Travaux Dirigés et Pratiques de Utilisation des Systèmes Informatiques (HAI103I)

Michel Meynard

Préambule

Connectez-vous à l'ENT de l'Université `https://ent.umontpellier.fr/`, puis cliquez sur Moodle `https://moodle.umontpellier.fr/` ou bien directement sur Moodle où vous trouverez dans le cours USI (HAI103I) les supports de cours, des TDs et des TPs.

Pour tout problème de connexion (login, mot de passe oublié, ...), les enseignants ne peuvent pas vous aider. Il vous faut contacter le centre de services de la DSIN et déposer un ticket `https://centre-de-services.umontpellier.fr`.

Les exercices de TD et de TP sont destinés à vous aider à comprendre le cours. En TP, vous devrez avoir recours aux pages du manuel Unix/Linux disponibles par la commande `man xxx`, ou par l'URL `http://manpagesfr.free.fr/consulter.html`. Certaines distributions utilisent aussi la commande `info` pour décrire les commandes complexes.

aide-mémoire minimal INDISPENSABLE

Terminal
- flêche vers le haut ↑ pour l'historique des commandes précédentes
- clear pour effacer le terminal
- Ctrl-C pour arrêter la commande courante en avant-plan
- kill -9 pid pour tuer définitivement un pus récalcitrant

Manuel Unix
- `man -k mot-clé` pour lister les entrées de manuel correspondant à ce mot-clé
- `man bash` pour toutes les commandes internes
- `/nom<Enter>` recherche d'occurrence de nom dans la page (passe en mode recherche)
- n permet de rechercher la prochaine occurrence
- q permet de quitter la page de man
- `<Space>` permet de passer à la page suivante

1 Arborescence Unix

Exercice 1 (TD)
En quel langage est écrit le système d'exploitation Unix ?

Exercice 2 (TD)
Quels sont les types de fichiers Unix que vous connaissez ?

Exercice 3 (TD)
Qu'est ce qu'un chemin (path) au niveau syntaxique ?

Exercice 4 (TD)
Soit le rendu de la commande suivante :

```
~$ ls /bin
...
ls
lscpu
lualatex
lxterm
lynx
make
man
...
```

1. Que représente ce répertoire ?
2. Citez 2 interpréteurs de commandes
3. Quelles commandes permettent de manipuler des répertoires ?

Exercice 5 (TD/TP Exécutables, localisation, PATH)
Dans l'exercice précédent, un certain nombre de commandes externes essentielles ont été localisées dans `/bin`.

1. Quelle commande permet de connaître le répertoire où est située une commande ?
2. Connaissez-vous un autre répertoire contenant des commandes aussi importantes que `mkdir` ?
3. Quel est le type de fichier contenant une commande ?
4. Existe-t-il des commandes non présentes dans l'arborescence des fichiers ?
5. Que pensez-vous de l'affichage suivant :

```
~$ which codium
/usr/bin/codium
~$ ls -l /usr/bin | grep codi
lrwxrwxrwx 1 root root 28 mars  21 11:03 codium -> /usr/share/codium/bin/codium
```

   Quel est le type de ce fichier `/usr/bin/codium` ?
6. Où se situe la liste des répertoires d'exécution ? Cette liste contient les répertoires dans lesquels bash cherche itérativement un fichier exécutable correspondant au premier mot de la ligne de commande.

Exercice 6 (TP Commandes et widgets (objets graphiques XFCE))

1. Explorez le répertoire `/usr/bin` qui contient également des commandes (4500) !
2. Repérez certaines commandes connues (firefox, chromium-browser, g++, gcc). Pour explorer ces commandes, utilisez le manuel Unix !
3. Grâce au whisker, sélectionnez Paramètres/Editeur de type MIME. Choisissez "text/plain" puis sélectionnez VSCodium plutôt que l'éditeur de texte par défaut `gedit` qui est extrêmement sommaire ... Vous pouvez ainsi personnaliser les applications lancées lorsque vous double-cliquez sur un fichier ayant une extension (txt, py, c, ...).
4. Cliquez avec le bouton droit de la souris sur un bouton graphique XFCE du bureau et sélectionnez `Propriétés` puis l'onglet `Lanceur` : dans le champ `Commande` apparaît la commande qui est exécutée lorsque vous double-cliquez desssus !

Exercice 7 (TP ls)
Exécutez puis décrivez en une phrase courte ce que fait chacune des commandes suivantes :

1. `ls ..`
2. `ls -a`
3. `ls -l`
4. `ls -R ~`
5. `ls -R /` (Tapez Ctrl-c pour arrêter cette tâche)

Exercice 8 (TP cd pwd)
Décrivez en une phrase courte chacune des commandes suivantes :

```
pwd
cd / ; pwd ; cd ~ ; pwd
cd / ; pwd ; cd $home ; pwd
cd ~toto ; pwd       (toto est un <nom d'utilisateur>)
cd ~/.. ; pwd
cd /usr ; pwd ;cd  ; pwd
cd /home ; pwd
cd ~ ; cd ../.. ; pwd
cd ../toto ; pwd
```

Exercice 9 (TP)
Depuis votre répertoire d'accueil :

1. créez un fichier `fable.txt` contenant le mot `été` (sans éditeur de texte)
2. quelle est la taille de ce fichier ?
3. créez un script bash `lsparent.sh` qui affiche "Contenu du répertoire parent de <nom du répertoire de travail>", puis qui liste son contenu SANS changer de répertoire !

Exercice 10 (TP mkdir rmdir mv)
Afin d'organiser votre espace, vous allez créer votre sous-arborescence :

```
~
  USI
    Script
     lsparent.sh
    Fable
      fable.txt
```

Donnez la liste des commandes nécessaires SANS changer de répertoire (d'accueil) et sans recopier les fichiers réguliers. Que pensez-vous des noms de répertoires contenant des espaces ?

Exercice 11 (TP commandes récursives)
Certaines commandes peuvent agir sur une sous-arborescence !

1. Effectuez une copie récursive de votre répertoire USI dans un autre répertoire "Backup" que vous aurez précédemment créé !
2. Affichez la liste récursive de vos répertoires contenus dans votre répertoire d'accueil !
3. Tentez de supprimer le répertoire Backup ! Que se passe-t-il ? Comment résoudre ce problème ?

Exercice 12 (TP commande find)
La commande `find path [expr]` permet de rechercher des fichiers en parcourant récursivement l'arborescence à partir du répertoire `path` et d'effectuer une action sur les fichiers parcourus. Un exemple simple de recherche des fichiers shells dont le nom termine par sh suit :

```
$ find /bin -name "*sh"
/bin/csh
/bin/bash
/bin/sh
/bin/tcsh
/bin/ksh
/bin/zsh
$
```

L'action réalisé sur les fichiers sélectionnés est par défaut de les afficher mais on peut réaliser d'autres actions grâce à l'expression `-exec` suivie de l'action à réaliser puis d'un `\;`

1. Recherchez tous les fichiers textes d'un de vos répertoires et listez leurs noms (print)
2. Recherchez tous les fichiers textes d'un de vos répertoires et listez leurs noms suivi de leur contenu
3. Comme la mise en page manque d'espace, refaites l'exercice précédent en ajoutant une ligne blanche entre chaque fichier et un soulignement du nom de fichier ...

Exercice 13 (TP fichiers compressés et archives)
Différents algorithmes et formats de compression coexistent (zip, gzip, bzip, ...). Un fichier régulier `toto.txt` pourra ainsi être compressé en `toto.txt.gz` par exemple. De plus, l'archivage de répertoires est possible avec la commande `tar` qui concatène récursivement tous les fichiers du répertoire cible dans un unique fichier régulier d'extension `tar`. Sous Unix, les archives `tgz` (tarballs) sont souvent utilisées comme moyens d'échange ou d'empaquetage. Un fichier d'extension `tgz` est le résultat d'un archivage de dossier (via `tar`) puis de la compression de cette archive par `gzip`.

1. Etudier le manuel de `gzip` puis testez la compression de 2 fichiers textes que vous aurez créés au préalable **en une seule commande !**
2. Que deviennent les fichiers originaux ? Recréez-les !
3. Quelle est l'option de `gzip` qui permet de conserver la version originale ?
4. Etudiez le manuel de `tar` notamment la partie Exemples à la fin puis testez l'archivage de 2 fichiers textes sans compression dans `archive.tar`. Afficher le contenu de l'archive avec cat puis avec `tar -tf archive.tar`...
5. Archivez avec compression ces deux mêmes fichiers et comparez la taille des archives. Quelle remarque peut-on faire sur les tailles ?
6. Archivez avec compression un gros répertoire puis recréez le à un autre endroit.
7. Comparez les deux versions du répertoire avec la commande `diff` récursive.

Exercice 14 (TP find tar gzip !)
On souhaite constituer une archive compressée de certains fichiers situés dans une arborescence (et pas des autres). Pour cela, on utilisera `find` pour rechercher ces fichiers et pour les ajouter à un fichier tar qu'on compressera ensuite :

1. Trouvez tous les fichiers textes de `monRep` et ajoutez les dans `mesTextes.tar` !
2. Compressez ce fichier grâce à gzip !
3. Réalisez les 2 actions en une seule ligne de commande !

Exercice 15 (TP pour ceux qui ont terminé !)
Le portail web sapiens de l'Université `https://sapiens.umontpellier.fr/` vous donne accès à de nombreux services et ressources. La plupart, service Web et MySQL ne sont accessibles **que lorsque vous êtes à l'Université** (depuis le domaine Internet umontpellier.fr). Cependant, l'application cliente `x2go` vous permet depuis votre domicile de vous connecter à un serveur x2go de l'université, et donc de travailler en mode graphique depuis chez vous (Windows ou Mac OS X ou Linux) sur une machine semblable à celles des salles de TPs en bénéficiant de votre espace disque de l'Université.

Parcourez la documentation du portail sapiens pour en savoir plus !

2 Système de gestion des fichiers

2.1 droits d'accès et de suppression de fichiers

Exercice 16 (TD)

1. Sous unix, quels sont les droits nécessaires pour pouvoir supprimer un fichier ?
2. Y a-t-il une relation entre le droit de supprimer un fichier et les droits d'accès à ce fichier ?
3. Quels sont les droits nécessaires pour modifier le contenu d'un fichier ?
4. Est-ce que le propriétaire d'un fichier peut s'enlever ses propres droits de lecture, écriture et exécution sur un fichier ? Est-ce grave ?
5. Obtient-on une meilleure protection si le propriétaire du fichier enlève ses propres droits d'accès ?
6. Dans quelle situation ne peut-on écrire sur un fichier alors qu'on a le droit de modifier (w) celui-ci ?

Exercice 17 (TD/TP)
Créez un fichier `USI/droits.txt` avec `vscodium` et saisissez-y du texte puis enregistrez-le. Dans un terminal xfce ou dans un terminal de `vscodium` testez la commande suivante :

```
ls -ali | grep droits
```

Expliquez à quoi correspond chacune des colonnes de l'affichage suivant ?

```
3615674885 -rw-r--r--  1 michel.meynard@umontpellier.fr p00000010501 8 août  30 10:54 droits.txt
```

2.2 droits sur les répertoires

Exercice 18 (TD)
Que signifie le droit `x` sur un répertoire ? On veut donner à quelqu'un l'accès à un fichier en lecture et écriture, mais on ne veut pas qu'il puisse prendre connaissance du répertoire contenant. Comment faire ? Que peuvent faire tous les autres utilisateurs ?

Exercice 19 (TD)
Quelles sont les possibilités offertes lorsqu'on a le droit d'écriture sur un répertoire ?

Exercice 20 (TD)
Pour les TP, préparer un ensemble de manipulations, afin de mettre en évidence la signification des droits. Par exemple :

- enlever le droit de lecture sur un répertoire et chercher à modifier un fichier dans ce répertoire,
- enlever le droit d'exécution sur un répertoire et chercher à atteindre un fichier inclus,
- enlever le droit d'écriture sur un répertoire et supprimer un fichier inclus (qu'on soit propriétaire de ce fichier ou non),
- etc, à votre imagination.

2.3 Cohérences dans le système de gestion de fichiers

Les situations décrites ci-après correspondent à la vision d'une partie de la gestion de l'espace disque sous unix. Vous devez répondre face à chaque situation si elle vous paraît cohérente ou non. Considérez chaque situation comme un cas séparé et ne pas tenir compte de la colonne *type*. N'oubliez pas de justifier votre réponse et de signaler toutes les anomalies que vous trouvez.

les blocs d'allocation seront pris de taille `8 kibi-octets` (1Kio=2^{10} octets). la notation `(0)` signifie « libre ». On admettra que tous les blocs qui suivent le premier bloc libre sont libres.

situation	numéro d'inode	type	taille	blocs d'adressage direct					...
1	148		29000	4776	4786	7021	7022	(0)	(0)
	272		38000	2415	4728	4014	17012	30202	(0)
2	2405		37000	2405	4718	4004	17002	(0)	(0)
3	256		27000	4102	8204	2307	6409	(0)	(0)
	498		38000	1205	6144	4102	1025	1026	(0)

Exercice 21 (TD)
Corriger les situations incohérentes et proposer pour chaque situation un ensemble cohérent.

2.4 Cohérence fichiers et répertoires

On regroupe l'ensemble des situations en une seule. On précise en outre les données relatives aux liens.

inode	nb. liens
148	3
272	1
2405	1
256	5
498	2

Exercice 22 (TD)
À la seule vue de ce tableau de liens, peut-on dire si une entrée est un fichier simple ou un répertoire ?

Exercice 23 (TD)
On propose maintenant deux organisations possibles de répertoires. Est-ce que chaque organisation (1 puis 2) est cohérente avec les données ci-dessus de l'exercice précédent ?

organisation	répertoire 1		répertoire 2	
1	fibre	148	fini	272
	finioui	2405	fininon	148
2	fibonacci	498	fibonnacci	498
	figue	148	figue	148
	filon	148		
	fichtre	498		

2.5 Manipulations sur les fichiers et les droits d'accès

Exercice 24 (TD)
La commande `chmod` (`man chmod`) et l'appel système `chmod()` (`man 2 chmod`) existent. Quelle est l'utilité de chacun ?

2.6 Droits d'accès : chmod et umask

On peut utiliser des symboles prédéfinis (`[ugo][+-=][rwx]+`) ou une valeur équivalente en octal (0755), partout où il est nécessaire de modifier les droits d'accès aux fichiers avec chmod.

Lors de la création de fichiers, il est tenu compte de la variable d'environnement `umask`. Les droits définitifs sont obtenus par l'opération :

`(droits demandés) et (non umask)`

La **commande interne** `umask` permet de déterminer les droits par défaut avec lesquels tout fichier sera créé. C'est bien un masque qui est mis en œuvre : `umask` *valUmask* provoque la prise en compte des droits de base `rw-rw-rw-` (666 en octal), puis on fait l'opération :

`(droits de base) et (non umask)`

Ce qui revient à retirer chaque élément correspondant à un 1 binaire de *valUmask*. Par exemple :

`umask` 022 crée les fichiers avec les droits `rw-r--r--`
`umask` 024 crée les fichiers avec les droits `rw-r---w-`
`umask` 066 crée les fichiers avec les droits `rw-------`

Exercice 25 (TP)
- Quelle est la valeur courante de votre umask ?
- Changez votre valeur de masque à 020 puis créez un nouveau fichier : quels sont ses droits ?
- Créez maintenant un nouveau répertoire et vérifiez ses droits.
- Peut-on s'interdire à soi-même tout accès à tout fichier nouvellement créé ? Est-ce grave (irréversible) ?

2.7 numéros d'inodes et liens physiques (durs)

Exercice 26 (TD)
Quelle commande permet de visualiser le numéro d'inode d'un fichier ? Quelle commande permet de visualiser le nombre de liens (liens *physiques* dits *durs*) ? Quelle commande permet de lister les fichiers cachés (commençant par un point) ?

Exercice 27 (TD)
Partant d'un fichier et de son numéro d'inode, comment peut-on trouver l'ensemble des noms qui référencent cet inode ?

Exercice 28 (TP lien dur)
- Créer plusieurs liens durs sur un fichier toto.txt vous appartenant dans le même répertoire et dans un répertoire différent. Peuvent-ils avoir le même nom ?
- vérifiez le compte de liens et l'inode dans chaque répertoire ?
- Supprimez le lien sur toto.txt. Que se passe-t-il ?
- modifiez le contenu du fichier grâce au lien du même répertoire. Que se passe-t-il pour le dernier lien créé ?

2.8 Une utilisation dangeureuse

Beaucoup d'éditeurs de texte prennent la peine de créer une sauvegarde du fichier édité avant de laisser l'utilisateur faire des modifications. Soit *figaro* le fichier à éditer. L'éditeur procède ainsi :
- au lancement il renomme *figaro* en *figaro.levieux* ou *figaro*
- il laisse ensuite l'utilisateur faire son travail, jusqu'à la demande de sauvegarde,
- la demande de sauvegarde se fait en recréant *figaro*, puis en lui attachant les droits d'accès définis dans l'ancienne version.
- En cas de liens durs multiples, les anciens liens pointent maintenant sur l'inode contenant l'ancienne version !
- Dans le cas d'un lien symbolique sur figaro avant édition, c'est le nouveau figaro après édition qui reste pointé. En effet, il y a bien attribution d'un nouvel inode au nouveau figaro, mais le lien symbolique ne référence que le chemin.

Exercice 29 (TP liens symboliques)
Créer des répertoires temporaires et faire des liens symboliques sur ces répertoires, toujours par vous-même et par un collègue. Cette fois-ci peut-on savoir qu'un lien existe sur ce répertoire ? Que se passe-t-il :
- lorsqu'on supprime un lien ?
- lorsqu'on supprime le répertoire sans supprimer les liens ?
- Comment produire un circuit de liens symboliques ?

3 Processus Unix

Exercice 30 (TD)
L'exécution d'une commande déclenche-t-elle toujours la génération d'un nouveau processus ? Donnez deux exemples.

Exercice 31 (TD)
Pourquoi tout processus a deux modes d'exécution (utilisateur, noyau (système)) ?

Exercice 32 (TD)
Quel est le nom du mécanisme qui permet à plusieurs utilisateurs de travailler en même temps sur une seule machine ? Sur une machine mono-processeur comment est réalisé la commutation de processus ?

Exercice 33 (TD)
Listez l'ensemble des mécanismes qui permettent à un processus de "communiquer" avec l'extérieur de ce processus.

Exercice 34 (TD)
Après avoir détaché un processus du terminal courant (`vscodium &`), on souhaite supprimer ce processus. Comment faire ?

3.1 Observations statistique concernant les processus

Un échantillonnage concernant l'état des processus à un instant donné (une photographie) est renvoyé par la commande `ps`. Différentes options renvoient, pour chaque processus, , l', ... , son état STAT (actif, endormi, inactif, swappé, en défaut de page, en basse priorité ...). Voir `man ps`.

Exercice 35 (TP)
Exécutez les commandes : `ps; ps l; ps vx; ps faux`

Essayez de trouver la signification des différents champs (man ps) et options. Repérez la filiation PID - PPID, c'est à dire la relation numéro du processus - numéro du processus père. Repérez votre processus d'ouverture de session !

3.2 Enchaînements , enchaînements conditionnels

Exercice 36 (TP)

1. Enchaînez, en une seule ligne de commande, le listage (ls) du répertoire courant puis l'affichage (cat) du fichier des mots de passe (fichier /etc/passwd) enfin l'affichage de l'heure (date).
2. Toute commande Unix renvoie un code de retour (status) qui vaut vrai (0) si la commande s'est déroulée sans échec et faux (tout entier différent de 0) sinon. Enchaînez une commande `cat toto`, puis en utilisant le caractère spécial convenable (`; || &&`), une seconde commande `echo "échec !"`.
3. même question que la précédente mais avec affichage de "réussite !" si le fichier toto existe

3.3 Exécution de processus concurrents : le caractère de détachement &

Exercice 37 (TP)
Exécutez la ligne suivante :
`( sleep 5 ; echo "fin des 5 s" ; ps f ) &  (sleep 4 ; echo "fin des 4 s" ; ps f ) &`

1. Combien de nouveaux processus ont été créés ?
2. Que signifie les nombres 1 et 2 entre crochets ?
3. Peut-on prédire quel nouveau processus à démarré en premier et l'ordre des suivants ?
4. Quel parallélisme et entrelacement observez-vous ?

3.4 Interrompre ou tuer des processus

Exercice 38 (TP)
Exécutez la commande : `find / -type d -print`

Cette commande cherche à partir de la racine (/) tous les objets de type répertoire et affiche leur nom. Attention cette commande est récursivement appliquée à tous les répertoires fils de la racine. L'exécution complète est très longue. On est bloqué tant que le processus find n'est pas terminé. Plusieurs moyens permettent de débloquer la situation :

1. interrompez le processus find en tapant le caractère Intr généralement Ctrl-C
2. On ne peut cependant interrompre que le processus s'exécutant en avant-plan. Exemple : `find / -type d -print &`. Le processus est lancé en arrière-plan ou tâche de fond. De plus ses sorties polluent la sortie standard. Que faire ? Ramener le processus en question en avant-plan : %1 puis l'interrompre Ctrl-C ou bien l'interrompre directement : `kill %1`. On suppose que 1 est le numéro de job associé au processus lancé en arrière plan (commande `jobs` pour les connaître).
   Parfois, certains processus sont plus coriaces, et ne se laissent pas tuer aussi facilement. On emploie alors la manière forte : `kill -9 PID ....`
3. Lancer un processus vscodium en arrière-plan puis le tuer avec la commande kill.
4. Pouvez-vous tuer des processus qui ne vous appartiennent pas ? Essayez.
5. Exécutez la commande `find / -type d -print`. Ouvrez une deuxième fenêtre shell ou un autre onglet, et depuis cette autre fenêtre, observez la liste des processus actifs en utilisant la commande `ps`. Tuez alors le processus find en utilisant la commande `kill -9` en lui passant le numéro de processus de find. Tuez de même le processus shell de l'autre fenêtre. Observez ce qui se passe.

3.5 Redirection des entrées/sorties : > < >> 2 >

De façon à permettre l'exécution de jobs sans présence d'êtres humains (sans interaction), il est souvent utile de "rediriger" les E/S dans des fichiers.

Exercice 39 (TP)

1. Reprenez la commande de la section 3.3 et redirigez les sorties des différents processus concurrents dans un fichier
2. Que fait la commande suivante : `ls /bin /toto > ~/commandes`
3. Que fait la commande suivante : `cat <tempo >tempo2` ?

3.6 Le mécanisme de "tube" ou "pipe" : les filtres

Un filtre est une commande qui reçoit un flot de données sur son entrée standard, les transforme et écrit un flot de données sur sa sortie standard. On compose les filtres par le mécanisme de tube pour obtenir de nouveaux filtres. Il suffit de connaître les filtres de base proposés avec le système Unix, pour résoudre une bonne part de problèmes "système". Les données rencontrées sont :

— de type caractère,
— ou de type ligne, une ligne étant une suite de caractères terminée par le caractère de fin de ligne,
— ou de type structuré avec un caractère séparateur de champs paramétrable ; un enregistrement est une ligne découpée en champs séparés par SEP.

La fin de flot est signifiée par un symbole virtuel de End Of File générable par Ctrl-d.

Les filtres de base :

— `more, less ou most` : affiche un flot de caractères page par page. Ex : `cat /etc/passwd | less`. autre exemple : man
— `expand` change les caractères de tabulation en espaces
— `unexpand` change les espaces multiples en caractères de tabulation
— `tr a b` (translate) convertit les a par b. Ex : `cat /etc/passwd | tr ':' '\t'`
— `wc` compte les caractères, mots, lignes du flot d'entrée.
— `pr` formate le flot de caractère en entrée suivant des critères longueur de ligne, taille de la page, en-têtes ... Ex : `cat monfic | pr -3` affiche le fichier en 3 colonnes avec troncatures !
— `nl` affiche un numéro de ligne pour chaque ligne du flot d'entrée.
— `head -n` tronque un flot de lignes à ses n premières lignes.
— `tee monfic` recopie son entrée sur la sortie **et** sur monfic.
— `sort` trie un flot de lignes, éventuellement suivant la valeur de champ(s) lorsque celles-ci sont structurées en champs. Ex : `cat /etc/passwd | sort -t: +5` trie le fichier des mots de passe suivant la valeur du champ 5 (répertoire d'accueil) avec comme séparateur de champ ' :'
— `cut` "coupe" chaque enregistrement en conservant les seuls champs spécifiés (peut être combinée avec la commande paste pour réordonner les champs dans un fichier). Ex : `cat /etc/passwd | cut -d: -f 1,3,5` restreint le fichier des mots de passe aux champs 1, 3 et 5, le séparateur de champ étant ' :'.
— `grep, fgrep, egrep ARG` sélectionne les lignes contenant ARG. L'argument peut être une expression définissant un modèle de chaîne de caractères suivant un langage. Ex : `cat /etc/passwd | grep root` affiche toute les lignes du fichier des machines contenant la chaîne root
— `sed OPTION` filtre d'édition. OPTION définit les transformations à faire subir au flot d'entrée. Ex : `cat /etc/passwd | sed "s/a/xxx/g"`. Ici OPTION est une substitution globale de la chaine xxx à l'expression a.

Exercice 40 (TD)

1. Enchaînez en utilisant le mécanisme de tube : ls, sort, pr, more, avec les bons arguments et les options appropriées pour chaque commande, de façon à obtenir en une seule commande : la liste des fichiers de /dev (ls) dans l'ordre inverse des noms (sort) mise en page par pr sur 3 colonnes avec une largeur de ligne de 60 caractères, 40 lignes par page et un titre "jolie impression" affichés sur l'écran page par page grâce à more.
2. Afficher le nom des fichiers de "type texte" de votre répertoire d'accueil. Idée : la commande `file fichier ...` détermine le "type" de l'argument en examinant le début du contenu de l'argument. Essayez : `file *` et `file /usr/bin/a*`.
3. De la même façon, affichez le nom des répertoires de votre répertoire d'accueil
4. Afficher le nom de connexion, le nom complet et le répertoire d'accueil de tous les utilisateurs triés par ordre du nom complet.

Exercice 41 (TP connexion à distance)

La commande `ssh` permet d'ouvrir une session à distance. Pour cela, il faut qu'un serveur `sshd` tourne sur la machine cible. A l'université, un groupe de machines sont accessibles via ssh depuis n'importe où (`x2go.umontpellier.fr`).

1. étudiez la commande ssh avec le man, notamment l'option Y
2. connectez-vous depuis un terminal local avec la commande suivante :

```
ssh -A -Y prenom.nom@etu.umontpellier.fr@x2go.umontpellier.fr
```

lancez quelques commandes (`emacs &`, ...) et vérifiez que les processus tournent bien sur la machione distante.

des clients x2go graphiques existent pour windows et MacOS X

4 Développement logiciel sous Unix

4.1 Scripts Bash

Exercice 42 (TD)
Quelques questions et réponses **justifiées** :

- un script bash est-il :
  - un fichier régulier ?
  - un fichier binaire ?
  - un fichier ayant l'extension `sh` **obligatoire** ?
  - un fichier se trouvant obligatoirement dans un des répertoires du `PATH` ?
- le lancement d'un script affiche le message `permission denied`. Pourquoi ?
- le lancement d'un script affiche le message `file not found`. Pourquoi ?
- le lancement d'un script affiche le message `syntax error`. Pourquoi ?

Exercice 43 (TP script bash lsparent.sh)
créez un script bash `lsparent.sh` qui, **sans changer le répertoire de travail**, affiche le nom du répertoire parent du répertoire de travail puis qui liste le contenu complet (même les fichiers cachés) du répertoire parent de façon exhaustive (voir tous les champs). Veillez à ce que ce script soit exécutable par vous-même !

Exercice 44 (TP)
Ecrire un script Bash `lsarg.sh` qui affiche la liste des arguments, y compris la commande, chacun sur une ligne différente numérotée de 0 à n-1 :

```
$ lsarg.sh a b c
Liste des arguments :
0 : ./lsarg.sh
1 : a
2 : b
3 : c
$
```

Exercice 45 (TP Factorielle)
Ecrire la fonction récursive factorielle qui calcule le produits des entiers consécutifs inférieurs au paramètre fourni et écrire le script correspondant :

```
Exemples$ ./fact.sh 5
120
```

Exercice 46 (TP Parcours récursif d'un répertoire)
Ecrire un script bash `lsrec.sh` qui affiche récursivement la liste des sous-répertoires (et pas les fichiers réguliers) en indentant cette liste afin de la représenter de manière arborescente. On utilisera une fonction récursive **parcours** qui se rappellera pour chacun des sous-répertoires de l'argument courant en utilisant un second paramètre indent qui grossira au fur et à mesure ...

```
Exemples$ lsrec.sh ..
../Exemples
  ../Exemples/Bidon2
  ../Exemples/USI
    ../Exemples/USI/Fable
    ../Exemples/USI/Script
  ../Exemples/bidon
```

4.2 Langage C : source, objet, compilation

Exercice 47 (TD)
Quelle différence existe-t-il entre un interpréteur et un compilateur ? Citez deux exemples pour chacun d'entre eux.

Exercice 48 (TD)
Quelles similitudes et quelles différences existe-t-il entre un fichier d'en-tête (toto.h) et une bibliothèque ? Quand les utilise-t-on dans le processus de compilation ? Citez deux exemples pour chacun d'entre eux.

Exercice 49 (TD)
Quelle différence entre l'inclusion d'un fichier d'entête standard tel que `stdio.h, stdlib.h, ctype.h` et un fichier d'entête personnel `monprog.h` ?

Exercice 50 (TD)
Comment les paramètres de la ligne de commande sont-ils passés à un programme C ? Comment l'environnement du processus (variables d'environnement telles que PATH, HOME, ...) est-il atteignable par un programme C ?

Exercice 51 (TD/TP (point d'entrée d'un programme C))
On souhaite écrire un programme affichant le nombre de paramètres passés à la ligne de commande, ces paramètres, ainsi que les variables d'environnement.

1. Ecrire l'algorithme de ce programme.
2. Ecrire le programme C correspondant.

Exercice 52 (TP (gcc))
A l'aide du programme précédent, testez les possibilités du compilateur `gcc` en produisant :
— le fichier prétraité ;
— le fichier assembleur ;
— le fichier objet ;
Observez ces différents fichiers.

Exercice 53 (TD/TP (scanf))
On souhaite réaliser le calcul de la moyenne d'une suite de 5 nombres flottants saisis au clavier.

1. Ecrire l'algorithme de ce programme.
2. Ecrire le programme C correspondant.

Exercice 54 (TD/TP (nombre variable d'arguments))
En utilisant la fonction `atoi` qui convertit une chaîne en entier, réaliser le calcul de la moyenne de la suite des paramètres passés à la ligne de commande : `moy 5 8 12 3`.

1. Ecrire l'algorithme de ce programme.
2. Ecrire le programme C correspondant.

4.3 Types de données, compilation séparée

Exercice 55 (TD/TP (représentation des données))
Soit une chaîne de caractères C contenant un nombre entier positif en représentation décimale. On veut écrire une fonction convertissant cette chaîne en un entier.

1. Ecrire un algorithme itératif de conversion.
2. Ecrire la fonction C correspondante.

Exercice 56 (TD/TP (formattage))
On veut réaliser la conversion inverse de l'exercice précédent. Soit un nombre entier positif que l'on veut convertir en une chaîne de caractères C contenant sa représentation décimale.

1. Ecrire un algorithme de conversion.
2. Ecrire la fonction C correspondante.

Exercice 57 (TD)
Ces algorithmes de conversion (entier vers chaîne, chaîne vers flottant, ...) sont-ils fréquemment employés ? Précisez à quelles occasions.

Exercice 58 (TD/TP (compilation séparée))
Soit la **définition** des fonctions suivantes :
— fonction pair

```c
int pair(unsigned int i){
  if (i==0)
    return 1;
  else
    return impair(i-1);
}
```

— fonction impair

```
int impair(unsigned int i){
  if (i==0)
    return 0;
  else
    return pair(i-1);
}
```

1. Créer un fichier `pair.c` (resp. `impair.c`) qui contienne la définition de la fonction pair (resp. impair). Créer un fichier `pair.h` (resp. `impair.h`) qui contienne la **déclaration** (prototype) de la fonction pair (resp. impair). Créer un programme principal `spair.c` qui réalise l'algorithme suivant :

   ```
   lire l'entier positif n passé à la ligne de commande
   si pair(n) alors
     afficher : n est pair !
   sinon
     afficher : n est impair !
   ```

   Bien entendu, il faut réfléchir au différentes inclusions à réaliser dans les différents fichiers puis compiler les 3 sources .c et lier avec le nom spair et enfin tester ce programme.
2. Ecrire un fichier ppair.c contenant les trois fonctions pair(), impair() et main(). Compiler et lier cet unique fichier en un exécutable ppair. Tester ppair.

Exercice 59 (TD/TP triangle de Pascal)
Le triangle de Pascal est un demi tableau qui contient les nombres de combinaisons possibles pour n objets dans p places. La combinaison de 4 objets dans 2 places est notée C_4^2 et vaut 6. Ce triangle est indicé sur les lignes par n et sur les colonnes par p et est infini :

```
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
...
```

La valeur de combinaison(n,p) : $C_n^p = n!/p!(n-p)!$
On souhaite écrire le programme C qui affiche le triangle jusqu'à la ligne n !

```
$ triangle 4
1
1   1
1   2   1
1   3   3   1
1   4   6   4   1
```

1. Écrire une première version utilisant la fonction factorielle récursive que vous aurez programmé
2. L'affichage étant erroné à partir d'un certain rang, écrire une seconde version en utilisant la programmation dynamique ...On tiendra compte de l'identité remarquable : $\forall p \notin \{0, n\}, C_n^p = C_{n-1}^{p-1} + C_{n-1}^p$
3. Prouvez l'identité remarquable utilisée ...
4. Pourquoi l'affichage devient-il faux dans la première version ? La deuxième version est-elle plus rapide ?

Exercice 60 (TD/TP strsplit (allocation dynamique))
Dans certains fichiers structurés, les articles sont représentés sur une ligne dont les champs sont séparés par un caractère séparateur (fichiers .csv). On souhaite écrire une fonction `char **strsplit(const char *s, const char sep)` qui découpe une chaîne de caractères correspondant à une ligne csv en un tableau de chaînes dynamiques. Par exemple, l'appel à
`strsplit("/bin:/usr/bin:/usr/local/bin",':')` doit générer le tableau suivant :

```
0 --> /bin
1 --> /usr/bin
2 --> /usr/local/bin
3 NULL
```

On écrira également une fonction `main` qui lira la chaîne et le séparateur depuis la ligne de commande.

1. Ecrire l'algorithme de ce programme.
2. Ecrire le programme C correspondant.