R1.04 - Systèmes - TP 7
Synthèse sur les filtres

Introduction

Les exercices suivants vont vous permettre de mettre à l'épreuve votre réflexion dans la mise en œuvre des Regex vues dans le TP 6, mais aussi des différents filtres vus dans les deux parties du TP 5, afin de produire le résultat attendu. C'est un TP de synthèse.

Sources

Le fichier de données qui va nous servir de base de travail (20.000 lignes, ce qui sera déjà correct pour faire des tests), est disponible sur Moodle.

Si vous voulez expérimenter chez vous sur un fichier beaucoup plus gros, vous en trouverez un est à cette URL **(ne cliquez pas sur ce lien à l'IUT !!)**, il fait 119 Mo et plus de 640.000 lignes.

`http://www.almhuette-raith.at/apache-log/access.log`

Contexte

Le fichier de travail est un fichier de *logs* d'un serveur **Apache**.

Apache est un serveur Web qui a connu son heure de gloire dans les années 2000-2015 avec plus de 70% de parts de marché.

Depuis ces dernières années, il est en net déclin au profit de **nginx** (prononcer engine-X), les deux se partageant environ 75% du marché.

Logs

Il est temps de parler du terme de *logs*, que vous avez sans doute déjà **(**vu **|** lu**)**[1], et qui est une chose essentielle en informatique, en particulier dans le domaine du développement logiciel.

Ce terme vient de *Logbook*, le journal de bord d'un bateau ou d'un avion. L'informatique s'est approprié le concept : il s'agit d'un outil ou d'une technique servant à consigner des événements.

[1] Vous avez repéré la Regex ? 😉

Quand on développe un logiciel, on prévoit toujours de produire des logs pour décrire ce qui se passe dans le logiciel. C'est utile notamment pour le débogage, pour comprendre, *a posteriori*, les étapes que le logiciel a suivies, les événements qui se sont produits etc. Les logs servent aussi pour la sécurité : qui a fait quoi, quand…

Apache ne fait pas exception à la règle, et pour cause : c'est un logiciel qui est une porte béante vers, ou plutôt depuis, la jungle de l'Internet. *Logguer* les événements, tous les événements, est crucial pour différentes raisons, comme par exemple :

- Savoir qui (adresse IP) demande quoi (URL), quand (date+heure très précises)
- Savoir quelles URL n'aboutissent à rien

Les finalités sont multiples :

- Produire des statistiques de fréquentation
- Produire des statistiques de consommation de bande passante[2]
- Produire des statistiques sur la nature des visiteurs (navigateurs, OS, versions)
- Produire des statistiques de la provenance géographique des visiteurs
- Détecter les liens cassés
- Détecter les tentatives d'accès abusif à certaines ressources

Fichier de travail

Ayant été très tôt le serveur le plus populaire (plus de 40% de parts de marché dès 1997), Apache est devenu une référence, notamment dans la façon de formater les logs Web.

Apache propose plusieurs formats qui portent des noms comme **`common`** ou **`combined`**.

Pour plus de détail, consultez **https://httpd.apache.org/docs/2.4/fr/logs.html**

Le format du fichier de travail est dérivé du **`combined`** :

- Adresse IP du client (client = généralement un navigateur Web)
- Un **-** (tiret), information non utilisée
- Nom d'utilisateur du client (quand il s'est identifié au préalable), sinon **-** (tiret)
- Date + Heure précises de la requête
- Méthode + Requête + Protocole.
- Code de retour de la requête
- Taille (en octets) de la réponse : c'est la taille de la ressource demandée (page HTML, image etc)
- URL d'où provient la requête (en cas de lien depuis un autre site par exemple)
- User-Agent : un chaîne de caractères spécifique à chaque navigateur
- Un **-** (tiret), information non utilisée

[2] Terme technique pour parler du volume de données entrantes et sortantes, mis en rapport avec la capacité maximale de la connexion au réseau Internet (ex : 1, 4, 20 mégabits en xDSL voire 1 ou plusieurs gigagbits sur de la fibre)

Codes de retour

Les principaux codes de retour sont :

- 200 : Ressource trouvée. La page HTML, l'image, le fichier CSS, etc. est renvoyé au client. C'est le cas le plus commun, c'est le code indiquant que tout est OK.
- 304 : Ressource non modifiée depuis la dernière fois (le client utilisera son cache)
- 403 : Accès interdit à la ressource demandée
- 404 : Ressource inexistante. Un code bien connu !
- 500 : Erreur interne au serveur

Vous en trouverez peut-être d'autres dans le fichier de travail. Pour plus d'informations sur les codes de retour possibles (le 418 vaut le détour et a sa petite histoire) :

https://fr.wikipedia.org/wiki/Liste_des_codes_HTTP

User-Agent

La chaîne User-Agent peut parfois être compliquée à analyser.

Voici une liste non exhaustive des valeurs communes rencontrées :

https://fr.wikipedia.org/wiki/User_agent

Méthode + Requête + Protocole

C'est une chaîne constituée de 3 parties séparées par un espace.

Partie 1 - La méthode : c'est la nature de la requête : **`GET`** et **`POST`** sont les plus communes. Il s'agit de la façon dont le client a effectué sa requête. Par exemple, **`POST`** correspond à l'envoi du contenu d'un formulaire depuis une page Web.

Partie 2 - La requête : c'est simplement l'URL de la ressource demandée par le client. C'est plus précisément ce qui se trouve après le **`http[s]://nom_de_domaine.xyz`**, un morceau qui ne figure donc pas dans cette chaîne.

Partie 3 - Le protocole : c'est une indication du protocole[3] utilisé par le client. Sans entrer dans les détails, vous y trouverez généralement quelque chose du genre **`HTTP/1.0`** ou **`HTTP/1.1`**. Le **`HTTP`** est le protocole, le **`1.x`** est la version de ce protocole. C'est utile au serveur pour savoir comment il peut répondre au client[4].

Dans certaines questions (vous le saurez au moment venu), vous aurez sans doute besoin de plus d'informations, que vous trouverez ici :

https://fr.wikipedia.org/wiki/Hypertext_Transfer_Protocol

[3] **HTTP** : **H**ypertext **T**ransfer **P**rotocol. Un protocole définit une façon de communiquer. En informatique, les appareils communiquent entre eux en suivant des règles, comme les humains : Client : Yo bro, t'es là ? Serveur : Ouaich, kes tu ve ? Client : ben, la page **`index.html`**

[4] Du genre : **`1.0`** = *Yo man* et **`1.1`** = *Bonjour, enchanté de faire votre connaissance*

Exercices

Répondez aux questions suivantes en utilisant les filtres que vous voulez.

Il y a souvent plusieurs routes pour mener à destination. Peu importe le chemin, l'essentiel est le résultat. Mais certains chemins sont plus élégants que d'autres !

Les lignes de logs concernent le mois de **`décembre 2020`**. Sauf indication contraire, on pourra donc s'appuyer sur cette information dans l'écriture des commandes.

Le fichier de travail fait 20.000 lignes. Vous risquez d'avoir de très très longs affichages. Une astuce, pendant la mise au point de vos commandes (certainement des enfilades de filtres), est de filtrer vos résultats sur un **`less`** (**`| less`**) pour avoir un affichage paginé et facilement interruptible, ou de ne conserver que les 10 ou 20 premières lignes de résultat (**`| head -20`**). Une fois votre mise au point terminée, n'oubliez pas de retirer ce filtrage afin de produire le résultat attendu, tel que compter les lignes par exemple ! Ne comptez pas directement sans être sûr que vous comptez les bonnes lignes.

> N'oubliez pas l'utilité
>
> d'un **`egrep --color`** ou d'un **`alias egrep='egrep --color'`**

NB : quand vous utilisez un autre filtre derrière un **`egrep --color`**, vous perdez la mise en évidence par coloration de **`egrep`**.

Attention, pour vous aider à vérifier vos commandes, vous avez les réponses attendues mises entre **`()`**, mais il est évident que vous ne devez pas en faire usage pour trouver le résultat… 😳

Q. 1 Comptez les (**`3607`**) lignes du **`24 décembre 2020`**.

Q. 2 Comptez les (**`67`**) lignes du **`24 décembre 2020`** de l'intervalle horaire **`[18:00 , 19:00[`**.

Q. 3 Comptez les (**`875`**) lignes de l'intervalle horaire **`[18:00 , 19:00[`**, quelle que soit la date.

Q. 4 Comptez les (**`436`**) lignes de l'intervalle horaire **`[18:00 , 18:30[`**, quelle que soit la date.

Q. 5 Comptez les (**`384`**) lignes de l'intervalle horaire **`[09:00:00 , 09:30:00]`**, quelle que soit la date.

Q. 6 Affichez la liste des (**`7`**) dates uniques (Jour, Mois, Année) présents dans le fichier. N'affichez aucune autre information que les dates.

Q. 7 Affichez la liste des (**`7`**) dates uniques (Jour, Mois, Année) avec le nombre de requêtes pour chaque jour, triés par nombre de requêtes décroissant.

Q. 8 Affichez l'adresse IP ayant fait le plus de requêtes. N'affichez que l'IP (**`45.144.0.179`**) et le nombre de requêtes faites (**946**).

Q. 9 Affichez les (**2**) lignes dont les adresses IP ont le même nombre en 1ère et 3ème positions (**`999.999.999.999`**).

Q. 10 Comptez les (**18581**) requêtes ayant abouti à un code **`200`**.

Q. 11 Affichez la liste des (**4**) codes des requêtes ayant abouti à un code autre que **`200`**, et comptez le nombre de requêtes pour chaque code (somme totale = **1419**).

Q. 12 Affichez les (**3**) lignes des requêtes qui ont été faites à un moment où Heures, Minutes et Secondes était un nombre identique (ex : **`12:12:12`**).

Q. 13 Affichez les (**7**) lignes des requêtes qui ne sont ni **`GET`** ni **`POST`**. Quelle est la taille des réponses ? Allez voir le lien, donné précédemment, concernant le protocole **`HTTP`**. Comprenez-vous pourquoi les tailles sont ainsi dans les réponses ? Demandez de l'aide à votre enseignant.e si besoin.

Q. 14 Affichez les (**18**) lignes des requêtes faites par des personnes clairement identifiées.

Q. 15 Pour chacune des ces (**18**) personnes clairement identifiées, affichez ces informations (et uniquement celles-là), une par ligne, séparées par des espaces et triées par taille de ressource :

- Nom de la personne
- Méthode de la requête
- Ressource demandée (URL)
- Taille de la ressource renvoyée

Q. 99 Calculez la taille totale (**`487775629 octets`**) des réponses aux requêtes ayant abouti à un code **`200`**. Ce problème n'est pas évident et vous aurez besoin d'utiliser la commande **`bc`** (**b**asic **c**alculator). C'est aussi un filtre. Voici un exemple : **`echo '1+2+3' | bc`**. Vous aurez aussi, sans doute, besoin d'un petit coup de pouce de votre enseignant.e !

Bonus : modifiez votre commande pour exprimer le résultat en **`Mo`**. (**465**)