Scripts en BASH

- Rappels
  - Généralités sur le bash
  - Caractères spéciaux
  - Redirection et Enchainement
  - Physionomie et lancement d'un script
- Variables
- Structures de contrôle

Généralités sur le BASH

- Programme qui principalement
  - attend et interprète vos commandes (ou scripts)
  - gère les processus
  - redirige les flux d'entrée et de sortie
- Dispose d'un contexte de travail contenant notamment des variables d'environnement
  - HOME le chemin menant à votre répertoire home
  - PWD le répertoire courant
  - PATH la liste des répertoires où chercher un exécutable

La commande set permet d'afficher toutes les variables d'environnement

Caractères spéciaux

- ␣ Séparateur d'arguments
- $ Accès au contenu d'une variable
- \ Annule l'effet d'un caractère spécial
- ' Bloquent l'interprétation de tous les caractères spéciaux dans la chaîne qu'elles encadrent
- " Bloquent l'interprétation des caractères spéciaux dans la chaîne qu'elles encadrent, à l'exception des caractères $ et `
- ` Remplacent la chaîne qu'elles encadrent par le résultat de son execution

Caractères spéciaux - exemple

- echo cette chaine est pleine despaces

  cette chaine est pleine despaces
- echo "cette chaine est pleine d’espaces"

  cette chaine est pleine d’espaces
- echo "la valeur de HOME est $HOME"

  la valeur de HOME est /root
- echo "le resultat de la commande pwd est `pwd`"

  le resultat de la commande pwd est /home/jld/SCRIPTS
- echo ‘la valeur de HOME est $HOME’

  la valeur de HOME est $HOME

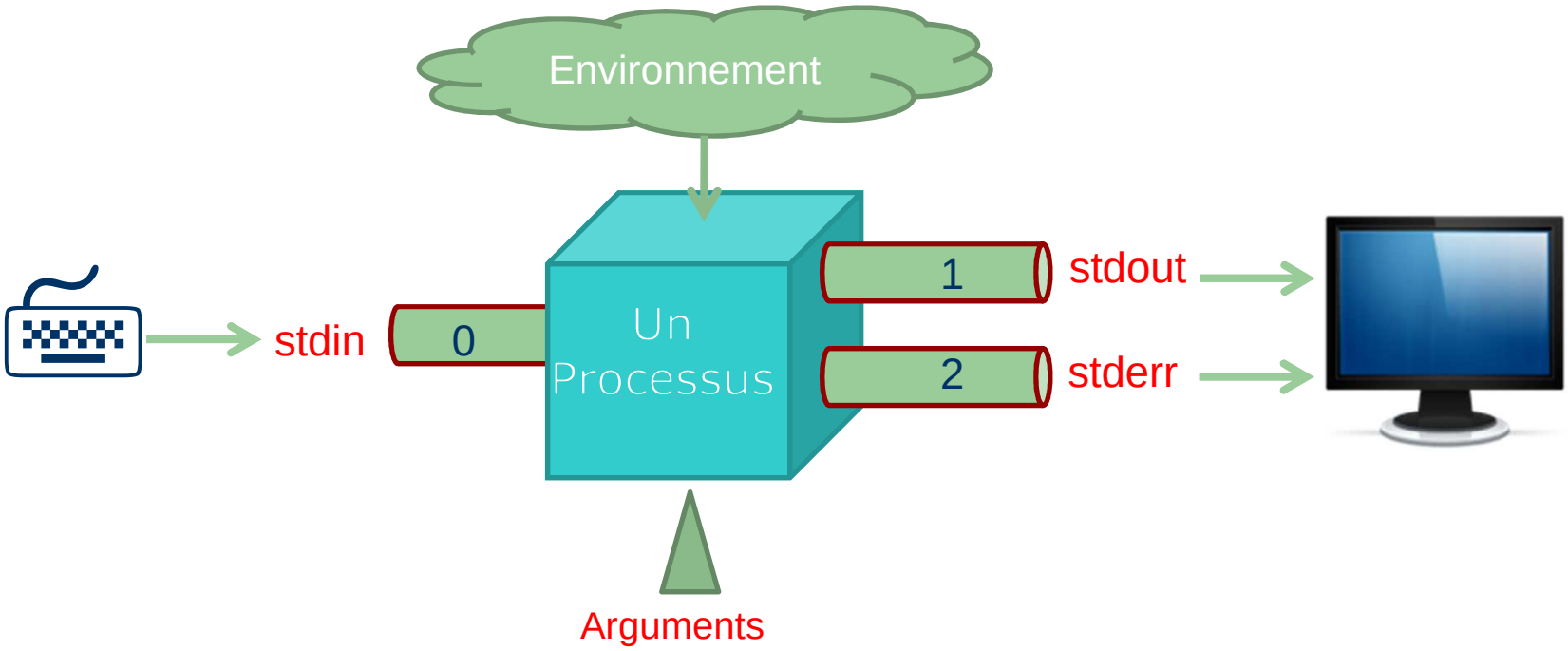
Caractères spéciaux liés à la réécriture "simplifiée" d'un nom de fichier

- ~ sera remplacé par le chemin du répertoire personnel de l'utilisateur courant
- * remplace 0 ou plusieurs caractères
- ? remplace 1 caractère
- [] remplace 1 caractère pris parmi ceux entre les crochets
- [^] remplace 1 caractère pris parmi ceux n'étant pas entre les crochets

Caractères spéciaux - exemple

- *ls ~jld/SCRIPTS/*.sh* tous les fichiers d'extension *.sh* et se trouvant dans le répertoire *SCRIPTS* du répertoire d'accueil de *jld*
- *ls [a-d]*.?* tous les fichiers commençant par une lettre entre en *a* et *d*, suivi de 0 ou n caractères quelconques, et dont l'extension ne comporte qu'un seul caractère
- *ls [^a-d]*.??* tous les fichiers ne commençant par une lettre entre en *a* et *d*, suivi de 0 ou n caractères quelconques, et dont l'extension ne comporte deux caractères

Entrées/sorties d'un processus



Caractères spéciaux liés à l'exécution d'un processus

; lancement séquentiel de processus

& lancement en "arrière plan" d'un processus

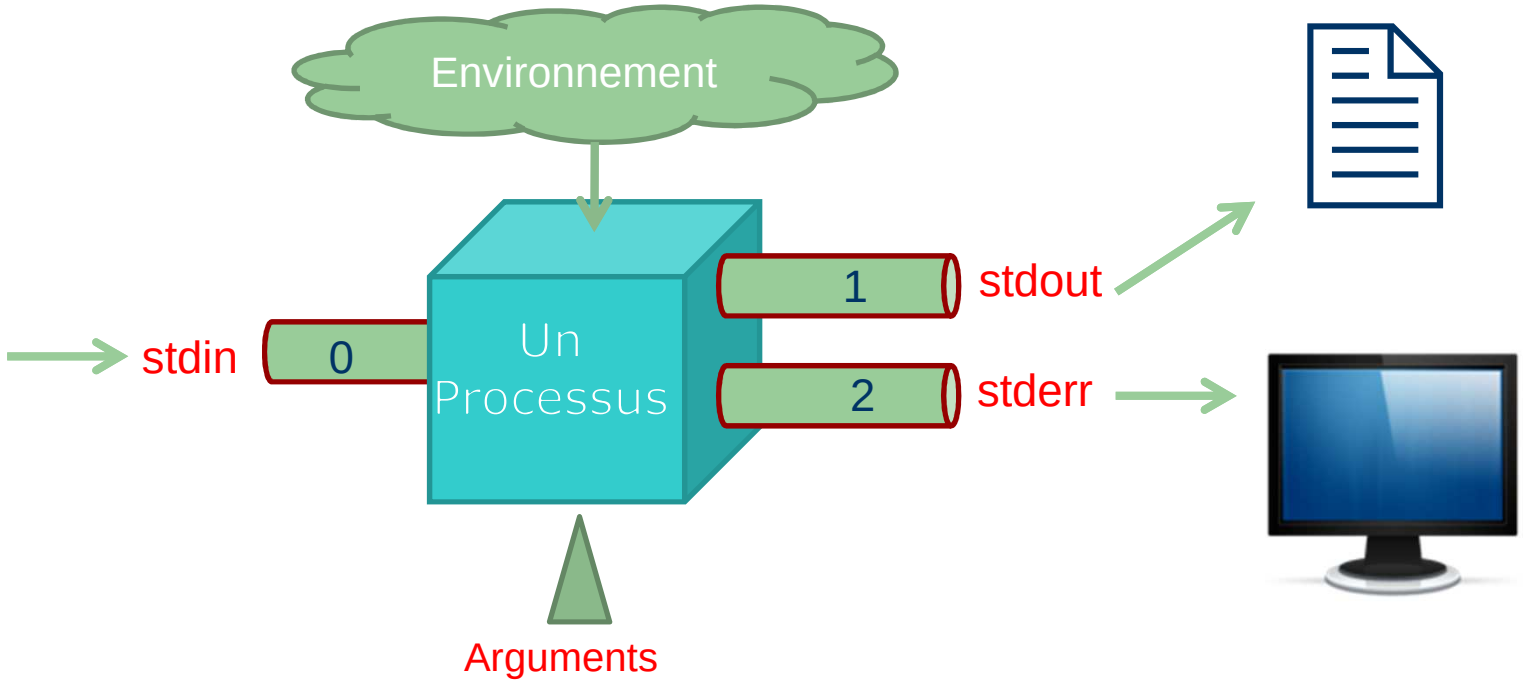
> redirection de la/des sorties d'un processus

>> redirection sans écrasement de la/des sorties d'un processus

< redirection de l'entrée d'un processus

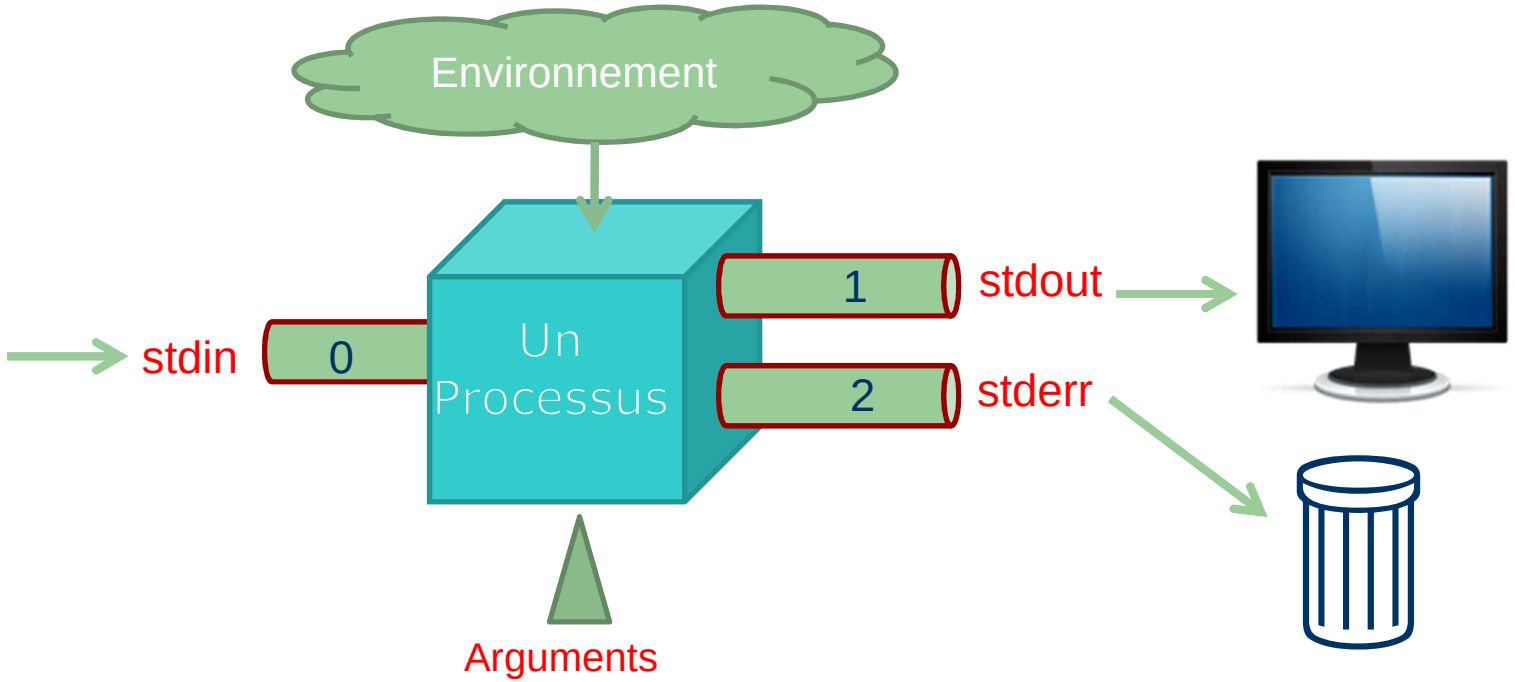
| lancement "chainé" de processus

Redirection de la sortie standard >



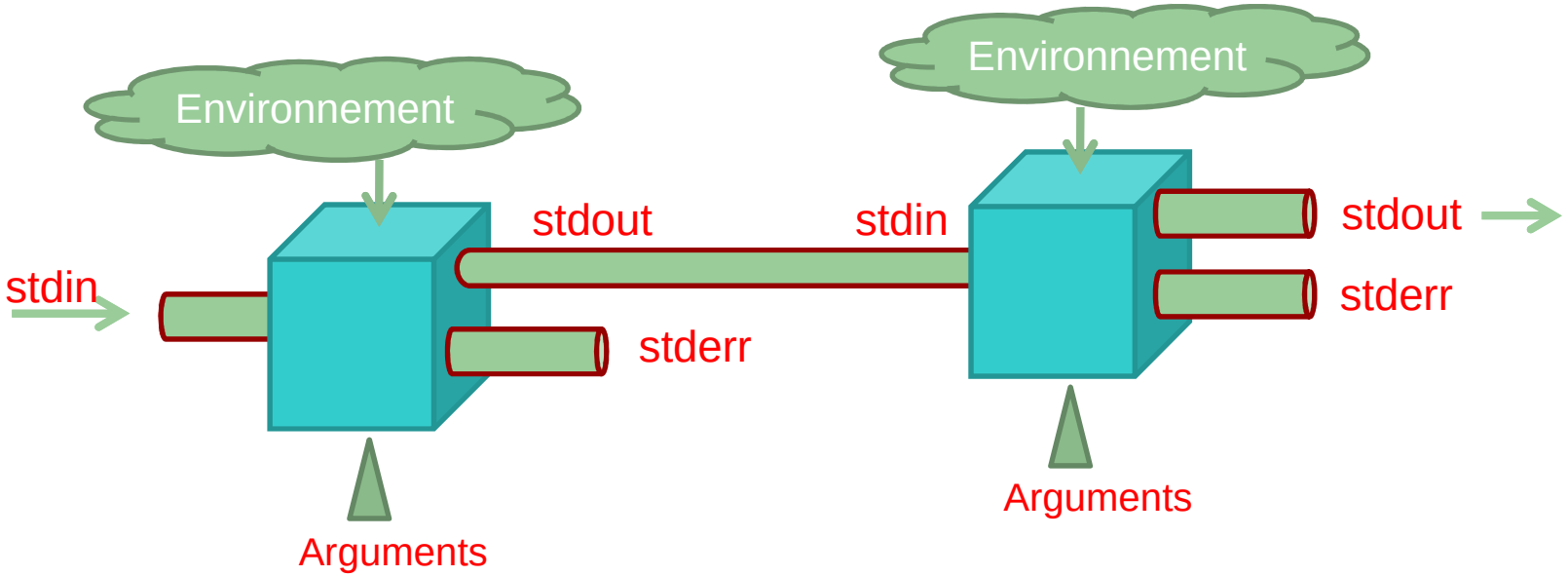
ls > saisie.txt

Redirection de la sortie d'erreurs dans la poubelle 2>



ls 2> /dev/null

Enchainement de processus |



ps aux | grep mozilla | wc -l

Physionomie et lancement d'un script

- Ecrire dans un fichier vos commandes

```
#!/bin/bash
echo "mon premier script "
echo $@
exit 0
```

script1.bash

- Rendre exécutable ce fichier

  chmod +x script1.bash

- Lancer l'interprétation du script

  ./script1.bash toto titi tutu

Code de retour

- Toute commande, suite de commandes ou script retourne une valeur de terminaison (0 indique qu'il n'y a pas eu d'erreurs)
- Ce code de retour est accessible via $?
- Dans un script, la fonction exit permet de renvoyer ce code de retour

- Rappels
- Variables
  - Définition et utilisation d’une variable
  - Définition et utilisation d’un tableau
  - Les arguments du script
- Structures de contrôle

Variables

- Les variables sont par défaut de type chaine de caractères
- Création de la variable

  chaine="ceci est une chaine"

  temperature=25

Attention à ne pas mettre d'espace avant et après le =

- Accès au contenu de la variable via le $

  echo $chaine *ceci est une chaine*

  echo $temperature *25*

Tableaux

- Les tableaux sont unidimensionnels et indicés à partir de 0

- Création nomDuTableau=(val_0 val_1 val_n)

  animaux=(lapin cheval vache)

- Accès à un élément ${nomDuTableau[indice]}

  echo ${animaux[1]} *cheval*

Tableaux (suite)

- Accès à tous les éléments ${nomDuTableau[@]}

  echo ${animaux[@]} *lapin cheval vache*

- Le nombre d'éléments ${#nomDuTableau[@]}

  echo ${#animaux[@]} *3*

- Copie d'un tableau

  zoo=("${animaux[@]}")

Arguments d’un script

- Les arguments d’un script sont stockés dans un genre de tableau indicé à partir de 0, et dont le premier élément est le nom du script, les éléments suivants étant les arguments

- Accès aux arguments

${i}	le $i^{ème}$ argument
$@	tous les arguments
$#	le nombre d’arguments

- Rappels
- Variables
- Structures de contrôle
  - Lecture au clavier
  - Evaluer une expression
  - Si Alors SinonSi Sinon
  - Aiguillage
  - Boucles for et while

Lecture au clavier

- Réalisée par l'instruction read
- Les valeurs saisies sont séparées par au moins un espace

```
read x y z ; echo $x $y $z
51 est          22
51 est 22
```

Evaluer une expression numérique

- Réalisée par la commande ((expression))
- Inutile de protéger les caractères spéciaux, ou de préfixer les variables par le $
- Tous les opérateurs classiques sont utilisables dans l'expression

```
x="67" ; ((x > 0)) && echo $x est positif
```

67 est positif

```
x=5 ; ((x > 0 && x < 10)) && echo "$x est entre 0 et 10"
```

5 est compris entre 0 et 10

Tester une condition composite

- Réalisé par la commande [condition]
- La condition peut porter sur des fichiers, des chaînes de caractères, des entiers
- Attention !
  - les espaces entourant la condition sont obligatoires et les différents termes de la condition devront être séparés par des espaces
  - Les caractères spéciaux doivent être protégés

Conditions sur les fichiers (extrait)

- -e fichier — vrai si le fichier existe
- -s fichier — vrai si le fichier existe et sa taille est supérieure à 0
- -d fichier — vrai si le fichier est un répertoire
- -r fichier — vrai si le fichier est en lecture
- -w fichier — vrai si le fichier est en écriture
- -x fichier — vrai si le fichier est exécutable

Conditions sur les chaînes (extrait)

- ch_1 == ch_2 vrai si les chaînes sont identiques
- ch_1 != ch_2 vrai si les chaînes sont différentes
- ch_1 < ch_2 vrai si ch_1 est plus petite que ch_2
- ch_1 > ch_2 vrai si ch_1 est plus petite que ch_2
- -z ch vrai si ch est vide
- -n ch vrai si ch n'est pas vide

Conditions sur les entiers

- n_1 –lt n_2 vrai si $n_1 < n_2$
- n_1 –le n_2 vrai si $n_1 <= n_2$
- n_1 –gt vrai si $n_1 > n_2$
- n_1 –ge n_2 vrai si $n_1 >= n_2$
- n_1 –eq n_2 vrai si $n_1 = n_2$
- n_1 –ne n_2 vrai si $n_1 <> n_2$

Pour évaluer une expression arithmétique, utilisez plutôt ((expr))

Si alors sinonsi sinon

if Instructions$_1$**; then**

Instructions

elif Instructions$_2$ **; then**

Instructions

elif Instructions$_3$ **; then**

Instructions

else

Instructions

fi

Instructions$_i$ est vrai si son code de retour est égal à 0

elif et else sont optionnels

Si alors - exemple

```
#!/bin/bash
if [ -s ${1} ]; then
  echo  "ce fichier n'est pas vide"
fi
exit 0
```

Si alors sinon - exemple

```
#!/bin/bash
read –s –p "Entrez un mot de passe " mdp
if [ "$mdp" != "vivaBash" ]; then
  echo "Echec d'authentification"
  exit -1
else
  echo "Bienvenue"
fi
```

Si alors sinonsi sinon - exemple

```
#!/bin/bash
read –p "Entrez un nombre ? " nb
if ((nb < 100)); then
  echo "$nb est inferieur a 100"
elif ((nb < 1000)); then
  echo "$nb est compris entre 100 et 999"
else
  echo "$nb est superieur a 999"
fi
```

Aiguillage

case mot **in**

motif $_1$ **)**

liste$_1$

;;

motif$_{2a}$ **|** motif$_{2b}$ **|** motif$_{2c}$ **)**

liste$_2$

;;

. . .

esac

Si mot et motif$_i$ contiennent des caractères spéciaux, ceux-ci son interprétés avant la recherche de la correspondance entre mot et motif$_i$

***;;** joue le rôle du break et il n'y a pas de cas par défaut*

Aiguillage- exemple

```
#!/bin/bash
case ${1} in
  t[oa]* | r[xy]*)
      echo "le premier argument commence par to ou par ta"
      echo " ou il commence par rx ou ry"
      ;;
  *[oa])
      echo " le premier argument finit par o ou par a"
     ;;
  *)
     echo " le premier argument ne commence pas par to ou ta"
     echo "et ne finit pas par o ou a"
  ;;
esac
exit 0
```

Boucle for in

for variable **in** liste_de_valeur **; do**

instructions

done

Boucle for in - exemple

```
#!/bin/bash
for i in mouton souris 345; do
  echo $i
done
exit 0
```

Boucle for in - exemple

```
#!/bin/bash
for i in `ls /etc/c*`; do
  echo $i
done
exit 0
```

Boucle pour "classique"

for ((expr1; expr2;expr3)) **do**

instructions

done

à la double paire de parenthèse près, c'est la boucle du java, php, javascript

Boucle pour "classique" - exemple

```
#!/bin/bash
for ((i=1; i <10; i++)) do
  echo $i
done
exit 0
```

Boucle while

while Instructions$_1$ **; do**

  Instructions$_2$

done

tant que le code de retour de Instructions$_1$ est égal à 0; faire Instructions$_2$

Boucle while - exemple

```
#!/bin/bash
while read -p "entrez 3 mots ? " mot1 mot2 reste ; do
        echo "mot1 : $mot1"
        echo "mot2 : $mot2"
        echo "reste : $reste"
done
exit 0
```

Boucle while - exemple

```
#!/bin/bash
i=0
while [ $PWD != / ]; do
  cd ..
  ((i++))
done
echo "$i niveaux remontés avant d'atteindre le sommet"
exit 0
```

Pour en savoir beaucoup plus

- http://infodoc.iut.univ-aix.fr/~cpb/enseignement/systeme/cours/2013-poly_systeme_dut1.pdf