Feb 11, 2019

1. Objet du TP

- Lire les argument d'un programme
- Lire et écrire un fichier texte.
- Utiliser `IO.popen` pour travailler en collaboration avec des outils externes
- Revoir un peu les expressions rationnelles
- Facultatif: Utiliser les fonctionnalité de pack/unpack pour lire des données binaires, en vue de reverse engineering.

2. Échauffement: Faire un logiciel de fortune

fortune est un logiciel destiné à afficher des fortunes, petits messages souvent humoristiques ou sibyllins. Le nom vient des *fortune cookies*, les gateaux chinois avec un petit message à l'intérieur.

- Écrivez un script ruby *rfortune* permettant de lire un fichier de fortune et d'en afficher une tirée au hasard. Pour ce faire, regardez le format des fichiers. Il est basé sur du texte.
- Ajoutez la gestion des arguments: *rfortune* accepte désormais en paramêtre le nom des fichiers de fortune qu'il peut utiliser.

3. Faire des logs du temps de ping moyen d'un serveur

Comment évolue votre connexion réseau dans le temps? On peut principalement étudier deux métriques: le débit, et la latence (le temps que met un paquet à atteindre un serveur).

La commande shell `ping` vous permet d'envoyer des paquets de contrôle `ICMP` et d'avoir une estimation de la latence. Celle ci, lancée sans arguments, ne s'arrête jamais. Testez un `ping londres`

Néanmoins, sur un réseau non privé, la latence varie beaucoup et une simple valeur ne représente pas grand chose. Une moyenne donne plus d'informations mais finit par agréger tellement de valeurs qu'elle ne veut plus dire grand chose sur l'état de votre ligne à un moment donnée.

Nous allons donc utiliser une *moyenne glissante* (une moyenne des *n* dernières valeurs).

3.1. Étape 1

Écrivez un programme ruby qui reçoit en argument un nom de serveur et effectue des `ping` sur celui ci, en affichant chacune des lignes données par le `ping`.

3.2. Étape 2

À l'aide d'une expression rationnelle, extrayez de cette ligne uniquement le temps

3.3. Étape 3

Écrivez une fonction qui calcule la moyenne d'un tableau (une ligne de ruby).

3.4. Étape 4

Stockez le temps obtenu à l'étape 2 dans un tableau, et limitez la taille du tableau aux *n* dernières valeurs. Appelez la fonction de moyenne dessus et affichez le résultat

3.5. Étape 5

Logguez l'intégralité dans un fichier, en ayant sur chaque ligne l'heure de la mesure et la moyenne, séparées par une espace?

3.6. Étape 6

Vous pouvez générer des courbes de débit en utilisant les données de l'étape 5 et un programme externe (`gnuplot` par exemple). Néanmoins, de nombreuses gemmes (paquets Ruby) sont destinées à générer des représentations graphiques de données, que ce soit sous format d'image (png, pdf…) ou de programme (javascript). Nous vous proposons d'utiliser la gemme `gruff`, une gemme qui utilise les bibliothèques d'imagemagick pour générer des images.

Vous pouvez l'installer par `gem install --user-install gruff`.

Rappel: vous devez avoir un proxy bien configuré. Si non:

```
cat >> ~/.bashrc <<EOF
export HTTPS_PROXY=193.49.118.36:8080
export HTTP_PROXY=193.49.118.36:8080
EOF
```

Voici un petit programme de démo pour générer un graphique simple:

```
require 'gruff'
g = Gruff::Line.new
g.title = "Graphique des trucs inutiles"
mots = %w{ compte donc les lettres des mots de cette phrase ou uniquement les voyelles}
g.labels = mots.map.each_with_index { | w , i | [ i, w]  }.to_h
g.marker_font_size = 12
g.data "Nombre de lettres par mot", mots.map(&:size)
g.data "Nombre de voyelles", mots.map{ |w| w.gsub(/[^aeiuoy]/,'')}.map(&:size)
g.write '/tmp/o.png'
```

Générez un graphique des ping moyens.

4. Déchiffrons des fichiers binaires en ruby

Cet exercice est facultatif. Il ne doit bien sûr pas vous servir à faire des progrès en reverse engineering.

4.1. La base

Ce TP a pour but de vous faire travailler un problème très courant en informatique: la récupération de données quand on a pas le code source du loficiel qui les a généré. C'est utile aussi bien dans la vie courante (quand a on planté son téléphone et qu'on veut récupérer son carnet d'adresse, quand on veut exploiter les fichiers de données d'un jeu) ou dans la vie professionnelle (en cas de *pipelining*: interfaçage entre des logiciels qui ne parlent pas la même langue).

Pour ce faire, je vous propose la description du format d'un fichier binaire (je vous fais grâce des questions d'endianness (little/big endian des processeurs)):

Le fichier de données à traiter (exemple pack) est d'un format connu:

- Il commence par une entête de 127 octets qui ne nous intéresse pas.
- Il est ensuite composé d'une série d'enregistrements de taille fixe (57 octets), qui comportent les champs suivants:
  1. Nom du personnage/Nom de la Tribu, codé en UTF-8 sur 50 octets. Indice: les versions récentes de Ruby travaillent nativement en UTF-8.
  2. Classe du personnage, sur un octet en binaire, codée de la façon suivante:
     - 0 -> Troll
     - 1 -> Humain
     - 2 -> Elfe
     - 3 -> Vulcain
     - 4 -> Wookie
     - 5 -> Gobelin
     - … -> Pas encore découvert. Ne pas bugguer pour autant!
  3. Points de vie, sur 2 octet (peut être négatif, sisi!).
  4. Pièces d'or, sur 4 octets (on ne peut avoir de dettes).
- On va éviter de lire tout le fichier en mémoire d'un coup (ça bouffe de la mémoire sans accélérer le traitement). À l'inverse, tout pervers qui voudrait lire le fichier octet par octet sera condamné à vider le Gour de Tazenat à la petite cuillère.
- Attention, certaines fonctions de lecture ont changé de nom entre Ruby1.9 et Ruby2.x. Adaptez vous à votre interpréteur (et préférez Ruby2).
- Pour faire ce TP simplement, vous aurez besoin de la méthode `String#unpack`. Vous pourrez compléter votre lecture du manuel de référence (`ri String#unpack`) par cette traduction d'un article en Français.

4.2. Générer un fichier CSV (Coma Separated Values)

- Ajouter au programme précédent l'exportation vers un fichier CSV (simple fichier texte, un enregistrement par ligne, champs séparés par des virgules).

4.3. Lectures complémentaires, voir annexes

- Une page sur l'exploitation des données binaires en ruby sur Practicing Ruby. Au passage, vous trouverez plein d'exemples de code sympa sur ce site, sous forme de cours/exercice/tutos.
- Il arrive souvent de se retrouver avec une application propriétaire qui produit des fichiers avec des données qui nous intéressent mais dont le format n'est pas traduisible vers quoi que ce soit d'ouvert. Pour ceux qui veulent en apprendre plus sur le reverse-engineering de fichier binaires, je vous conseille la lecture du Definitive Guide to Exploring File Formats.

Supports de cours pour l'IUT
francois.delobel@uca.fr

TPs mis à disposition le plus simplement possible pour mes étudiants.