INSTITUT Mines-Télécom

TELECOM SudParis

Petit bilan à mi-module

CSC3102 – Introduction aux systèmes d'exploitation

Gaël Thomas

Le langage bash

- Des variables : `x=42; echo $x`
- Des structures algorithmiques : `if, for, while`
- Des paramètres : `shift, "$@", "$0", "$1", "$2"...`
- Des codes de retour : `exit n`
- Des imbrications de commandes : `x=$(expr $x + 1)`

Des commandes de base

- Lecture/écriture :
  - `echo, read`
- Le système de fichier :
  - `ls, rm, cp, mv, ln,`
  - `find, df, du, tar`
- Le contenu d'un fichier :
  - `cat, grep, cut, sort, tr`
- Le calcul :
  - `expr`

Des commandes de base

- Lecture/écriture :
  - `echo, read`
- Le système de fichier :
  - `ls, rm, cp, mv, ln,`
  - `find, df, du, tar`
- Le contenu d'un fichier :
  - `cat, grep, cut, sort, tr`
- Le calcul :
  - `expr`

Nota bene :

`expr` affiche son résultat sur la sortie standard

```
$ expr 1 + 2
3
$
```

Des commandes de base

- Lecture/écriture :
  - `echo, read`
- Le système de fichier :
  - `ls, rm, cp, mv, ln,`
  - `find, df, du, tar`
- Le contenu d'un fichier :
  - `cat, grep, cut, sort, tr`
- Le calcul :
  - `expr`

Nota bene :

`expr` affiche son résultat sur la sortie standard

```
$ expr 1 + 2
3
$ x=$(expr 1 + 2)
$
```

Des commandes de base

- Lecture/écriture :
  - `echo, read`
- Le système de fichier :
  - `ls, rm, cp, mv, ln,`
  - `find, df, du, tar`
- Le contenu d'un fichier :
  - `cat, grep, cut, sort, tr`
- Le calcul :
  - `expr`

Nota bene :

`expr` affiche son résultat sur la sortie standard

```
$ expr 1 + 2
3
$ x=$(expr 1 + 2)
$ echo $x
3
$
```

Interprétation de commandes

- Quand `bash` interprète une commande, il exécute :
  1) Analyse « déclaration de variables » – « commande et arguments » – « redirections »
  2) Substitution des variables et des motifs
  3) Puis ouverture des flux si redirections
  4) Puis exécution de la commande

```
$ motif="rep/198*" echo $motif   >80s.txt
```

Variables

Flux

```
0 (stdin)  : terminal
1 (stdout) : terminal
2 (stderr) : terminal
```

80s.txt

(n'existe pas)

Interprétation de commandes

- Quand `bash` interprète une commande, il exécute :
  1) Analyse « déclaration de variables » – « commande et arguments » – « redirections »
  2) Substitution des variables et des motifs
  3) Puis ouverture des flux si redirections
  4) Puis exécution de la commande

```
$ motif="rep/198*"  echo $motif  >80s.txt
```

Variables

Flux

```
0 (stdin)  : terminal
1 (stdout) : terminal
2 (stderr) : terminal
```

80s.txt

(n'existe pas)

Interprétation de commandes

- Quand `bash` interprète une commande, il exécute :
  1) Analyse « déclaration de variables » – « commande et arguments » – « redirections »
  2) Substitution des variables et des motifs
  3) Puis ouverture des flux si redirections
  4) Puis exécution de la commande

```
$ motif="rep/198*" echo $motif >80s.txt
```

Variables

```
motif : rep/198*
```

Flux

```
0 (stdin)  : terminal
1 (stdout) : terminal
2 (stderr) : terminal
```

80s.txt

```
(n'existe pas)
```

Interprétation de commandes

- Quand `bash` interprète une commande, il exécute :
  1) Analyse « déclaration de variables » – « commande et arguments » – « redirections »
  2) Substitution des variables et des motifs
  3) Puis ouverture des flux si redirections
  4) Puis exécution de la commande

```
$ motif="rep/198*" echo rep/198* >80s.txt
```

Variables

```
motif : rep/198*
```

Flux

```
0 (stdin)  : terminal
1 (stdout) : terminal
2 (stderr) : terminal
```

80s.txt

```
(n'existe pas)
```

Interprétation de commandes

- Quand `bash` interprète une commande, il exécute :
  1) Analyse « déclaration de variables » – « commande et arguments » – « redirections »
  2) Substitution des variables et des motifs
  3) Puis ouverture des flux si redirections
  4) Puis exécution de la commande

```
$ motif="rep/198*" echo rep/1989 >80s.txt
```

Variables

```
motif : rep/198*
```

Flux

```
0 (stdin)  : terminal
1 (stdout) : terminal
2 (stderr) : terminal
```

80s.txt

```
(n'existe pas)
```

Interprétation de commandes

- Quand `bash` interprète une commande, il exécute :
  1) Analyse « déclaration de variables » – « commande et arguments » – « redirections »
  2) Substitution des variables et des motifs
  3) Puis ouverture des flux si redirections
  4) Puis exécution de la commande

```
$ motif="rep/198*" echo rep/1989 >80s.txt
```

Variables

```
motif : rep/198*
```

Flux

```
0 (stdin)  : terminal
1 (stdout) : terminal
2 (stderr) : terminal
```

80s.txt

```
(existe, vide)
```

Interprétation de commandes

- Quand `bash` interprète une commande, il exécute :
  1) Analyse « déclaration de variables » – « commande et arguments » – « redirections »
  2) Substitution des variables et des motifs
  3) Puis ouverture des flux si redirections
  4) Puis exécution de la commande

```
$ motif="rep/198*" echo rep/1989 >80s.txt
```

Variables

```
motif : rep/198*
```

Flux

```
0 (stdin)  : terminal
1 (stdout) : terminal
2 (stderr) : terminal
3          : 80s.txt
```

80s.txt

```
(existe, vide)
```

Interprétation de commandes

- Quand `bash` interprète une commande, il exécute :
  1) Analyse « déclaration de variables » – « commande et arguments » – « redirections »
  2) Substitution des variables et des motifs
  3) Puis ouverture des flux si redirections
  4) Puis exécution de la commande

```
$ motif="rep/198*" echo rep/1989 >80s.txt
```

Variables

```
motif : rep/198*
```

Flux

```
0 (stdin)  : terminal
1 (stdout) : &3
2 (stderr) : terminal
3          : 80s.txt
```

80s.txt

```
(existe, vide)
```

Interprétation de commandes

- Quand `bash` interprète une commande, il exécute :
  1) Analyse « déclaration de variables » – « commande et arguments » – « redirections »
  2) Substitution des variables et des motifs
  3) Puis ouverture des flux si redirections
  4) Puis exécution de la commande

```
$ motif="rep/198*" echo rep/1989 >80s.txt
```

Variables

```
motif : rep/198*
```

Flux

```
0 (stdin)  : terminal
1 (stdout) : &3
2 (stderr) : terminal
3          : 80s.txt
```

80s.txt

```
rep/1989
```

Interprétation de commandes

- Quand `bash` interprète une commande, il exécute :
  1) Analyse « déclaration de variables » – « commande et arguments » – « redirections »
  2) Substitution des variables et des motifs
  3) Puis ouverture des flux si redirections
  4) Puis exécution de la commande

```
$
```

Variables

```
motif : rep/198*
```

Flux

```
0 (stdin)  : terminal
1 (stdout) : terminal
2 (stderr) : terminal
```

80s.txt

```
rep/1989
```

Les entrées/sorties

- Des redirections
  - `echo coucou >fic`
  - `read line <fic`
  - `exec 3>fic; echo coucou >&3`
- Des tubes anonymes
  - `cat /etc/passwd | grep root | cut -d':' -f3`

Les entrées/sorties

- Des redirections
  - `echo coucou >fic`
  - `read line <fic`
  - `exec 3>fic; echo coucou >&3`

```
$ ls
bap     bip
$
```

Les entrées/sorties

- Des redirections
  - `echo coucou >fic`
  - `read line <fic`
  - `exec 3>fic; echo coucou >&3`

```
$ ls
bap     bip
$ ls -l >plop
$
```

Le flux, et donc le fichier `plop`, sont créés avant de lancer la commande `ls`

Les entrées/sorties

- Des redirections
  - `echo coucou >fic`
  - `read line <fic`
  - `exec 3>fic; echo coucou >&3`

```
$ ls
bap     bip
$ ls -l >plop
$ cat plop
total 96
-rw-r--r--  1 gthomas  staff   5925 11 oct 16:38 bap
-rwxr-xr-x  1 gthomas  staff  38512 11 oct 16:38 bip
-rw-r--r--  1 gthomas  staff      0 11 oct 16:39 plop
$
```

Le flux, et donc le fichier `plop`, sont créés avant de lancer la commande `ls`

⇒ `plop` apparaît dans `plop` !

Les entrées/sorties

- Des redirections
  - `echo coucou >fic`
  - `read line <fic`
  - `exec 3>fic; echo coucou >&3`

```
$ ls
bap     bip
$ ls -l >plop
$ cat plop
total 96
-rw-r--r--  1 gthomas  staff   5925 11 oct 16:38 bap
-rwxr-xr-x  1 gthomas  staff  38512 11 oct 16:38 bip
-rw-r--r--  1 gthomas  staff      0 11 oct 16:39 plop
$ echo * >plip
$ cat plip
bap bip plop
```

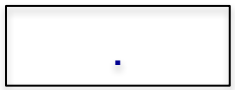
Le motif est évalué avant la redirection

⇒ `plip` n'apparaît pas dans `plip` !

Motifs `bash`, motifs de commandes

- `echo rep/19[7-9][[:digit:]][-_][[:upper:]]*`

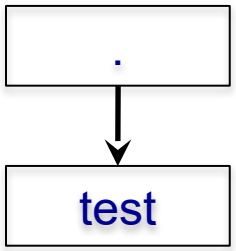
```
$ ls
$
```



Motifs `bash`, motifs de commandes

- `echo rep/19[7-9][[:digit:]][-_][[:upper:]]*`

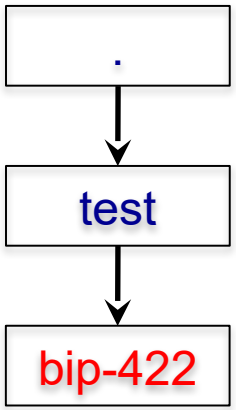
```
$ ls
$ mkdir test
$
```



Motifs `bash`, motifs de commandes

- `echo rep/19[7-9][[:digit:]][-_][[:upper:]]*`

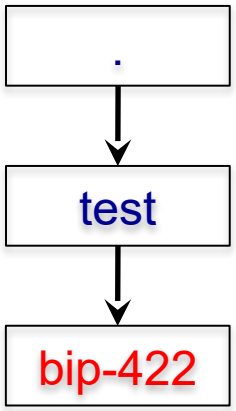
```
$ ls
$ mkdir test
$ touch test/bip-422
$
```



Motifs `bash`, motifs de commandes

- `echo rep/19[7-9][[:digit:]][-_][[:upper:]]*`

```
$ ls
$ mkdir test
$ touch test/bip-422
$ find . -name bip*
./test/bip-422
$
```

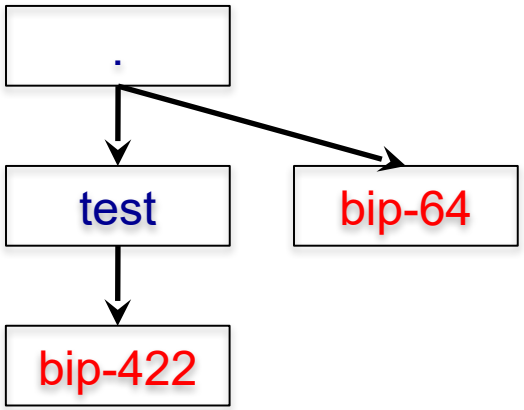


`bash` commence par chercher le motif `bip*`
⇒ pas de correspondance trouvée dans le répertoire courant
⇒ `bash` conserve `bip*`
⇒ `bash` exécute `find` avec comme paramètre `bip*`

Motifs `bash`, motifs de commandes

- `echo rep/19[7-9][[:digit:]][-_][[:upper:]]*`

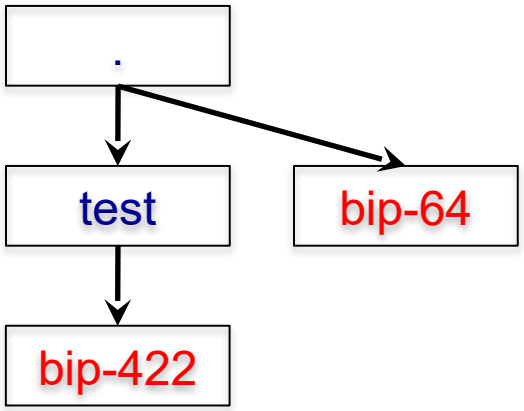
```
$ ls
$ mkdir test
$ touch test/bip-422
$ find . -name bip*
./test/bip-422
$ touch bip-64
$
```



Motifs `bash`, motifs de commandes

- `echo rep/19[7-9][[:digit:]][-_][[:upper:]]*`

```
$ ls
$ mkdir test
$ touch test/bip-422
$ find . -name bip*
./test/bip-422
$ touch bip-64
$ find . -name bip*
./bip-64
$
```

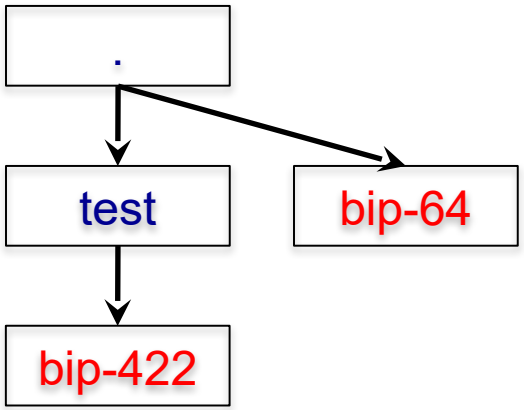


`bash` commence par chercher le motif `bip*`
⇒ une correspondance trouvée dans le répertoire courant
⇒ `bash` transforme `bip*` en `bip-64`
⇒ `bash` execute `find` avec comme paramètre `bip-64` !

Motifs `bash`, motifs de commandes

- `echo rep/19[7-9][[:digit:]][-_][[:upper:]]*`

```
$ ls
$ mkdir test
$ touch test/bip-422
$ find . -name bip*
./test/bip-422
$ touch bip-64
$ find . -name bip*
./bip-64
$ find . -name "bip*"
./bip-64
./test/bip-422
$
```



`bash` n'interpère pas le motif `bip*` car entre guillements

⇒ `bash` execute `find` avec comme paramètre `bip*`