
debian

NANO
Racourcies

Ctrl + G : afficher l'aide

Ctrl + K : couper la ligne de texte

Ctrl + U : coller la ligne de texte

Ctrl + W : rechercher dans le fichier

(puis entrée et Ctrl +C pour quitter)

Ctrl + Y : page précédente

Ctrl + V : page suivante

Ctrl + O : enregistrer le fichier

Ctrl + X : quitter

On retrouve ces raccourcies en bas de l'éditeur Nano

NANO
Paramètres

-m → permet d'activer l'utilisation de la souris pour se déplacer sur le fichier.

Installation de **gpm** prérequise sur certaines distributions linux

-i → active l'indentation automatique

Exercice 1

Utiliser nano pour éditer votre fichier .bashrc

Faites une recherche sur les alias

Ajouter un nouvel alias de votre choix

Shell
Introduction

Ce langage est un **langage intégré à linux**. Il n'est **pas compilé** ce qui simplifie grandement son utilisation au sein de linux.

Il existe plusieurs Shell (sh / bash / zsh / etc).

Nous travaillerons avec bash.

Premier script (nano monscript.sh) :

```
#!/bin/bash


# ceci est un commentaire
cd
```

Lancer son script

```
chmod +x monscript.sh
./monscript.sh
```

En mode débug :

```
bash -x monscript.sh
```

Variables

Déclaration d'une variable

variable='value'

Appel de variable

echo $variable

Quotes

Simples quotes → affiche le texte tel quel

string='mon texte'

echo $string → *mon texte*

Doubles quotes → affiche le texte et interprète les variables (si présentent)

name='Jean'

string="Bonjour $name"

echo $string → *Bonjour Jean*

Back quotes → exécute ce qui se trouve entre les ` `

here=`pwd`

string="Je suis ici : $here"

echo $string → *Je suis ici : /home/jean/*

Exercice 2

Il est temps de faire son premier script !

Ecrire un script bash avec une variable name composé de votre prénom
Afficher cette phrase :

Bien le bonjour petit padawan NAME

Variables d’environnement / globales

Sur linux, il existe des variables d’environnement que l’on trouve via la commande **env**

On peut travailler avec ses variables en bash très simplement

echo “La position de votre dossier home est $HOME”

On peut ajouter des variables d’environnement directement dans son .bashrc

export VARIABLE=value

Variables des paramètres du .sh

Il est très utile de pouvoir avoir des paramètres passé lors de l'appel de votre .sh

./monscript.sh param1 param2

$# → contient le nombre de paramètres passés à votre script

$0 → contient le nom du script exécuté (/monscript.sh)

$1 → contient le premier paramètre

$2 → contient le second paramètre

...

$XXX → contient le XXXe paramètre

Read

read est une méthode permettant de demander à l'utilisateur une saisie

et de l'enregistrer dans une variable

read -p 'Question ? ' variable

read -p 'Question multiple ? ' variable1 variable2

Limitation du nombre de caractère d'une variable

read -p 'Question ? (10 chars max)' -n 10 variable

Cacher le texte saisie (pour les passwords)

read -p 'Password ? (10 chars max)' -s -n 10 variable

Let

Bash ne permet pas les opérations mathématiques car une variable est un string quoi qu'il arrive.

Il existe une méthode pour effectuer des opérations : let

let "a = 10"

let "b = 5"

let "c = a + b"

echo $c → *15*

On retrouve comme dans certain langage (C / PHP)une contraction d'affectation.

let "a = a /5" **<>** **let "a /= 5"**

Tableaux

table=(‘val0’ 'val1' ‘val2’)

echo ${table[2]} → *val2*

table_asso=(['un']="one" ['deux']="two" ['trois']="three")

table_asso*[“quatre”]=”four”*

echo ${table_asso[“trois”]} → *three*

echo ${table[*]} → *val0 val1 val2*

Exercice 3

1
Changer votre précédent script pour que la variable name
soit une variable passée en paramètre de votre appel de script.

1bis
Changer de nouveau votre script pour cette fois avoir une question :
Quel est votre nom ? (20 caracères max)

2
Faire un script qui prend deux paramètres.
Demander quel opérateur souhaite l'utilisateur (+ - / *).
Faites l'opération et afficher le résultat avec le détail du calcul

Conditions

```
if [ test ]
then
    echo "true"
fi
```

```
if [ test ]; then
    echo "true"
fi
```

```
if [ test ]
then
    echo "true"
elif [ test2 ]
then
    echo "second true"
else
    echo "false"
fi
```

Tests
String / Int

If identique

$string1 = $string2

If not identique

$string1 != $string2

If string is empty

-z $string

If string is not empty

-n $string

If **eq**ual =

$num1 -eq $num2

If **n**on**e**qual !=

$num1 -ne $num2

If **l**ower**t**han <

$num1 -lt $num2

If **l**owero**r****e**qual <=

$num1 -le $num2

If **g**reater**t**han >

$num1 -gt $num2

If **g**reatero**r****e**qual >=

$num1 -ge $num2

Tests
Files

-e $file → Vérifie si le fichier existe.

-d $file → Vérifie si le fichier est un répertoire.

-f $file → Vérifie si le fichier est un fichier.

-r $file → Vérifie si le fichier est lisible (r).

-w $file → Vérifie si le fichier est modifiable (w).

-x $file → Vérifie si le fichier est exécutable (x).

```
if [ -x $file ]
then
        ...
```

Switch

```
case $variable in
    “val1”)
        commands
        ;;
    “val2”)
        commands
        ;;
    “val3”|“val4”)
        commands
        ;;
    *)
        commands
        ;;
esac
```

Exercice 3

1
Créer un script qui vous permet d'indiquer si le param1 est un dossier ou un fichier.
Préciser également s'il est lisible, modifiable et/ou executable.

2
Améliorer votre calculatrice avec un switch.
Et un test sur le param (il doit être uniquement + - * /)

Boucles

```
while [ test ]
do
    echo 'Boucle jusqu'à que test soit faux'
done < $data


for data in 'val1' 'val2' 'val3'
do
    echo "$data"
done
```

Boucles

```
for i in `seq 1 10`;
do
        echo “$i”
done
```

21

Exercice 3

1
Votre script doit afficher la liste des fichiers avec le message suivant :

J’ai trouvé un fichier $name1
J’ai trouvé un fichier $name2
….

2
Créer un fichier avec plusieurs (Nom Prénom Note : Jean Dupond 18)
Lire ce fichier et afficher les étudiants ayant une note supérieur à 11

Fonctions

```
print_something () {
        echo “Hey I am a function”
}
print_something
print_something

function print_something2 {
        echo “Hey I am a function too”
}
print_something2
print_something2
```

```
print_something_with_params () {
        return “Hello $1”
}
print_something_with_params Jean
print_something_with_params Marcel
echo “Return of previous function is $”

function_var_scope () {
        local var='value'
}
```

Exercice 4

Convertisseur d'image
(convert -quality 20 image.png image.jpg)

Créer un dossier et y ajouter 5 images en PNG

Créer un script permettant de convertir les images d'un dossier en JPG
Il faudra pour ce faire demander à l'utilisateur :

- Quelle qualité souhaitez-vous (de 1 à 100) ?
- Souhaitez vous déplacer les images dans un nouveau dossier (Y/N) ?
- Si oui → Où ?

Utilisez des functions pour avoir un code lisible