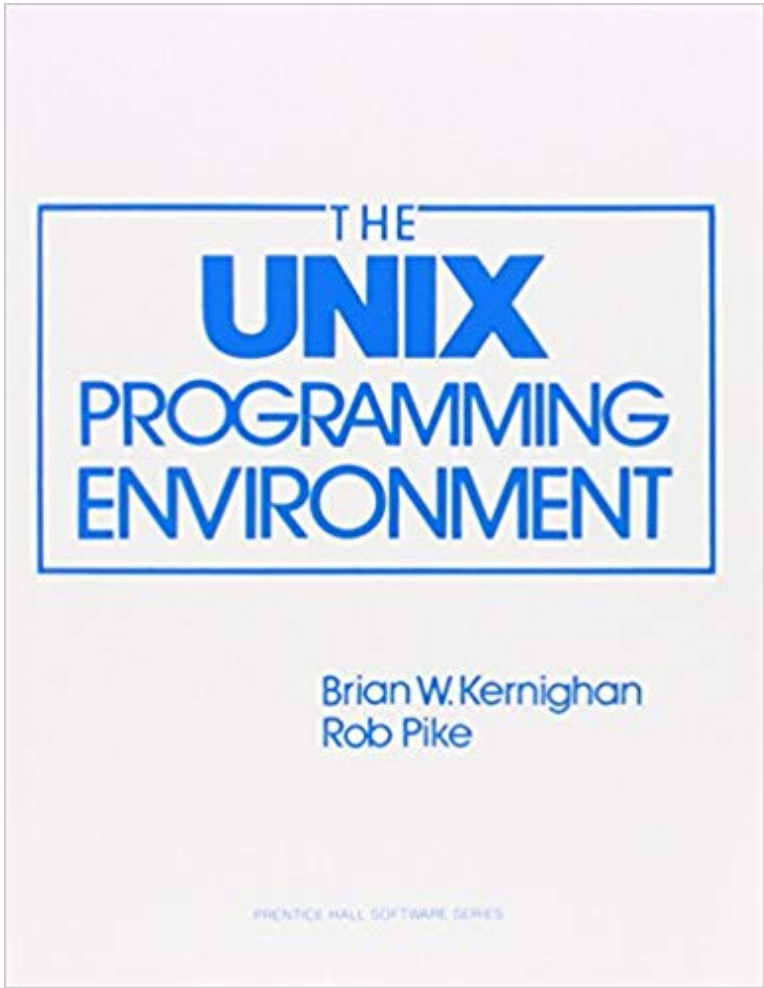
Initiation à l'environnment Unix
CM2 : La petite cuisine du shell

Pierre Rousselin
rousselin@univ-paris13.fr

Université Sorbonne Paris Nord
L1 informatique et double-licence
septembre 2024



Kernighan et Pike (de Bell labs), 1984

Because it must satisfy both the interactive and programming aspects of command execution, [the shell] is a strange language, shaped as much by history as by design.

Comme il doit satisfaire à la fois les aspects interactifs et de programmation de l'exécution de commandes, le shell est un langage étrange, façonné autant par l'histoire que par des décisions réfléchies.

Petite cuisine

Le shell peut transformer la ligne de commande entrée *avant même d'exécuter la commande* en faisant les transformations suivantes dans cet ordre :

1. Développement du tilde `~` (*tilde expansion*)
2. Développement de variable `$var` ou `${var}` (*parameter expansion*), développement arithmétique `$((calcul))` (*arithmetic expansion*) et substitution de commande `$(commande)` (*command substitution*)
3. Séparation des champs (*field splitting*)
4. Développement de noms de chemins `* ? [...]` (*pathname expansion*)
5. Suppression des caractères inhibiteurs (*quote removal*)

Petite cuisine (2)

On peut inhiber les caractères spéciaux (et en particulier empêcher les développements) avec les caractères inhibiteurs :

- contre-oblique (*backslash*) `\`
- apostrophes (*single quotes*) `'...'`
- guillemets anglais (*double quotes*) `"..."`

Voir plus loin pour les règles d'inhibition.
Pour voir les commandes lancées par le shell après sa petite cuisine, on peut activer l'option `-x` du shell avec la commande

```
set -x
```

(désactiver avec `set +x`).

Variables du shell

Un nom de variable peut contenir

- des lettres majuscules ou minuscules (non accentuées) ;
- des chiffres décimaux ;
- le caractère `_` (blanc souligné ou *underscore*) ;
- ne doit pas commencer par un chiffre.

Exemples :

`ma_var` `_RESULTAT` `lol` `arg_1`

Contre-exemples :

`3petits_chats` `résultat` `un-deux`

Variables du shell

On affecte une valeur à une variable avec la syntaxe

```
nom_de_variable=valeur_affectée
```

sans espace autour du signe =

On accède à la valeur affectée avec l'une des syntaxes

```
$nom_de_variable      ${nom_de_variable}
```

Exemples :

```
$ rep=/home/pierre/depots/ieunix/
$ cd $rep
$ wc -c $rep/cm2/unix_cm2.tex
10457 /home/pierre/depots/ieunix/unix_cm2.tex
$ rep=/usr/include
$ echo $rep
/usr/include
```

C'est le développement des variables.

Variables du shell

- Une variable contient toujours une chaîne de caractère (le shell n'est pas un langage typé).
- Une variable est toujours globale.
- Une variable peut être vide (*empty*), ou non définie (*unset*).
- Le développement d'une variable non définie donne une chaîne vide, et pas une erreur !
- Un caractère qui n'est pas valide pour identifier une variable délimite le nom de la variable (mais il est plus lisible dans ce cas d'utiliser la syntaxe `${...}`).

Variables du shell

- Une variable contient toujours une chaîne de caractère (le shell n'est pas un langage typé).
- Une variable est toujours globale.
- Une variable peut être vide (*empty*), ou non définie (*unset*).
- Le développement d'une variable non définie donne une chaîne vide, et pas une erreur !
- Un caractère qui n'est pas valide pour identifier une variable délimite le nom de la variable (mais il est plus lisible dans ce cas d'utiliser la syntaxe `${...}`).

```
$ rad=chant
$ echo je $rade tu $rades elle $rade
je tu elle
$ echo je ${rad}e tu ${rad}es elle ${rad}e
je chante tu chantes elle chante
$ echo $radé $rad-0 $rad/ $radeur.euse
chanté chant-0 chant/ .euse
```

Développement arithmétique

Le shell sait faire des calculs sur des nombres entiers avec la syntaxe

```
$(( ... ))
```

Le mot `$(( ... ))`, où `...` ne contient que des nombres entiers et des opérations sur ces nombres est *remplacé par le shell* par le résultat du calcul.

Développement arithmétique

Le shell sait faire des calculs sur des nombres entiers avec la syntaxe

```
$(( ... ))
```

Le mot `$(( ... ))`, où `...` ne contient que des nombres entiers et des opérations sur ces nombres est *remplacé par le shell* par le résultat du calcul.

```
$ echo 7 fois 8 fait $(( 7 * 8 ))
7 fois 8 fait 56
$ echo $((3 * 8)) / 5 = $(( (3 * 8) / 5 ))
24 / 5 = 4
$ i=3
$ i=$(($i + 1)); echo $i
4
```

Substitution de commande

`commande` ou $(commande)

Le shell exécute la commande `commande` et remplace `$(commande)` (syntaxe préférée) ou `` `commande` `` (ancienne syntaxe, moins lisible) par ce qu'afficherait la commande sur le terminal.

```
$ echo le système est $(uname)
Le système est Linux
$ echo la machine est $(uname -s)
La machine est x86_64
$ rep=$(pwd)
$ cd /
$ cd $rep
```

Caractère spéciaux pour le shell

Caractères spéciaux pour le shell :

`;` `<newline>` `&` `|` `<` `>` `$` `` ` `` `<space>` `<tab>` `'` `"` `\`

Caractères spéciaux dans certains contextes :

`*` `?` `[` `#` `~` `=` `%`

On dit qu'on les inhibe lorsqu'on leur rend leur sens littéral.

- Inhibition par contre-oblique `\`
- inhibition entre apostrophes (*single quotes*) `'...'`
- inhibition entre guillemets anglais (*double quotes*) `"..."`

Inhibition par contre-oblique

La contre-oblique \ inhibe le caractère qui la suit puis est supprimée par le shell.

Inhibition par contre-oblique

La contre-oblique \ inhibe le caractère qui la suit puis est supprimée par le shell.

```
$ echo \* \' \" \$ \? \>
* ' " $ ? > \
$ mkdir Ma\ Musique
$ rmdir Ma Musique
rmdir: impossible de supprimer Ma ...
rmdir: impossible de supprimer Musique ...
$ rmdir Ma\ Musique
$ echo A\ \ \ B
A   B
```

Inhibition par contre-oblique

La contre-oblique \ inhibe le caractère qui la suit puis est supprimée par le shell.

```
$ echo \* \' \" \$ \? \>
* ' " $ ? > \
$ mkdir Ma\ Musique
$ rmdir Ma Musique
rmdir: impossible de supprimer Ma ...
rmdir: impossible de supprimer Musique ...
$ rmdir Ma\ Musique
$ echo A\ \ \ B
A   B
```

Une seule subtilité : \ suivi directement par une fin de ligne est simplement supprimé par le shell (permet de séparer une ligne de commande en plusieurs lignes).

```
$ echo A B\
> C D
A BC D
```

Inhibition entre apostrophes

Entre apostrophes, tous les caractères ont leur sens littéral...
... sauf l'apostrophe qui met fin à l'inhibition.

Inhibition entre apostrophes

Entre apostrophes, tous les caractères ont leur sens littéral...
... sauf l'apostrophe qui met fin à l'inhibition.

```
$ echo '" $ * ? ~'
" $ * ? ~
$ echo 'A
> B'
A
B
$ echo '"les fichiers :"' *
"les fichiers :" a.out prog.c
$ echo 'Pour afficher une '\'' on sort des '\'
Pour afficher une ' on sort des '
```

Inhibition entre guillemets anglais

Entre guillemets anglais, les seuls caractères spéciaux sont :

- `$ ` pour le développement de variable, le développement arithmétique et la substitution de commandes,
- `"` pour mettre fin à l'inhibition.
- `\` n'est spéciale que si elle précède `$ ` " ou `\`, auquel cas elle rend au caractère suivant son sens littéral puis est supprimée.

```
$ echo "*** $(pwd) ***"
*** /home/ada ***
$ echo "2 + 2 font      : $((2 + 2))"
2 + 2 font      : 4
$ echo "*** \$(pwd) ***"
*** $(pwd) ***
$ echo "2 + 2 font      : \$((2 + 2))"
2 + 2 font      : $((2 + 2))
$ echo "\"\$\`\\a\b"
"$`\a\b
```

Les limites de `echo`

La commande `echo` est la commande historique pour faire des sorties dans le shell, mais elle a souffert dès le début de problèmes de conception.

- Retour à la ligne automatique.
- Certaines versions de `echo` se sont mises à avoir une option `-n` pour ne pas avoir de retour à la ligne.
- Tandis que d'autres proposaient la séquence d'échappement `\c` pour l'enlever.
- Différentes versions trop fortement incompatibles, dès les années 1980...
- Un poème a été écrit dessus : « The Unix and the echo ».
- Le standard POSIX dit : « *Implementations shall not support any options.* » Presque aucune des implémentations de `echo` ne respectent le standard.

La commande `printf`

La commande `printf` est :

- standard et le standard est (plutôt) bien respecté ;
- plus puissante ;
- très agréable d'utilisation.

La commande `printf`

- `printf FORMAT [args]...`
- Pas de retour à la ligne automatique.
- Le format peut contenir :
  - des séquences d'échappement (`\n`, etc)
  - des spécifications de format (`%s`, `%d`, etc).
- Si plus d'arguments que de spécifications de format, le format est réutilisé !

```
$ printf 'Bonjour %s !\n' Alice Bob Charlie
Bonjour Alice !
Bonjour Bob !
Bonjour Charlie !
```

- Les spécificateurs de format ne correspondant pas à un argument sont remplacés par des valeurs par défaut raisonnables.

```
$ printf '%d %s %d %s\n' 23 salut
23 salut 0
```

Morale de l'histoire

Utilisez `printf` autant que possible plutôt que `echo`, surtout dans les scripts.

Premiers exemples

```
$ var="mon petit lapin"
$ printf '%s\n' "$var\n"
mon petit lapin
$ printf '%s\n' mon petit lapin
mon
petit
lapin
$ printf '%s\n' $var
```

Premiers exemples

```
$ var="mon petit lapin"
$ printf '%s\n' "$var\n"
mon petit lapin
$ printf '%s\n' mon petit lapin
mon
petit
lapin
$ printf '%s\n' $var
```

Question : `$var` $\longrightarrow$ 3 mots ou 1 seul ?

Premiers exemples

```
$ var="mon petit lapin"
$ printf '%s\n' "$var\n"
mon petit lapin
$ printf '%s\n' mon petit lapin
mon
petit
lapin
$ printf '%s\n' $var
```

Question : `$var` $\longrightarrow$ 3 mots ou 1 seul ? Réponse :

```
mon
petit
lapin
```

Premiers exemples

```
$ var="a bien du chagrin"
$ printf '%s\n' "$var"
```

Premiers exemples

```
$ var="a bien du chagrin"
$ printf '%s\n' "$var"
```

Question : `"$var"` $\longrightarrow$ 4 mots ou 1 seul ?

Premiers exemples

```
$ var="a bien du chagrin"
$ printf '%s\n' "$var"
```

Question : `"$var"` $\longrightarrow$ 4 mots ou 1 seul ? Réponse :

```
a bien du chagrin
```

Premiers exemples

```
$ var="a bien du chagrin"
$ printf '%s\n' "$var"
```

Question : `"$var"` $\longrightarrow$ 4 mots ou 1 seul ? Réponse :

```
a bien du chagrin
```

```
$ date
mar. avril  7 18:52:38 CEST 2020
$ printf "%s\n" $(date)
```

Premiers exemples

```
$ var="a bien du chagrin"
$ printf '%s\n' "$var"
```

Question : `"$var"` $\longrightarrow$ 4 mots ou 1 seul ? Réponse :

```
a bien du chagrin
```

```
$ date
mar. avril  7 18:52:38 CEST 2020
$ printf "%s\n" $(date)
```

Question : `$(date)` $\longrightarrow$ 6 mots ou 1 seul ?

Premiers exemples

```
$ var="a bien du chagrin"
$ printf '%s\n' "$var"
```

Question : `"$var"` $\longrightarrow$ 4 mots ou 1 seul ? Réponse :

```
a bien du chagrin
```

```
$ date
mar. avril  7 18:52:38 CEST 2020
$ printf "%s\n" $(date)
```

Question : `$(date)` $\longrightarrow$ 6 mots ou 1 seul ? Réponse :

```
mar.
avril
8
18:52:38
CEST
2020
```

Premiers exemples

```
$ var="a bien du chagrin"
$ printf '%s\n' "$var"
```

Question : `"$var"` $\longrightarrow$ 4 mots ou 1 seul ? Réponse :

```
a bien du chagrin
```

```
$ date
mar. avril  7 18:52:38 CEST 2020
$ printf "%s\n" $(date)
```

Question : `$(date)` $\longrightarrow$ 6 mots ou 1 seul ? Réponse :

```
mar.
avril
8
18:52:38
CEST
2020
```

```
$ printf '%s\n' "$(date)"
mar. avril  7 18:52:38 CEST 2020
```

Un développement peut donner plusieurs mots

- La séparation en champs ne peut se produire que sur les développements de variable, arithmétique et les substitutions de commandes.
- Ce mécanisme est inhibé entre guillemets anglais.
- Sans modification de votre part, les champs sont séparés par un ou plusieurs éléments de :
  - `<space>`
  - `<tab>`
  - `<newline>`

Un développement peut donner plusieurs mots

- La séparation en champs ne peut se produire que sur les développements de variable, arithmétique et les substitutions de commandes.
- Ce mécanisme est inhibé entre guillemets anglais.
- Sans modification de votre part, les champs sont séparés par un ou plusieurs éléments de :
  - `<space>`
  - `<tab>`
  - `<newline>`

Exemple :

```
$ var='a<tab> bc
> d   e'
$ printf '%s\n' $var
a
bc
d
e
```

La variable `IFS`

C'est la variable `IFS` pour *internal field separator* (séparateur interne de champ) qui contient les séparateurs des champs.

- Elle vaut au départ `<space><tab><newline>`.

La variable IFS

C'est la variable `IFS` pour *internal field separator* (séparateur interne de champ) qui contient les séparateurs des champs.

- Elle vaut au départ `<space><tab><newline>`.
- Mais vous pouvez la redéfinir :

```
$ IFS=-+
$ var=mon-petit+lapin
$ printf "%s\n" $var
mon
petit
lapin
$ printf "%s\n" "$var"
mon-petit+lapin
```

La variable IFS

C'est la variable IFS pour *internal field separator* (séparateur interne de champ) qui contient les séparateurs des champs.

- Elle vaut au départ <space><tab><newline>.
- Mais vous pouvez la redéfinir :

```
$ IFS=-+
$ var=mon-petit+lapin
$ printf "%s\n" $var
mon
petit
lapin
$ printf "%s\n" "$var"
mon-petit+lapin
```

- Si IFS est vide (IFS=), la séparation en champs n'a pas lieu (c'est souvent ce qu'on veut !)
- Si IFS est non définie (unset IFS), la séparation en champs se fait comme dans le cas par défaut (<space><tab><newline>).

La variable IFS

- Subtilité : les caractères « blancs » (espace, tabulation et nouvelle ligne) qui sont dans l'IFS sont ignorés lorsqu'ils sont en début, en fin de chaîne, ou à côté d'un autre séparateur.
- Alors que des caractères non-blancs (par exemple -) dans l'IFS en début ou en fin de chaîne ou à côté d'autres séparateurs non blancs produisent des champs vides.

```
$ IFS='+- '
$ var='   mon -  + petit    lapin   '
$ printf '%s\n' $var
mon


petit
lapin
```

Morale de l'histoire

- La séparation en champs a lieu dans les cas suivants :
  - développement de variable ou
  - développement arithmétique ou
  - substitution de commandes
  - qui n'apparaissent pas entre guillemets anglais (`""`) ;
  - ou encore lorsqu'on lit sur l'entrée standard avec la commande `read` (voir plus tard).

Morale de l'histoire

- La séparation en champs a lieu dans les cas suivants :
  - développement de variable ou
  - développement arithmétique ou
  - substitution de commandes
  - qui n'apparaissent pas entre guillemets anglais (`""`) ;
  - ou encore lorsqu'on lit sur l'entrée standard avec la commande `read` (voir plus tard).
- Il est très fréquent qu'on n'en veuille pas : penser à protéger ces développements par `""`.

Morale de l'histoire

- La séparation en champs a lieu dans les cas suivants :
  - développement de variable ou
  - développement arithmétique ou
  - substitution de commandes
  - qui n'apparaissent pas entre guillemets anglais (`""`) ;
  - ou encore lorsqu'on lit sur l'entrée standard avec la commande `read` (voir plus tard).
- Il est très fréquent qu'on n'en veuille pas : penser à protéger ces développements par `""`.
- La séparation en mots n'a jamais lieu :
  - à droite du signe `=` dans une affectation (comme par exemple dans `var=$truc` ou `la_date=$(date)`) ;
  - dans la chaîne et les motifs de la construction `case` (voir plus tard) ;
  - mais dans le doute vous pouvez toujours inhiber avec `""`.

Morale de l'histoire

- La séparation en champs a lieu dans les cas suivants :
  - développement de variable ou
  - développement arithmétique ou
  - substitution de commandes
  - qui n'apparaissent pas entre guillemets anglais (`""`) ;
  - ou encore lorsqu'on lit sur l'entrée standard avec la commande `read` (voir plus tard).
- Il est très fréquent qu'on n'en veuille pas : penser à protéger ces développements par `""`.
- La séparation en mots n'a jamais lieu :
  - à droite du signe `=` dans une affectation (comme par exemple dans `var=$truc` ou `la_date=$(date)`) ;
  - dans la chaîne et les motifs de la construction `case` (voir plus tard) ;
  - mais dans le doute vous pouvez toujours inhiber avec `""`.
- La séparation en champs est souvent mal comprise. C'est l'une des causes principales de bugs dans les scripts (ou commandes) shell.

Parenthèse sur `zsh`

- Rappel : `zsh` est un shell assez moderne et populaire qui est le shell par défaut sur les MacOS récents.
- Le shell `zsh` ne cherche pas à être conforme par défaut au standard POSIX, mais peut l'être avec certaines options.
- Par défaut, `zsh` n'active pas la séparation en champs, celle-ci doit être demandée explicitement avec la syntaxe `${=variable}`.

```
$ echo $0 # quel est mon shell ?
zsh
$ var="dans zsh"
$ printf '%s\n' $var
dans zsh
$ printf '%s\n' ${=var}
dans
zsh
```

Voir `man zshexpn` pour plus d'informations.

Options pour rendre `zsh` standard

- La séparation en champs est contrôlée par l'option `SH_WORD_SPLIT`.

```
$ set -o SH_WORD_SPLIT
$ printf '%s\n' $var
dans
zsh
$ set +o SH_WORD_SPLIT
$ printf '%s\n' $var
dans zsh
```

- Invoquer `zsh` de la façon suivante

```
$ zsh --emulate sh
```

  rendra `zsh` (presque) conforme au standard POSIX.

- On peut également entrer la commande

```
$ emulate sh
```

  directement dans `zsh`.

- Voir `man zshoptions` pour plus d'informations.