TP n°9



I. Manipulation de fichiers dans le terminal

Télécharger l'archive **TP9** dans les ressources de TP depuis cahier-de-prepa. L'extraire et s'y rendre avec un terminal. C'est votre salle 20 ! Explorer rapidement l'arborescence ainsi récupérée, en affichant le contenu de quelques fichiers, comme **tableau.txt**. Dans les commandes suivantes, <f> et <f'> désignent un fichier ou un chemin vers un fichier.

```
cp <f> <f'>
```

1. La commande **cp** (pour **copy**) permet de copier un fichier, en indiquant d'abord le fichier à copier puis le nouvel emplacement souhaité sous forme d'un chemin absolu ou relatif. Utiliser cette commande pour copier **tableau.txt** dans un fichier dans votre sac.

```
mv <f> <f'>
```

2. Similairement, la commande **mv** (pour **move**) permet de déplacer un fichier. Ranger la salle 20 en déplaçant les ordures à la poubelle et les livres dans l'armoire. On peut également utiliser cette commande pour renommer un fichier, en lui donnant un nouveau nom.

```
rm <f>
```

3. Pour supprimer un fichier, on utilise la commande **rm** (pour **remove**). Aller dans la poubelle et supprimer quelques fichiers (pas tous, c'est long...). Attention, la suppression est définitive !

```
rm -r <dossier>
```

4. Pour supprimer un dossier, on ajoute le flag **-r**. Revenir dans le dossier **salle20** et supprimer tout le dossier **poubelle** d'un coup. Vous pouvez ensuite recréer un nouveau dossier **poubelle**. Ou pas.

```
ls -a
```

5. Certains dossiers sont cachés : ce sont les dossiers dont le nom commence par un point. Cela est utilisé par les dossiers de configurations par exemple. Ajouter le flag **-a** (pour **all**) à la commande **ls** afin de révéler deux endroits secrets et trouver un indice caché.

```
grep -r <motif>
```

6. La commande **grep** permet de chercher une chaîne de caractères dans tous les fichiers de l'arborescence. Utiliser cette commande pour localiser le code secret dans les livres de l'armoire.

II. Droits d'accès aux fichiers

Utilisateurs et groupes

Les systèmes **Unix** (comme vos distributions Linux, macOS, Android...) suivent le **standard POSIX**, qui définit des conventions pour uniformiser le nom des commandes, la gestion des entrées/sorties et bien d'autres aspects.

Cette norme définit un système d'utilisateurs : chaque utilisateur possède un nom et un identifiant propre nommé **UID**. Chaque utilisateur appartient à des groupes : un groupe principal et des groupes secondaires, qui possèdent également un nom et un identifiant chacun.

```
id
```

7. Utiliser la commande **id** pour afficher votre UID, ainsi que tous vos groupes et leurs identifiants.

Droits d'accès

Chaque fichier dispose de restrictions sur ses droits d'accès : lecture (valeur 4), écriture (valeur 2), exécution (valeur 1). On peut résumer le tout en un seul nombre : on fait la somme des valeurs des droits dont on dispose. Par exemple avoir le droit de lecture et écriture mais pas exécution donne la valeur 4 + 2 = 6. Avoir tous les droits donne la valeur 4 + 2 + 1 = 7. N'avoir aucun droit donne la valeur 0.

Ces droits permettent d'éviter que n'importe qui puisse accéder à n'importe quoi. Concrètement, on divise les utilisateurs en trois catégories : (1) le propriétaire du fichier, (2) le groupe principal du propriétaire du fichier, et (3) tout le monde. Chaque catégorie peut posséder des droits différents sur ce fichier.

```
ls -l
```

8. Ajouter le flag **-l** à la commande **ls** pour afficher le propriétaire d'un fichier et les différent droits d'accès pour chacune des trois catégories. Prendre le temps de bien comprendre l'affichage produit.

```
chmod <num> <f>
```

9. La commande **chmod** (pour **change mode**) permet de modifier les droits d'accès pour chacune des trois catégories. Il suffit pour cela d'indiquer trois chiffres correspondant aux nouveaux droits pour les trois catégories. Par exemple **chmod 777 f** donne tous les droits à toutes les catégories. Que vont faire les valeurs **762** ? **100** ? **421** ?

Inode (ou Inœud)

De manière générale, les méta-données d'un fichier (dernière date de modification, propriétaire, références permettant de retrouver ses données sur le système de stockage...) sont gardées dans un **inode**, une structure de données utilisée par le gestionnaire de fichiers. Ce dernier ne manipule qu'une petite arborescence ne stockant pas le contenu des fichiers : uniquement leurs noms, chacun pointant vers son inode.

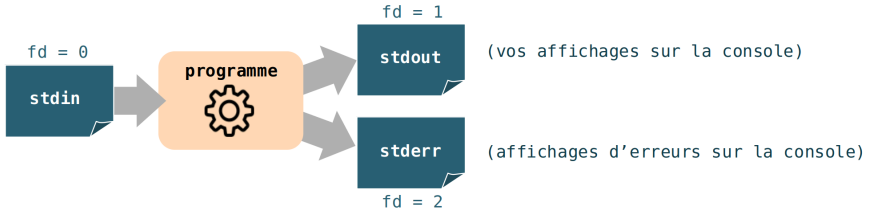
```
stat <f>
```

10. La commande **stat** permet d'afficher des informations sur l'inode d'un fichier, contenant diverses méta-données. Tester sur un exemple et prendre le temps de comprendre ce qui s'affiche.

III. Une 1ère approche pour écrire dans un fichier : la redirection de canaux standards

Canaux standards

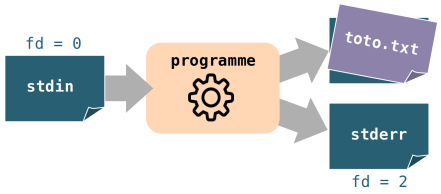
Pour communiquer avec l'extérieur, un programme utilise trois fichiers "virtuels" spéciaux. Le premier, **stdin** s'appelle l'entrée standard. Le second, **stdout**, s'appelle la sortie standard. C'est lui qui est relié à la console quand on lance un programme et contient les affichages de `printf`. Le dernier, **stderr**, sert aux messages d'erreurs.



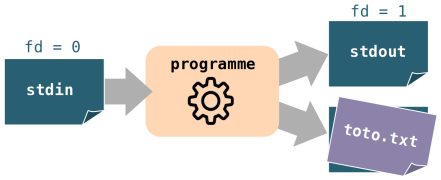
Redirection de canaux standards

Il est possible de rediriger les canaux standards vers d'autres fichiers. Cela permet à votre programme utilisant `printf` de directement écrire dans votre propre fichier, plutôt que d'écrire les affichages dans la console. Pour cela, on utilise l'opérateur `>` (respectivement `2>`) dans le terminal, qui va créer un nouveau fichier (ici **toto.txt**), ou effacer son contenu s'il existe déjà, puis rediriger le canal **stdout** (respectivement **stderr**) vers ce nouveau fichier.

```
./programme > toto.txt
```




```
./programme 2> toto.txt
```



11 Utiliser la redirection de canaux et un petit programme C pour produire un fichier **premiers.txt** contenant les 1000 premiers nombres premiers, un par ligne. Ouvrir le fichier ainsi produit.

Flux en C

En C, les canaux d'entrée-sortie standards sont représentés par des flux de type `FILE *`. Au-delà des canaux standards **stdin**, **stdout** et **stderr**, nous allons désormais manipuler des flux ouverts sur nos propres fichiers pour pouvoir directement lire et écrire dans un fichier quelconque.

IV. Écriture et lecture directement en C

Ouverture et fermeture d'un fichier

En C, on peut directement ouvrir un fichier quelconque et récupérer son flux avec `fopen`.
Cette fonction a pour premier paramètre le nom du fichier, et pour second paramètre un mode d'ouverture.

```c
FILE* f = fopen("toto.txt", "w");
...
fclose(f); // On n'oublie pas de fermer
```

Mode d'ouverture	`"r"`	`"w"`	`"a"`
Signification	Lecture	Écriture	Ajout

C'est vous qui êtes alors responsable de la fermeture du fichier, qui se réalise en appelant la fonction `fclose` avec pour argument le flux ouvert du fichier à fermer. Attention à ne pas utiliser un fichier fermé par la suite.

La fonction fprintf

Une fois un fichier ouvert avec `fopen` en mode écriture, il est possible d'écrire directement dedans avec la fonction `fprintf`. Cette dernière marche comme la fonction `printf`, à ceci près qu'on lui donne tout d'abord un flux (ici `f`) indiquant dans quel fichier on veut écrire, contrairement à `printf` qui écrit par défaut dans **stdout**.

```c
fprintf(f, "Dans un fichier j'écris ce texte et quelques valeurs : %d, %f.\n", v1, v2);
```

12 Modifier votre programme C pour qu'il ouvre lui-même un nouveau fichier nommé **premiers.txt** avec `fopen` en mode écriture. Utiliser la fonction `fprintf` avec le flux renvoyé par `fopen` pour y écrire un par un les 1000 premiers nombres premiers. Vérifier en ouvrant le fichier ainsi produit.

La fonction fscanf

Une fois un fichier ouvert avec `fopen` en mode lecture, il est possible de lire et récupérer son contenu avec la fonction `fscanf`. Cette dernière prend aussi un flux et essaie de faire correspondre une chaîne de format avec ce qu'elle lit dans le fichier. Les arguments supplémentaires ne sont pas des valeurs comme dans `fprintf` : ce sont des pointeurs vers des variables de bons types, où l'on souhaite stocker les valeurs lues dans le fichier.

Si la lecture s'est bien passée, la fonction aura modifié les valeurs au bout des pointeurs donnés, pour correspondre aux valeurs écrites dans le fichier. Ici par exemple, on récupère un entier et un flottant depuis le fichier précédent :

```c
int v1;
float v2;
fscanf(f, "Dans un fichier j'écris ce texte et quelques valeurs : %d, %f.\n", &v1, &v2);
// Les variables v1 et v2 ont bien été mises à jour avec les valeurs lues dans le fichier
```

13 Dans un autre programme C, utiliser `fopen` pour ouvrir le fichier **premiers.txt** en mode lecture. Appeler la fonction `fscanf` à répétition avec la chaîne de format `"%d\n"` pour récupérer un par un les 1000 nombres premiers : chaque appel à `fscanf` lira un entier et mettra à jour la variable au bout du pointeur. Attention à ne pas appeler `fscanf` plus de 1000 fois, ou la lecture échouera.