Enoncé du TP 5 Système

C. Pain-Barre

INFO - IUT Aix-en-Provence

version du 20/11/2012

ⓘ Démarrer les PC sous Linux.

1 Découpage, recherche, compression et archivage de fichiers

Exercice 1

Utiliser **split** *pour générer des fichiers en morcelant de l'information*

1. Depuis une nouvelle fenêtre terminal, ouvrir une connexion SSH sur allegro en déportant l'affichage graphique avec l'option **-X** :

   ssh -X *etxxxx***@allegro**

   La suite se passe sur allegro.
2. Créer un répertoire `split` dans `tpunix` et s'y placer
3. Découpage de `../des_lignes.txt` par blocs de lignes :
   (a) Découper le fichier `../des_lignes.txt` en fichiers contenant au plus 3 lignes
   (b) Afficher le contenu du répertoire de travail, il devrait y avoir les fichiers `xaa` à `xag`
   (c) Avec ces morceaux, reconstituer le fichier `des_lignes.txt` dans le répertoire de travail
   (d) L'afficher pour vérifier que tout est correct
4. Découpage de `~cpb/public/Firefox_wallpaper.png` par la taille :
   (a) Taper la commande :

   eog ~cpb/public/Firefox_wallpaper.png &

   afin de visualiser l'image sur laquelle on va travailler.

   ⓘ **eog** est une application graphique. Elle ne s'affichera sur votre poste que si la connexion SSH s'est faite avec l'option **-X**, sinon une erreur se produit.

   Cela fait, quitter **eog**.
   (b) Découper le fichier `~cpb/public/Firefox_wallpaper.png` en fichiers dont le nom commence par **`fond`** (à la place de **`x`**) et de taille maximale 100 Kio
   (c) Afficher le contenu du répertoire de travail, il devrait y avoir 9 fichiers
   (d) Avec ces morceaux, reconstituer le fichier d'origine en l'appelant `lefond.png`
   (e) Taper la commande **`eog lefond.png`** pour vérifier que l'image a été correctement reconstituée puis quitter **eog**

(f) Supprimer tous les fichiers du répertoire

5. Découpage de l'entrée standard par blocs de lignes :
   (a) **split** ne découpe qu'un seul fichier, ou éventuellement ce qui est lu sur l'entrée standard. Découper la liste détaillée de tous les processus du système, pour fabriquer des fichiers contenant les informations sur au plus 40 processus. Le nom de ces fichiers doit commencer par **procs_**
   (b) Afficher le contenu du répertoire de travail, il devrait y avoir un certain nombre de fichiers
   (c) Retourner dans `tpunix`
   (d) Supprimer le répertoire `split`

Exercice 2

Utiliser **find** *pour rechercher et manipuler des fichiers vérifiant certaines conditions*

1. Utiliser **find** pour rechercher le **fichier ordinaire** `cigale.txt` dans l'arborescence de votre répertoire d'accueil (pas de travail).
2. Utiliser **find** pour rechercher le **répertoire** `tpunix` dans l'arborescence de votre répertoire personnel (d'accueil).
3. Utiliser **find** pour rechercher tous les **fichiers ordinaires vous appartenant** dans toute l'arborescence du système de fichiers (c'est à dire depuis la racine).

> ⓘ Les fichiers vous appartenant contenus dans le répertoire `/proc` sont des fichiers virtuels informant sur les processus que vous exécutez.

4. Reprendre la question précédente et détourner les sorties de manière à éliminer les messages d'erreur et paginer l'affichage des fichiers trouvés.
5. Utiliser **find** pour rechercher les fichiers ordinaires vous appartenant, contenus dans l'arborescence de votre répertoire personnel, ayant été modifiés il y a moins d'une semaine mais plus d'un jour.
6. Reprendre la question précédente pour stocker les références des fichiers trouvés dans le fichier `hebdo.txt` (à placer dans `tpunix`).
7. Utiliser le manuel en ligne pour étudier le fonctionnement de la commande externe **tee**.

> ⓘ Cette commande tire son nom des raccords en "T" en plomberie. Elle s'insère dans un pipeline de façon transparente, car elle écrit sur sa sortie ce qu'elle lit sur son entrée, comme le ferait **cat**. Mais au passage, elle stocke ce qui passe par elle dans le(s) fichier(s) donné(s) en argument(s).

   Reprendre la question précédente pour, en plus de créer le fichier `hebdo.txt`, utiliser **wc** pour faire afficher le nombre de fichiers trouvés, **le tout en un seul *pipeline***.
8. Avec une seule commande, faire afficher les informations détaillées de chacun des fichiers ordinaires qui se trouvent dans l'arborescence du répertoire d'accueil de l'utilisateur `cpb`, dont le nom commence par **ts_** ou par **algo**, et dont l'extension est **adb** ou **ads**. Pour cela, utiliser le parenthésage, le "ou", les motifs et l'option **-exec** de **find**.
   Remarque : il existe le prédicat (critère) **-ls** de **find** qui, comme **-print**, est toujours vrai et qui exécute **ls -dils** sur le fichier en cours d'analyse. Ici, on lui préfèrera l'option **-exec** pour réaliser un simple **ls -ld** sur chaque fichier trouvé.

9. Reprendre la question précédente mais au lieu d'afficher les informations sur ces fichiers, les copier en mode bavard dans votre répertoire `tpunix`.
10. Rechercher les fichiers ordinaires d'extension `.txt` contenus dans l'arborescence de votre répertoire personnel, et modifier leurs permissions (en mode bavard) en supprimant tous les droits aux membres du groupe et aux autres.
11. Exécuter le programme **`~cpb/public/unix/alim`**. Il crée 4 fichiers parfaitement inutiles (!) dans votre arborescence, et dont le nom se termine par **`~`** ou par **`.bak`**

> ⓘ Cette terminaison dans le nom est typique des fichiers de sauvegarde de version antérieure créés par certains logiciels quand on enregistre un fichier. D'autres extensions sont possibles (`.backup`, `.old`, `.orig`, ...).

12. > **Cette question doit être faite avec soin pour ne pas supprimer tous ses fichiers ! !**

    Rechercher et supprimer en mode bavard avec **demande de confirmation** (sans pour les plus téméraires), les fichiers ordinaires contenus dans l'arborescence de votre répertoire personnel, qui se terminent par **~** ou par **`.bak`**
13. On veut savoir quels sont les terminaux sur lesquels on peut écrire. Ce sont ceux qui nous appartiennent ou ceux dont (au moins) la permission d'écriture est accordée aux autres (ce qui n'est pas vraiment courant...). Les faire rechercher (dans l'arborescence appropriée) par **find**.
14. Consulter le manuel en ligne de **find** pour vous faire une idée du nombre de critères et d'actions disponibles. Étudier en particulier le critère **-writable** puis écrire une commande plus simple répondant à la question précédente.

Exercice 3

Utilisation de **gzip/gunzip** *et de* **zcat**

1. Noter la taille de votre fichier `cigale.txt`
2. Compresser `cigale.txt` avec **gzip** (en mode verbeux). On remarque que le gain indiqué par **gzip** est de l'ordre de 40%. Comparer la taille du fichier `cigale.txt.gz` avec celle de `cigale.txt` (qui n'existe plus)
3. Décompresser `cigale.txt.gz` (pour restituer `cigale.txt`) avec **gunzip**
4. Utiliser **gzip** pour compresser récursivement et en mode verbeux tous les fichiers de l'arborescence de votre répertoire `tpunix`. On remarquera sûrement que pour certains petits fichiers, la compression a eu l'effet inverse (compression > 100%) car les informations de compression dépassent le gain de la compression...
5. Afficher le contenu de `tpunix` et observer que l'opération précédente a remplacé ses fichiers par leur version compressée.
6. Utiliser **zcat** pour regarder le contenu de `cigale.txt.gz` sans créer le fichier décompressé
7. Obtenir le même résultat (et toujours sans créer le fichier décompressé) en utilisant uniquement **gzip**.
8. Utiliser **gzip** ou **gunzip** (au choix) pour décompresser récursivement tous les fichiers de votre arborescence `tpunix`

Exercice 4

Utilisation de **tar**

1. Faire afficher le contenu de l'archive `~cpb/public/unix/images.tar`, sans l'extraire, en utilisant l'action **t** et l'option **f** de **tar**
2. Depuis votre répertoire `tpunix`, extraire tous les fichiers de cette archive. Utiliser l'option **v** afin d'avoir une trace du déroulement de l'extraction. Vous devrez obtenir un répertoire `images` qui contient des images GIF
3. Afficher, sans l'extraire, le contenu de l'archive `~cpb/public/unix/files.tgz` (utiliser l'option **z** de **tar** car l'archive est compressée avec **gzip**)
4. Refaire la question précédente sans utiliser l'option **z** mais en combinant **gzip** (ou **gunzip**) et **tar** dans un *pipeline* : **gzip** doit décompresser l'archive et **tar** doit en afficher le contenu.
5. Supprimer de votre répertoire `tpunix` le répertoire `images` ainsi que les fichiers `images.tar` et `files.tgz` si vous les avez copiés
6. Se placer dans `tp`
7. Créer `tpunix.tgz` comme l'archive compressée (avec **gzip**) de votre répertoire `tpunix`. Utiliser l'option **v** pour faire afficher la progression de l'archivage.
8. Vérifier en faisant afficher le contenu de cette archive (ne pas l'extraire !)
9. Supprimer `tpunix.tgz`

2 Liens et bits set-uid, set-gid, sticky-bit

Exercice 5

Création de liens

1. Se placer dans `tpunix`
2. Créer un lien physique appelé `phys` sur votre fichier `cigale.txt`
3. Créer un lien symbolique appelé `symb` sur votre fichier `cigale.txt`
4. Faire afficher les informations détaillées sur les fichiers de `tpunix`. Remarquer que `cigale.txt` et `phys` ont 2 comme nombre de liens, alors que les autres fichiers ordinaires n'en ont qu'un
5. Faire afficher les informations détaillées sur `symb`. Ce fichier spécial n'a pas de permissions propres : les permissions effectives sont celles du fichier pointé (éventuellement aucune si ce dernier est dans un chemin inaccessible).
6. Refaire afficher les informations détaillées sur `symb` mais en ajoutant l'option **-L** qui demande à déréférencer le lien symbolique afin d'obtenir les informations sur le fichier cible (si c'est possible) et non pas celles du lien symbolique.
7. Modifier (avec **vi**) la première ligne de `cigale.txt`
8. Afficher `phys` et `symb` : ils ont aussi changé
9. Supprimer `cigale.txt`
10. Tenter d'afficher le contenu de `symb`. Cela échoue car ce lien pointe désormais sur un (nom de) fichier inexistant. En revanche, afficher `phys` doit réussir (il contient la version modifiée de `cigale.txt`)

11. Renommer `phys` en `cigale.txt`. Cette fois, on doit pouvoir à nouveau afficher le contenu de `symb`

Exercice 6

Bits set-uid *et* set-gid *(sur les fichiers)*

1. Afficher les informations détaillées sur les fichiers `infos0`, `infos1`, `infos2` et `infos3` contenus dans `~cpb/public/unix` et observer la présence éventuelle des bits *set-uid* et/ou *set-gid* dans leurs permissions.
2. Exécuter chacun de ces fichiers. Remarquer la différence qu'il peut exister entre les utilisateurs réel et effectif et les groupes réel et effectif
3. Exécuter **infos3** avec l'option **-c**. Il crée alors un fichier dans `/tmp` dont le nom est affiché et suit le format `traceinfos_`*pid*`.txt` où *pid* est le PID du processus. Afficher les informations détaillées sur ce fichier. Remarquer que son propriétaire et son groupe sont ceux effectifs du processus créateur. Noter le nom du fichier.

   > ⓘ Vous ne pouvez pas supprimer ce fichier car vous n'en êtes pas propriétaires ! Nous reviendrons sur cette question à l'exercice suivant.

4. Copier `infos3` dans votre répertoire `public`. Remarquer que votre copie ne possède pas les bits *set-uid* et *set-gid* (du moins, en principe ;-))
5. Modifier les permissions de `infos3` pour que le propriétaire et les autres aient le droit d'exécution mais pas les membres du groupe. Ne pas positionner les bits spéciaux.
6. L'exécuter pour s'assurer qu'il se lance bien. Puis, le faire exécuter par un voisin qui n'est pas membre de votre groupe et observer que les identités (utilisateur et groupe) effectives restent celles réelles de l'exécutant
7. Activer le bit *set-gid* de `infos3`. Vérifier en affichant les informations détaillées sur le fichier.
8. Le faire à nouveau exécuter par son voisin. On doit remarquer qu'à ce stade, l'activation de ce bit n'a pas d'effet...
9. Pour que le bit *set-gid* soit effectif, il faut aussi le droit d'exécution pour le groupe. Ajouter ce droit à `infos3`. Vérifier en demandant les informations détaillées sur le fichier.
10. Le faire exécuter par son voisin, avec l'option **-c**. Cette fois, le processus garde l'identité du voisin comme effective mais son groupe effectif est le vôtre. Observer que le groupe du fichier créé est bien le vôtre. Cela vous donne probablement certains droits sur ce fichier (pas forcément plus qu'un utilisateur quelconque), mais toujours pas le droit de le supprimer. Demander au voisin de le supprimer. Lui, le peut.
11. Modifier `infos3` pour ajouter le bit *set-uid*, mais supprimer les droits d'exécution au propriétaire. Vérifier en demandant les informations détaillées sur le fichier.
12. Le faire exécuter par son voisin, avec l'option **-c**. Cette fois, le processus a votre identité et votre groupe comme effectifs. Observer que le fichier créé vous appartient et a votre groupe. Le supprimer (ce que le voisin ne peut faire).

    > ✍ On note au passage que le bit *set-uid* suffit pour changer l'utilisateur effectif, même si le propriétaire n'a pas les permissions d'exécution, alors que le changement de groupe effectif nécessite à la fois le bit *set-gid* et le droit d'exécution pour le groupe...

13. Supprimer `infos3`.

Exercice 7

Quelques utilitaires set-uid *et/ou* set-gid

1. De nombreux utilitaires du système sont *set-uid* et/ou *set-gid*. Par exemple :
   - **chfn** et **chsh** sont *set-uid* (`root`) car ils doivent modifier le fichier `/etc/passwd` ;
   - **passwd** est *set-uid* (`root`) car il modifie `/etc/shadow` ;
   - **gpasswd** est *set-uid* (`root`) car il modifie `/etc/group` et/ou `/etc/gshadow` ;
   - **su** est *set-uid* (`root`) afin de permettre d'exécuter un programme avec l'identité d'un autre utilisateur ;
   - **write** est *set-gid* (`tty`) afin de permettre à un utilisateur d'afficher un message sur le terminal d'un autre utilisateur ;
   - etc.

   Utiliser **type** pour savoir où se trouvent les utilitaires mentionnés ci-dessus, et afficher les informations détaillées sur chacun d'eux afin de vérifier l'activation de ces bits. Utiliser l'option **-L** de **ls** car certains sont des liens symboliques.
2. Ouvrir un nouveau terminal sur le PC et se connecter à nouveau sur allegro avec **ssh**.
3. Sur cette nouvelle connexion, taper **`mesg y`**
4. Afficher les informations détaillées sur le terminal utilisé sur cette connexion. Remarquer que le droit d'écriture est accordé au groupe (`tty`) sur ce terminal.
5. Demander à un voisin de taper (sur allegro) **`write etxxxx`** où `etxxxx` est votre nom d'utilisateur, puis de saisir quelques lignes de texte, en terminant par CTRL-**D**. Remarquer que le texte tapé s'affiche sur votre terminal accessible en écriture au groupe.
6. Terminer cette connexion SSH puis fermer ce terminal.

Exercice 8

Observation de l'utilité du sticky-bit *et du bit* set-gid *sur les répertoires*

1. Faire afficher les informations détaillées sur le répertoire `/tmp`. Remarquer que ses permissions sont totales pour tout le monde, et que son *sticky-bit* est activé.
2. Le *sticky-bit* protège le contenu du répertoire. Essayer de supprimer le fichier `traceinfos_`*pid*`.txt` créé au début de l'exercice précédent. Malgré vos droits sur le répertoire, la suppression échoue car le fichier ne vous appartient pas !
3. Demander les informations détaillées sur le répertoire `/tmp/depot`. Il n'a pas de *sticky-bit*
4. Créer un fichier ordinaire (quelconque, même vide) `/tmp/etxxxx` où `etxxxx` correspond à votre nom d'utilisateur. De même créer un autre fichier ordinaire `/tmp/depot/etxxxx`
5. Demander à son voisin de (tenter de) les supprimer, en confirmant la suppression (**y**) si c'est demandé. Noter que seul `/tmp/etxxxx` résiste...
6. Supprimer votre fichier `/tmp/etxxxx`
7. Créer un répertoire `/tmp/etxxxx` où `etxxxx` correspond à votre nom d'utilisateur. Donner tous les droits pour tout monde sur ce répertoire et activer son *sticky-bit*. Vérifier les droits.

8. Copier votre fichier `~/tp/tpunix/amphigouri.txt` dans (votre) `/tmp/etxxxx`
9. Demander à votre voisin de créer les fichiers `fic1` et `fic2` dans votre répertoire `/tmp/etxxxx`
10. Afficher le contenu détaillé de votre `/tmp/etxxxx`. Le propriétaire et le groupe de ces fichiers sont ceux de votre voisin.
11. Activer le bit *set-gid* sur votre répertoire `/tmp/etxxxx`. Vérifier ses permissions.
12. Demander à votre voisin de créer un fichier `fic3` et un répertoire `rep` dans votre répertoire `/tmp/etxxxx`
13. Afficher les informations détaillées sur le contenu de `/tmp/`*nnnppp* : le groupe de `fic3` et de `rep` devrait être le vôtre ! De plus, `rep` devrait lui aussi avoir le bit *set-gid* activé

> ⓘ Le bit *set-gid* sur un répertoire est très pratique pour le travail collaboratif sur un projet dans un répertoire commun.

14. Demander au voisin de supprimer le fichier `amphigouri.txt` de votre répertoire `/tmp/etxxxx`. Il ne le peut pas à cause du *sticky-bit*.
15. Lui demander de supprimer le fichier `fic1` de votre répertoire `/tmp/etxxxx`. En tant que propriétaire du fichier, il en a le droit.
16. Supprimer de votre répertoire `/tmp/etxxxx` le fichier `fic2` du voisin. En tant que propriétaire du répertoire, vous en avez le droit !
17. Supprimer entièrement votre répertoire `/tmp/etxxxx`

> 💣 **N'oubliez pas que vous êtes administrateur du groupe qui porte votre nom d'utilisateur (sur** allegro**). Vérifiez les membres de votre groupe en consultant le fichier** `/etc/group`**, et vérifiez les permissions de vos fichiers et répertoires !**

> ✍ En principe, au cours du TP 4 vous avez créé puis supprimé le répertoire `~/public/groupe`, sur lequel tous les membres du groupe avaient tous les droits. Si vous avez recréé un répertoire de ce type parce qu'il vous est utile, il est certainement plus judicieux de le doter du *sticky-bit*, et peut-être aussi du bit *set-gid*...

3 Variables et tableaux

Exercice 9

Manipulation de variables

1. Créer une variable **`var1`** contenant **`3`**
2. Utiliser **echo** pour faire afficher le contenu de **`var1`**
3. Créer une variable **`var2`** contenant exactement "**`une␣␣␣suite␣␣de␣␣␣caractères`**" (respecter les espaces)
4. Utiliser **echo** pour faire afficher le **contenu exact** (y compris les espaces) de **`var2`**

5. Créer une variable **var3** contenant le résultat de la multiplication de **var1** par 4
6. Faire afficher le contenu de **var3**
7. Modifier **var3** pour qu'elle contienne le résultat du ET bit à bit entre sa valeur et 7
8. Faire afficher le contenu de **var3** (qui doit valoir 4)
9. Supprimer la variable **var3**
10. Faire afficher le contenu de **var3** (cela doit écrire une ligne vide)

Exercice 10

Manipulation de quelques variables prédéfinies

1. Faire afficher le contenu de la variable **USER**

> ✍ En principe, cette variable contient la même chose que ce qu'écrit la commande **whoami**, et son utilisation est parfois plus pratique que d'exécuter **whoami**. On s'en sert principalement dans des programmes bash, appelés des scripts bash, quand on doit exploiter le nom de l'utilisateur exécutant le programme.

2. Faire afficher le contenu de la variable **UID** qui contient votre numéro d'utilisateur.
3. Faire afficher le contenu de la variable **HOME**

> ⓘ Pour développer (remplacer) le tilde dans les chemins ~ (seul) ou ~/..., bash utilise le contenu de **HOME**. Elle est aussi utilisée comme défaut pour **cd** quand elle est appelée sans argument. À nouveau, c'est une variable parfois employée dans les programmes bash, notamment là où le développement du tilde n'est pas possible.

4. Se placer dans `/etc` **puis** se placer (directement) dans `tpunix`
5. Faire afficher le contenu de la variable **PWD** (qui contient le chemin du répertoire de travail, soit `/home/etxxxx/tp/tpunix` pour l'utilisateur *etxxxx* sur allegro)

> ⓘ Cette variable est mise à jour par la commande interne **cd**, et contient la même chose que ce qu'écrit la commande **pwd**.

6. Faire afficher le contenu de la variable **OLDPWD** (qui contient le chemin du répertoire de travail précédent, soit `/etc`)

> ⓘ Cette variable est mise à jour par la commande interne **cd**, qui s'en sert quand son argument est **-** (tiret).

Les manipulations qui suivent ont un intérêt pratique très limité ! On les effectue ici pour montrer que bash ne fait pas que mettre à jour certaines variables. Il s'en sert aussi :

7. Affecter la valeur **/etc** à la variable **HOME**. Remarquer que le chemin affiché dans le prompt est différent (le **~** n'apparaît pas)
8. Taper **cd ~/init.d**. Le tilde réapparaît dans le prompt.
9. Taper **pwd**. Cela devrait afficher `/etc/init.d`. En effet, bash utilise le contenu de **HOME** pour développer le tilde seul. Ainsi, `~/init.d` est équivalent à `$HOME/init.d` (soit, dans notre cas, `/etc/init.d`)
10. Taper **cd** seule (équivalente à **cd ~** ou **cd "$HOME"**)
11. Taper **pwd**. Cela devrait afficher `/etc`
12. Restituer à **HOME** sa valeur normale.
13. Affecter à **OLDPWD** la valeur **~/tp/tpunix**
14. Taper **cd -** (équivalente à **cd "$OLDPWD"**)
15. Taper **pwd**. Le répertoire de travail devrait être `tpunix`
16. Afficher à nouveau le contenu de **OLDPWD**, ce devrait être `/etc`

Exercice 11

Utilisation des substitutions étendues de variables

Se reporter à la partie de cours sur les substitutions étendues de variables afin de :

1. Faire afficher le nombre de caractères contenus dans **var2** (sans utiliser **wc**). Cela devrait afficher 28.
2. Ne faire afficher de **var2** que la sous-chaîne de 5 caractères à partir du 6e. Cela devrait afficher **suite**
3. Créer une variable **reffic** contenant la chaîne `/home/etxxxx/tp/tpunix/cigale.txt` où `etxxxx` est votre nom d'utilisateur
4. Taper **cat $reffic**. Cela devrait afficher le contenu du fichier `cigale.txt`
5. On suppose maintenant que `reffic` est un chemin générique, absolu ou relatif, d'un fichier comportant une extension. Sa valeur actuelle est une possibilité, mais elle pourrait contenir d'autres répertoires, et ne pas commencer par `/`. Appliquer la substitution avec motifs sur **reffic** pour ne faire afficher :
   (a) que le chemin du fichier. Pour la valeur actuelle de **reffic**, cela doit afficher `/home/etxxxx/tp/tpunix`
   (b) que le nom du fichier. Pour la valeur actuelle de **reffic**, cela doit afficher `cigale.txt`
   (c) que l'extension du fichier (sans le point). Pour la valeur actuelle de **reffic**, cela doit afficher `txt`
   (d) que la racine du fichier, c'est à dire qu'il faut tout afficher sauf l'extension du fichier. Pour la valeur actuelle de **reffic**, cela doit afficher `/home/etxxxx/tp/tpunix/cigale`
6. En utilisant **reffic** et la substitution précédente, créer une copie de `cigale.txt` dans `tpunix` en la nommant `cigale.old`, sans utiliser explicitement la chaîne **cigale**
7. En utilisant **reffic** et la substitution de remplacement pour remplacer **cigale** par **fff**, créer une copie de `cigale.txt` nommée `fff.txt`

Les substitutions qui suivent sont surtout utilisées dans des programmes bash, lorsqu'on doit tester et utiliser la valeur d'une variable ou d'un paramètre :

8. En principe, la variable **var3** n'existe plus. Taper **echo ${var3:-inexistante}** qui doit afficher **inexistante** car **var3** n'exsite pas.
9. Taper **echo ${var3:=inexistante}** qui produit le même résultat, mais en plus affecte à **var3** la chaîne **inexistante**
10. Vérifier en faisant afficher le contenu de **var3**