ai mis un peu de temps à comprendre). **blanc** est le nom du calque et 100.0 est simplement l'opacité du calque.

Ensuite, une fois le calque définit, il faut l'ajouter à l'image, c'est le rôle de la ligne suivante. -1 est la position du calque (en gros, s'il est devant ou derrière les autres calques. En mettant -1, on place le calque devant tous les autres calques (en mettant des valeurs positives, on recule peu à peu le calque dans la liste.

Remarque : 0 permet de mettre le calque en arrière plan je crois, mais à vérifier.

5.1 Copier un calque

```
layer_cleaned = pdb.gimp_layer_copy (layer_original, False)
layer_cleaned.name = "cleaned"
pdb.gimp_image_add_layer(image, layer_cleaned, -1)
```

Ici on copie un calque, on change son nom, et on l'ajoute à l'image.

5.2 Courbe de niveaux

La commande suivante permet d'ajuster la plage de couleurs pour que la plus claire soit le blanc et la plus foncée soit le noir. Ceci ne marche qu'avec des images en niveau de gris je crois.

```
pdb.gimp_levels_stretch(layer_cleaned)
```

Notez que cette commande s'applique à un calque.

5.3 Définir le calque actif

Avec les logiciels de retouche d'image, les modifications sont la plupart du temps appliquées au calque courant. En particulier, si on veut faire une sélection, celle-ci sera fait à partir du calque courant. Il faut donc pouvoir définir le calque actif avant de faire une opération.

```
pdb.gimp_image_set_active_layer(image, layer_cleaned)
```

Remarque : Ceci n'est toutefois pas vrai tout le temps. Certaines commandes permettent de spécifier un calque sur lequel appliquer la modification, qu'il soit actif ou pas.

5.4 Sélection par couleur

```
pdb.gimp_by_color_select (layer_cleaned, (250, 250, 250),
                          15,2,False,False,0,False)
```

On cherche à sélectionner toutes les couleurs proches de la couleur RGB (250.250.250) avec une tolérance de 15 dans le calque **layer_cleaned**.

Remarque : Pour tout désélectionner, il suffit de faire :

```
pdb.gimp_selection_none(image)
```

5.5 Diminuer la sélection

Pour réduire la sélection d'un certain nombre de pixel, il faut faire :

```
pdb.gimp_selection_shrink (image,2)
```

Cette commande diminue la sélection de 2 pixels (valeur que l'on peut bien entendu modifier).

5.6 Augmenter la sélection

Pour augmenter la sélection d'un certain nombre de pixel, il faut faire :

```
pdb.gimp_selection_grow (image,2)
```

Cette commande augmente la sélection de 2 pixels (valeur que l'on peut bien entendu modifier).

5.7 Définir la couleur de premier plan (utile pour colorier)

Avant d'utiliser le pinceau, le crayon ou n'importe quel outil utilisant les couleurs, il faut définir la couleur, de premier plan et/ou arrière plan.

Toujours dans un soucis de simplification, je ne montre que la couleur de premier plan. Ça suffit pour mes exemples (certaines commandes existent en double, l'une utilisant la couleur de premier plan et l'autre la couleur d'arrière plan. D'autres commandes ont une option qui permet de choisir si on veut utiliser la couleur de premier ou arrière plan.

```
pdb.gimp_palette_set_foreground ((255, 255, 255))
```

5.8 Remplissage avec une couleur

Une fois une sélection effectuée, on peut remplir cette dernière avec la couleur de premier plan

```
pdb.gimp_bucket_fill(layer_blanc, 0, 0, 100, 10, 0, 0, 0)
```

Avec une opacité de 100, on colorie toute la sélection (voire tout le calque si aucune sélection n'est effectuée) sur le calque **layer_blanc**.

5.9 Utiliser le plugin G'MIC/Greycstoration

Cette commande m'a vraiment donné beaucoup de mal. **Greycstoration** n'étant plus maintenu, il est conseillé d'utilisé **G'MIC** à la place, sauf que ce dernier est beaucoup plus compliqué et complet et je souhaitais uniquement retrouver le comportement de greycstoration.

Pour celà, il suffit de faire

```
pdb.plug_in_gmic(image, layer_cleaned, 1,
                 "␣-smooth[-1]␣20␣-denoise␣5,5,8",
                 run_mode=RUN_NONINTERACTIVE)
```

La commande s'appliquant à l'image **image** et au calque associé **leayer_cleaned**.

> **Remarque** : Il est bien entendu possible de rajouter d'autres commandes propres à **G'MIC** qui en a des dizaines. Ces deux là sont justes celles qui permettent de retrouver le comportement de **Greycstoration**

6 Astuces et bouts de code

6.1 Sélectionner un calque à partir de son nom

```
layer_ids = image.layers

for layer_id in layer_ids:
  if (layer_id.name == "blanc"):
    pdb.gimp_image_set_active_layer(image, layer_id)
```

Ici, on cherche un calque dont le nom est **blanc**, et on passe ce calque en tant que calque actif de l'image associée.

6.2 Trouver les couleurs extrémales d'une région

N'oubliez pas d'importer le module **array** via `from array import array`

```
# If the darkest color is black,
# then darkest will be equal to 0. Setting
# whitest to 255 ensure that we will get the right value.
darkest = 255

# If the lightest color is white,
# then lightest will be equal to 255. Setting w
# hitest to 0 ensure that we will get the right value.
lightest = 0

# initialize the regions and get their contents into arrays:
srcRgn = drawable.get_pixel_rgn(0, 0, width, height, False, False)
src_pixels = array("B", srcRgn[0:width, 0:height])

darkest = min(src_pixels)
lightest = max(src_pixels)

# Here we define colors. Grey colors are just colors with R=G=B
light = (lightest, lightest, lightest)
dark = (darkest, darkest, darkest)
```

Ce bout de code, que j'ai eu du mal à faire, permet de récupérer la valeur maximale et minimale des pixels (pour le cas d'une image en niveaux de gris). Je voulais ça pour utiliser les niveaux de gris, mais au final, la commande **gimp_levels_stretch** permet de faire ça sans avoir besoin de connaître ces valeurs.