Sujet de TP - Shell

Remarques préliminaires

Les problèmes et oublis constatés dans ce support sont à signaler à l'adresse `clot@univ-lyon1.fr`.

La philosophie des systèmes Unix repose sur une idée simple : la puissance du système prend plus ses racines dans le fait que les programmes peuvent communiquer entre eux que dans les programmes eux-mêmes. La capacité de rediriger les flux associés aux programmes est une caractéristique du shell à partir duquel les programmes sont invoqués. Le shell joue le rôle du ciment permettant l'assemblage des programmes basiques, pour la plupart, en programme plus sophistiqués. C'est dans cette optique que «*Small is beautiful*» prend son sens : on attend d'un programme qu'il rende un service de la façon la plus simple. L'utilisation de l'information qu'il produit ne le concerne pas.

Dans une première partie, nous observerons les différents mécanismes d'interprétation du shell (tel que bash ou ksh) sur des commandes simples puis nous présenterons des commandes plus élaborées reposant sur les diverses structures de contrôle propres au shell.

Commandes simples

Dans cette section, nous nous bornerons à l'emploi de commandes simples faisant appels aux différents mécanismes du shell précédant l'exécution de la commande construite :

1. alias
2. développement des accolades
3. développement des caractères tilde (~)
4. développements de paramètres et variables
5. substitution de commandes
6. développements arithmétiques
7. découpage des champs
8. développement des noms de fichiers
9. traitements des quotes

Ces mécanismes prennent place dans l'ordre d'énumération ci-dessus, ce qui a évidemment une grande importance.

alias

La commande `alias` permet de définir une association entre une chaîne de caractère et une commande.

- Exécutez la commande `echo "DEBUT"`. Ceci trouvera son utilité plus loin.
- Dans quelle page du manuel trouve-t-on de l'aide pour l'emploi de cette commande ? Quelle commande permet de supprimer un alias ?
- Utilisez la commande `alias` afin d'obtenir la liste des alias définis dans votre environnement

- Définissez un alias associant la chaîne `o` au pager `less`. Testez ce nouvel alias.
- Définissez un alias permettant de lister tous les fichiers, répertoire et autres types de fichiers du répertoire courant. Cet alias utilisera la chaîne `W`. Testez l'alias.
- Après avoir identifié l'option de `ls` permettant d'indiquer la nature d'un fichier, modifiez l'alias `W` afin qu'il invoque cette option. Testez l'alias.
- Définissez un alias nommé `m1` permettant de réaliser une connexion ssh vers `minisfa` utilisant votre login. Nous verrons plus tard comment définir cet alias afin qu'il soit actif lors de chaque ouverture de session.
- Définissez un alias nommé `wwwm1` permettant d'ouvrir le navigateur `w3m` sur la page `isfaserveur¬.univ-lyon1.fr/~denis.clot/IRM1.html`.

Développement des accolades

Le mécanisme d'expansion d'accolades permet la création de chaînes quelconques selon le principe suivant : `PREF_{B,C,D}_SUF` est développé en `PREF_B_SUF PREF_C_SUF PREF_D_SUF`, où le préfixe `PREF_` et le suffixe `_SUF` sont optionnels.

- Utilisez ce mécanisme pour créer les fichiers `file1`, `file2`, `file3`.
- Utilisez ce mécanisme pour construire la chaine `00 01 02 03 04 05 .. 99`
- Générez les 256 premiers entiers en notation binaire.

Caractère tilde (~)

- Affichez le chemin de votre «home ».
- Affichez le chemin du «home » de votre voisin.
- Supposez que les logins `user1` et `user2` soient valides. Comment se développe la commande suivante : `echo ~user{1,2}`. Et si `user2` n'est pas un login valide ?

Développement des paramètres et des variables

Le shell permet de stocker des informations dans des variables sous forme de chaîne de caratères. Pour accéder aux informations stockées dans une variable nommée `VAR`, il suffit de référencer son contenu par `$VAR` ou `${VAR}`. Toutefois le premier mécanisme de référencement est à banir car il conduit à des effets de bord.

Le tableau ci-dessous présente quelques substitutions réalisables à partir d'une variable `VAR` :

Invocation	Action résumée	Détail
`${VAR}`	utilisation de la valeur de `VAR`	est substitué par la valeur de `VAR`
`${VAR:-val}`	utilisation d'une valeur par défaut	est substitué par la valeur de `VAR` si différente de la chaîne vide ou par `val` sinon.
`${VAR:=val}`	attribution d'une valeur par défaut	est substitué par la valeur de `VAR` si différente de la chaîne vide ou par `val` sinon. Dans ce dernier cas, `VAR` reçoit la valeur `val`.
`${VAR:?val}`	test de non nullité	produit une erreur si `VAR` n'est pas définie ou contient une chaîne vide en utilisant `val` pour construire le message d'erreur. Sinon, utilise la valeur de `VAR`.
`${VAR:+val}`	utilisation d'une valeur alternative	si `VAR` est non vide, utilise la valeur `val`.
`${VAR:n1}`	extraction de sous-chaîne	est substitué par `VAR` dont les `n1` premiers caractères ont été supprimés.
`${VAR:n1:n2}`	extraction de sous-chaîne	idem, mais seuls les `n2` caractères suivants sont conservés.
`${#VAR}`	longueur de la valeur de `VAR`	est substitué par la longueur de la valeur de `VAR`

- Stockez dans une variable `V` la chaîne `val`. A l'aide de la commande `echo`, produisez la chaîne «`La variable V contient >valeurDeV<`» où `valeurDeV` est substitué par la valeur de `V`.
- Stockez dans une variable `F` la chaîne `f` et dans une variable `F1` la chaîne `g`. Affichez leur valeurs avec la commande `echo`. En utilisant la variable `F`, testez les deux référencements possibles pour afficher la chaîne `f1`.
- Définissez une variable `OLD_PAGER` qui prend la valeur de la variable `PAGER` et affectez à `PAGER` la valeur "`less`". Affichez la valeur des deux variables, puis définissez une affection de sorte que `PAGER` retrouve sa valeur initiale si elle était non nulle.
- Faites en sorte que `PAGER` reçoive la valeur "`more`" s'il n'est pas nul et la valeur "`less`" sinon. Vous pouvez utiliser deux affectations successives...
- Placez dans la variable `MAX` un entier de votre choix et définissez `NB_CHF_MAX` de sorte qu'elle contienne le nombre de chiffres de `MAX`.
- Placez dans la variable `ME` votre login et exécutez la commande `echo ~${ME}`. Comment expliquez-vous le résultat ?

D'autres substitutions permettent de manipuler le contenu de la variable `VAR` par rapport à un motif donné. Le motif est l'analogue des expressions régulières dans le cadre du shell. Les deux tableaux ci-dessous présentent les constructions particulières permettant de définir un motif et le suivant présente les dernières opérations relatives aux variables du shell.

Symbole	Correspondance
`*`	N'importe quelle chaîne, y compris la chaîne vide
`?`	N'importe quel caractère
`[..]`	Comme pour les expressions régulières...Toutefois, le caractère `^` est remplacé par `!` pour l'inversion.
`mot1|mot2|..|motN`	motif `mot1` ou motif `mot2` ou...motN. Ceci permet de définir une liste de motifs. Par convention, une liste de motif peut comporter un seul motif.

Les opérateurs ci-dessous apportent un niveau de description plus détaillé. Ils sont toujours utilisables avec `ksh` mais `bash` nécessite l'activation de l'option `extglob` par la commande `shopt` (cf. manuel) pour leur emploi.

Symbole	Correspondance
`?(liste-motif)`	Correspond à zéro ou une occurence d'un des motifs de `liste-motif`
`*(liste-motif)`	Correspond à zéro ou plusieurs occurences...
`+(liste-motif)`	Correspond à une ou plusieurs occurences...
`@(liste-motif)`	Correspont à une occurence d'un des motifs...
`!(liste-motif)`	Correspond à tout sauf aux motifs de `liste-motif`

Invocation	Action résumée	Détail
`${VAR#motif}`	suppression du préfixe motif	est substitué par la valeur de `VAR` de laquelle est retirée la plus petite concordance du motif considéré comme préfixe.
`${VAR##motif}`	suppression du préfixe motif	est substitué par la valeur de `VAR` de laquelle est retirée la plus longue concordance du motif considéré comme préfixe.
`${VAR%motif}`	suppression du suffixe motif	est substitué par la valeur de `VAR` de laquelle est retirée la plus petite concordance du motif considéré comme suffixe.
`${VAR%%motif}`	suppression du suffixe motif	est substitué par la valeur de `VAR` de laquelle est retirée la plus longue concordance du motif considéré comme suffixe.
`${VAR/motif/rpl}`	`bash` uniquement ! remplacement de la première occurence du motif par rpl	est substitué par la valeur de `VAR` dans laquelle la première occurence de `motif` est remplacée par la chaîne `rpl`
`${VAR//motif/rpl}`	`bash` uniquement ! remplacement de toute les occurences du motif par rpl	idem, mais toutes les occurences de `motif` sont remplacées.

- Placez dans la variable `HOME_PART` le chemin vers votre «home directory». A partir de cette nouvelle variable, affichez le répertoire père de ce répertoire. Vous proposerez au moins deux façons différentes d'y parvenir.
- Dans le shell `bash`, activez l'option `extglob` afin de pouvoir utiliser les opérateurs étendus.
- Trouvez une nouvelle façon d'affichez le répertoire père en utilisant les opérateurs supplémentaires.
- Affichez le nom du répertoire courant en utilisant la variable `PWD` i.e. sans le chemin complet à ce répertoire.
- Certains fichiers comportent des extentions dans leur nom et d'autres n'en comportent pas. Etant donné un nom de fichier stocké dans la variable `FILE`, définissez une variable `EXT` qui recevra l'extention du nom d'un fichier. Si le nom ne comporte pas d'extention, `EXT` devra contenir la chaîne vide. Vous affecterez à `FILE` les chaînes appropriées permettant de réaliser des tests.

Substitution de commandes

- Utilisez la commande `whoami` pour vous envoyer un mail de sujet `Automail`.
- Utilisez la variable `PRIVATE_MAIL` pour stocker votre adresse mail personelle et construisez une commande permettant d'envoyer un mail pour destination l'adresse contenue dans `PERSO_MAIL` et de sujet `"MEMO - x_DATE_x"` où `x_DATE_x` est à remplacer par une chaîne donnant dans l'ordre l'année, le mois, le jour du mois, l'heure, les minutes et secondes séparées par des caractères «`:`».
- Construisez l'alias `memo` permettant d'exécuter la commande ci-dessus.
- Parmi les fichiers de votre répertoire courant ainsi que ceux de votre `~` , cherchez avec `grep` la chaîne "`alias`" dans ceux qui ont été accédés depuis moins de 120 minutes.
- Dans votre répertoire courant, créez un fichier nommé "`VotreNom VotrePrenom`" (où `VotreNom` est votre nom...). Assurez-vous d'avoir créé un seul fichier et non deux nommés `VotreNom` et `VotrePrenom`. Ajoutez la ligne `alias` avec la commande `echo`. Relancez la commande ci-dessus et observez l'erreur produite par `grep`. Comment expliquez-vous l'erreur produite par `grep` ?

Développement arithmétique

Rappelons que les calculs sont réalisés sur des entiers longs. Ces derniers peuvent être exprimés en base n, où $2 \leq n \leq 36$. Il suffit pour cela de faire précéder le nombre de `n#` où `n` est la base (exprimée en base décimale :o). Par défaut, la base décimale est utilisée et il n'est pas nécessaire de la préciser

- Affichez la chaîne "`Il est HH:MM. Il reste XX minutes avant 12h. (x_X)`" où `HH:MM` est remplacé par l'heure courante et `XX` est le nombre de minutes avant 12h.
- Affichez la valeur de $(!5 - 5^2) \mod 5$ (où ! représente l'opérateur factoriel).
- Placez dans la variable `N` la chaîne `2#10` et utilisez-la pour afficher la chaîne : `Il existe X sortes de gens : ceux qui savent lire le binaire et les autres.` où `X` est la valeur de `N`.
- Stockez une valeur dans `A` et une autre dans `B` et affichez la chaîne : "`Le minimum entre valDeA et valDeB est Min.`" où `valDeA` est la valeur de `A` et `Min` est la valeur minimum entre `A` et `B`.

Découpage des champs

Le découpage des champs (ou mots) concerne les résultats des développements des paramètres, des évaluations arithmétiques et des substitutions de commandes lorsqu'ils ne sont pas protégés par des doubles quotes. Par défaut, les caractères délimitant les champs sont les caractère d'espace, de tabulation et de retour-chariot.

- Exécutez les commandes suivantes : `touch "AA BB";ls $(echo AA*BB);ls "$(echo AA*BB)"`. Comment expliquez-vous l'erreur engendrée par la première instance de `ls` ?
- Exécutez les commandes suivantes : `touch "AA BB" "A B";ls "$(echo A*B)"`. Comment expliquez-vous l'erreur engendrée ? Pensez-vous pouvoir trouver une solution sans abandonner l'obtention des deux noms de fichiers par substitution de commande ?

Développement des noms de fichiers

- Créez les fichiers `file.dvi`, `file.ps`, `file1.dvi`, `file1.ps`, `file2.dvi`, `file2.ps`. Listez tous les fichiers, puis tous les fichiers d'extension `.ps`. Puis listez tous les fichiers comportant un chiffre 1 ou 2 et les fichiers d'extension `.ps`. Listez les fichiers qui ne sont pas d'extension `.ps`. Pour finir, listez les fichiers qui ne comportent ni chiffre, ni d'extension `.ps`.
- Expliquez pourquoi le fait de faire figurer dans un nom de fichier l'un des caractères `*`, `'`, `?` comporte des risques.
- D'après les documents de cours, les mécanismes de découpage en mot et de développement de noms de fichiers ne concernent pas les affectations de variables. Réalisez l'affectation `F=*` puis trouvez un moyen de tester si la valeur de `F` est toujours `*` ou si ce caractère a été développé en la liste de fichiers du répertoire courant.

Traitement des quotes

Toutes les occurences non-protégées des caractères `'`, `"` et `\` et qui ne résultent pas de substitution ou développement sont retirées.

- Exécutez les commandes `echo "A B";echo $(echo \"A B\")` et comparez les résultats obtenus. Comment s'explique la différence ?
- Affichez la chaîne `'"\`.
- Affichez la chaîne `Saisissez la commande R=${PWD} puis echo "Test de \${R}=${R}"`

Et les redirections !!

- Appliquer la commande `ls` récursivement aux répertoires `~` et `~root` dont vous aurez redirigé la sortie standard vers le fichier `LS_OUT` et la sortie d'erreur `LS_ERR`. Contrôlez le contenu des deux fichiers créés.
- Ajouter la ligne `STDIN -- FIN` (resp. `STDERR -- FIN`) au fichier `LS_OUT` (resp. `LS_ERR`) en redirigeant la sortie de la commande `echo`. Notez bien qu'il s'agit d'ajouter des choses aux fichiers mentionnés et non de les écraser !

- Appliquez la commande `ls` comme ci-dessus et redirigez-la vers le pager `less`. Les erreurs qui apparaissent à l'écran n'ont pas été transmises à `less` comme vous le savez. Un rafraîssement d'écran permet de s'en débarasser. Procedez à une redirection de façon à transmettre également les erreurs à `less`.
- Un flux redirigé vers le fichier `/dev/null` est perdu. Procédez à une redirection de sorte que seule la sortie d'erreur soit transmise à `less`.
- Procédez à une intervertion des flux de sortie de sorte qu'un rafraichissement dans `less` permette de ne conserver que les messages d'erreurs.
- Envoyez-vous un mail dont le corps du message sera le fichier `LS_ERR`. Vous pourrez utiliser votre alias memo.
- Envoyez-vous un mail dont le corps sera alimenté par vous jusqu'à ce qu'il rencontre la chaîne `FIN_`.

Commandes composées

Pipelines

- En utilisant en particulier la commande `wc`, affichez le nombre de fichiers présents dans le répertoire courant. Vous pourrez utiliser cette commande pour générer la chaïne "`N fichiers sont presents dans le repertoire REP`" où `N` (resp. `REP`) est le nombre de fichiers (resp. nom du repertoire courant).
- Affichez la chaîne "`Le repertoire REP contient X entrees au total, dont NF fichiers, NR repertoires et NL liens symboliques.`".
- En une ligne de commande, transferez-vous par mail toutes les lignes de l'historique de votre session relative à la commande `alias` et aux affectations de variables. Les lignes formant le corps de votre mail devront isoler les lignes relatives à `alias` des autres et la totalité devra être triée selon l'ordre lexicographique (cmd `sort`). La commande `history` pour permettra d'obtenir l'historique de vos commandes. Vous ne filtrerez que les lignes suivants la ligne `echo "DEBUT"` engendrée par cette commande en début de session.
- Filtrez les lignes fournies par `du` appliquée toutes les entrées du répertoire (tous les fichiers et répertoires qu'il contient) courant de sorte à obtenir ces lignes triées selon l'ordre décroissant de la taille des fichiers, la taille de chaque fichier étant exprimée avec les unités appropriées (en octets si la taille est inférieure à 1024 octets, en kilo-octets si la taille est inférieure à 1024^2 octets, etc).

Exécutions conditionnelles

- L'aboutissement de la commande `rmdir`, permettant la suppression d'un répertoire, dépend de la présence de fichier dans ce répertoire. Créez un répertoire et ainsi qu'un fichier à l'intérieur de celui-ci. Utilisez les opérateurs d'exécution conditionnelle de sorte que le succès soit signalé par l'affichage de "`suppression effectuee`" et que l'echec provoque l'affichage du message `Le repertoire contient les fichiers ...` avec la liste des fichiers contenu dans le répertoire.
- A l'aide des opérateurs d'exécution conditionnelle, construisez une commande qui affiche la chaîne "`tout seul`" si votre login est le seul utilisé pour les connexions en cours et "`pas seul`" sinon.
- Modifiez votre commande afin que le message "`tout seul`" soit remplacé par un message du type "`login1 login2 ... connectes`", où `login1 login2` sont les autres logins en cours d'utilisation. `awk` pourra être considéré comme moyen de construction de cette chaîne.
- Construisez une commande permettant d'afficher la chaîne "`Encore XX minutes avant la fin`" s'il reste plus de 15 minutes avant la fin de la séance et `"Moins de XX minutes avant la`

`deconnexion!"` s'il reste moins de 15 minutes avant la fin de la séance. Le test pourra se baser sur le résultat d'une évaluation arithmétique.


Sur `minisfa`, construisez une commande permettant de tester l'existence dans le répertoire courant d'un répertoire `NomPrenom` où `NomPrenom` résulte de la concaténation de vos nom et prénom et qui crée ce répertoire s'il n'existe pas dans le répertoire courant.

Boucles et autres structures de contrôles

Nous allons considérer les structures de contrôle suivantes :
— `if liste ; then liste ; [ elif liste ; then liste ; ] ... [ else liste ; ] fi`
— `for nom [ in mot ] ; do liste ; done`
— `case mot in [ motif [ | motif ] ... ) liste ;; ] ... esac`
— `while liste ; do liste ; done`

Les détails sont consultables sur les pages du manuel du bash et du ksh.


Supposez que la variable `FILE` contienne le nom d'un fichier compressé. Plusieurs programmes de compression sont courants : `zip`, `gzip`, `bzip2` (les commandes de décompression associées sont `unzip`, `gunzip` et `bunzip2`). Les extensions de fichiers associées sont `zip`, `gz` et `bz2`. Utilisez une structure comportant des `if...` pour permettre l'usage du programme approprié pour la décompression du fichier dont le nom est `${FILE}`. Vous pourrez baser l'analyse sur l'extension du nom de fichier [1].


Proposez une commande similaire basée sur l'usage d'un `case`.


L'instanciation de la variable `RANDOM` permet d'obtenir des valeurs entières pseudo-aléatoires. Utilisez cette variable pour générer dans votre répertoire `NomPrenomParam` une dizaine de fichiers préfixés par votre nom et suffixé par une valeur aléatoire. Vous utiliserez d'abord une boucle `while` puis une boucle `for`. Un délai d'au moins une seconde sera observé entre les créations de fichiers.


Afin de retrouver la trace du temps dans les noms de fichiers créés précédement, vous décidez de préfixer chaque nom de fichier par le numéro correspondant à son ordre de création. Utilisez une boucle permettant de simuler cette procédure de renommage. Par exemple votre boucle pourrait engendrer une suite de chaînes :

```
Martin2654 -> 1-Martin2654
Martin7612 -> 2-Martin7612
...
```


Modifiez votre boucle précédente de sorte que les numéros produits comportent tous le même nombre de chiffres en utilisant un préfixage par des chiffres «0». Ainsi, la suite `1`, `2`, `3`, `4`, `5`, `6`, `7`, `8`, `9`, `10` engendrera la suite `01`, `02`, `03`, `04`, `05`, `06`, `07`, `08`, `09`, `10`. Procédez au renommage réel des fichiers.


Vous décidez de la création d'une nouvelle dizaine de fichiers dans ce même répertoire. Utilisez pour cela la boucle élaborée plus haut. Modifiez la boucle de la question précédente pour renommer vos fichiers. Sans précaution, vous obtiendrez des fichiers nommés

```
01-01-Martin2654
02-02-Martin7612
...
11-Martin1324
..
```

Corrigez cette boucle de sorte à construire une suite de fichier cohérente :

```
01-Martin2654
02-Martin7612
...
11-Martin1324
```

1. Notez que cette façon de procéder est fragile. Nous verrons plus tard l'emploi d'une commande plus fiable pour l'identification des types de fichiers.

..


En combinant un `while` et l'instruction `read`, proposez une boucle qui lit l'entrée standard jusqu'à ce que l'utilisateur saisisse la chaîne "`Clef`". Vous pourrez tester la robustesse de votre commande en lui envoyant un signal de fin de fichier. [2]

Les structures ci-dessous pourront être testées par les curieux :
— `until liste ; do liste ; done`
— `select nom [ in mot ] ; do liste ; done`

liste de commandes

Les `{..}` permettent de construire des commandes groupées. Considérées de l'extérieur du groupe qu'elles forment, les commandes groupées partagent la même entrée, la même sortie et la même sortie d'erreurs.


Former un groupe permettant de générer le flot formé de la chaîne "`START`", suivi des lignes produites par les commandes `w`, `ps -ef` et `free`, et achevée par la chaîne "`STOP`". Vous pourrez observer ce flot de sortie dans votre pager préféré.

Sous-shell

Qu'est-ce qu'un sous-shell ? Un shell exécuté par un shell. Plusieurs façons permettent d'invoquer un nouveau shell dans `bash` et `ksh` : son appel explicite (comme dans `bash -c 'echo "il est $(date)"'` ou `sh fichier_script`) ou son appel par l'utilisation des parenthèses.

Une liste de commandes placée entre accolades est exécutée dans le shell courant. Une liste de commandes placée entre parenthèses est exécutée dans un sous-shell qui est la copie du shell courant (obtenu par un `fork`). Des commandes exécutées dans un sous-shell partagent les mêmes descripteurs de fichiers standard, vue de l'extérieur du sous-shell. En particulier, leurs outputs sont groupés, comme s'ils provenaient d'un seul programme. Toutefois, il n'y a pas d'effet de bord sur le shell courant. Ainsi, toutes les variables du shell ne sont pas modifiées par les actions réalisées dans le sous-shell.

Il existe une différence importante entre les shells obtenus par les méthodes indiquées : si un shell est obtenu par un `fork`, il est la copie exacte du shell qui l'a engendré. En conséquence, toutes les variables existant dans le père avant le fork existent dans le fils. En revanche, seules certaines variables sont transmises au sous-shell s'il est invoqué directement. Plus bas, nous verrons comment transmettre des variables à un tel sous-shell.

Ci-dessous, nous désignons par «sous-shell» le shell invoqué par une liste de commande placée entre parenthèses. L'autre type sera mentionné de façon explicite.


Contrôlez votre répertoire courant. Exécutez un changement de répertoire (e.g. vers `/tmp`) dans un sous-shell, puis contrôlez le répertoire courant en sortie du sous-shell.


Dans un sous-shell, affectez la variable `ANNEE` du résultat de la commande `date +%Y`. Affichez le contenu de cette variable dans le sous-shell puis hors du sous-shell. Qu'observez-vous ?


Réalisez cette affectation dans le shell courant puis testez son contenu dans le shell puis dans un sous-shell. Qu'observez-vous ?


Réalisez cette affectation dans le shell, puis modifiez cette variable dans le sous-shell, en testant sa valeur, puis re-testez sa valeur dans le shell père. Qu'observez-vous ?


Forcez l'exécution d'un sous-shell par la commande `bash` ou `ksh` puis affichez le contenu de la variable `ANNEE`. Revenez au shell précédent en envoyant un signal de fin de fichier.


Exportez la variable `ANNEE` par la commande appropriée puis retestez la variable dans un sous-shell invoqué directement.

2. Vous pourrez également tester la précision de votre commande en saisissant `Clef␣␣␣␣`. Les plus motivés pourront chercher une solution rigoureuse au problème posé.

Former un groupe permettant de générer le flot formé de la chaîne "`START`", suivi des lignes produites par la commande `env` limitées à celles commençant par `PA` puis ces mêmes lignes après réaffectations des variables qu'elles indiquent à la chaîne vide, et achevée par la chaîne "`STOP`". Quel problème rencontrez-vous ? Pourquoi ?

Fonctions

Elles ne seront pas au programmes. Les motivés peuvent toutefois manifester leur impatience `:o)~`.