Linux

Les bases du shell
Les structures de contrôle du shell
Exécution de sous-shell

P. MALINGE

Objectifs

- Être capable de programmer des shell scripts simples

Qu'est-ce qu'un script ?

- Un programme de haut niveau souvent court écrit dans un langage de script :
  - Shell
  - Perl
  - Ruby
  - Python
  - Javascript
  - ...

Caractéristiques d'un script

- 'Glue' entre des programmes
- Traitement intensif de chaînes de caractères
- Manipulation de fichiers et répertoires
- Un code adapté à un problème
- Quelques scripts permettent de maîtriser un gros systèmes
- Interface graphique optionnelle
- Portable sur divers OS
- Pas de compilation

Rappel sur `stdin stdout stderr`

- Pour chaque processus, 3 fichiers sont ouverts par le système : `stdin, stdout, stderr`
- Ils correspondent respectivement par défaut à : le clavier, l'écran, l'écran
- Il est très intéressant de rediriger ces 3 fichiers vers d'autres fichiers

Descripteur de fichier	Symbole	Digit associé	Description
stdin	<	0	Entrée standard
stdout	>	1	Sortie standard
stderr	>	2	Erreur standard

Redirection

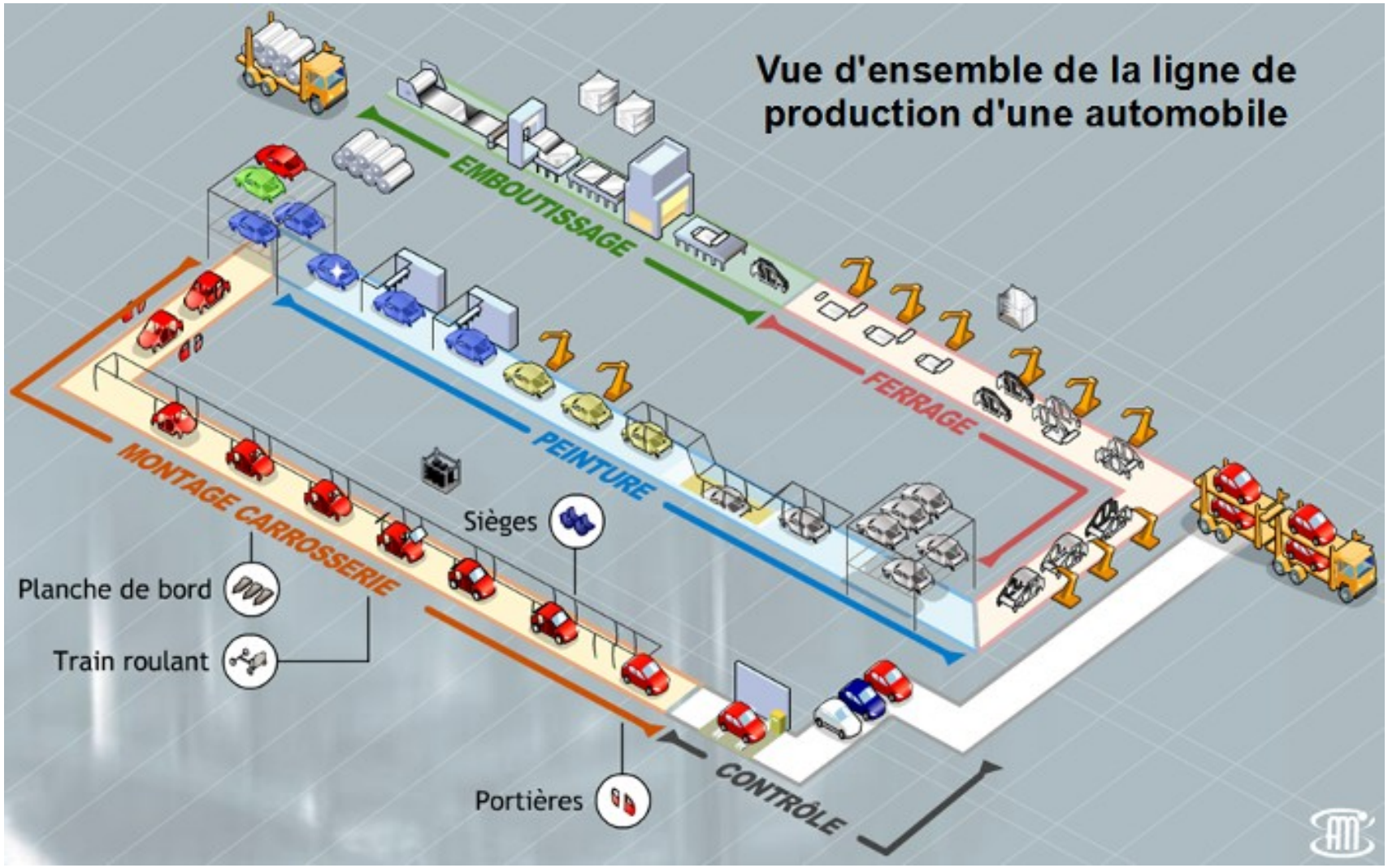
- On utilisera :

  `>` pour rediriger en créant un *fichier*

  `>>` pour rediriger en ajoutant à la fin du *fichier*

  Exemples
- Pour rediriger `stdout` dans *fichier* : `>fichier`
- Pour rediriger `stderr` dans *fichier* : `2>fichier`
- Pour rediriger `stderr` et écrire à la suite de *fichier* : `2>>fichier`

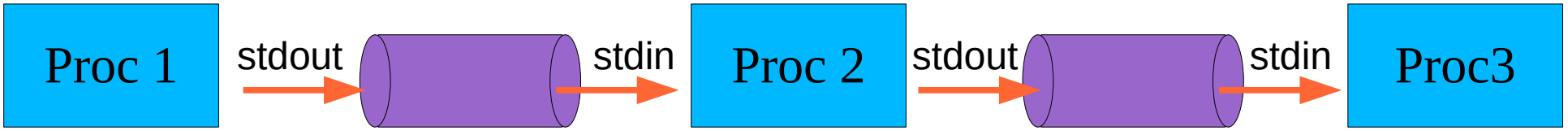
Pipe ?



P. MALINGE

Le pipe

- Le pipe permet de faire communiquer 2 processus en créant un canal de communication entre eux, on parle de flux.
- La sortie standard de l'un sera l'entrée standard du suivant



Ex. : dans le fichier `produit`, recherche des lignes contiennent `DSC`. Cette liste sera transformée en majuscules puis triée dans l'ordre du dictionnaire.

```
grep DSC produit | tr '[a-z]' '[A-Z]' | sort -d
```

Entête de script (exemple)

```
#!/bin/bash


# Objet :
# Usage :
# Ex.   :
# Date création      :
# Date modification :
# Révision :
```

Les variables

- Il n'est pas nécessaire de déclarer les variables
- Les variables sont de type chaine de caractères
- On accède au contenu d'une variables grâce à l'opérateur $
- Ex. :

```
X=12
Y="/tmp/devel"
echo "$Y est créé pour le user $X"
```

Les chaines de caractères

- Les chaines de caractères sont délimitées par les caractères :
  - ‘ (*single quote*)
  - " (*double quote*)
- !! Les variables à l’intérieur de ‘ (*single quote*) ne sont pas évaluées
- Ex. :

```
X=12
echo "Il fait $X" affichera Il fait 12
echo ’Il fait $X’ affichera Il fait $X
```

Conditionnel : if

```
if [ condition ]
then
    traitement_1
else
    traitement_2
fi
```

Boucle: for

```
for variable in liste
do
    traitement_1
done
```

Boucle: While

```
while [ condition ]
do
    traitement
done
```

Choix multiples: case

```
case variable in
    val_1) traitement_1;;
    val_2) traitement_2;;
    *)     traitement_defaut;;
esac
```

Résultat d'un sous-shell

variable=`traitement` ← *VIELLE ÉCOLE*

ou

variable=$(traitement) ← ACTUEL

Remarque :

` est la *back quote* à ne pas confondre avec

' (*single quote*)

Quelques opérateurs sur les fichiers

- `-e fichier` : fichier existe-t-il ?
- `-f fichier` : fichier est-il un fichier régulier ?
- `-s fichier` : fichier est-il non vide ?
- `-d fichier` : fichier est-il un répertoire ?
- `-r fichier` : fichier est-il readable par l'utilisateur courant ?
- `-w fichier` : fichier est-il writable par l'utilisateur courant ?
- `-x fichier` : fichier est-il executable par l'utilisateur courant ?

Quelques opérateurs

`E1 -eq E2` => E1 est égale à E2 ?

`E1 -ne E2` => E1 est différente de E2 ?

`E1 -lt E2` => E1 est inférieure à E2 ?

`E1 -gt E2` => E1 est supérieure à E2 ?

`E1 -le E2` => E1 est inf. ou égale à E2 ?

`E1 -ge E2` => E1 est sup. ou égale à E2 ?

`!(expression)` => négation de expression

`f1 -nt f2` => fichier f1 est plus récent que f2 ?

Exemple avec if

```
if [ "$REPONSE" = "hello" ]
then
    echo "Bonjour Monsieur"
else
    echo "Bye"
fi
```

Exemple avec for

```
for i in 1.jpg 2.jpg 3.jpg
do
    cp $i /tmp
done
```

Ou un autre exemple avec une liste dynamique (créée par un sous-shell)

```
for i in $(ls /polo/*jpg /titi/*mkv /titi/*avi)
do
    cp $i /tmp
done
```

Exemple avec while

```
typeset -i num=1


while [ $num -le 6 ]
do
    touch toto${num}
    num=num+1
done
```

Exemple avec case

```
case "$reponse" in
    "hello")  echo "salut";;
    "HELLO")  echo "SALUT";;
    "Hello")  echo "Salut";;
          *)  traitement_defaut;;
esac
```

Les arguments

- $# : nombre d'arguments passés au programme
- $* : Tous les arguments
- $$x$$: argument en $$x^{\text{ème}}$$ position

Les variables systèmes

- `$?` : code retour de la dernière commande exécutée (`0` = OK, `!=0` pas OK)
- `$!` : PID du processus lancé en tâche de fond
- `$$` : PID du processus lui même

Lire une entrée

- `read toto` : Affecte à la variable `toto` la valeur saisie par l'utilisateur (stdin)
- `read -p 'User ' toto` : Affiche le prompt `'User '` et affecte à la variable `toto` la valeur saisie par l'utilisateur (stdin)
- `read toto titi` : Affecte aux variables `toto` et `titi` les valeurs saisie par l'utilisateur (stdin)

Lire une entrée (suite)

Le fichier nommé `fichier_client` contient :

```
145 Dupont
541 Waterson
6   Simon
521 Pirul
```

On veut d'abord afficher le champ nom du client puis le champ numéro de client

Lire une entrée (suite)

```
cat fichier_client | \
while read client_id client_name
do
  echo "$client_name $client_id"
done
```

Lit stdin (`fichier_client`) et affiche les champs stockés dans les variables `client_id` et `client_name`

Personnalisation du shell

Les fichiers `.bashrc` ou `.bash_profile` du home directory permettent de :

- définir :
  - des variables d'environnement
  - des alias
  - des fonctions
- lancer des programmes

Pour aller plus loin ou très loin !

- Advanced Bash-Scripting Guide
  - http://www.tldp.org/LDP/abs/html/
  - http://abs.traduc.org/