Exercice 1 : Explorons vos fichiers avec le web

Les deux programmes suivants permettent de déléguer à un script nommé **`process-request`**, le traitement de requêtes lues, via le script nommé **`get-request`**, sur le port réseau 8080 d'une machine.

get-request

```
#!/bin/bash

PATH=$(cd $(dirname $0) ; pwd):$PATH

tube=/tmp/requete
mkfifo $tube

while true
do
    cat $tube | process-request | nc.openbsd -l 127.0.0.1 8080 > $tube
done
```

process-request

```
#!/bin/bash

read line

echo $line
```

En se basant sur ces scripts vous allez devoir écrire un serveur web dialoguant exclusivement en `HTTP/1.1` et supportant uniquement la méthode `GET` (les autres méthodes provoque donc une errer `405`). Il devra permettre d'obtenir des fichiers stockés à partir d'un répertoire ⟨*racine*⟩ qui lui sera spécifié.

Ce serveur devra traiter de manière différenciée les requêtes en fonction de la forme de leur cible :

`/contenu/`⟨***chemin***⟩
- si ⟨*chemin*⟩ correspond à un fichier régulier de ⟨*racine*⟩ son contenu est renvoyé par le serveur
- si ⟨*chemin*⟩ correspond à un répertoire de ⟨*racine*⟩ le serveur considère la cible comme non acceptable (406)

`/html/`⟨***chemin***⟩
- si ⟨*chemin*⟩ correspond à un fichier CSV une version sous forme de table HTML de son contenu est renvoyé par le serveur
- si ⟨*chemin*⟩ correspond à un fichier texte une version HTML de son contenu est renvoyé par le serveur
- si ⟨*chemin*⟩ correspond à un répertoire de ⟨*racine*⟩ une version HTML de son contenu est renvoyé par le serveur sous la forme d'une liste non ordonnée (`<ul>`) dans laquelle chaque item (`<li>`) contient une ancre (`<a>`) dont la cible est :
  - de type **`/html/`** pour les fichiers CSV, les fichiers texte et les répertoires
  - de type **`/contenu/`** pour les autres fichiers
- si ⟨*chemin*⟩ correspond à un autre *type* de fichier de ⟨*racine*⟩ le serveur considère la cible comme non acceptable (406)
- sinon le serveur avertit que la cible n'est pas présente (404)

`/html/`⟨***chemin***⟩**`/triepar/`**⟨***nombre***⟩

`/html/`⟨***chemin***⟩**`/triepar/`**⟨***nom***⟩
- si ⟨*chemin*⟩ correspond à un fichier CSV une version HTML de son contenu est renvoyée par le serveur en triant sur la colonne numéro ⟨*nombre*⟩ ou la colonne portant le label ⟨*nom*⟩. Si la colonne demandée est inexistante le serveur avertit que la cible n'est pas acceptable (406)
- si ⟨*chemin*⟩ correspond à un autre `type` de fichier de ⟨*racine*⟩ le serveur avertit que la cible n'est pas acceptable (406)
- sinon le serveur avertit que la cible n'est pas présente (404)

Toutes les autres cibles sont considérées par le serveur comme non acceptables (406).

Toutes les versions HTML des fichiers renvoyées auront une forme identique construite à partir d'un fichier ⟨*modele*⟩.

Pour mettre en place ce serveur vous devrez donc écrire plusieurs commandes afin de rendre votre développement modulaire.

`http-server` est le programme principal qui doit accepter les options :

- `-p` ⟨*port*⟩ spécifiant le port d'écoute
- `-d` ⟨*chemin*⟩ spécifiant la ⟨*racine*⟩ contenant les fichiers servis
- `-t` ⟨*chemin*⟩ spécifiant l'emplacement du modèle utilisé par générer les pages HTML

Par ailleurs pour chaque requête reçu il doit envoyer sur sa sortie d'erreur une ligne décrivant le traitement effectué sur celle-ci. La forme de cette ligne doit être ⟨*script*⟩`:` ⟨*nature*⟩`:` ⟨*cible*⟩`:` ⟨*code*⟩`:` ⟨*chaîne*⟩ avec

- ⟨*script*⟩ le nom de base du script
- ⟨*nature*⟩ qui peut prendre comme valeur `erreur` ou `info`
- ⟨*cible*⟩ la cible demandé par la requête
- ⟨*code*⟩ qui est le code HTTP utilisé dans la réponse envoyée
- ⟨*chaîne*⟩ qui est une chaîne expliquant le code envoyé

`http-request` est le programme qui lit une requête HTTP sur son entrée standard et qui écrit une réponse adaptée sur sa sortie standard.

`cvs2html` est le programme qui lit des lignes CSV sur son entrée standard et qui les transforme en fragment HTML (`<table>`), trié ou non, avant de les envoyer sur sa sortie standard

`remplacer-dans` est le programme qui remplace une zone délimitée par les lignes `<!-- DEBUT -->` et `<!-- FIN -->` dans un fichier ⟨*modele*⟩ par ce qu'il reçoit sur son entrée standard.

Toutes les programmes que vous écrivez doivent accepter une option `-h` décrivant, *à la manière UNIX*, leur fonctionnement.