

**Christophe Blaess**

# **Programmation par scripts**

**Ecriture de scripts shell**

Ce document a été réalisé sur système Linux avec les logiciels libres :

- **Formation+** pour préparer le cours complet,
- **L<sup>A</sup>T<sub>E</sub>X** pour produire les transparents et le support de cours imprimé,
- **Xfig** pour les graphiques vectoriels,
- **The Gimp** pour les images et les photos.

Support de cours : version 17

© CHRISTOPHE BLAESS 2001-2007 – Tous droits réservés

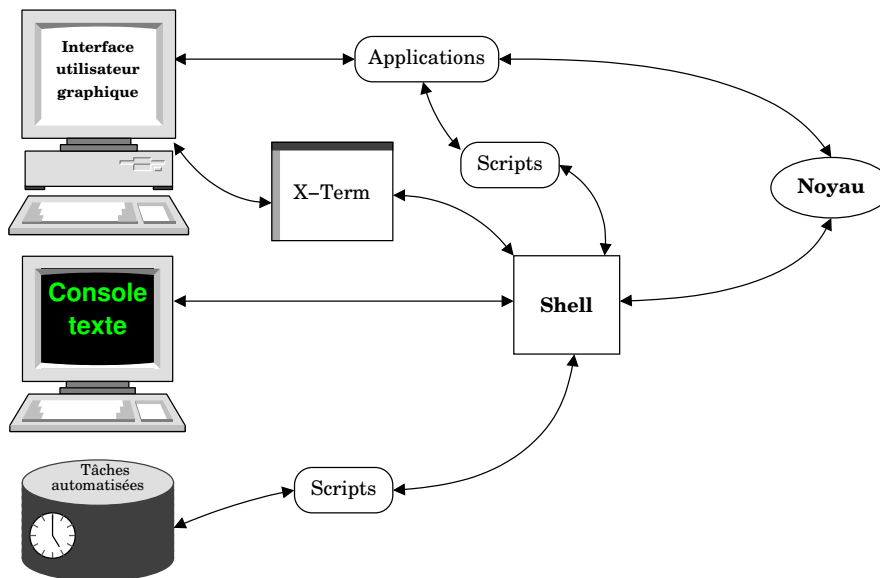
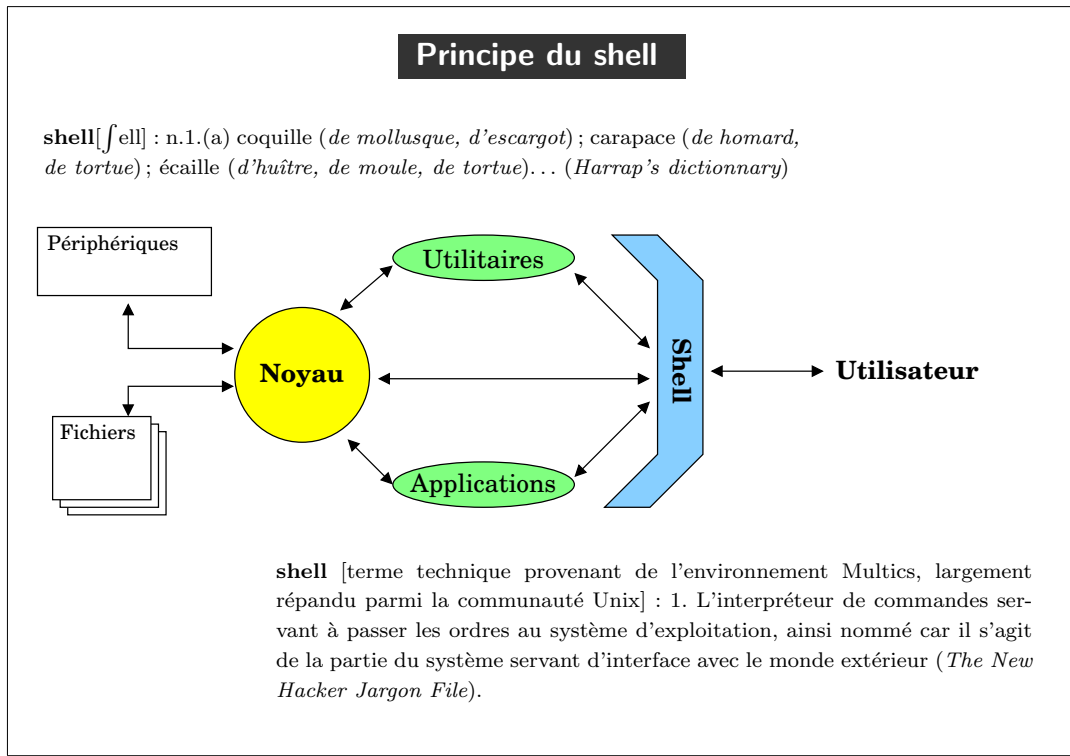
Aucune partie de ce cours ne peut être reproduite ou transmise à quelque fin ou par quelque moyen que ce soit, électronique ou mécanique, sans l'autorisation expresse et écrite de l'auteur.

*Ecriture de scripts shell***Programmation  
par scripts**

Christophe Blaess

– <b>Introduction et historique</b> .....	<b>3</b>
– Principe du shell .....	3
– Évolution des shells Unix .....	4
– <b>Principaux types de shells</b> .....	<b>6</b>
– Shells de type Bourne .....	6
– Shells de type C .....	7
– Normalisation .....	8
– <b>Exécution des scripts</b> .....	<b>12</b>
– Interprétation et compilation .....	12
– Editeurs de texte .....	13
– Invocation de l'interpréteur .....	14
– Invocation directe du script .....	17
– Ligne shebang .....	18
– <b>Travaux pratiques</b> .....	<b>19</b>
– Présentation .....	19
– Exercices .....	21





Voir aussi : Le Jargon File (<http://catb.org/esr/jargon/>) d'Eric S. Raymond.

**Évolution des shells Unix****1970 : Ken Thompson et Denis Ritchie (AT&T Bell Labs)**

Interpréteur de commandes simplifié intégré directement dans le noyau Unix.

**1973 : Steve Bourne (AT&T Bell Labs)**

Interpréteur hors du noyau, dans l'espace utilisateur : `/bin/sh`.

**1978 : William Joy (Université de Berkeley)**

Shell au comportement interactif plus agréable : `csh` (*C-Shell*).

**1983-1993 : David Korn (AT&T Bell Labs)**

Puissance interactive du C-shell et performances des scripts Bourne : `ksh` (*Korn Shell*).

**1985 : Brian Fox, puis Chet Ramey (Free Software Foundation)**

Shell libre et puissant pour le projet Gnu : `bash` (*Bourne Again Shell*).

**Voir aussi :**

*La plupart des documents suggérés dans cette section sont disponibles sous forme électronique sur le CD-rom accompagnant la formation.*

**Articles**

- `Ritchie_1974.pdf` : Dennis Ritchie & Ken Thompson – "*The UNIX Time-Sharing System*"
- `Joy_1978.pdf` : William Joy – "*An introduction to the C shell*"
- `Ritchie_1979.html` : Dennis Ritchie – "*Evolution of the Unix Time-Sharing System*"
- `Korn_1993.html` : David Korn, Charles Nothrup, Jeffery Korn – "*The New Korn Shell*"
- `Christiansen_1995.txt` : Tom Christiansen – "*CSH programming considered harmful*"
- `Hauben_1996.txt` : Hauben – "*On the Early History and Impact of Unix Tools to Build the Tools for a New Millenium*"

**F.A.Q.**

- `unix-faq/shell/shell-differences` : "*UNIX shell differences and how to change your shell*"
- `unix-faq/shell/bash` : "*The BASH F.A.Q.*"

**Web**

- **The Free Software Foundation** (<http://www.fsf.org>)

**1987 : Eric Gisin puis Michael Rendell**

Shell de type Korn libre : `pdksh` (*Public Domain Korn Shell*).

**1987 : Paul Placeway, Christos Zoulas (Cornell University)**

Shell corrigeant les défauts du C-Shell : `tcsh` (*Tenex-C-Shell*)

**1989 : Kenneth Almquist**

Shell minimal pour système sur disquettes : `ash` (*Almquist Shell*).

**1992 : Paul Falstad**

Shell alliant compatibilité Ksh-88 et Bash : `zsh` (*Z Shell*).

Interpréteur `COMMAND.COM` du Ms-Dos et fichiers `.BAT`.

**Voir aussi :****F.A.Q.**

– `unix-faq/shell/zsh` : "Z-shell F.A.Q."

**Web**

– TCSH Home Page (<http://www.tcsh.com/doc/>)

## Shells de type Bourne

Symbole d'invite : \$

**sh** : le shell original écrit par Steve Bourne, présent sur tous les Unix propriétaires (*AIX, HPUX, Irix, OSF/1, SCO, Solaris, Ultrix...*)

**ksh** : le shell Korn présent sur la plupart des Unix propriétaires et sous Windows NT. Depuis 2000, Ksh 93 est disponible sur les Unix libres.

**bash** : le shell de la FSF, disponible pour tous les Unix propriétaires, et installé sur tous les Unix libres - sous Linux il s'agit du shell par défaut. Disponible sous MacOS X et Windows (projet *Cygwin*).

**pdksh** : une version libre du shell Korn, disponible pour de nombreux Unix libres et propriétaires.

### Utilisation interactive

<b>Bash ou Ksh</b>	<b>set -o emacs</b>	<b>set -o vi</b>
Reculer dans l'historique	Ctrl-p (↑)	k
Avancer dans l'historique	Ctrl-n (↓)	j
Début de ligne	Ctrl-a (^)	0
Fin de ligne	Ctrl-e (Fin)	\$
Avancer d'un caractère	Ctrl-f (→)	l
Reculer d'un caractère	Ctrl-b (←)	h
Saut au mot suivant	Alt-f	w
Retour au mot précédent	Alt-b	b
Insérer littéralement	Ctrl-v	Ctrl-v
Effacer caractère précédent	Ctrl-h (←=)	X
Effacer caractère suivant	Ctrl-d (Suppr)	x
Effacer jusqu'à la fin	Ctrl-k	D

<b>Bash</b>	<b>set -o emacs</b>	<b>set -o vi</b>
Demande de complétion	↳	↳
Liste des complétions	↳↳	↳↳

<b>Ksh</b>	<b>set -o emacs</b>	<b>set -o vi</b>
Demande de complétion	Echap. Echap.	↳
Liste des complétions	Echap.?	↳↳

Pour Ksh en mode *vi*, activer l'option `set -o vi-tabcomplete`.

## Shells de type C

Symbole d'invite : % ou >

`csh` : le shell C original de Bill Joy, présent de nos jours sur de nombreux Unix propriétaires.

`tcsh` : version libre et améliorée du shell C, disponible sur la plupart des Unix propriétaires et libres, ainsi que sous Windows NT.

Les shells de type C sont couramment employés en utilisation interactive, mais très peu appropriés pour l'écriture de scripts.

### Utilisation interactive

	Tcsh
Reculer dans l'historique	<code>Ctrl-p</code> (↑)
Avancer dans l'historique	<code>Ctrl-n</code> (↓)
Début de ligne	<code>Ctrl-a</code> (↶)
Fin de ligne	<code>Ctrl-e</code> (Fin)
Avancer d'un caractère	<code>Ctrl-f</code> (→)
Reculer d'un caractère	<code>Ctrl-b</code> (←)
Saut au mot suivant	<code>Alt-f</code>
Retour au mot précédent	<code>Alt-b</code>
Insérer littéralement	<code>Ctrl-v</code>
Effacer caractère précédent	<code>Ctrl-h</code> (⇐)
Effacer caractère suivant	<code>Ctrl-d</code>
Effacer jusqu'à la fin	<code>Ctrl-k</code>
Complétion	<code>↩ / ↩↩</code>

**Voir aussi :**

**Pages de manuel :** `csh(1)`, et `tcsh(1)`.

**Fiche-conseil**

– Fiche d'aide à la conversion de script pour shell C en script pour shell Bourne

**Articles**

– `Christiansen_1995.txt` : Tom Christiansen – "*CSH programming considered harmful*"

## Normalisation

Les normes apportent une garantie importante de portabilité :

- entre systèmes Unix différents ;
- entre diverses versions du même système.

Plusieurs standards concernent le shell et ses outils :

- **Posix** : référence IEEE Std 1003.2 (*Posix.2*) ;
- **Single Unix Specifications v.2** : Open Group SUS v.2 ;
- **Single Unix Specifications v.3** : Open Group + IEEE (*SUSv3*).

Les shells Bash et Korn sont conformes à SUSv3.

**Voir aussi :**

**Normes**

- <doc/susv3/index.html> : Spécifications Single unix version 3

**Web**

- **Open Group Home Page** (<http://www.opengroup.org/>)

**AIX 4.3.2**

```

/bin          -> /usr/bin
/usr/bin/sh   -> /usr/bin/ksh
/usr/bin/bsh  (Bourne)
/usr/bin/csh  (C)
/usr/bin/ksh  (Korn 88)
/usr/dt/bin/dtksh (Korn 93)

```

**HP/UX 11.22**

```

/bin          -> /usr/bin
/usr/bin/sh   -> /usr/bin/ksh
/usr/bin/posix/sh -> /usr/bin/sh
/usr/bin/csh  (C)
/usr/bin/ksh  (Korn 88)

```

**Irix 6.5**

```

/bin          -> /usr/bin
/usr/bin/*sh  -> /sbin/*sh
/sbin/sh      -> /sbin/ksh
/sbin/bsh     (Bourne)
/sbin/csh     (C)
/sbin/ksh     (Korn 88)

```

**Linux Fedora**

```

/bin/sh       -> /bin/bash
/bin/csh      -> /bin/tcsh
/bin/ash      -> (Korn 88)
/bin/bash     (Bash)
/bin/ksh      (Pdksh, Korn 88)
/bin/tcsh     (C)

```

**OSF/1 v.5**

```

/bin          -> /usr/bin
/usr/bin/posix/sh -> /usr/bin/ksh
/usr/bin/sh   (Bourne)
/usr/bin/csh  (C)
/usr/bin/ksh  (Korn 88)
/sbin/sh     (Bourne static)

```

**Solaris 9**

```

/bin          -> /usr/bin
/usr/bin/sh   (Bourne)
/usr/bin/csh  (C)
/usr/bin/ksh  (Korn 88)
/usr/dt/bin/ksh (Korn 93)

```

**Web**

– Various system shells (<http://www.in-ulm.de/~mascheck/various/shells>)

**A vous...**

Utilisez la commande `uname -a` pour déterminer le type de système d'exploitation dont vous disposez :

Vérifiez la présence des shells décrits précédemment :

Shell	Présent ?	Lien symbolique ?	Destination ?
/bin/sh			
/bin/csh			
/bin/ksh			

La commande `chsh` permet de préciser le shell employé à la connexion de l'utilisateur.

**Voir aussi :****Manuel Unix**

- `chsh(1)`
- `uname(1)`

Outre le shell, SUSv3 normalise :

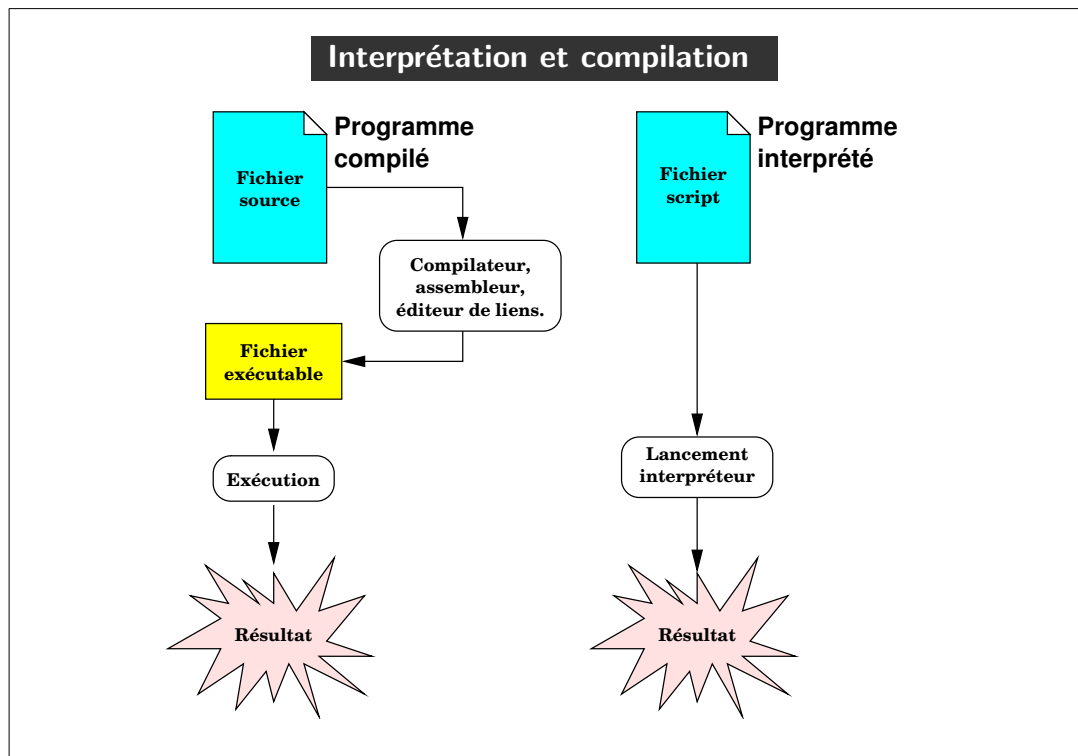
Les outils de manipulation de fichiers : `ls`, `cp`, `mv`, `rm`, `mkdir`,  
`rmdir`, `touch`, `ln`, `basename`, `dirname`...

Les utilitaires de traitement de texte : `cat`, `grep`, `find`, `sort`,  
`head`, `tail`, `wc`, `csplit`, `diff`, `file`, `fmt`, `tr`, `lpr`...

Les outils système de gestion des processus : `at`, `date`, `nice`, `kill`,  
`ps`, `stty`...

De nombreux utilitaires annexes du shell : `bc`, `expr`, `false`, `true`,  
`echo`, `printf`...

Des langages de script : `sed`, `awk`.



message.c

```
#include <stdio.h>

int main(void)
{
    printf("Bonjour !\n");
    return 0;
}
```

message.sh

```
echo "Bonjour"
```

## Editeurs de texte

L'essentiel du travail de développement des scripts se fait dans la fenêtre de l'éditeur de texte.

Il est important de choisir un éditeur adapté à un travail régulier et prolongé.

Quelques points à considérer :

- disponibilité sur différentes plates-formes,
- facilité d'apprentissage,
- ergonomie, (extension, coloration syntaxique...),
- esthétique et perception.

**\$ emacs mon\_script.sh**

```

# encrypted but /var/lib/random-seed is needed to initialize swap.
init_crypto() {
  local have_random dst src key opt mode owner params makeswap skip arg opt
  local param value rc ret nks2fs adir

  ret=0
  have_random=1
  while read dst src key opt; do
    [ -z "$dst" -o "$dst#/" != "$dst" ] && continue
    [ -b /dev/urandom/$dst ] && continue
    if [ "$have_random" = 0 ] && key_is_random "$key"; then
      continue
    fi
    if [ -n "$key" -a "$skkey" != "xnone" ]; then
      if test -e "$key"; then
        mode=$(ls -l "$key" | cut -c 5-10)
        owner=$(ls -l "$key" | cut -c 1-4)
        if [ "$mode" != "-----" ] && ! key_is_random "$key"; then
          echo "$INSECURE MODE FOR $key"
        fi
        if [ "$owner" != root ]; then
          echo "$INSECURE OWNER FOR $key"
        fi
      fi
    fi
  done
}

rc=$?
if [ "$rc" = 0 ]; then
  echo "Script executed successfully"
else
  echo "Script execution failed with error code $rc"
fi

```

**\$ gvim mon\_script.sh**

```

# encrypted but /var/lib/random-seed is needed to initialize swap.
init_crypto() {
  local have_random dst src key opt mode owner params makeswap skip arg opt
  local param value rc ret nks2fs adir

  ret=0
  have_random=1
  while read dst src key opt; do
    [ -z "$dst" -o "$dst#/" != "$dst" ] && continue
    [ -b /dev/urandom/$dst ] && continue
    if [ "$have_random" = 0 ] && key_is_random "$key"; then
      continue
    fi
    if [ -n "$key" -a "$skkey" != "xnone" ]; then
      if test -e "$key"; then
        mode=$(ls -l "$key" | cut -c 5-10)
        owner=$(ls -l "$key" | cut -c 1-4)
        if [ "$mode" != "-----" ] && ! key_is_random "$key"; then
          echo "$INSECURE MODE FOR $key"
        fi
        if [ "$owner" != root ]; then
          echo "$INSECURE OWNER FOR $key"
        fi
      fi
    fi
  done
}

rc=$?
if [ "$rc" = 0 ]; then
  echo "Script executed successfully"
else
  echo "Script execution failed with error code $rc"
fi

```

**\$ nedit mon\_script.sh**

```

# encrypted but /var/lib/random-seed is needed to initialize swap.
init_crypto() {
  local have_random dst src key opt mode owner params makeswap skip arg opt
  local param value rc ret nks2fs adir

  ret=0
  have_random=1
  while read dst src key opt; do
    [ -z "$dst" -o "$dst#/" != "$dst" ] && continue
    [ -b /dev/urandom/$dst ] && continue
    if [ "$have_random" = 0 ] && key_is_random "$key"; then
      continue
    fi
    if [ -n "$key" -a "$skkey" != "xnone" ]; then
      if test -e "$key"; then
        mode=$(ls -l "$key" | cut -c 5-10)
        owner=$(ls -l "$key" | cut -c 1-4)
        if [ "$mode" != "-----" ] && ! key_is_random "$key"; then
          echo "$INSECURE MODE FOR $key"
        fi
        if [ "$owner" != root ]; then
          echo "$INSECURE OWNER FOR $key"
        fi
      fi
    fi
  done
}

rc=$?
if [ "$rc" = 0 ]; then
  echo "Script executed successfully"
else
  echo "Script execution failed with error code $rc"
fi

```

## Invocation de l'interpréteur

On peut exécuter un script en appelant explicitement l'interpréteur :

```
$ sh mon_script.sh
$ . mon_script.sh
$ source mon_script.sh
```

La commande `source` est spécifique à Bash, elle est synonyme de la commande « point » (`.`) normalisée par SUSv3.

```
$ sed mon_script.sed
$ awk -f mon_script.awk
```

Sous Unix, les suffixes (`.sh`, `.sed`, `.awk`, `.pl...`) n'ont qu'un rôle d'information pour l'utilisateur, mais n'ont aucune signification pour le système.

**A vous...**

Utilisez l'un des éditeurs disponibles sur votre système pour écrire le script suivant :

test.sh

```
echo "Je suis le script de test"
```

Exécutez-le successivement avec les commandes suivantes :

```
$ sh test.sh
```

```
$ . test.sh
```

Voyez-vous une différence ?

Modifiez le script précédent pour ajouter une ligne ainsi :

test.sh

```
echo "Je suis le script de test"  
exit 0
```

Ré-essayez les deux types d'invocation. Voyez-vous une différence ?

Pour pouvoir invoquer le script directement par son nom, il faut d'abord que le noyau sache où le trouver.

Essayez d'appeler directement :

```
$ test.sh
```

Quel est le résultat ?

On peut placer le script dans un endroit connu par le noyau. Ce dernier ira le chercher dans l'ensemble des répertoires indiqués dans la variable d'environnement `PATH`.

Affichez le contenu de votre variable d'environnement :

```
$ echo $PATH
```

Quels sont les répertoires parcourus pour rechercher les fichiers ?

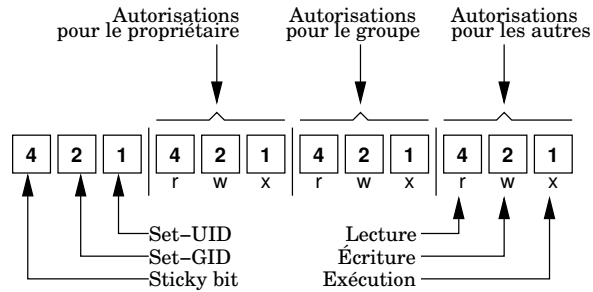
On peut indiquer explicitement que le fichier se trouve dans le répertoire en cours :

```
$ ./test.sh
```

Quel est le résultat ?

### Invocation directe du script

Pour que le noyau sache que le fichier contient des instructions et non des données, il faut l'indiquer en rendant le fichier *exécutable*.



Classiquement on utilise la commande

```
$ chmod 755 test.sh
```

pour rendre un script exécutable.

Après avoir modifié les droits de votre script, vérifiez-les avec `ls -l` et puis appelez-le.

**Voir aussi :**

**Manuel Unix**

– `chmod(1)`

## Ligne shebang

Par défaut le shell considère que le script doit être interprété par `/bin/sh`.

Pour tous les autres scripts (`/bin/ksh`, `/bin/sed`, `/bin/awk`, `/usr/bin/perl...`) il est indispensable de préciser quel est l'interpréteur à utiliser pour exécuter le script.

Par convention, si les deux premiers octets du fichier sont `#` et `!`, le reste de la ligne représente l'interpréteur à utiliser.

`script.sh`

```
#!/bin/sh  
...
```

`script.sed`

```
#!/bin/sed -f  
...
```

`script.awk`

```
#!/bin/awk -f
```

`script.pl`

```
#!/usr/bin/perl -w
```

## Travaux pratiques

### Présentation

Des travaux pratique ont été prévus en accompagnement de chaque module de la formation

*Programmation Shell et Langages de Script*

Ces travaux pratiques sont présentés sous forme d'exercice dont l'énoncé est donné sur une page HTML permettant un accès direct à un fichier source offrant un squelette de réponse, dans lequel les points principaux ont été remplacés par des commentaires, et un fichier source de solution complète.

On a mentionné une estimation de la difficulté de résolution des exercices, afin de permettre à chacun de choisir en fonction de son niveau et du temps disponible.

Vous trouverez ci-après les indications nécessaires pour installer les fichiers de travaux pratiques à partir du CD-Rom fourni aux stagiaires.

#### Première étape Copie des fichiers de travail

Insérez dans le lecteur de votre machine le CD-Rom qui vous a été fourni lors de la formation.

Suivant la configuration du système, il se peut qu'une fenêtre s'ouvre automatiquement avec le gestionnaire de fichiers de l'environnement (Kde, Gnome...). Dans ce cas utilisez cet outil pour copier le fichier `tp.tar` depuis le CD-Rom vers votre répertoire personnel.

Sinon ouvrez une fenêtre *X-Term* et tapez les commandes suivantes

```
mount /mnt/cdrom
```

```
cp /mnt/cdrom/tp.tar .
```

La première commande *monte* le CD-Rom pour rendre son contenu accessible, en lecture sous le répertoire `/mnt/cdrom`.

La seconde commande copie le fichier `tp.tar` se trouvant dans le répertoire représentant le CD-Rom, dans le répertoire courant.

## Deuxième étape Décompression du fichier archive

Dans une fenêtre *X-Term*, tapez la commande :

```
tar -xf tp.tar
```

Cette commande développe l'archive pour installer tous les fichiers utiles.

L'arborescence ainsi obtenue est la suivante :

```
doc/articles/  
doc/FAQ/  
doc/man/  
doc/memo/  
doc/photos/  
doc/susv3/  
src/  
tp/01_programmation_scripts/  
tp/02_fonctionnement_shell/  
tp/03_deroulement_script/  
tp/04_commandes/  
tp/06_er_grep/  
tp/07_sed_awk/
```

## Troisième étape Contenu des travaux pratiques

Pour explorer le contenu des documents installés, invoquez :

```
firefox &
```

puis utilisez le menu "*Fichier*", l'option "*Ouvrir un fichier*", pour charger le fichier `index.html`.

On trouvera dans le sous répertoire `doc/` des documents supplémentaires divers concernant les sujets abordés durant la formation. Ces documents se présentent sous forme de texte brut, de pages *html* ou de fichiers *pdf*.

Dans le répertoire `tp/` se trouve un fichier `index.html` donnant accès aux énoncés et fichiers source des travaux pratiques. Vous pouvez le consulter avec *Firefox* ou l'explorateur de votre environnement de travail. Chaque module de la formation est représenté par un sous-répertoire contenant un fichier `index.html` particulier, les fichiers des squelettes et des solutions des exercices.

Un sous-répertoire `src/` au même niveau que `doc/` et `tp/` contient des exemples d'applications diverses en rapport avec les thèmes de la formation.

**Exercices****Exercice numéro 1**

Interpréteurs de commandes disponibles

**Exercice numéro 2**

Editeurs de texte

**Exercice numéro 3**

Invocation d'un script

**Exercice numéro 4**

Configuration du chemin de recherche

**Exercice numéro 5**

Utilité de la variable PATH

**Exercice numéro 6**

Variable PATH et cheval de Troie

**Exercice numéro 7**

Shell interactif et shell de script

**Exercice numéro 8**

Utilité de la ligne *Shebang*



**Index**

/bin/ash	5	Joy (Bill)	7
/bin/bash	4, 6, 8, 9	Joy (William)	4, 5
/bin/csh	4, 7, 9, 9, 10	Korn (David)	4
/bin/ksh	4, 6, 8, 9, 9, 10, 18	Korn (shell)	4, 6, 8
/bin/pdksh	5, 6, 9	Linux (Unix)	6, 9
/bin/sh	4, 6, 9, 9, 10, 14, 18	MacOs (Unix)	6
/bin/tcsh	5, 7, 9	Open Group	8
/bin/zsh	5	OSF/1 (Unix)	6, 9
AIX (Unix)	6, 9	PATH (variable)	13, 16
Almquist (Kenneth)	5	Placeway (Paul)	5
awk (outil)	11, 14	Posix (norme)	8
Bash (shell)	4, 6, 8	Ramey (Chet)	4
Bourne (shell)	4, 6	Rendell (Michael)	5
Bourne (Steve)	4, 6	Ritchie (Denis)	4
C (shell)	4, 5, 7	Sco (Unix)	6
chmod(1)	17	sed (outil)	11, 14
chsh(1)	10, 10	shebang (ligne)	15, 18
command.com	5	shell (définition)	3
Cygwin	6	Solaris (Unix)	6, 9
Falstad (Paul)	5	source (commande)	14
Fox (Brian)	4	SUSv3 (norme)	8, 8, 11, 14
FSF	4	Tcsh (shell)	5
Gisin (Eric)	5	Thompson (Ken)	4
GNU	4	Ultrix (Unix)	6
HPUX (Unix)	6, 9	uname(1)	10, 10
IEEE	8	Zoulas (Christos)	5
Irix (Unix)	6, 9		