

CI 1 – Shell et Script Shell

CSC3102 - Introduction aux systèmes d'exploitation
Elisabeth Brunet

Porte d'entrée dans le système : Le Terminal (ou console)



- Interface entre l'utilisateur et le système d'exploitation
 - Dialogue à base d'appels de commandes interprétées par le shell
 - Prompt = Invite de commande (configurable)
 - Différentes versions : xterm (la plus utilisée), Konsole, rxvt, etc.

Shell

- Interpréteur de commandes
 - Lit les commandes saisies par l'utilisateur dans le terminal,
 - Les interprète,
 - Les exécute,
 - Et transmet les résultats à l'utilisateur.
 - CTRL-C pour arrêter en cas de problème
- Langage script shell
 - structures algorithmiques classiques : if, while, for, etc.
- Variables d'environnement shell
- Shells les plus utilisés : bash, sh (, zsh, csh...)

Commandes interprétées par le shell

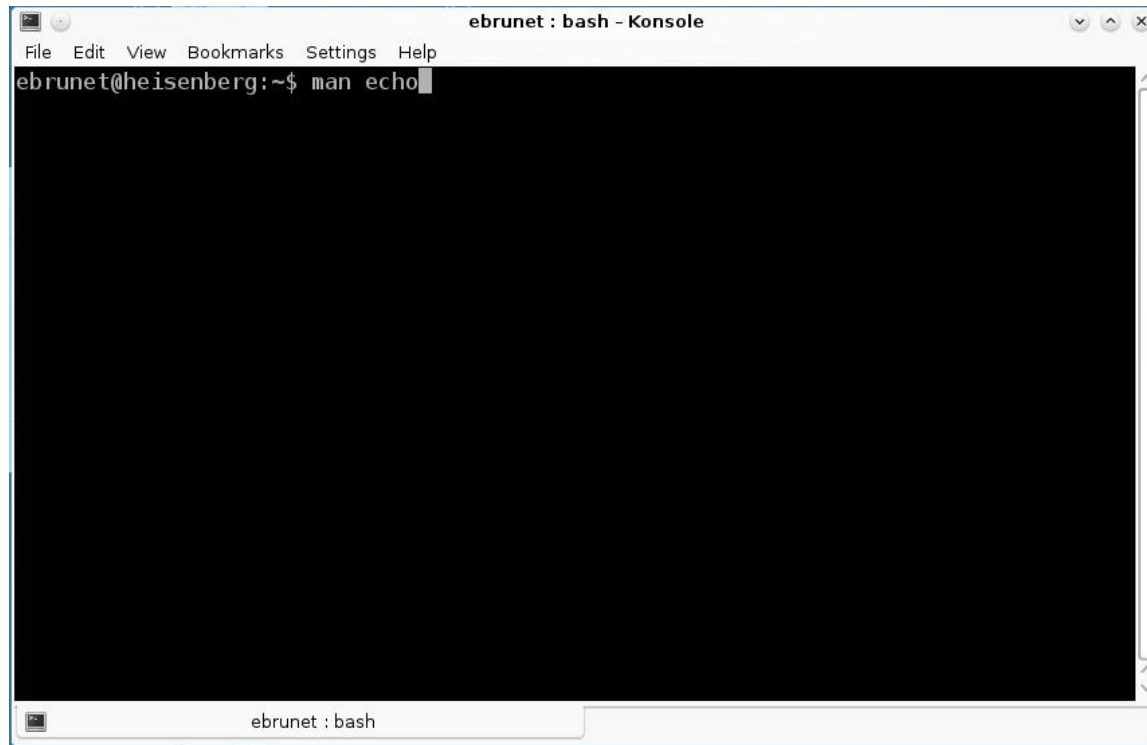
- Fournies par le système d'exploitation, par des logiciels installés en supplément ou créées par l'utilisateur
- Syntaxe
 - `<cmd> [-options] [arguments]`
 - `< >` → champ à fournir obligatoirement
 - `[]` → champ optionnel
- Manuel des commandes
 - Section 1 de la documentation Unix
 - Commande `man`
 - `man [1] <cmd>` : préciser la section si la cmd existe par ailleurs
 - Recherche dans le manuel : commande `apropos`

Première commande : echo



```
ebrunet : bash - Konsole
File Edit View Bookmarks Settings Help
ebrunet@heisenberg:~$
```

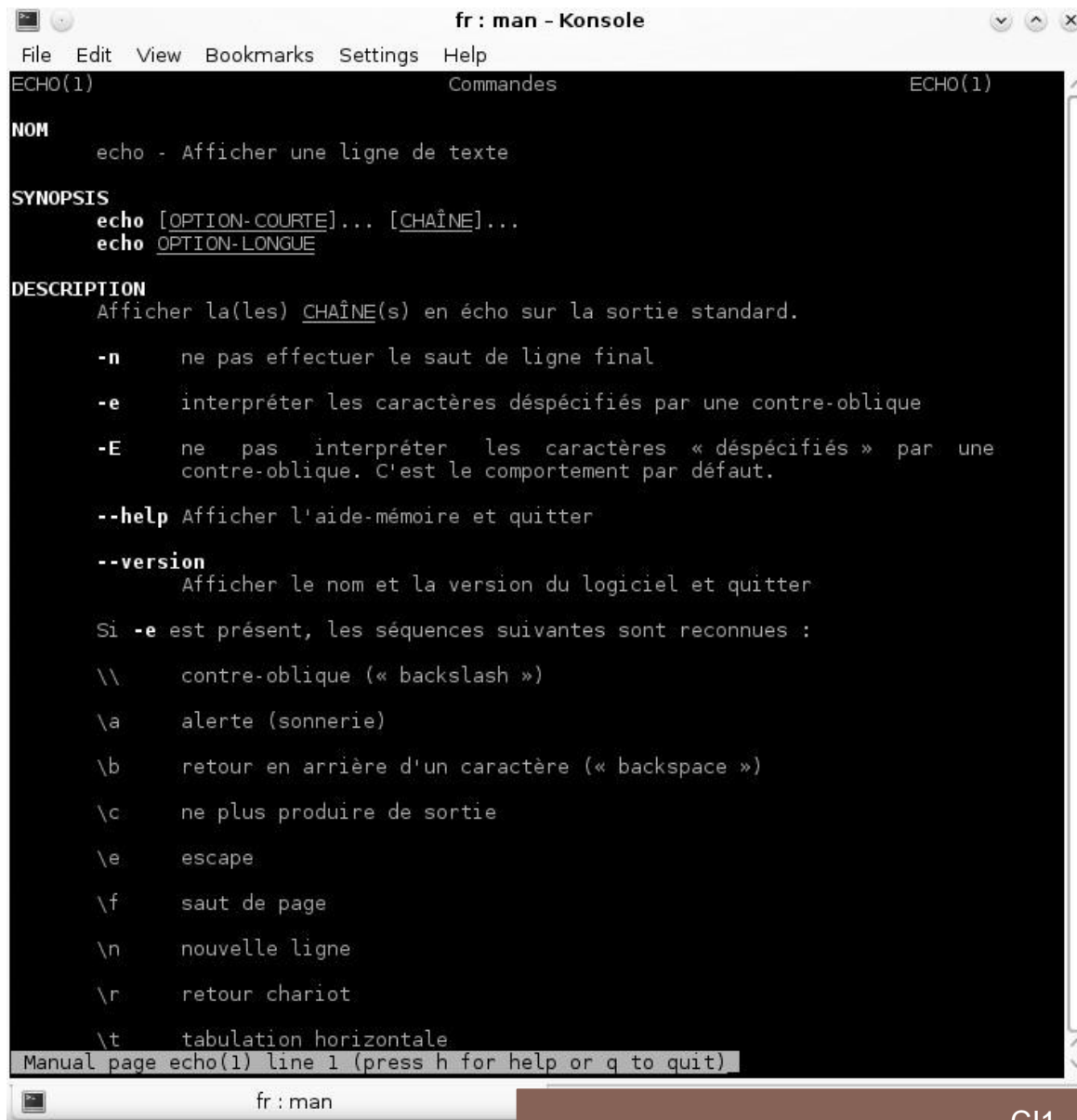
Première commande : echo



```
ebrunet : bash - Konsole
File Edit View Bookmarks Settings Help
ebrunet@heisenberg:~$ man echo
```

- Premier réflexe face à une commande inconnue
⇒ Consulter la page de manuel !

Première commande : echo



```
fr : man - Konsole
File Edit View Bookmarks Settings Help
ECHO(1) Commandes ECHO(1)

NOM
    echo - Afficher une ligne de texte

SYNOPSIS
    echo [OPTION-COURTE]... [CHAÎNE]...
    echo OPTION-LONGUE

DESCRIPTION
    Afficher la(les) CHAÎNE(s) en écho sur la sortie standard.

    -n      ne pas effectuer le saut de ligne final
    -e      interpréter les caractères déspecifiés par une contre-oblique
    -E      ne pas interpréter les caractères « déspecifiés » par une
            contre-oblique. C'est le comportement par défaut.

    --help  Afficher l'aide-mémoire et quitter

    --version
            Afficher le nom et la version du logiciel et quitter

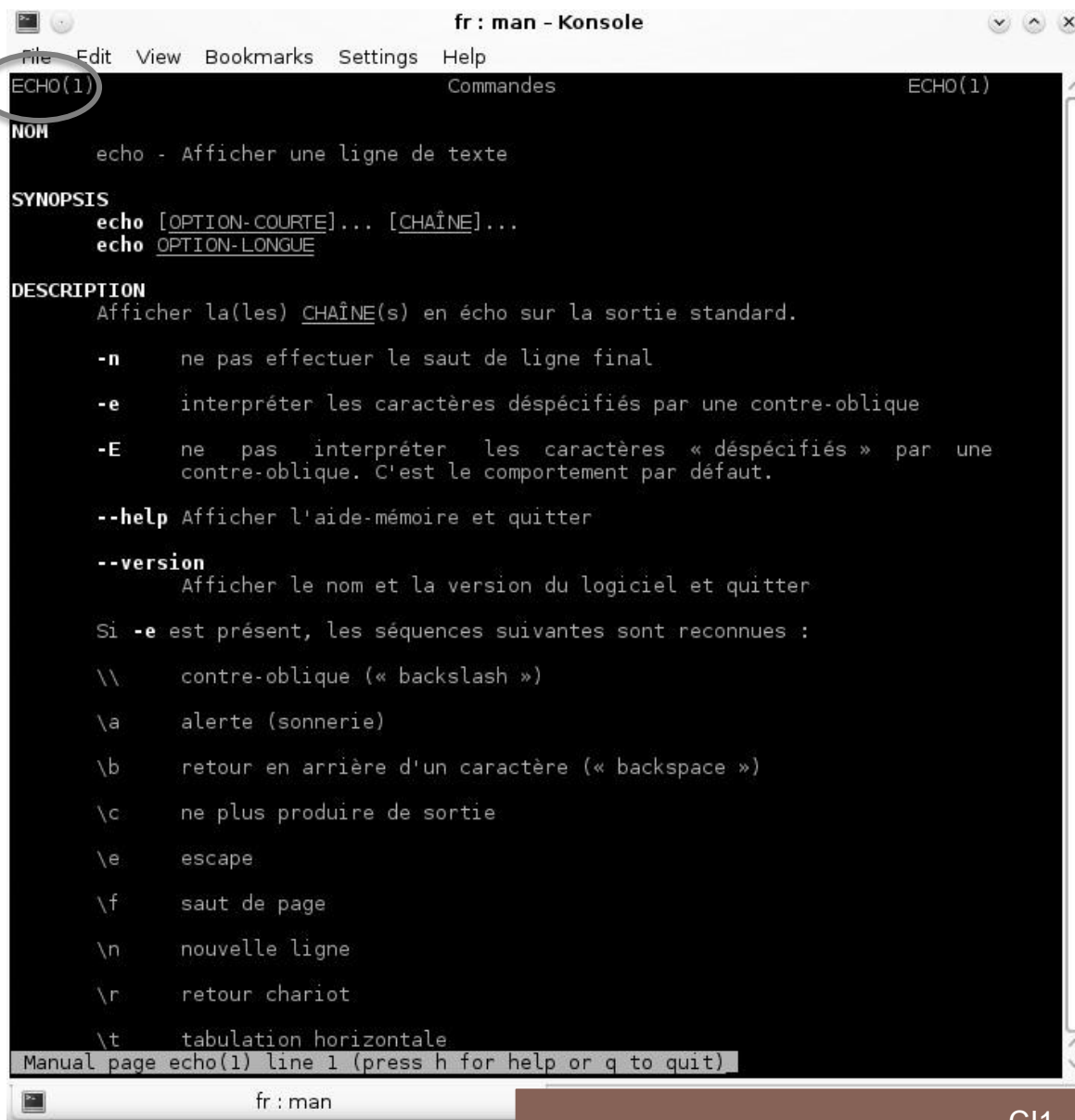
    Si -e est présent, les séquences suivantes sont reconnues :

    \\      contre-oblique (« backslash »)
    \a      alerte (sonnerie)
    \b      retour en arrière d'un caractère (« backspace »)
    \c      ne plus produire de sortie
    \e      escape
    \f      saut de page
    \n      nouvelle ligne
    \r      retour chariot
    \t      tabulation horizontale

Manual page echo(1) line 1 (press h for help or q to quit)
```

Première commande : echo

Section 1



```
fr : man - Konsole
File Edit View Bookmarks Settings Help
ECHO(1) Commandes ECHO(1)
NOM
  echo - Afficher une ligne de texte
SYNOPSIS
  echo [OPTION-COURTE]... [CHAÎNE]...
  echo OPTION-LONGUE
DESCRIPTION
  Afficher la(les) CHAÎNE(s) en écho sur la sortie standard.

  -n      ne pas effectuer le saut de ligne final
  -e      interpréter les caractères déspecifiés par une contre-oblique
  -E      ne pas interpréter les caractères « déspecifiés » par une
          contre-oblique. C'est le comportement par défaut.
  --help  Afficher l'aide-mémoire et quitter
  --version
          Afficher le nom et la version du logiciel et quitter

  Si -e est présent, les séquences suivantes sont reconnues :

  \\      contre-oblique (« backslash »)
  \a      alerte (sonnerie)
  \b      retour en arrière d'un caractère (« backspace »)
  \c      ne plus produire de sortie
  \e      escape
  \f      saut de page
  \n      nouvelle ligne
  \r      retour chariot
  \t      tabulation horizontale
Manual page echo(1) line 1 (press h for help or q to quit)
```


Première commande : echo

□ Démonstration

```
$
```

Première commande : echo

□ Démonstration

```
$ echo bonjour
```

Première commande : echo

□ Démonstration

```
$ echo bonjour  
bonjour  
$
```

Première commande : echo

□ Démonstration

```
$ echo bonjour  
bonjour  
$ echo bonjour le monde
```

Première commande : echo

□ Démonstration

```
$ echo bonjour
bonjour
$ echo bonjour le monde
bonjour le monde
$
```

Première commande : echo

□ Démonstration

```
$ echo bonjour
bonjour
$ echo bonjour le monde
bonjour le monde
$ echo
```

Première commande : echo

□ Démonstration

```
$ echo bonjour
bonjour
$ echo bonjour le monde
bonjour le monde
$ echo
$
```

□ Illustration interactive en mode commande (IIEMC) « Première commande : echo »

Caractères spéciaux du shell

□ Ligne de commande = chaîne de caractères

□ Caractères spéciaux

- `\ ' " > < $ # * ~ ? ; () espace { } ``
(attention ! Ce dernier caractère est un accent grave)
- Signification propre au shell
- Explication de chacun donnée dans la suite du cours

□ Désactive l'interprétation des caractères spéciaux

- `\` désactive l'interprétation du caractère spécial suivant
- `'...'` → désactive l'interprétation de toute la chaîne
- `"..."` → seuls sont interprétés les caractères `$ \ `` (accent grave)

Démonstration

```
$ echo 45 > 40  
$
```

Démonstration

```
$ echo 45 > 40  
$ ls  
40 delme umask.tmp  
$
```

Démonstration

```
$ echo 45 > 40
$ ls
40 delme umask.tmp
$ more 40
45
$
```

Démonstration

```
$ echo 45 > 40
$ ls
40 delme umask.tmp
$ more 40
45
$ echo 45 \> 40
45 > 40
$
```

Démonstration

```
$ echo 45 > 40
$ ls
40 delme umask.tmp
$ more 40
45
$ echo 45 \> 40
45 > 40
$ echo '45 > 40 > 30 * 1'
45 > 40 > 30 * 1
$
```

Démonstration

```
$ echo 45 > 40
$ ls
40 delme umask.tmp
$ more 40
45
$ echo 45 \> 40
45 > 40
$ echo '45 > 40 > 30 * 1'
45 > 40 > 30 * 1
$ echo "45 > 40 > 30 * 1"
45 > 40 > 30 * 1
$
```

Démonstration

```
$ echo 45 > 40
$ ls
40 delme umask.tmp
$ more 40
45
$ echo 45 \> 40
45 > 40
$ echo '45 > 40 > 30 * 1'
45 > 40 > 30 * 1
$ echo "45 > 40 > 30 * 1"
45 > 40 > 30 * 1
$ echo "$dollar"

$
```

Démonstration

```
$ echo 45 > 40
$ ls
40 delme umask.tmp
$ more 40
45
$ echo 45 \> 40
45 > 40
$ echo '45 > 40 > 30 * 1'
45 > 40 > 30 * 1
$ echo "45 > 40 > 30 * 1"
45 > 40 > 30 * 1
$ echo "$dollar"

$ echo "\$dollar"
$dollar
$
```


Démonstration

```
$ echo 45 > 40
$ ls
40 delme umask.tmp
$ more 40
45
$ echo 45 \> 40
45 > 40
$ echo '45 > 40 > 30 * 1'
45 > 40 > 30 * 1
$ echo "45 > 40 > 30 * 1"
45 > 40 > 30 * 1
$ echo "$dollar"

$ echo "\$dollar"
$dollar
$
```

- Illustration interactive en mode commande
« Caractères spéciaux du shell »

Script shell

- Ensemble structuré de commandes shell dans un fichier texte
 - Interprétable par le shell au lancement par l'utilisateur
 - Modifiable par un éditeur de code (p. ex. emacs, vi, mais pas word !)
 - Par défaut, pas exécutable :
 - Pour le rendre exécutable : `chmod u+x <fic.sh>`
 - Notion vue dans le CI 2 sur le système de fichiers
 - Par convention, les noms de script sont suffixés par l'extension « .sh »
 - p. ex., `mon_script.sh`

- Lancement similaire à celui d'une commande shell
 - `./mon_script.sh`
 - Avec ses options et arguments :
`./mon_script.sh opt1 opt2 arg1 arg2`

Structure d'un script shell

□ Première ligne : `#!/bin/sh`

- `#!` : indique au système que ce fichier est un ensemble de commandes à exécuter par l'interpréteur dont le chemin suit
 - p. ex : `/bin/sh`, `/usr/bin/perl`, `/bin/awk`, etc.
- `/bin/sh` fait appel au shell par défaut du système
 - sous Linux, `/bin/bash`

□ Puis succession structurée d'appels à des commandes shell

```
#!/bin/sh

commande1
commande2
...
mon_script.sh
```

□ Sortie du script implicite à la fin de l'exécution

- Sortie explicite grâce à la commande `exit`

Variables shell

- Déclaration/modification : `ma_var=<valeur>`
- Consultation : `$ma_var`
- Portée des variables
 - Locales au contexte de déclaration
 - Propagation descendante grâce au mot clé `export`
- Saisie interactive d'une variable par l'utilisateur
 - Commande `read`

Démonstration : portée des variables

```
$ ma_var_term="terminal"  
$
```

```
#!/bin/sh  
ma_var_script="script"  
echo $ma_var_script  
echo $ma_var_term
```

variable.sh

Démonstration : portée des variables

```
$ ma_var_term="terminal"
$ ./variable.sh
script

$
```

```
#!/bin/sh
ma_var_script="script"
echo $ma_var_script
echo $ma_var_term
```

variable.sh

Démonstration : portée des variables

```
$ ma_var_term="terminal"
$ ./variable.sh
script

$ export ma_var_term
$
```

```
#!/bin/sh
ma_var_script="script"
echo $ma_var_script
echo $ma_var_term
```

variable.sh

Démonstration : portée des variables

```
$ ma_var_term="terminal"
$ ./variable.sh
script

$ export ma_var_term
$ ./variable.sh
script
terminal
$
```

```
#!/bin/sh
ma_var_script="script"
echo $ma_var_script
echo $ma_var_term
```

variable.sh

Démonstration : portée des variables

```
$ ma_var_term="terminal"
$ ./variable.sh
script

$ export ma_var_term
$ ./variable.sh
script
terminal
$ echo $ma_var_script

$
```

```
#!/bin/sh
ma_var_script="script"
echo $ma_var_script
echo $ma_var_term
```

variable.sh

Démonstration : portée des variables

```
$ ma_var_term="terminal"
$ ./variable.sh
script

$ export ma_var_term
$ ./variable.sh
script
terminal
$ echo $ma_var_script

$
```

```
#!/bin/sh
ma_var_script="script"
echo $ma_var_script
echo $ma_var_term
```

variable.sh

```
#!/bin/sh
export ma_var_script="script"
echo $ma_var_script
echo $ma_var_term
```

variable_exportee.sh

Démonstration : portée des variables

```
$ ma_var_term="terminal"
$ ./variable.sh
script

$ export ma_var_term
$ ./variable.sh
script
terminal
$ echo $ma_var_script

$ ./variable_exportee.sh
script
terminal
$
```

```
#!/bin/sh
ma_var_script="script"
echo $ma_var_script
echo $ma_var_term
```

variable.sh

```
#!/bin/sh
export ma_var_script="script"
echo $ma_var_script
echo $ma_var_term
```

variable_exportee.sh

Démonstration : portée des variables

```
$ ma_var_term="terminal"
$ ./variable.sh
script

$ export ma_var_term
$ ./variable.sh
script
terminal
$ echo $ma_var_script

$ ./variable_exportee.sh
script
terminal
$ echo $ma_var_script

$
```

```
#!/bin/sh
ma_var_script="script"
echo $ma_var_script
echo $ma_var_term
```

variable.sh

```
#!/bin/sh
export ma_var_script="script"
echo $ma_var_script
echo $ma_var_term
```

variable_exportee.sh

Démonstration : portée des variables

```
$ ma_var_term="terminal"
$ ./variable.sh
script

$ export ma_var_term
$ ./variable.sh
script
terminal
$ echo $ma_var_script

$ ./variable_exportee.sh
script
terminal
$ echo $ma_var_script

$
```

```
#!/bin/sh
ma_var_script="script"
echo $ma_var_script
echo $ma_var_term
```

variable.sh

```
#!/bin/sh
export ma_var_script="script"
echo $ma_var_script
echo $ma_var_term
```

variable_exportee.sh

➤ IIMC « Portée des variables shell »

Démonstration : saisie de variable

\$

```
#!/bin/sh  
echo "Veuillez saisir x "  
read x  
echo "x=$x"
```

variable_read.sh

Démonstration : saisie de variable

```
$ ./variable_read.sh  
Veuillez saisir x
```

```
#!/bin/sh  
echo "Veuillez saisir x "  
read x  
echo "x=$x"
```

variable_read.sh

Démonstration : saisie de variable

```
$ ./variable_read.sh  
Veuillez saisir x  
23
```

```
#!/bin/sh  
echo "Veuillez saisir x "  
read x  
echo "x=$x"
```

variable_read.sh

Démonstration : saisie de variable

```
$ ./variable_read.sh  
Veuillez saisir x  
23  
x=23  
$
```

```
#!/bin/sh  
echo "Veuillez saisir x "  
read x  
echo "x=$x"
```

variable_read.sh

Démonstration : saisie de variable

```
$ ./variable_read.sh
Veillez saisir x
23
x=23
$ ./variable_read.sh
Veillez saisir x
```

```
#!/bin/sh
echo "Veillez saisir x "
read x
echo "x=$x"
```

variable_read.sh

Démonstration : saisie de variable

```
$ ./variable_read.sh
Veillez saisir x
23
x=23
$ ./variable_read.sh
Veillez saisir x
678
```

```
#!/bin/sh
echo "Veillez saisir x "
read x
echo "x=$x"
```

variable_read.sh

Démonstration : saisie de variable

```
$ ./variable_read.sh
Veillez saisir x
23
x=23
$ ./variable_read.sh
Veillez saisir x
678
x=678
$
```

```
#!/bin/sh
echo "Veillez saisir x "
read x
echo "x=$x"
```

variable_read.sh

Schéma algorithmique séquentiel

- Suite de commandes les unes après les autres
 - Sur des lignes séparées
 - Sur une même ligne en utilisant le caractère point virgule (;) pour séparateur

Tests (1/2)

□ Schéma alternatif classique

- Si alors ... sinon si alors... sinon ...
- Elif et else sont optionnels

```
if <cond> ; then  
    <cmds>  
elif <cond> ; then  
    <cmds>  
else  
    <cmds>  
fi
```

Conditions de test

□ Tests sur des valeurs numériques

- `[n1 -eq n2]` : vrai si $n1 = n2$
- `[n1 -ne n2]` : vrai si $n1$ est différent de $n2$
- `[n1 -gt n2]` : vrai si $n1 > n2$ strictement
- `[n1 -ge n2]` : vrai si $n1$ supérieur ou égal à $n2$
- `[n1 -lt n2]` : vrai si $n1 < n2$ strictement
- `[n1 -le n2]` : vrai si $n1$ est inférieur ou égal à $n2$

□ Tests sur des chaînes de caractères

- `[chaîne1 = chaîne2]` : vrai si chaîne1 est égale à chaîne2
- `[c1 != chaîne2]` : vrai si $c1$ n'est pas égale à chaîne2
- `[-z chaîne]` : vrai si chaîne est une chaîne de caractères vide
- `[-n "chaîne"]` : vrai si chaîne est une chaîne non vide

□ Attention! Les `[]` sont ici obligatoires!!

Tests (1/2)

- Schéma alternatif classique
 - Si alors ... sinon si alors ... sinon ...
 - Elif et else sont optionnels

```
if <cond> ; then  
    <cmds>  
elif <cond> ; then  
    <cmds>  
else  
    <cmds>  
fi
```

```
x=1  
y=2  
if [ $x -eq $y ] ; then  
    echo "$x = $y"  
elif [ $x -ge $y ] ; then  
    echo "$x > $y"  
else  
    echo "$x < $y"  
fi
```


Tests (2/2)

□ Schéma alternatif classique

- Si alors ... sinon si alors ... sinon ...
- Elif et else sont optionnels

```
if <cond> ; then
    <cmds>
elif <cond> ; then
    <cmds>
else
    <cmds>
fi
```

□ Schéma alternatif multiple

- Cas où ... vaut ... ou ... ou ...
- Motif : chaîne de caractères pouvant utiliser des méta-caractères (voir CI3)
 - Symbole | : signifie OU
- *) correspond au cas par défaut

```
case mot in
    motif1)
        ...;;
    motif2)
        ...;;
    *)
        ...;;
esac
```

Tests (2/2)

□ Schéma alternatif classique

- Si alors ... sinon si alors ... sinon ...
- Elif et else sont optionnels

```
if <cond> ; then
    <cmds>
elif <cond> ; then
    <cmds>
else
    <cmds>
fi
```

□ Schéma alternatif multiple

- Cas où ... vaut ... ou ... ou ...
- Motif : chaîne de caractères pouvant utiliser des méta-caractères (voir C13)
 - Symbole | : signifie OU
- *) correspond au cas par défaut

```
case mot in
    motif1)
        ...;;
    motif2)
        ...;;
    *)
        ...;;
esac
```

```
res="fr"
case $res in
    "fr")
        echo "Bonjour";;
    "it")
        echo "Ciao";;
    *)
        echo "Hello";;
esac
```

Boucles

□ Schémas itératifs

- **while**
 - Tant que ... faire ...
 - Condition similaire à celle du test
 - Mot clé break pour rompre la boucle

```
while <cond> ; do  
    <cmds>  
done
```

Boucles

□ Schémas itératifs

- **while**
 - Tant que ... faire ...
 - Condition similaire à celle du test
 - Mot clé break pour rompre la boucle

```
while <cond> ; do  
    <cmds>  
done
```

```
x=10  
while [ $x -ge 0 ] ; do  
    read x  
    echo $x  
done
```

Boucles

□ Schémas itératifs

- **while**

- Tant que ... faire ...
- Condition similaire à celle du test
- Mot clé break pour rompre la boucle

```
while <cond> ; do  
    <cmds>  
done
```

```
x=10  
while [ $x -ge 0 ] ; do  
    read x  
    echo $x  
done
```

- **for**

- Pour chaque ... dans ... faire ...
- var correspond à la variable itératrice
- liste, à l'ensemble sur lequel var itère

```
for var in liste ; do  
    <cmds>  
done
```

Boucles

□ Schémas itératifs

- **while**

- Tant que ... faire ...
- Condition similaire à celle du test
- Mot clé break pour rompre la boucle

```
while <cond> ; do  
    <cmds>  
done
```

```
x=10  
while [ $x -ge 0 ] ; do  
    read x  
    echo $x  
done
```

- **for**

- Pour chaque ... dans ... faire ...
- var correspond à la variable itératrice
- liste, à l'ensemble sur lequel var itère

```
for var in liste ; do  
    <cmds>  
done
```

```
for var in 1 2 3 4 ; do  
    echo $var  
done
```

Interactions avec l'utilisateur

Trois façons de transmettre des données utilisateur à un script :

- Affectation de variable exportée
- Saisie interactive de variable avec la commande `read`
- Passage d'arguments sur la ligne de commande

Arguments sur la ligne de commande

- Ligne de commande
= chaîne de caractères découpée mot-à-mot
 - Stockés dans un tableau
- Lancement d'un script : `mon_script.sh opt1 opt2 arg1 arg2 ...`

<code>mon_script.sh</code>	<code>opt1</code>	<code>opt2</code>	<code>arg1</code>	<code>arg2</code>	<code>...</code>
<code>\$0</code>	<code>\$1</code>	<code>\$2</code>	<code>\$3</code>	<code>\$4</code>	<code>...</code>

- `$0` : toujours le nom de la commande
- `$1 ... $9` : les paramètres de la commande
- `$#` : nombre de paramètres de la commande
- `"$@"` : liste des paramètres = `"opt1" "opt2" "arg1" "arg2" ...`
- `shift` : permet de décaler la liste des paramètres

Illustration

```
#!/bin/sh
for i in "$@" ; do
    echo $i
done
```

mon_echo.sh

\$

Illustration

```
#!/bin/sh
for i in "$@" ; do
    echo $i
done
```

mon_echo.sh

```
$ ./mon_echo.sh
$
```

Illustration

```
#!/bin/sh
for i in "$@" ; do
    echo $i
done
```

mon_echo.sh

```
$/mon_echo.sh
$/mon_echo.sh toto titi
toto
titi
$
```

Illustration

```
#!/bin/sh
for i in "$@" ; do
    echo $i
done
```

mon_echo.sh

```
$/mon_echo.sh
$/mon_echo.sh toto titi
toto
titi
$/mon_echo fin de la demo
fin
de
la
demo
$
```

Illustration

```
#!/bin/sh
for i in "$@" ; do
    echo $i
done
```

mon_echo.sh

➤ IIEMC « Passage de paramètres »

```
$/mon_echo.sh
$/mon_echo.sh toto titi
toto
titi
$/mon_echo fin de la demo
fin
de
la
demo
$
```

Imbrication de commandes

- Forcer l'interprétation d'une commande au sein d'une autre
 - `$ (cmd)`
 - Attention à ne pas confondre avec `$cmd` qui permet l'accès à la valeur de la variable `cmd`

Imbrication de commandes

- Forcer l'interprétation d'une commande au sein d'une autre
 - `$ (cmd)`
 - Attention à ne pas confondre avec `$cmd` qui permet l'accès à la valeur de la variable `cmd`

```
$
```

Imbrication de commandes

- Forcer l'interprétation d'une commande au sein d'une autre
 - `$ (cmd)`
 - Attention à ne pas confondre avec `$cmd` qui permet l'accès à la valeur de la variable `cmd`

```
$ date
lundi 27 juillet 2015, 12:47:06 (UTC+0200)
$
```


Imbrication de commandes

- Forcer l'interprétation d'une commande au sein d'une autre
 - `$ (cmd)`
 - Attention à ne pas confondre avec `$cmd` qui permet l'accès à la valeur de la variable `cmd`

```
$ date
lundi 27 juillet 2015, 12:47:06 (UTC+0200)
$ echo "Nous sommes le $(date). "
Nous sommes le lundi 27 juillet 2015, 12:47:06
(UTC+0200).
$
```

Imbrication de commandes

- Forcer l'interprétation d'une commande au sein d'une autre
 - `$ (cmd)`
 - Attention à ne pas confondre avec `$cmd` qui permet l'accès à la valeur de la variable `cmd`

```
$ date
lundi 27 juillet 2015, 12:47:06 (UTC+0200)
$ echo "Nous sommes le $(date). "
Nous sommes le lundi 27 juillet 2015, 12:47:06
(UTC+0200).
$ echo "Nous sommes le $date."
Nous sommes le .
$
```

Imbrication de commandes

- Forcer l'interprétation d'une commande au sein d'une autre
 - `$ (cmd)`
 - Attention à ne pas confondre avec `$cmd` qui permet l'accès à la valeur de la variable `cmd`

```
$ date
lundi 27 juillet 2015, 12:47:06 (UTC+0200)
$ echo "Nous sommes le $(date). "
Nous sommes le lundi 27 juillet 2015, 12:47:06
(UTC+0200).
$ echo "Nous sommes le $date."
Nous sommes le .
$ date="test"
$
```

Imbrication de commandes

- Forcer l'interprétation d'une commande au sein d'une autre
 - `$ (cmd)`
 - Attention à ne pas confondre avec `$cmd` qui permet l'accès à la valeur de la variable `cmd`

```
$ date
lundi 27 juillet 2015, 12:47:06 (UTC+0200)
$ echo "Nous sommes le $(date). "
Nous sommes le lundi 27 juillet 2015, 12:47:06
(UTC+0200).
$ echo "Nous sommes le $date."
Nous sommes le .
$ date="test"
$ echo "Nous sommes le $date."
Nous sommes le test.
$
```

Imbrication de commandes

- Forcer l'interprétation d'une commande au sein d'une autre
 - `$ (cmd)`
 - Attention à ne pas confondre avec `$cmd` qui permet l'accès à la valeur de la variable `cmd`

```
$ date
lundi 27 juillet 2015, 12:47:06 (UTC+0200)
$ echo "Nous sommes le $(date). "
Nous sommes le lundi 27 juillet 2015, 12:47:06
(UTC+0200).
$ echo "Nous sommes le $date."
Nous sommes le .
$ date="test"
$ echo "Nous sommes le $date."
Nous sommes le test.
$
```

➤ IEMC « Imbrication de commandes »

Aide au débogage d'un script shell

- Affichage de la suite de commandes exécutées
 - Commande `set -x` (`set +x` pour désactiver le mode)
 - A (dés)activer à l'intérieur des scripts pour étendre le mode

```
$
```

Aide au débogage d'un script shell

- Affichage de la suite de commandes exécutées
 - Commande `set -x` (`set +x` pour désactiver le mode)
 - A (dés)activer à l'intérieur des scripts pour étendre le mode

```
$ set -x  
$
```

Aide au débogage d'un script shell

- Affichage de la suite de commandes exécutées
 - Commande `set -x` (`set +x` pour désactiver le mode)
 - A (dés)activer à l'intérieur des scripts pour étendre le mode

```
$ set -x
$ date
+ date
lundi 27 juillet 2015, 12:47:06 (UTC+0200)
$
```


Aide au débogage d'un script shell

- Affichage de la suite de commandes exécutées
 - Commande `set -x` (`set +x` pour désactiver le mode)
 - A (dés)activer à l'intérieur des scripts pour étendre le mode

```
$ set -x
$ date
+ date
lundi 27 juillet 2015, 12:47:06 (UTC+0200)
$ echo "Nous sommes le $(date). "
++ date
+ echo `Nous sommes le lundi 27 juillet 2015, 12:47:06
(UTC+0200).`
Nous sommes le lundi 27 juillet 2015, 12:47:06 (UTC+0200).
$
```

Aide au débogage d'un script shell

- Affichage de la suite de commandes exécutées
 - Commande `set -x` (`set +x` pour désactiver le mode)
 - A (dés)activer à l'intérieur des scripts pour étendre le mode

```
$ set -x
$ date
+ date
lundi 27 juillet 2015, 12:47:06 (UTC+0200)
$ echo "Nous sommes le $(date). "
++ date
+ echo `Nous sommes le lundi 27 juillet 2015, 12:47:06
(UTC+0200).`
Nous sommes le lundi 27 juillet 2015, 12:47:06 (UTC+0200).
$ echo "Nous sommes le $date."
+ echo "Nous sommes le ."
Nous sommes le .
$
```

Aide au débogage d'un script shell

- Affichage de la suite de commandes exécutées
 - Commande `set -x` (`set +x` pour désactiver le mode)
 - A (dés)activer à l'intérieur des scripts pour étendre le mode

```
$ set -x
$ date
+ date
lundi 27 juillet 2015, 12:47:06 (UTC+0200)
$ echo "Nous sommes le $(date). "
++ date
+ echo `Nous sommes le lundi 27 juillet 2015, 12:47:06
(UTC+0200).`
Nous sommes le lundi 27 juillet 2015, 12:47:06 (UTC+0200).
$ echo "Nous sommes le $date."
+ echo "Nous sommes le ."
Nous sommes le .
$ date="test"
+date=test
$
```

Aide au débogage d'un script shell

- Affichage de la suite de commandes exécutées
 - Commande `set -x` (`set +x` pour désactiver le mode)
 - A (dés)activer à l'intérieur des scripts pour étendre le mode

```
$ set -x
$ date
+ date
lundi 27 juillet 2015, 12:47:06 (UTC+0200)
$ echo "Nous sommes le $(date). "
++ date
+ echo `Nous sommes le lundi 27 juillet 2015, 12:47:06
(UTC+0200).`
Nous sommes le lundi 27 juillet 2015, 12:47:06 (UTC+0200).
$ echo "Nous sommes le $date."
+ echo "Nous sommes le ."
Nous sommes le .
$ date="test"
+date=test
$ echo "Nous sommes le $date."
+ echo "Nous sommes le test."
Nous sommes le test.
$
```

Aide au débogage d'un script shell

- Affichage de la suite de commandes exécutées
 - Commande `set -x` (`set +x` pour désactiver le mode)
 - A (dés)activer à l'intérieur des scripts pour étendre le mode

```
$ set -x
$ date
+ date
lundi 27 juillet 2015, 12:47:06 (UTC+0200)
$ echo "Nous sommes le $(date). "
++ date
+ echo `Nous sommes le lundi 27 juillet 2015, 12:47:06
(UTC+0200).`
Nous sommes le lundi 27 juillet 2015, 12:47:06 (UTC+0200).
$ echo "Nous sommes le $date."
+ echo "Nous sommes le ."
Nous sommes le .
$ date="test"
+date=test
$ echo "Nous sommes le $date."
+ echo "Nous sommes le test."
Nous sommes le test.
$ set +x
```

Conclusion

□ Concepts clés

- Terminal, prompt
- Interpréteur de commande shell
 - Commandes, langage script shell, variable d'environnement
- Documentation
- Caractères spéciaux du shell
- Script shell

□ Commandes clés

- `man`, `apropos`

□ Commandes à connaître

- `echo`, `date`, `hostname`

En route pour le TP !!