R1.04 - Systèmes - TP 2
Chemins, Jokers

Rappels

Fichier

Un fichier contient des données (texte, image, son, vidéo, programme etc.)

Un fichier est un élément terminal dans l'arborescence. On ne peut pas aller plus loin, plus profondément, comme pour une feuille sur un arbre, ce n'est pas une branche.

Dossier

Un dossier ne contient pas vraiment de données, il peut contenir d'autres objets (fichiers ou dossiers), et c'est même très souvent le cas. Mais il peut aussi être un élément terminal, s'il ne contient rien, comme une branche sans feuille ni branche sur un arbre.

Pour matérialiser qu'on parle d'un dossier, il est fréquent, même si ce n'est pas nécessaire, de placer un **/** à la fin du nom. Par exemple, **`Documents`** et **`Documents/`** représentent le même objet, mais la seconde écriture ne laisse aucun doute sur le fait qu'il s'agit d'un dossier, alors qu'avec la première forme d'écriture cela n'a rien d'évident. D'ailleurs, il ne faut pas se laisser influencer par le nom d'un objet, la première écriture peut très bien représenter un fichier texte contenant une liste bibliographique d'une thèse ! Qui peut le dire ? Avec un **/** final, aucun doute, il s'agit forcément d'un dossier.

Attention, ce **/** final ne fait pas partie du nom !

Parent

Tout objet du système de fichiers possède un dossier parent.

Il n'est pas nécessaire de connaître le nom du parent pour y faire référence. Le parent possède un nom générique qui est **`..`** (c'est une façon de dire "papa" dans le langage *Système de fichiers Unix*). On dit *point point* et non *deux points*. Par exemple, pour se déplacer dans le dossier parent de l'endroit où on est actuellement :

```
cd ..
```

Composition d'un chemin

Dans un Système de fichiers, un chemin décrit la façon dont on peut accéder à un *objet* (fichier ou dossier).

Un chemin est une succession de dossiers séparés par des **/**

Chaque *morceau* entre les **/** ne peut être qu'un *dossier*.

Le dernier *morceau* (après le dernier **/**) peut être un *dossier*, dans ce cas le chemin mène vers ce *dossier*.

Le dernier morceau, et uniquement celui-là, peut aussi être un *fichier*, dans ce cas le chemin mène vers ce *fichier*.

Exemples :

`/home/etu/mlucas` ⇦ mène vers un dossier, le *Home* de l'utilisateur **`mlucas`**
`/home/etu/mlucas/Documents/exo1.c` ⇦ mène vers un fichier *source C*

Chemin absolu

Un chemin absolu est un chemin **commençant** par un **/**

Le **/** s'appelle la racine de l'arborescence.

Un chemin absolu donné mène toujours vers le même objet, peu importe où on est quand on l'utilise.

Chemin relatif

Un chemin relatif est un chemin qui **ne commence pas** par un **/**

Un chemin est relatif par rapport à l'endroit où on est quand on l'utilise.

Nommage des objets du Système de fichiers

Le nom d'un objet (fichier ou dossier) est le dernier *morceau* d'un chemin. Exemples :

Pour **`/home/etu/mlucas`** il s'agit de **`mlucas`** (c'est le *Home* de **`mlucas`**, c'est un dossier)
Pour **`/home/`** il s'agit de **`home`** (c'est certainement un dossier car il est suivi d'un **/**)
Pour **`/home/etu/mlucas/Documents/exo1.c`** il s'agit de **`exo1.c`** (à priori c'est un fichier, mais rien ne permet de l'affirmer avec une certitude absolue)

Un nom peut être constitué de deux parties :

- Un *nom de base* : tout ce qui se trouve à la gauche du dernier **`.`** (point) du nom, ou le nom complet s'il n'y a aucun **`.`** dans le nom
- Une *extension* : tout ce qui se trouve à la droite du dernier **`.`** y compris ce **`.`**

S'il n'y a aucun **.** dans le nom, alors il n'y a pas d'extension.

Si le nom commence par un **.** alors il n'y a pas de nom de base, seulement une extension.

Un nom peut contenir plusieurs **.** C'est le dernier **.** qui sépare le nom de base et la partie extension. Ceci signifie qu'en présence de plusieurs **.** le nom de base contiendra aussi des **.**

Exemples :

Dans **../tp1/exo1.c**, le nom de base est **exo1** et l'extension est **.c**

Dans **exo1.c.old**, le nom de base est **exo1.c** et l'extension est **.old**

Dans **.old**, le nom de base est *vide* et l'extension est **.old**

Complément à propos des dossiers

"Vous êtes ici" 🎯

Dans le TP 1, nous avons eu l'occasion de voir la commande

```
ls -l -a
```

qui permet d'afficher de façon détaillée (**-l**) tous les objets, y compris (**-a**) ceux qui sont normalement masqués. Cette commande nous affiche deux lignes spéciales, et ce, peu importe où on se trouve. Exemple :

```
drwxr-xr-x 4 jbond  mi6    4096 nov.  27  2020 .
drwxr-xr-x 3 root   root   4096 mai    9  2020 ..
```

On a évoqué un peu plus haut le rôle de **..** (point point, le parent) et on avait remis à plus tard l'explication de l'autre ligne, le **.** (point). Nous voici maintenant à l'explication de sa présence et de son rôle.

Ce **.** signifie *ici*, le dossier courant, celui dans lequel on est actuellement.

Il est très utile car, tout comme on a souvent besoin de se référer au parent, on a aussi très souvent besoin de se référer au dossier où on se trouve actuellement. Voici un exemple pour déplacer un fichier **exo1.c** du dossier parent au dossier courant :

```
mv ../exo1.c .
```

La source est **`../exo1.c`** (le fichier **`exo1.c`** qui se trouve dans le parent, donc dans **`..`**)

La destination est **`.`** le dossier courant, autrement dit *ici*.

Et en procédant ainsi, on n'a pas besoin de préciser le nom du fichier destination, il gardera le même nom lors de son déplacement *ici*.

Vous connaissez déjà cette manière de procéder car vous en faites usage en TP d'Algo/Prog quand vous exécutez un programme que vous venez de compiler :

```
cc exo1.c -o exo1 -Wall
./exo1
```

Cette écriture **`./exo1`** est à lire ainsi : "exécute le programme **`exo1`** qui se trouve *ici*". On reviendra plus tard sur l'explication de la nécessité de mettre le **`./`** devant le programme qu'on veut exécuter.

"Maison" 🏠

On a eu l'occasion de parler de nombreuses fois du *Home* qui est le dossier d'accueil d'un utilisateur.

Ce dossier *Home* a aussi un nom raccourci, on peut y faire référence en utilisant un **`~`** (*tilde*, **`AltGr-2`** sur un clavier PC, **`Option-n`** sur Mac)

Exemple pour copier un fichier vers votre *Home* :

```
cd /tmp
cp doc.txt ~
```

"Je suis mon père" 🎅

Dans le TP 1, nous avons vu que **`..`** représente le dossier parent d'un dossier et que tout dossier possède un dossier parent.

Nous avons aussi vu que **`/`** est la racine du Système de fichiers.

Il est donc naturel de penser que **`/`** ne possède pas de père. Pourtant ce n'est pas le cas. **`/`** possède, lui aussi, un père. **Ce père c'est lui-même !**

C'est une caractéristique bizarre qui tire sa raison d'être d'un besoin d'homogénéité : tous les dossiers ont les mêmes attributs (notamment un père), y compris **`/`**

Ainsi, peu importe le nombre de **`..`** qu'on placera en tête d'un chemin, quand on a atteint la racine, ces **`..`** en surplus n'auront aucun effet, on bouclera sur la racine à chacun d'eux.

Exemple :

`/../../../../../tmp` est finalement identique à **`/tmp`**

Arbre mystère

Voici différents noms complets de fichiers dans un système fictif dans lequel les éléments portent des noms tous différents (il n'y a qu'un seul **`d1`**, un seul **`f2`**, etc). Les répertoires sont nommés **`d...`** et les fichiers **`f...`** On suppose que l'utilisateur qui voit ces noms se trouve dans un répertoire appelé **`d1`** quelque part dans cet arbre. Initialement, on n'en sait pas plus que ça.

Dessinez l'arbre des fichiers correspondant aux noms qui suivent. Les répertoires seront représentés par des rectangles et les fichiers par des cercles :

```
../../../d7/f1          ../../d8/f2          /f3
f4                      /d2/f5               d6/f6
../../../d7/d4/f7       ./f8                 ../d9/f9
/d5/d3/d9/f10
```

Voici maintenant quelques commandes qui s'appliquent à cet arbre. Vous indiquerez quel est ou quels sont les répertoires dans lesquels elles peuvent fonctionner. Par exemple, **`more /f3`** peut fonctionner dans n'importe quel répertoire, mais **`more f5`** ne peut fonctionner que dans **`d2`**.

NB : la commande **`more`** est strictement identique, dans son fonctionnement, à la commande **`less`**, vue en TP 1. Donc, si vous préférez utiliser **`less`**, pas de problème.

Et si elles sont possibles quelque part, alors quels sont les effets des commandes suivantes ?

```
1. more f1
2. more d8/f2
3. more ../d9/f9
4. mv ./f6 ../..
5. rm ../f1 f7
6. mv d6/f6 .
7. cp ../f3 d8/f3
8. cp ../d2/f5 d3
9. mkdir d8
10. mkdir d6/d11
```

Jokers

Définition

Dans un jeu de cartes, un *Joker* est une carte qui remplace n'importe quelle autre carte.

Le *Shell* (chez nous il s'appelle **`Bash`**) utilise aussi ce principe de Jokers dans le ciblage des objets (noms et chemins). Il peut être utile de cibler tous les fichiers dont les noms suivent un modèle de nommage. Par exemple : tous les fichiers dont l'extension est **`.c`** ou encore tous les dossiers dont les noms contiennent **`2019`**.

C'est là que les Jokers entrent en jeu. Comme une carte Joker peut remplacer n'importe quelle autre carte, un Joker Shell peut remplacer un ou plusieurs caractères dans la description du nom. Ainsi, on peut globaliser la cible en une seule écriture générique du nom. Au lieu de dire : *Je cible les fichiers nommés* ***`exo1.c`***, ***`exo2.c`***, ***`exo3.c`*** *et* ***`exo4.c`***, on dit *Je cible les fichiers nommés* ***quelque chose terminé par `.c`***

Le *quelque chose*, c'est ce qu'on appelle un *Joker*.

Types de Jokers

Il existe plusieurs types de Jokers Shell. Pour illustrer les exemples, nous allons partir de cette liste d'objets.

`doc1.txt`	**`doc2.txt`**	**`doc3.txt`**	**`document.txt`**
`ancien_doc.md`	**`docs/`**	**`mes_docs/`**	**`doc_2019/`**
`Documentaire.mp4`			

Peu importe qu'il s'agisse de fichiers ou de dossiers, les Jokers ont le même effet, ils décrivent des noms d'objets.

Dans les exemples suivants, les noms qui **correspondent** au motif sont mis **en vert**, ceux qui **ne correspondent pas** sont ~~**en rouge et barrés**~~.

Joker "Séquence quelconque de caractères" : * (étoile)

L'étoile remplace une séquence de caractères quelconque, y compris une séquence vide ou un seul caractère. Exemples :

Exemple 1 **`doc*`** correspond aux objets dont le nom commence par **`doc`**, suivi d'un nombre quelconque de caractères :

`doc1.txt`	**`doc2.txt`**	**`doc3.txt`**	**`document.txt`**
~~**`ancien_doc.md`**~~	**`docs/`**	~~**`mes_docs/`**~~	**`doc_2019/`**
~~**`Documentaire.mp4`**~~	~~**`cntxt`**~~	~~**`context`**~~	

Identifiez dans chaque cas à quoi correspond l'étoile.

Exemple 2 ***doc*** correspond aux objets dont le nom contient **doc**, car ce **doc** est précédé et suivi d'un nombre quelconque de caractères :

doc1.txt	doc2.txt	doc3.txt	document.txt
ancien_doc.md	docs/	mes_docs/	doc_2019/
~~Documentaire.mp4~~	~~cntxt~~	~~context~~	

Identifiez dans chaque cas à quoi correspond chaque étoile.

Exemple 3 ***txt*** correspond aux objets dont le nom se termine par **txt** :

doc1.txt	doc2.txt	doc3.txt	document.txt
~~ancien_doc.md~~	~~docs/~~	~~mes_docs/~~	~~doc_2019/~~
~~Documentaire.mp4~~	cntxt	~~context~~	

Identifiez dans chaque cas à quoi correspond chaque étoile.

Placer plusieurs ***** successivement (**doc*****) ne sert à rien puisque le 1er englobe déjà n'importe quelle séquence.

La description des noms d'objets avec des Jokers s'appelle un **motif**.

Q.4 Proposez un motif pour décrire tous les objets portant un nom :

a) commençant par un **d**
b) contenant un **_**
c) se terminant par un **1**
d) ayant une *extension*

Vous pouvez vérifier en créant tous ces objets dans un dossier **~/systeme/TP2** et, depuis ce dossier, en faisant un simple **ls ....**, où **....** est à remplacer par votre motif de test.

—

Joker “Un et un seul caractère quelconque” : **?** (point d’interrogation)

Le **?** remplace obligatoirement un caractère mais pas plus d’un caractère. Exemples :

Exemple 1 **doc?.txt** correspond aux objets dont le nom commence par **doc**, suivi d’un et un seul caractère quelconque, suivi de **.txt** :

doc1.txt	doc2.txt	doc3.txt	~~document.txt~~
~~ancien_doc.md~~	~~docs/~~	~~mes_docs/~~	~~doc_2019/~~
~~Documentaire.mp4~~	~~cntxt~~	~~context~~	

Q.5 Identifiez dans chaque cas à quoi correspond le **?**

Plusieurs **?** successifs permettent de remplacer autant de caractères qu’il y a de **?**.

Exemple 2 **doc?????** correspond aux objets dont le nom commence par **doc**, suivi d’exactement 5 caractères quelconques, qui terminent le nom (rien d’autre après ces 5 caractères) :

doc1.txt	doc2.txt	doc3.txt	~~document.txt~~
~~ancien_doc.md~~	~~docs/~~	~~mes_docs/~~	doc_2019/
~~Documentaire.mp4~~	~~cntxt~~	~~context~~	

Q.6 Identifiez dans chaque cas à quoi correspond le **?????**

Notez que le **.** de l’extension est considéré comme un caractère comme les autres.

On peut aussi combiner les Jokers entre eux.

Exemple 3 ***.???** correspond aux objets dont le nom se termine par un **.** suivi de 3 caractères quelconques :

doc1.txt	doc2.txt	doc3.txt	document.txt
~~ancien_doc.md~~	~~docs/~~	~~mes_docs/~~	~~doc_2019/~~
Documentaire.mp4	~~cntxt~~	~~context~~	

Q.7 Identifiez dans chaque cas à quoi correspond l’étoile et les **???**

Joker “Un caractère parmi une liste” : **[...]**

Le **[...]** remplace obligatoirement un et un seul caractère pris parmi les caractères listés entre les crochets (c'est-à-dire la partie : **...**).

Exemple 1 **[Dd]*** correspond aux objets dont le nom commence par un **D** ou un **d** suivi d'un nombre quelconque d'autres caractères, autrement dit, un nom commençant par un **d** (min ou MAJ) :

doc1.txt	doc2.txt	doc3.txt	document.txt
~~ancien_doc.md~~	docs/	~~mes_docs/~~	doc_2019/
Documentaire.mp4	~~cntxt~~	~~context~~	

Joker “Un caractère parmi un intervalle” : **[...-...]**

Le **[...-...]** remplace obligatoirement un et un seul caractère pris parmi les caractères listés par l'intervalle entre les crochets. Exemples :

Exemple 1 ***[0-9]*** correspond aux objets dont le nom contient un caractère entre **0** et **9**, autrement dit, un chiffre :

doc1.txt	doc2.txt	doc3.txt	~~document.txt~~
~~ancien_doc.md~~	~~docs/~~	~~mes_docs/~~	doc_2019/
Documentaire.mp4	~~cntxt~~	~~context~~	

Q.8 Identifiez dans chaque cas à quoi correspond le **[0-9]**

Exemple 2 ***[a-z]*** correspond aux objets dont le nom se termine par un caractère entre **a** et **z**, autrement dit, une lettre minuscule :

doc1.txt	doc2.txt	doc3.txt	document.txt
ancien_doc.md	docs/	mes_docs/	~~doc_2019/~~
~~Documentaire.mp4~~	cntxt	context	

Voici comment fonctionne le mécanisme de l'intervalle **[...-...]**

L'intervalle est développé pour former la liste exhaustive des caractères qui le compose. On se retrouve alors dans la situation de la liste de caractères **[...]**.

Exemples :

[0-9] est développé en **[0123456789]**

[a-z] est développé en **[abcdefghijklmnopqrstuvwxyz]**

Ces listes développées s'entendent ensuite comme *un et un seul caractère parmi la liste de caractères indiqués entre crochets*.

Les intervalles sont donc des versions condensées des listes de caractères vues précédemment.

Pour comprendre le développement des intervalles, il faut connaître la table **ASCII**[1] afin de savoir comment sont classés les caractères. Voici un extrait des caractères affichables. La table **ASCII** représente la table des 128 caractères de base, sans les caractères spéciaux ni les caractères accentués. Certains ne sont pas affichables, il ne sont donc pas listés ici :

```
  ! " # $ % & ' ( ) * + , - . /
0 1 2 3 4 5 6 7 8 9 : ; < = > ?
@ A B C D E F G H I J K L M N O
P Q R S T U V W X Y Z [ \ ] ^ _
` a b c d e f g h i j k l m n o
p q r s t u v w x y z { | } ~
```

NB : le 1er caractère en haut-gauche est le caractère "espace".

Notez que les majuscules viennent avant les minuscules mais que ces deux blocs (MAJ et min) ne sont pas consécutifs, il y ces 6 autres caractères entre les deux blocs : **[\ ] ^ _** et **`**. Ainsi :

- **[A-Z]** : signifie une lettre majuscule
- **[a-z]** : signifie une lettre minuscule
- **[0-9]** : signifie un chiffre

Mais :

- **[a-Z]** : ne signifie pas une lettre minuscule ou majuscule car les majuscules étant avant les minuscules, l'intervalle entre **a** et **Z** est vide car **a** est après **Z**, un peu comme l'intervalle mathématique **[10, 0]** est aussi un intervalle vide.
- **[A-z]** : ne signifie pas non plus uniquement une lettre majuscule ou minuscule car, même si **A** vient avant **z**, cet intervalle contient effectivement toutes les majuscules et toutes les minuscules, mais aussi les 6 caractères **[\ ] ^ _** et **`**

[1] **ASCII** : **A**merican **S**tandard **C**ode for **I**nformation **I**nterchange - **http://asciitable.com**

en plus. C'est donc un peu plus large que simplement les majuscules et les minuscules. On risque alors de cibler de mauvais fichiers si leur nom contient un de ces 6 caractères, sans satisfaire à la présence en même temps d'une minuscule ou d'une majuscule.

On peut aussi combiner des intervalles avec des caractères supplémentaires qui sont ajoutés à la liste une fois l'intervalle développé. Exemple :

Pour **`[ab0-9yz]`**, on développe d'abord l'intervalle, à savoir la partie **`0-9`** qui devient donc **`0123456789`**, à laquelle viennent s'ajouter le **`a`**, le **`b`**, le **`y`** et le **`z`**, pour former finalement la liste **`[ab0123456789yz]`**

Attention, il s'agit ici d'un **intervalle de caractères** et non pas un intervalle mathématique.

On ne peut pas dire que **`[0-100].txt`** représente les fichiers dont le nom est entre **`0.txt`** et **`100.txt`**

En effet, même s'il est tentant et naturel de penser *intervalle mathématique*, si on applique le développement des intervalles tel que vu un peu plus haut, le **`[0-100]`** se développe ainsi :

- On s'occupe d'abord de développer **`0-1`** (on parle en caractères de la table ASCII, pas en valeurs numériques), ce qui donne donc la liste des caractères du **`0`** au **`1`**, soit finalement uniquement le caractère **`0`** et le **`1`**.
- Ensuite, on ajoute les caractères supplémentaires présents dans la liste et qui ne sont pas impliqués dans un intervalle, à savoir un **`0`** et encore un **`0`** (les deux **`0`** présents après la partie **`0-1`**). Même si ça ne sert à rien, il n'est pas interdit de spécifier plusieurs fois le même caractère.
- Ainsi, on obtient la liste des quatre caractères **`[0100]`**, qui peut se réduire à simplement **`[01]`**.

Finalement, **`[0-100].txt`** ne peut représenter que les fichiers **`0.txt`** ou **`1.txt`** et rien d'autre ! (et certainement pas **`10.txt`** ou **`100.txt`**)

Poursuivons nos exercices sur la même base de fichiers que précédemment.

Q.9 Que représentent les listes suivantes, et quels sont les fichiers ciblés :

a) **`*[0-9]*`**
b) **`*[0-9]`**
c) **`*[0-9][0-9]*`**
d) **`[aeiouy]*`**
e) **`[A-Za-z]*`**
f) **`*[2000-3000]`**

Développement des motifs de Jokers

Il est **TRÈS IMPORTANT** de comprendre à quel moment se déroule le développement des *motifs de Jokers*.

Nous avons vu que le *Shell* (qui s'appelle *Bash* chez nous), est l'outil qui sert d'interface entre l'utilisateur et le système. C'est ce logiciel qui interprète les commandes de l'utilisateur et passe la main au système pour leur exécution.

Il se trouve que c'est aussi au Shell que va revenir la tâche du développement des motifs de Jokers. C'est lui qui va faire en sorte de transformer un **`*.c`** en une liste exhaustive de tous les fichiers **`.c`** présents dans le répertoire courant.

Avant de demander au système d'exécuter la commande ordonnée par l'utilisateur, le Shell va donc développer le motif de Jokers en une liste d'objets (fichiers et/ou dossiers) correspondant au motif.

Ensuite, la commande, quelle qu'elle soit, recevra la liste résultant du développement du motif et non pas le motif initial. Elle n'aura donc aucun moyen de savoir si cette liste a été donnée *in extenso* par l'utilisateur ou si elle est le résultat d'un motif de Jokers. Elle n'a, de toute façon, pas besoin de le savoir.

Avant de passer aux exemples, préparons le terrain :

```
mkdir test_jokers
cd test_jokers
touch toto tutu riri fifi loulou
```

Exemple, si on tape ceci :

```
ls
```

ça doit afficher tous les fichiers, mais la commande :

```
ls t*
```

sera, en réalité, passée par le Shell au système sous la forme suivante :

```
ls toto tutu
```

Le Shell s'est chargé au passage de transformer le **t*** en **toto tutu**

Cas particulier, si le motif de Jokers ne correspond à aucun objet lors de son développement, alors il est conservé sous sa forme initiale. Exemple avec les mêmes fichiers que précédemment, la commande :

```
ls z*
```

affichera simplement une erreur :

```
ls: cannot access 'z*': No such file or directory
```

Le motif a été transmis inchangé cette fois-ci, puisque rien commençant par un **z** n'a été trouvé par le Shell, provoquant évidemment une erreur puisque, si le Shell n'a pas trouvé de correspondance, la commande **ls** ne fera pas mieux mais c'est à elle de gérer cette situation, d'afficher cette erreur, car le Shell ne connaît pas le rôle de la commande **ls**, peut-être qu'elle ne traite pas des fichiers mais qu'elle affiche simplement les paramètres qu'on lui passe ? Comment le Shell pourrait-il le savoir ?

Justement, vous allez maintenant apprendre une nouvelle commande qui… affiche à l'écran ce qu'on lui passe en paramètre ! Hé hé, avouez que ça tombe à pic !

Cette commande s'appelle **echo**. Cette fois-ce les barbus ont été assez sympas, on comprend ce qu'elle fait.

Donc, **echo** affiche à l'écran tout ce qu'on lui passe en paramètres derrière son nom. Exemple (à tester) :

```
echo Linux est formidable
```

Autre exemple (à tester aussi) :

```
echo t*
```

Cette fois-ci on obtient l'affichage, comme avec **ls**, de : **toto tutu**. C'est normal puisque, comme on l'a vu, la commande réellement confiée au système est simplement :

```
echo toto tutu
```

Essayons avec :

```
echo z*
```

Cette fois-ci on obtient **z***

Le Shell a, encore une fois, tenté de développer le **z*** en une liste d'objets, sans trouver aucune correspondance. Il a donc passé à **echo** la chaîne **z*** intacte. La commande **echo** a alors affiché ce qui lui a été passé en paramètre, à savoir la chaîne **z***, tout simplement ! **echo** ne travaillant pas sur des fichiers, ça ne pose aucun problème, pas d'erreur. Pour **echo** c'est juste du texte, contrairement à **ls** qui avait affiché une erreur, car **ls** s'intéresse à des fichiers qui doivent exister, bien entendu. En tout cas, ce n'est pas le Shell qui avait pris la décision d'afficher une erreur, mais la commande elle-même qui est la seule à savoir, de son point de vue, ce qui est une erreur et ce qui ne l'est pas.

Astuce : Désormais vous pouvez tester vos Jokers avec des commandes **echo** au lieu de commandes **ls**, ce sera plus simple, efficace et compact.

Exercices de synthèse

Toujours sur le même jeu de fichiers, proposez un motif de Jokers, **le plus court possible**, pour représenter les fichiers suivants et uniquement ceux-là :

Q.10

doc1.txt	doc2.txt	~~doc3.txt~~	~~document.txt~~
~~ancien_doc.md~~	~~docs/~~	~~mes_docs/~~	~~doc_2019/~~
~~Documentaire.mp4~~			

Q.11

~~doc1.txt~~	~~doc2.txt~~	~~doc3.txt~~	~~document.txt~~
~~ancien_doc.md~~	docs/	mes_docs/	~~doc_2019/~~
~~Documentaire.mp4~~			

Q.12

doc1.txt	doc2.txt	doc3.txt	document.txt
~~ancien_doc.md~~	~~docs/~~	~~mes_docs/~~	~~doc_2019/~~
Documentaire.mp4			

Q.13

```
doc1.txt        doc2.txt        doc3.txt        document.txt
ancien_doc.md   docs/           mes_docs/       doc_2019/
Documentaire.mp4
```

Q.14

```
doc1.txt        doc2.txt        doc3.txt        document.txt
ancien_doc.md   docs/           mes_docs/       doc_2019/
Documentaire.mp4
```

Q.15

```
doc1.txt        doc2.txt        doc3.txt        document.txt
ancien_doc.md   docs/           mes_docs/       doc_2019/
Documentaire.mp4
```

Q.16

```
doc1.txt        doc2.txt        doc3.txt        document.txt
ancien_doc.md   docs/           mes_docs/       doc_2019/
Documentaire.mp4
```

Bonus sur les Jokers

Créez maintenant un nouveau dossier **~/systeme/tp2/jokers** (à vous d'adapter), et déplacez-vous dedans. Continuez en créant les fichiers suivants à l'aide de cette commande :

> **touch prog.c prog.o projet.c projet.o projet.out presentation scene**

Donnez les commandes les plus courtes possibles pour effectuer les opérations demandées ci-dessous. *Montrez* : signifie qu'on utilise la commande **ls** et qu'il faut lui donner un paramètre qui ne lui fait afficher que les fichiers concernés.

Exemple : **ls tot*o**

1. montrez **`prog.c`** et **`prog.o`**
2. montrez **`prog.c`** et **`projet.c`**
3. montrez **`projet.o`** et **`projet.out`**, (mais ni **`projet.c`** ni les autres)
4. montrez **`projet.c`**, **`projet.o`**, **`projet.out`** et **`presentation`**
5. montrez tous les fichiers sauf **`presentation`**
6. montrez **`projet.c`** et **`projet.o`** mais pas **`projet.out`**
7. montrez **`presentation`** et **`scene`** mais aucun autre.
8. montrez **`scene`** seulement
9. montrez **`projet.out`** et **`presentation`**