

TP - Expressions régulières

1 Outils du Shell GNU

- 1 - Une Debian classique fournit deux outils de recherche de texte, grep et egrep, ce dernier utilisant par défaut les Regex "ERE". A l'aide de la commande which, donner la commande qui va donner la localisation de ces deux outils, et le résultat :

Commande :

localisation : grep :

egrep :

- 2 - En utilisant une substitution de commande utilisant la commande file, donner la commande qui va donner la nature de ces deux fichiers, ainsi que le résultat :

Commande :

type : grep :

egrep :

- 3 - En utilisant une substitution de commande utilisant , donner la commande qui va donner la taille de ces deux fichiers, ainsi que leur taille :

Commande :

taille : grep :

egrep :

- 4 - Pour ce dernier, l'afficher avec cat. Que pouvez-vous dire ?

2 Mise en pratique

Pour les exercices suivants, vous pourrez faire des essais préliminaires avec les outils en ligne donnés dans le cours, mais on demande une validation avec grep avec l'option -P (utilisation du dialecte "PCRE").

2.1 Noms de fichiers

Proposer une Regex qui va permettre de détecter des noms de fichiers de type image, à la condition qu'ils respectent les critères habituels : extension séparée du nom par un point, pas de fichier caché (commençant par un point)

Vous validerez votre expression avec ce fichier test :

https://github.com/skramm/but3_rt/blob/main/ressources_TP/regex/noms_fichiers.txt

```
img0912.jpg
updated_img0912.png
favicon.gif
documentation.html
.bash_profile
img0912.jpg.tmp
access.lock
workspace.doc
.a.b.jpg
azertyjpg
```

2.2 Nombres

Proposer une regex qui va matcher sur les chaînes suivantes :

en évitant de matcher sur les suivantes :

```
720p 384$ 248.22€
```

Vous utiliserez ce fichier pour la validation :

```
https://github.com/skramm/but3\_rt/blob/main/ressources\_TP/regex/nombres.txt
```

2.3 Numéros de téléphone (Américains)

Dans un champ de texte, les utilisateurs doivent saisir des numéros de téléphone américains. Par exemple :

```
415-555-1234 | 650-555-2345 | (416)555-3456 | 202 555 4567 | 4035555678 | 1 416 555 9292
```

Vous utiliserez ce fichier pour la validation :

```
https://github.com/skramm/but3\_rt/blob/main/ressources\_TP/regex/no\_tel\_USA.txt
```

2.4 Adresses IPv4

Matcher une adresse IP n'est pas si simple qu'il n'y paraît : Il s'agit de 3 groupes de 3 chiffres, mais qui doivent respecter la contrainte de ne pas dépasser 255.

Commencer par écrire une Regex qui matche sur une valeur entre 1 et 255 avec un point derrière. Vous utiliserez ce fichier pour la validation :

```
https://github.com/skramm/but3\_rt/blob/main/ressources\_TP/regex/adresses\_ipv4\_1.txt
```

```
grep -E "^(1[0-9]|2[0-4][0-9]|25[0-5])\.(1[0-9]|2[0-4][0-9]|25[0-5])\.(1[0-9]|2[0-4][0-9]|25[0-5])\.(1[0-9]|2[0-4][0-9]|25[0-5])\$" addresses_ipv4_1.txt
```

Puis l'étendre pour avoir la regex complète :

```
grep -E "^(\$ip\.)\{3\}\$ip\$" addresses_ipv4_2.txt
```

Vous utiliserez ce fichier pour la validation :

```
https://github.com/skramm/but3\_rt/blob/main/ressources\_TP/regex/adresses\_ipv4\_2.txt
```

3 Analyse d'un fichier système

Le fichier `/etc/services` liste les services réseau et le port qui leur est assigné. Examiner ce fichier (**sans le modifier!!!!**) et donner les commandes suivantes :

1 - Analyser la construction de ce fichier. En particulier, essayer l'outil `xxd` pour voir le/les caractère(s) utilisé(s) pour séparer les colonnes.

2 - Donner le nombre de services utilisant un port TCP :

```
grep tcp /etc/services | wc -l → Nbe= 218
```

3 - En prenant à la place du fichier réel le fichier ici :

```
https://github.com/skramm/but3\_rt/blob/main/ressources\_TP/regex/services.txt
```

Donner le nombre de services utilisant un port UDP et avec un numéro de port inférieur à 1023.

```
grep -P "\x09([1-9]([0-9]?)\{2\}|10[0-2][0-3])/udp" services.txt | wc -l → Nbe= 54
```

4 Validation d'entrée utilisateur

Reprendre le code du TP4 : web dynamique Python/Flask dockerisé (on pourra laisser de coté la BDD MySql)

Ajouter le "endpoint" `/newuser/` à votre code Python/Flask. Faites en sorte que ceci produise une page web à partir d'un template Jinja contenant un simple formulaire web permettant de saisir un identifiant et un bouton d'envoi (page type d'une inscription à un service)

Puis, ajouter le code qui va récupérer la saisie utilisateur et valider au moyen d'une regex ce qui a été saisi. Il faudra utiliser le package Python "re" pour gérer les expressions régulières.

(doc ici : <https://docs.python.org/3/library/re.html>)

Pour la regex, deux approches sont envisageables :

- soit on fait une regex globale, qui vérifie que la chaîne saisie respecte tous les critères
- soit on fait plusieurs regex qui vont tester **un** des critères, et on les teste successivement (plus simple et plus robuste).

Le plus simple consiste à utiliser la fonction `fullmatch(patt, string)` qui va renvoyer `None` si la Regex `patt` ne matche pas **exactement** sur la chaîne `string`.

On peut ainsi enchaîner les tests :

```
if re.fullmatch( patt1, str ) == None:  
    print( "Echec sur la condition 1" )  
else:  
    if re.fullmatch( patt2, str ) == None:  
        print( "Echec sur la condition 2" )  
    else:  
        ...
```

On demande que les identifiants respectent les critères suivants :

- Au moins 6 caractères
- Au moins 1 chiffre
- Au moins 1 majuscule et 1 minuscule
- Au moins 1 caractère parmi les 5 suivants : `#%{}@`

Afficher un message indiquant si la saisie respecte ces critères ou pas.

Amélioration : indiquer sur la page quel est le critère que la saisie utilisateur ne respecte pas.