IUP 1re Année T.P. de Système d'Exploitation UNIX

Les buts fixés pour cette série de tps sont les suivants :

- Se familiariser avec un système d'exploitation professionnel : UNIX.
- Maîtriser les commandes de base de ce système afin de réaliser des tâches simples (gestion de fichiers, courrier...).

Évaluation et notation :

- Le ou les compte-rendus comporteront les commandes saisies, les résultats obtenus ainsi que les réponses aux questions.

Table des matières

1 Introduction **7**
- 1.1 Reconnaître votre `shell` . . . 7
- 1.2 Reconnaître votre système d'exploitation . . . 7

2 Commandes Générales **7**
- 2.1 Changer votre mot de passe . . . 7
- 2.2 Afficher la ligne (terminal) sur laquelle vous êtes connectés . . . 7
- 2.3 Afficher les paramètres de votre terminal . . . 7
- 2.4 Reconfigurer la touche `erase` . . . 8
- 2.5 Qui êtes-vous ? . . . 8
- 2.6 Qui est connecté sur la station ? . . . 8
- 2.7 Afficher du texte et les valeurs des variables système . . . 8
- 2.8 Afficher les variables de votre système . . . 9
- 2.9 Création d'un fichier simple . . . 9
- 2.10 Visualisation de votre fichier . . . 9
- 2.11 Compter le nombre de lignes, de mots, de caractères du fichier . . . 10
- 2.12 Recherche d'une chaîne dans un fichier . . . 10
- 2.13 Trier le contenu du ou des fichiers passés en argument . . . 10
- 2.14 Envoyer un courrier électronique . . . 10
- 2.15 Converser en ligne . . . 10
- 2.16 Envoyer des messages sur la console d'autres utilisateurs . . . 10
- 2.17 Refuser les `talk` et les `write` des utilisateurs . . . 11

3 Système de fichiers **12**
- 3.1 Les caractères spéciaux . . . 12
  - 3.1.1 Les caractères d'abréviations . . . 12
  - 3.1.2 Les caractères spéciaux . . . 12
- 3.2 Quel est le répertoire courant . . . 12
- 3.3 Lister les fichiers dans le répertoire courant . . . 12
- 3.4 Copier un fichier . . . 13
- 3.5 Renommer un fichier . . . 13
- 3.6 Effacer un fichier . . . 13
- 3.7 Créer un lien symbolique sur un fichier . . . 14
- 3.8 Créer un lien sur un fichier . . . 14
- 3.9 Visualiser le fichier `fic_lien.txt` . . . 14
- 3.10 Créer un répertoire . . . 14
- 3.11 Changer de répertoire . . . 15
- 3.12 Revenir au répertoire précédent . . . 15
- 3.13 Effacer un répertoire . . . 15

3.14 Effacer un répertoire : le retour . . . 15
3.15 Comprendre les droits d'accès . . . 16
3.16 Modification des droits d'accès . . . 16
3.17 Donner le type du fichier . . . 17
3.18 Trouver un fichier dans l'arborescence . . . 17
3.19 Afficher les caractéristiques des disques . . . 17

4 Les éditeurs de texte 18
4.1 vi . . . 18
4.1.1 Mode saisie . . . 18
4.1.2 Mode commande . . . 19
4.1.3 Déplacement du curseur . . . 19
4.1.4 Commandes d'effacement et remise de texte . . . 19
4.1.5 Insertion d'un fichier extérieur . . . 19
4.1.6 Annulation de la dernière commande . . . 19
4.1.7 Recherche et remplacement . . . 19
4.1.8 Déplacement de texte . . . 20
4.1.9 Mode exécution . . . 20
4.2 emacs . . . 20
4.2.1 Les modes d'édition d'`emacs` . . . 21
4.2.2 Notion de buffer . . . 21
4.2.3 Commande de déplacement du curseur . . . 21
4.2.4 Insertion et suppression de texte . . . 21
4.2.5 Édition simultanée de plusieurs fichiers . . . 21
4.2.6 Recherche et remplacement . . . 22
4.2.7 Insertion d'un fichier . . . 22
4.2.8 Suppression de fenêtres . . . 22
4.2.9 Sauvegarder et quitter `emacs` . . . 22
4.3 nedit . . . 22

5 Gestion des processus 23
5.1 Introduction . . . 23
5.2 Obtenir l'`UID` et le `GID` . . . 23
5.3 Visualiser les processus . . . 23
5.4 Lancer un processus en tâche de fond . . . 24
5.5 Tuer un processus . . . 24
5.6 Bloquer un processus . . . 24
5.7 Passer un processus en arrière plan . . . 24
5.8 Lancer une commande et afficher les temps consommés . . . 24

6 Redirections des entrées/sorties, tubes et filtres 26
6.1 Introduction . . . 26
6.2 Créer un fichier contenant la liste des utilisateurs connectés . . . 26
6.3 Envoyer ce fichier par mail à l'un de vos collaborateurs . . . 26
6.4 Faites les deux opérations précédentes en une seule fois . . . 26
6.5 Enchaînement de processus . . . 27
6.6 Lancement en séquence de plusieurs commandes . . . 27
6.7 Communication entre processus . . . 27

7 Le Shell 28
7.1 Introduction . . . 28
7.2 Les variables du Shell . . . 28
7.3 Définition de nouvelles variables . . . 28
7.3.1 Créer une variable `var` contenant la valeur `etudiant_iup` . . . 28
7.4 Mécanisme de l'accent grave . . . 28
7.4.1 Créer une variable `var2` qui renvoie la date du système lorsque l'on veut afficher son contenu . . . 29
7.5 La commande interne `read` . . . 29
7.5.1 Utiliser la commande `read` afin de créer différentes variables contenant votre adresse 29

7.6 Variables prédéfinies . . . 29
7.7 Les commandes `set` et `unset` . . . 29
7.7.1 Utiliser la commande `set` pour visualiser les variables prédéfinies . . . 30
7.8 Variables exportables . . . 30
7.9 Exécution d'un fichier de commande . . . 30
7.10 Variables maintenues par le shell . . . 30
7.10.1 Variables de contrôle . . . 30
7.10.2 Variables de position et paramètres d'un fichier de commande . . . 30
7.10.3 Utiliser les paramètres positionnels . . . 30
7.10.4 La commande `expr` . . . 31

8 Structure de contrôle 32
8.1 Réalisation de tests . . . 32
8.2 la commande `test` . . . 32
8.2.1 Tests sur les chaînes de caractères . . . 32
8.2.2 Tests sur les chaînes numériqies . . . 32
8.2.3 Tests sur les fichiers . . . 32
8.2.4 Tests sur les droits d'accès . . . 32
8.2.5 Test sur la taille . . . 32
8.3 La conditionnelle : `if ... fi` . . . 33
8.4 L'aiguillage : `case ... esac` . . . 33
8.5 L'itération bornée : `for ...in ...do ...done` . . . 33
8.6 Les itérations non bornées . . . 33
8.6.1 L'itération `while...do...done` . . . 33
8.6.2 L'itération `until...do...done` . . . 33
8.7 Les ruptures de séquence `break` et `continue` . . . 33

Table des figures

1 Connexion à une machine (ici Crabe) . . . 4
2 Connexion réussie vous pouvez travailler . . . 4
3 Signification des champs du résultat de la commande `ls -al` . . . 16
4 Fenêtre d'édition pleine page vi . . . 18
5 Fenêtre d'édition `emacs` . . . 20
6 Fenêtre de l'éditeur graphique `nedit` . . . 22

Liste des tableaux

1 Tableau récapitulatif des droits d'accès . . . 16

Première Partie : **Avant de commencer un peu de lecture**

Procédure de Login

Suivant la salle où vous vous trouvez, le type de terminal et surtout le système d'exploitation utilisé, de petites différences dans le mode de connexion peuvent survenir.
Pour se connecter à l'une des machines, il faut bien entendu, avoir un accès à celle-ci c-à-d avoir un compte.
La plupart du temps il suffit d'ouvrir une nouvelle connexion telnet. Vous obtenez une nouvelle fenêtre ressemblant à la figure 1.

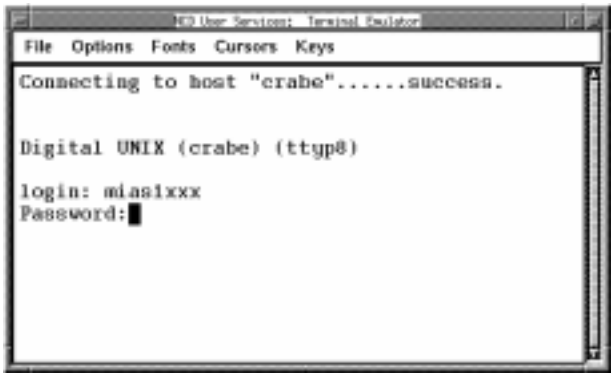

FIG. 1: Connexion à une machine (ici Crabe)

Login : saisissez ici votre **identifiant** ou **login** et `enter`
Password : saisissez ici votre **mot de passe** (Attention les lettres frappées ne s'affichent pas sur l'écran.) et `enter`.
Vous vous trouvez alors dans l'environnement UNIX, et vous avez une fenêtre du type de la figure 2.

FIG. 2: Connexion réussie vous pouvez travailler

Votre curseur clignote juste à côté du *caractère d'invite (prompt)* `$`, indiquant que le système est près à recevoir vos commandes.

Remarque sur le mot de passe

- Il faut impérativement changer le mot de passe après la première connexion.
- Vous seul connaissez le mot de passe.
- Si vous l'oubliez, la seule solution est de contacter l'administrateur réseau pour l'effacement de l'ancien mot de passe.
- Le mot de passe est composé d'au moins 6 caractères et au maximum 8 dont 2 au moins ne sont pas des lettres.
- Conseil : mélanger les lettres, chiffres, majuscules et minuscules.

Pour changer de mot de passe

- *yppasswd* ou *passwd* et `enter`
- La machine vous demande l'**ancien** mot de passe et `enter`
- Elle vous demande le **nouveau** et `enter`
- Puis elle redemande le **nouveau** pour confirmation et `enter`

Si vous n'avez pas de message d'erreur, le mot de passe est alors changé.

Terminer votre session

Pour terminer votre session UNIX (se *deloger*), vous pouvez saisir le commande `logout` ou `exit` et `enter` ou encore `ctrl-d`.

Forme et validation d'une commande

Une ligne de commande possède la forme générale suivante :

nom-de-commande [options] arguments

où l'*option* a la forme *-lettre*.

Exemple : Dans la commande
`ls -s -i toto titils` est le nom de la commande, `-s -i` sont les options, `toto` et `titi` sont les arguments. Plusieurs options peuvent être regroupées derrière le signe `-`.
`ls -si toto titi`
Une ligne de commande n'est reçue et exécutée par le système qu'après *validation* par `enter`.

Commande importante : `man`

La syntaxe est la suivante :
`man` *commande*
ou
`man -k` *commande*
Cette commande affiche la page de manuel correspondante à la *commande*. On y trouve tous les détails concernant son utilité, sa syntaxe et ses options.

Exemple : réponse du système à la commande `man ls` sur la machine `OSF1 flore V4.0 1091 alpha`

```
ls(1)

NAME

  ls - Lists and generates statistics for files

SYNOPSIS

  ls [-aAbcCdfFgilLmnopqrRstux1] [file...|directory...]

STANDARDS

  Interfaces documented on this reference page conform to industry standards
  as follows:

  ls:  XPG4, XPG4-UNIX

  Refer to the standards(5) reference page for more information about indus-
```

```
try standards and associated tags.
```

Pour toutes questions au sujet de la commande `man`, taper :

```
man man
```

Deuxième Partie : **Travaillons un peu**

1 Introduction

1.1 Reconnaître votre `shell`

`echo $SHELL`
Réponse :

1.2 Reconnaître votre système d'exploitation

`uname -a`
Réponse :

2 Commandes Générales

2.1 Changer votre mot de passe

`passwd` ou `yppasswd`
Réponse :

Observations :

2.2 Afficher la ligne (terminal) sur laquelle vous êtes connectés

`tty`
Réponse :

2.3 Afficher les paramètres de votre terminal

`stty`
Réponse :

Observations :

2.4 Reconfigurer la touche `erase`

`stty erase ^H`
Réponse :

2.5 Qui êtes-vous ?

`whoami`
Réponse :

`who am i`
Réponse :

Observations :

2.6 Qui est connecté sur la station ?

`who`
Réponse :

`w`
Réponse :

`users`
Réponse :

Observations :

2.7 Afficher du texte et les valeurs des variables système

`echo "Bonjour"`
Réponse :

`echo $DISPLAY`
Réponse :

2.8 Afficher les variables de votre système

`setenv`
Réponse :

Observations :

2.9 Création d'un fichier simple

Saisissez les lignes suivantes
`cat > fic.txt`
`mon premier fichier <enter>`
`bbbbbbbbbbbbbbbbbbb <enter>`
`ccccccccccccccccccc <enter>`
`aaaaaaaaaaaaaaaaaaa <enter>`
`ctrl-d`
Réponse :

2.10 Visualisation de votre fichier

`cat fic.txt`
Réponse :

`more fic.txt`
Réponse :

`less fic.txt`
Réponse :

Observations :

2.11 Compter le nombre de lignes, de mots, de caractères du fichier

`wc fic.txt`
Réponse :

2.12 Recherche d'une chaîne dans un fichier

`grep aa fic.txt`
Réponse :

Observations :

2.13 Trier le contenu du ou des fichiers passés en argument

`sort fic.txt`
Réponse :

2.14 Envoyer un courrier électronique

`mail` *login_utilisateur*
Envoyer un mail à l'un de vos collaborateurs
Réponse :

2.15 Converser en ligne

`talk` *login_utilisateur*
Essayer la commande `talk` avec l'un de vos collaborateurs
Réponse :

2.16 Envoyer des messages sur la console d'autres utilisateurs

`write` *login_utilisateur*
Essayer la commande `write` avec l'un de vos collaborateurs
Réponse :

2.17 Refuser les `talk` et les `write` des utilisateurs

`mesg` *yes|no*
Essayer la commande `mesg` avec l'un de vos collaborateurs
Réponse :

Troisième Partie : **Passons la vitesse supérieure**

3 Système de fichiers

3.1 Les caractères spéciaux

Il existe plusieurs caractères qui sont traités d'une manière différente pas l'interpréteur de commande. On les appelle des **caractères spéciaux** ou **métacaractères**. Ils comprennent :

- des caractères d'abréviations ;
- des caractères spéciaux liés au lancement d'une commande.

3.1.1 Les caractères d'abréviations

`*` remplace toute chaîne de caractères permettant de compléter un nom de fichier sauf une chaîne commençant par le caractère "." lequel doit être explicitement écrit.

`?` peut représenter un caractère quelconque (à l'exception de ".").

`[`***liste de caractères***`]` désigne un caractère quelconque écrit entre crochet. Par exemple `foo.[cp]` désigne la liste des 2 fichiers `foo.c` et `foo.p`.

3.1.2 Les caractères spéciaux

`;` sépare les instructions successives d'une ligne de commande.

`()` servent à regrouper les commandes.

`>` redirection de la sortie standard.

`>>` redirection de la sortie standard sans écrasement.

`2>` redirection de l'erreur standard.

`2>>` redirection de l'erreur standard sans écrasement.

`<` redirection de l'entrée standard.

`&` lancement d'un processus en arrière plan (tâche de fond).

`|` communication par tube entre deux processus.

Exemple :

- La commande `ls *.txt` permet de lister tous les fichiers possédant l'extension `txt`.
- La commande `ls t?t?.txt` permet de lister tous les fichiers `titi.txt`, `toto.txt`, `tctc.txt` ...
- La commande `ls t[ao]t[ao].txt` permet de lister tous les fichiers possédant l'extension `tata.txt`, `tota.txt`, `toto.txt`, `tato.txt`.

3.2 Quel est le répertoire courant

`pwd`
Réponse :

3.3 Lister les fichiers dans le répertoire courant

`ls`
Réponse :

`ls -l`

Réponse :

`ls -al`
Réponse :

Observations :

3.4 Copier un fichier

`cp fic.txt fic_copie.txt`
Réponse :

Vérification :

3.5 Renommer un fichier

`mv fic_copie.txt nouveau_nom.txt`
Réponse :

Vérification :

3.6 Effacer un fichier

`rm nouveau_nom.txt`
Réponse :

Vérification :
`rm -i nouveau_nom.txt`

Réponse :

Vérification :

Observations :

3.7 Créer un lien symbolique sur un fichier

`ln -s fic.txt fic_lien.txt`
Réponse :

Vérification :

3.8 Créer un lien sur un fichier

`ln fic.txt fic_lien.txt`
Réponse :

Vérification :

Observations par rapport au lien symbolique :

3.9 Visualiser le fichier `fic_lien.txt`

Réponse :

Observations par rapport au lien symbolique :

3.10 Créer un répertoire

`mkdir mon_repertoire`
Réponse :

Vérification :

3.11 Changer de répertoire

`cd mon_repertoire`
Réponse :

Vérification :

3.12 Revenir au répertoire précédent

`cd ..`
Réponse :

Vérification :

3.13 Effacer un répertoire

`rmdir mon_repertoire`
Réponse :

Vérification :

3.14 Effacer un répertoire : le retour

Créer un répertoire `parent`, se déplacer dans celui-ci, créer un répertoire `enfant`
Revenir dans votre répertoire `home` et lancer :
`rm -r parent`
Réponse :

Vérification :

Observations :

3.15 Comprendre les droits d'accès

La commande `ls -al` affiche le résultat suivant :

```
drwxr-xr-x 5 ymorere prof 1024 Sep 21 14 :57 public_html
```

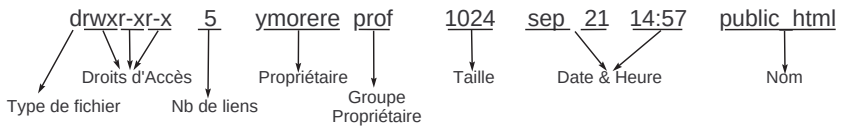

FIG. 3: Signification des champs du résultat de la commande `ls -al`

Les droits d'accès sont les ensembles d'autorisations, qui déterminent *qui* peut avoir accès aux fichiers *en vue de quelle utilisation.*

droits d'accès	**utilisateurs**	**commandes**
lecture = **r**	propriétaire = **u**	ajouter = +
écriture = **w**	groupe = **g**	enlever = -
exécutable = **x**	autre = **o**	initialiser = =
	tous = **a**	

TAB. 1: Tableau récapitulatif des droits d'accès

3.16 Modification des droits d'accès

Commande `chmod` *utilisateurs+/-droit fichier/répertoire*

Changer les droits d'accès de votre fichier `fic.txt` : vous devez être le seul à pouvoir lire et modifier votre fichier.

Réponse :

Vérification :

Observations :

Créer un répertoire et visualiser ses droits d'accès. Modifier ses droits d'accès afin que tout le monde puisse accéder à celui-ci mais ne puisse pas y faire de modification.

Réponse :

Vérification :

Observations :

3.17 Donner le type du fichier

Commande `file` *`nom_du_fichier`*
Exécuter la commande avec un fichier
Réponse :

Exécuter la commande avec un répertoire
Réponse :

Observations :

3.18 Trouver un fichier dans l'arborescence

Commande `find / -name "talk" -print`
recherche le fichier `talk` à partir du répertoire racine `/`
Rechercher les fichiers commençant par `x` dans le répertoire `/usr/bin`
Réponse :

3.19 Afficher les caractéristiques des disques

Commande `df`
Réponse :

Observations :

Quatrième Partie : **On a toujours besoin d'un éditeur de texte**

4 Les éditeurs de texte

Un éditeur de texte est un outil pour écrire un fichier texte pur c'est à dire sans mise en forme (il ne contient que des caractères de la table ASCII) (`edit` sous DOS, `turbo` pour la programmation TurboPascal), alors qu'un traitement de texte est un outil pour écrire et mettre en forme du texte (*WordPerfect*, *MS Word*).

4.1 vi

`vi` est un éditeur vidéo, c'est à dire pleine page et interactif. Il est possible d'insérer du texte à tout endroit dans le fichier en édition. (Cf. figure 4)
L'appel de `vi` s'effectue de la manière suivante :

- `vi`
- `vi` *fichier*
- `vi` *+n fichier* pour se placer directement à la *n*-ième ligne du *fichier*
- `vi` *+/motif fichier* pour se placer directement à la première occurrence du *motif* dans le *fichier*

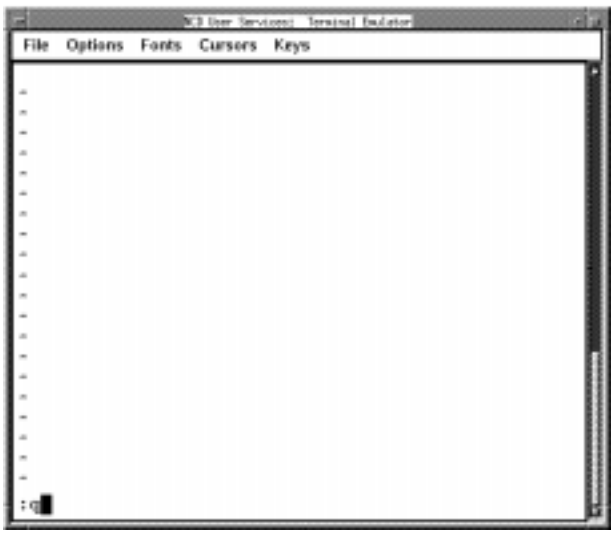

FIG. 4: Fenêtre d'édition pleine page vi

Il dispose de 3 modes de travail :

Mode commande permet les déplacements, les recherches, les destructions, etc... C'est le mode par défaut.

Mode exécution permet l'utilisation de toutes les commandes de `ed` (editeur en ligne UNIX). Mode utilisé pour les commandes globales et les commandes gérant les fichiers.

Mode saisie permet la saisie de texte.

4.1.1 Mode saisie

Pour passer en mode saisie (depuis le mode commande) il faut taper une des commandes d'édition :

- `a` pour "append" (ajout) ajoute du texte après le curseur.
- `A` pour "append" (ajout) ajout du texte en fin de ligne.
- `i` pour "insert" (insertion) insère du texte devant le curseur.
- `I` pour "insert" (insertion) insère du texte en début de ligne.
- `o` pour "open" (ouvrir) ouvre une ligne après le curseur.
- `O` pour "open" (ouvrir) ouvre une ligne en fin de ligne.

4.1.2 Mode commande

Pour repasser en mode commande il faut appuyer sur `ESC`.

4.1.3 Déplacement du curseur

Déplacement du curseur en mode commande :

- `x`,`j`,`k` et `l` déplacent le curseur dans les 4 directions (O, S, N et E). Ces touches peuvent être remplacées par les flèches du clavier.
- `w` place le curseur au début du mot suivant
- `b` place le curseur au début du mot précédent
- `e` place le curseur à la fin du mot courant
- `0` place le curseur en début de ligne
- `$` place le curseur en fin de ligne
- `enter` place le curseur au début de la ligne suivante
- `^F` avance le curseur d'une page
- `^B` recule le curseur d'une page
- `G` place le curseur en fin de fichier
- *n*`G` place le curseur à la *n*-ième ligne du fichier
- `/`*motif*`/` place le curseur à la prochaine occurrence du *motif*
- `m`*c* définit une *marque* : elle associe la position du curseur au caractère *c*. On peut alors retourner d'un autre endroit du fichier à cette position par la commande `` `c `` ou au début de la ligne par `'c`

4.1.4 Commandes d'effacement et remise de texte

- `x` efface le caractère sous le curseur
- `dw` efface un mot
- `db` efface le mot précédent
- `D` efface la fin de la ligne
- `dd` efface la ligne du curseur
- `r`*c* remplace le caractère courant par *c*
- `~` remplace une minuscule par une majuscule et vice-versa
- `p` réinsère le texte dernièrement effacé

4.1.5 Insertion d'un fichier extérieur

- `:r` *fichier* insère après la ligne courante un fichier extérieur

4.1.6 Annulation de la dernière commande

- `u` annule la dernière commande effectuée

4.1.7 Recherche et remplacement

- `:/`*motif* place le curseur sur la prochaine occurrence de *motif*
- `:s/`*motif*`/`*chaîne*`/` remplace dans la ligne courante la première occurrence de *motif* par la *chaîne*
- `:s/`*motif*`/`*chaîne*`/g` remplace dans la ligne courante toute occurrence de *motif* par la *chaîne*
- `:1,10s/`*motif*`/`*chaîne*`/g` remplace toute occurrence de *motif* par la *chaîne* de la première à la dixième ligne
- `:.,$%s/`*motif*`/`*chaîne*`/g` remplace toute occurrence de *motif* par la *chaîne* depuis la ligne courante (désignée par `"."`) jusqu'à la dernière ligne du fichier (désignée par `"$"`)
- `:%s/`*motif*`/`*chaîne*`/g` remplace dans tout le fichier toute occurrence de *motif* par la *chaîne*

4.1.8 Déplacement de texte

- `mc` définit le début de la section à déplacer
- se déplacer à la fin de la section à déplacer et frapper `d'c`, la fraction de texte est alors supprimée et placée dans le tampon
- se déplacer à l'endroit voulu et frapper `p` qui insère le texte contenu dans le tampon.

4.1.9 Mode exécution

Le mode exécution est activé depuis le mode commande par la caractère ' :' suivi de la commande :

- `:w` *nom_fichier* sauvegarde le fichier *nom_fichier*
- `:w` sauvegarde du fichier
- `:wq` sauvegarde et quitte `vi`
- `:q!` quitte `vi` sans sauvegarde
- `:x` équivalent à `:wq`

4.2 emacs

Les principaux avantages d'`emacs` sont l'extensibilité, la personnalisation et l'auto-documentation. Il possède de nombreuses fonctionnalités autres que celles de l'édition.
On peut compiler un programme, lire du courrier électronique, lire les forums, récupérer un fichier par ftp... `emacs` signifie editor macros.
L'appel d'`emacs` s'effectue de la manière suivante :

- `emacs`
- `emacs` *fichier*
- `emacs` *+n fichier* pour se placer directement à la *n*-ième ligne du *fichier*

FIG. 5: Fenêtre d'édition `emacs`

La plupart des commandes `emacs` font intervenir :

- la touche `<CTRL>` frappée en même temps qu'un autre caractère. Elle est habituellement notée `C-`. Par exemple `C-p` désigne la touche `<CTRL>` frappée en même temps que `p`.
- la touche `META`, existante sur la clavier, soit la touche `<ALT>` ou encore la touche `<ESC>` (dans ce cas elle doit être relâchée avant la touche qui suit). Elle est notée `M-`.

4.2.1 Les modes d'édition d'`emacs`

Selon que l'on travaille sur un fichier texte, un programme C, un fichier de données, les besoins d'édition sont différents. `emacs` propose donc plusieurs modes d'édition qui définissent un environnement de travail adapté au type de fichier.
Modes principaux :

- le mode **Fundamental**
- le mode **Text**
- le mode **Lisp**
- le mode **C**

Il existe des modes secondaires utilisés en conjonction avec un mode principal :

- le mode **Fill** : les lignes sont automatiquement coupées quand elles dépassent la marge droite.
- le mode **Abbrev** : l'expansion des abréviations est impossible.
- le mode **Ovwrt** : mode recouvrement.
- le mode **Narrow** : rend accessible qu'une partie du buffer.

4.2.2 Notion de buffer

C'est une partie de la mémoire d'`emacs`. A chaque ouverture de fichier un buffer portant le même nom est créé. Les modifications n'affectent pas tout de suite le fichier sur disque mais le buffer. Il faut sauver le fichier pour que les modifications soient effectuées sur le fichier d'origine.

4.2.3 Commande de déplacement du curseur

- `C-p` place le curseur sur la ligne précédente
- `C-n` place le curseur sur la ligne suivante
- `C-f` avance le curseur d'un caractère
- `M-f` avance le curseur à la fin du mot courant ou suivant
- `C-b` recule le curseur d'un caractère
- `M-b` recule le curseur au début du mot courant ou du précédent
- `C-a` place le curseur en début de ligne
- `C-e` place le curseur en fin de ligne
- `M-<` place le curseur en début de fichier
- `M->` place le curseur en fin de fichier
- `C-v` avance la fenêtre d'un écran
- `M-v` recule la fenêtre d'un écran

Toutes ces commandes peuvent être envoyées avec un argument numérique *n*. L'argument numérique est introduit par `C-u`.
La commande `C-u 8 C-n` avance le curseur de 8 lignes.

4.2.4 Insertion et suppression de texte

- `C-d` efface le caractère sur lequel se trouve le curseur
- `C-k` efface la fin de la ligne
- `C-y` réinsère le texte effacé
- `C-x u` annule l'effet de la dernière modification

4.2.5 Édition simultanée de plusieurs fichiers

- `C-x C-f` *fichier* crée un nouveau tampon et y place le *fichier*
- `C-x C-b` ouvre un nouveau tampon et une nouvelle fenêtre sur l'écran et y affiche la liste de tous les tampons ouverts (cette fenêtre peut être supprimée avec `C-x 1`)
- `C-x b` *tampon* place la curseur dans le *tampon*
- `C-x C-v` *fichier* place le *fichier* dans le tampon courant, son contenu actuel est éliminé.
- `C-x k` *tampon* supprime le *tampon* (par défaut c'est le tampon courant qui est supprimé)

4.2.6 Recherche et remplacement

- `C-s` *mot* recherche la première occurrence du *mot* dans la suite du texte. A chaque fois que l'utilisateur frappe une nouvelle lettre du *mot*, le curseur se place sur la prochaine occurrence de ce qui a été frappé. La répétition de `C-s` recherche l'occurrence du *mot* suivant.
- `C-r` *mot* a le même effet que `C-s`, mais la recherche se fait sur le texte qui précède.
- `M-%` *chaîne nouvelle_chaîne* remplace une *chaîne* de caractères par une *nouvelle_chaîne*. L'utilisateur doit valider chaque remplacement. S'il frappe `!` les validations sont omises

4.2.7 Insertion d'un fichier

- `C-x i` *fichier* insère le *fichier* à l'endroit du curseur

Pour n'insérer qu'une partie du fichier effectuer les commandes suivantes :

- `C-x C-f` *fichier* insère le *fichier* dans un nouveau tampon
- Placer le curseur au début de la région à insérer
- `C-@` ou `C-ESPACE` marque le début de la région
- Placer le curseur à la fin de la région à insérer
- `C-w` efface la région délimitée
- `C-x b` revient au tampon initial
- Se placer à l'endroit voulu de l'insertion
- `C-y` insère le contenu du tampon auxiliaire

4.2.8 Suppression de fenêtres

- `C-x 1` supprime toutes les fenêtres qui ont été ouvertes à l'exception de celle ou se trouve le curseur

4.2.9 Sauvegarder et quitter `emacs`

- `C-x C-s` sauvegarde les modifications effectuées si `emacs` connaît le nom de fichier. Sinon l'utilisateur est invité à entrer un nom de fichier dans le mini tampon.
- `C-z` quitte `emacs` provisoirement (le processus est suspendu). Le retour à `emacs` se fait en frappant `fg` ou encore `%emacs`.
- `C-x C-c` quitte `emacs` définitivement.

4.3 nedit

`nedit` est un éditeur graphique qui ressemble beaucoup aux éditeurs rencontrés sous les environnements PC, Mac, Amiga (Linux, Windows 3.xx et 9x, DOS, Mac OS, Amiga OS). `nedit` peut être complètement géré à la souris et possède des menus conviviaux pour l'édition, la recherche de texte, le copier coller, la sélection de texte.

FIG. 6: Fenêtre de l'éditeur graphique `nedit`

Il possède aussi des prédispositions pour la programmation. Il gère l'auto-indentation des lignes de programmes, la gestion du nombre des parenthèses, l'affichage des mots-clés du langage.

Cinquième Partie : **Unix : Un vrai OS multi-tâches**

5 Gestion des processus

5.1 Introduction

Un processus est un programme binaire en cours d'exécution. Il possède les caractéristiques suivantes :

- identificateur : `PID`
- identificateur du père : `PPID`
- identificateur de l'utilisateur : `UID`
- le répertoire courant
- les fichiers ouverts
- le masque de création : `umask`
- la taille maximum des fichiers créés par le processus : `ulimit`
- la priorité
- les temps d'exécution
- le terminal de contrôle

5.2 Obtenir l'`UID` et le `GID`

`id`
Réponse :

Observations :

5.3 Visualiser les processus

`ps`
Réponse :

`ps -u`*votre_login*
Réponse :

`ps -aj`
Réponse :

Observations :

5.4 Lancer un processus en tâche de fond

nom_du_processus&
Lancer `nedit` en tâche de fond
Réponse :

Lancer `nedit`
Réponse :

Observations :

5.5 Tuer un processus

`kill -`*nom_signal numéro_de_processus*
Tuer le processus `nedit`
Réponse :

Observations :

5.6 Bloquer un processus

Si vous avez lancer un processus long en oubliant de le placer en tâche de fond, il est intéressant de pouvoir le bloquer afin de "reprendre la main" et de le relancer en tâche de fond.
Frapper `<CTRL-Z>`, stoppe le processus en cours et vous "rend la main", il vous est alors possible de la placer en tâche de fond.

5.7 Passer un processus en arrière plan

`bg` *numéro_de_job* ou `bg`
Lancer `nedit`, bloquer le, et passer le en tâche de fond
Réponse :

Observations :

5.8 Lancer une commande et afficher les temps consommés

`time` *commande*
Lancer une recherche sur les fichiers commençant par `X` dans toute l'arborescence et afficher les temps consommés.
Réponse :

Observations :

Sixième Partie : **Vous serez bientôt des gourous UNIX**

6 Redirections des entrées/sorties, tubes et filtres

6.1 Introduction

Tout processus communique avec l'extérieur par l'intermédiaire de trois fichiers appelés *fichiers standards* :

- Le fichier *entrée standard* sur lequel le processus lit ses données
- Le fichier *sortie standard* sur lequel le processus écrit ses résultats
- le fichier *sortie erreur standard* sur lequel le processus écrit ses messages d'erreurs

Par défaut ces fichiers sont associés au terminal :

- l'*entrée standard* est le clavier
- la *sortie standard* et *sortie erreur* sont l'écran

Il est possible de *rediriger* les entrées/sorties standards d'un processus. On peut leur associer un fichier autre que le terminal.

- *commande* < *référence* : redirige l'entrée standard de la *commande* sur le fichier dont on donne la *référence*
- *commande* > *référence* : redirige la sortie standard *avec écrasement* du fichier nommé
- *commande* » *référence* : redirige la sortie standard *sans écrasement* du fichier nommé
- *commande* 2> *référence* : redirige la sortie d'erreur *avec écrasement* du fichier nommé
- *commande* 2» *référence* : redirige la sortie d'erreur *sans écrasement* du fichier nommé

6.2 Créer un fichier contenant la liste des utilisateurs connectés

Réponse :

Vérification :

6.3 Envoyer ce fichier par mail à l'un de vos collaborateurs

Réponse :

Vérification :

6.4 Faites les deux opérations précédentes en une seule fois

Réponse :

Vérification :

Observation :

6.5 Enchaînement de processus

Il est possible, dans une même ligne de commande, de lancer plusieurs commandes qui vont s'exécuter soit *séquentiellement*, soit *concurremment* avec communication entre elle par l'intermédiaire d'une zone mémoire appelée *tube (pipe)*.

6.6 Lancement en séquence de plusieurs commandes

commande1 ; *commande2* ou (*commande1* ; *commande2*)
Mettre la date et la liste des utilisateurs connectés dans un fichier `essai`
Réponse :

Vérification :

6.7 Communication entre processus

commande1 | *commande2* envoie directement la sortie de la *commande1* vers l'entrée de la *commande2*
Lister les fichiers de votre répertoire et afficher ceux qui contiennent `ess` dans leur nom
Réponse :

Vérification :

Septième Partie : `$SHELL : csh` **répondit l'**`echo`

7 Le Shell

7.1 Introduction

Sous Unix, il existe plusieurs *langages de commande*, les plus connus sont les suivants : le Bourne-Shell (`sh`), le C-Shell (`csh`), le Bash (Bourne Again Shell de Linux) (`bash`) et le Korn-Shell (`ksh`).

7.2 Les variables du Shell

Le Shell donne à l'utilisateur la possibilité de définir des **variables** qui peuvent être utilisées dans la construction de commandes complexes.
Un certain nombre de variables sont prédéfinies dès le moment où l'utilisateur se loge.

Une variable possède un *nom* et une *valeur*. Son *nom* est une chaîne de caractères commençant par une lettre et composée de lettres, de chiffres et du caractère `_`. Sa *valeur* est une chaîne de caractère quelconque.

7.3 Définition de nouvelles variables

Le mécanisme d'*affectation* avec la syntaxe *nom=valeur* permet de définir une nouvelle variable et de lui affecter une valeur. La valeur de la variable *nom* est donnée par la chaîne `$nom` ou `${nom}` s'il faut isoler la variables des caractères qui suivent.

Exemple :

```
$ x=gh
$ echo $x
gh
$echo $xijk

$echo ${x}ijk
ghijk
$
```

7.3.1 Créer une variable `var` contenant la valeur `etudiant_iup`

Réponse :

Vérification :

7.4 Mécanisme de l'accent grave

Le shell substitue un commande placée entre des accents graves ‘ par la chaîne de caractères qui serait envoyée sur la sortie standard.

Exemple :

```
$ a=‘pwd‘
$ echo $a
/usr/lib
$
```

7.4.1 Créer une variable `var2` qui renvoie la date du système lorsque l'on veut afficher son contenu

Réponse :

Vérification :

7.5 La commande interne `read`

On peut affecter à une ou plusieurs variables des valeurs lues sur l'entrée standard au moyen de la commande interne `read` avec la syntaxe suivante : `read` var_1 $var_2 \ldots var_n$.
La commande `read` analyse la ligne lue sur l'entrée standard et affecte les chaînes successives aux différentes variables var_1 $var_2 \ldots var_n$.

Exemple :

```
$ read nom adresse

lucifer 35, rue de la Gehenne    <-- entre au clavier
$ echo $nom
lucifer
$ echo $adresse
35, rue de la Gehenne
$
```

7.5.1 Utiliser la commande `read` afin de créer différentes variables contenant votre adresse

Réponse :

Vérification :

7.6 Variables prédéfinies

Les variables prédéfinies par défaut sont les suivantes.

`PS1` a pour valeur le premier caractère de l'invite (prompt).

`PS2` a pour valeur le second caractère de l'invite.

`HOME` a pour valeur la référence absolue du répertoire de l'utilisateur.

`LOGNAME` a pour valeur l'identification de l'utilisateur.

`PATH` est une variable très importante. Sa valeur est une chaîne de caractères indiquant une liste de références de tous les répertoires susceptibles de contenir des commandes utilisées par l'utilisateur.

`IFS` a pour valeur l'ensemble des caractères interprétés comme *séparateurs de chaîne* par le shell.

`TERM` est également une variable importante. Elle indique le type de terminal utilisé.

7.7 Les commandes `set` et `unset`

La commande `set` permet d'obtenir la liste des variables de l'environnement et de leurs valeurs. La commande `unset` permet de supprimer une variable.

7.7.1 Utiliser la commande `set` pour visualiser les variables prédéfinies

Réponse :

Observations :

7.8 Variables exportables

Afin d'ajouter une nouvelle variable à l'environnement shell on utilise la commande `export` ou `setenv`. On peut visualiser les variables à l'environnement shell par la commande `env`.

7.9 Exécution d'un fichier de commande

On peut rendre un fichier de commande exécutable de 4 manières :

1. en envoyant la commande `sh fichier`, il suffit dans ce cas que `fichier` soit accessible en lecture.
2. en lançant la commande `. fichier`, il suffit dans ce cas que `fichier` soit accessible en lecture.
3. en envoyant la commande `fichier`, mais là il faut que `fichier` soit accessible en lecture, en écriture et en **exécution**.
4. en envoyant la commande `exec fichier` ou `source fichier`, mais là il faut que `fichier` soit accessible en lecture, en écriture et en **exécution**.

7.10 Variables maintenues par le shell

Grâce à ces variables, il est possible, à l'intérieur d'un fichier de commande, de faire référence aux arguments de la ligne de commande.

7.10.1 Variables de contrôle

Elles donnent des informations sur les processus en cours.

`$` a pour valeur le numéro de processus shell en cours

`!` a pour valeur le numéro du dernier processus lancé en background

`?` a pour valeur le code de retour de la dernière commande exécutée. 0 si la dernière commande s'est exécutée convenablement, non nulle sinon.

7.10.2 Variables de position et paramètres d'un fichier de commande

Tout processus shell maintient une liste de chaînes de caractères que l'on peut connaître par la valeur des variables `*`, `#`,`1`, `2`, `3`, `...`, `9`.

`#` a pour valeur le nombre de chaînes présentes dans la liste

`*` a pour valeur la liste des chaînes de caractères

`i` pour `i=1,...,9` a pour valeur la `i`-ème chaîne de caractères

Remarque : Si le processus shell est crée pour exécuter une commande avec des arguments, alors la variable `0` prend pour valeur le nom de la commande et `*` prend pour valeur la liste des arguments.

7.10.3 Utiliser les paramètres positionnels

Saisir dans un fichier le script suivant :

```
echo procedure
echo il y a $# paramètres
echo qui sont [$*]
echo le troisième est $3
echo tous les paramètres sont contenus dans la liste [$@]
echo ce script a comme PID [$$]
```

Rendre le script exécutable et le lancer avec les paramètres `a b c d e f g h`
Réponse :

Observations :

7.10.4 La commande `expr`

La commande `expr` considère la suite de ses arguments comme une expression (numérique ou chaîne de caractères). Elle l'évalue et affiche le résultat sur la sortie standard.

```
$ expr 4 + 7
11
$
```

Attention :

1. les différents termes intervenant doivent être séparés par des espaces.
2. si les opérateurs utilisés sont des caractères spéciaux, il doivent être déspécialisés (exemple : `>` doit être écrit `\>`).
3. on peut utiliser les parenthèses `(` et `)` pour regrouper des termes (il faut également les déspécialiser).

Les opérateurs utilisables sont les suivants :

- Le **OU** logique `|` : $expression_1$ `|` $expression_2$ a pour valeur celle de $expression_1$ si $expression_1$ n'est pas nulle (chaîne vide ou 0) et sinon pour valeur celle de $expression_2$ (ou 0 si $expression_2$ est vide).
- Le **ET** logique `&` : $expression_1$ `&` $expression_2$ a pour valeur celle de $expression_1$ si $expression_1$ et $expression_2$ sont toutes 2 non nulles et non vides ; vaut 0 dans le cas contraire.
- Les opérateurs de **comparaisons** : `<`,`>`,`=`,`>=`,`<=`, `!` (différent de). $expression_1$ *opérateur* $expression_2$ vaut 1 si le résultat de la comparaison est vrai, 0 sinon.
- Les opérateurs **additifs** : `+` et `-`
- Les opérateurs **multiplicatifs** : `*` multiplication, `/`division, `%` reste.

Excercice : Écrire le script qui permet de renvoyer la valeur en € d'une valeur entrée en francs
Réponse :

Observations :

Huitième Partie : **Script, programmation et autres douceurs**

8 Structure de contrôle

8.1 Réalisation de tests

En shell on peut tester le code de retour d'une commande. On sait que tout processus se termine en délivrant un code de retour. Le code de retour du processus est affecté à la variable "`?`". La valeur 0 représente la valeur logique VRAI, et toute autre valeur non nulle la valeur logique FAUX.

Exemple : la commande `ls` délivre un code de retour nul si et seulement si son argument est un fichier du répertoire de travail.

8.2 la commande `test`

Cette commande permet de réaliser des tests sur les chaînes de caractères et sur des fichiers. Deux syntaxes sont possibles :

`test` *expression* ou `[` *expression* `]`

La commande `test` délivre un code de retour 0 si l'expression évaluée est vrai et non nul sinon.

8.2.1 Tests sur les chaînes de caractères

Voici les différents types de test disponibles :

`test -z` *chaîne* est VRAI si et seulement si *chaîne* est la chaîne vide
`test -n` *chaîne* est VRAI si et seulement si *chaîne* n'est pas vide
`test` $chaîne_1$ `=` $chaîne_2$
`test` $chaîne_1$ `!=` $chaîne_2$

8.2.2 Tests sur les chaînes numériqies

Une *chaîne numérique* est une suite de chiffres.

`test` $chaîne_1$ `-eq` $chaîne_2$ ("equal")
`test` $chaîne_1$ `-nq` $chaîne_2$ ("not equal")
`test` $chaîne_1$ `-lt` $chaîne_2$ ("less than")
`test` $chaîne_1$ `-le` $chaîne_2$ ("less or equal")
`test` $chaîne_1$ `-gt` $chaîne_2$ ("greater then")
`test` $chaîne_1$ `-ge` $chaîne_2$ ("greater or equal")

8.2.3 Tests sur les fichiers

On peut tester sur un fichier son *type*, les *droits d'accès* de l'utilisateur et le fait que sa *taille* est non nulle.

`test -p` *reference* est vrai ssi *reference* est un tube (nommé).
`test -f` *reference* est vrai ssi *reference* est un fichier ordinaire.
`test -d` *reference* est vrai ssi *reference* est un répertoire.
`test -c` *reference* est vrai ssi *reference* est un fichier spécial *en mode caractère*.
`test -b` *reference* est vrai ssi *reference* est un fichier spécial *en mode bloc*.

8.2.4 Tests sur les droits d'accès

On peut tester si un fichier *reference*, est autorisé en lecture, écriture ou exécution pour le propriétaire du processus.

`test -r` *reference* (autorisation en lecture).
`test -w` *reference* (autorisation en écriture).
`test -x` *reference* (autorisation en exécution).

8.2.5 Test sur la taille

`test -s` *reference* est vrai ssi *reference* est un fichier de taille non nulle.

8.3 La conditionnelle : `if ... fi`

La structure de contrôle `if ... fi` sa syntaxe est la suivante :

`if` *commande*$_1$
`then` *commande*$_2$
`else` *commande*$_3$
`fi`

commande$_1$ est exécutée ; si son code de retour est vrai (0), *commande*$_2$ est exécutée et on sort de la structure ; si son code de retour est faux (différent de 0), *commande*$_3$ est exécutée et on sort de la structure.

8.4 L'aiguillage : `case ... esac`

La structure de contrôle `case ... esac` permet d'exécuter telle ou telle commande en fonction de la valeur d'une certaine chaîne de caractères :

`case` *chaîne*
motif$_1$`)` *commande*$_1$ `; ;`
motif$_2$`)` *commande*$_2$ `; ;`
`..............`
`esac`

On examine si *chaîne* appartient à l'ensemble d'expressions décrit par le modèle *motif*$_1$; Dans ce cas *commande*$_1$ est exécutée et on sort de la structure ; sinon on examine si *chaîne* satisfait *motif*$_2$ et ainsi de suite. Ainsi la commande associée au premier modèle satisfait est exécutée.

8.5 L'itération bornée : `for ...in ...do ...done`

La structure de contrôle

`for` *variable* `in` *chaîne*$_1$, *chaîne*$_2$ ...
`do` *commande*
`done`

affecte successivement à *variable* les chaînes *chaîne*$_1$, *chaîne*$_2$ ..., en exécutant à chaque fois la *commande*.

8.6 Les itérations non bornées

8.6.1 L'itération `while...do...done`

La structure de contrôle

`while` *commande*$_1$
`do` *commande*$_2$
`done`

exécute répétitivement *commande*$_1$ puis *commande*$_2$ tant que le code de retour de *commande*$_1$ est vrai sinon on sort de la structure.

8.6.2 L'itération `until...do...done`

La structure de contrôle

`until` *commande*$_1$
`do` *commande*$_2$
`done`

exécute répétitivement *commande*$_1$ puis *commande*$_2$ tant que le code de retour de *commande*$_1$ est faux sinon on sort de la structure.

8.7 Les ruptures de séquence `break` et `continue`

Les commandes `break` et `continue` permettent d'interrompre ou de modifier le déroulement d'un boucle d'itération.

1. La commande `break` fait sortir d'une itération `for`, `while`, `until`. Sous la forme `break` *n* elle permet de sortir de *n* niveaux d'imbrication.

2. La commande `continue` permet de passer au pas suivant de l'itération. Sous la forme `continue` *n* elle permet de sortir de *n-1* niveaux d'imbrication et de passer au pas suivant du *n*-ième niveau.

Références

[CHAMPARNAUD et HANSEL, 1995] J.M. CHAMPARNAUD et G. HANSEL. *Passeport pour UNIX et C*. Passeport pour l'informatique. Internationnal Thomson Publishing, 1995.

[POULAIN, 1994] T. POULAIN. Cours unix. 1994.