sh: Les droits d'accès Spéciaux

- Sticky-Bit (bit « collant »)
  - Positionnement par le root
  - Valeur octal : 1000
    - Visible au 9e digit des droits
    - C'est un droit eXecute
    - « **t** » si le droit « **o+x** » est levé, « **T** » sinon
  - Fichier
    - Préservation en mémoire au terme de l'exécution
      - Rechargement plus rapide

64

sh: Les droits d'accès Spéciaux

- Sticky-Bit (bit « collant »)
  - Positionnement par le root
  - Valeur octal : 1000
    - Visible au 9e digit des droits
    - C'est un droit eXecute
    - « **t** » si le droit « **o+x** » est levé, « **T** » sinon
  - Dossier
    - Seul le propriétaire d'un fichier a le droit de le supprimer si le dossier qui le contient a le sticky-bit levé
      - Exemple: le dossier /tmp

65

sh: Les droits d'accès « droits d'endossement »

- Set-UID Bit
  - Valeur octal : 4000
    - Visible au 3e digit des droits
    - C'est un droit eXecute (le « x » du user)
    - « **s** » si le droit « **u+x** » est levé, « **S** » sinon
  - Fichier
    - Exécution d'un fichier avec les droits du propriétaire du fichier
      - Exemple: passwd
  - Pas de sens pour les dossiers

66

sh: Les droits d'accès
« droits d'endossement »

- Set-GID Bit
  - Valeur octal : 2000
    - Visible au 6e digit des droits
    - C'est un droit eXecute (le « x » de group)
    - « **s** » si le droit « **g+x** » est levé, « **S** » sinon
  - Fichier
    - Exécution d'un fichier avec les droits du groupe du fichier

67

sh: Les droits d'accès
« droits d'endossement »

- Set-GID Bit
  - Valeur octal : 2000
    - Visible au 6e digit des droits
    - C'est un droit eXecute (le « x » de group)
    - « **s** » si le droit « **g+x** » est levé, « **S** » sinon
  - Dossier
    - La création d'un fichier lui attribue comme group le même que celui du dossier qui le contient et non pas celui du login qui l'a créé
    - Un nouveau dossier « hérite » du droit set-GID

68

sh: Les droits d'accès

- Le changement du groupe

```
% chgrp [-Rfh] groupname {file|directory} …
```

  - « CHange GRouP »
    - Nécessité d'être propriétaire et/ou root
    - Disparition des bits « **s** »
    - -R : mode « recursive »
    - -h : agit sur le lien le cas échéant et non pas sur le fichier pointé.
    - -f : « force » mode

69

sh: Les droits d'accès

- Le changement du propriétaire

```
% chown [-R] NEWVALUE {file|directory} …
```

  - « CHange OWNer »
    - Nécessité d'être propriétaire et/ou root
  - NEWVALUE = [username[:groupname]]
    - « username »
      - Changement du propriétaire
    - « :groupname »
      - Même impact que chgrp
    - « username:groupname »
      - Changement du propriétaire et du groupe

70

Un tour d'horizon des

COMMANDES SHELL

71

sh: Commandes Usuelles

- Copie de fichiers : cp

```
% cp [options] source target
```

  - Copie du fichier « source » sous le nom « target »
    - Ecrasement de « target » s'il existe déjà.
- Copie de fichiers dans un dossier

```
% cp [options] sourcefile targetdirectory
```

  - « targetdirectory » doit exister
    - cf. cas précédent sinon
  - Copie de « sourcefile » sous le nom « targetdirectory/sourcefile »

72

sh: Commandes Usuelles

- Copie de dossiers
  - Copie d'une branche complète

```
% cp [options] sourcedirectory targetdirectory
```

  - Copie impossible sans options
    - « recursive » mode obligatoire

[options]

- -i : « interactive » mode
- -d : pas de déréférencement des liens symboliques
- -p : recopie des droits et dates de modification
- -r : « recursive » mode

73

sh: Commandes Usuelles

- Déplacement et Renommage

```
% mv [options] {source} {target}
```

- « source » est un fichier
- « target » est un fichier
  - Renomme « source » en « target »
  - Ecrasement de « target » s'il existe déjà
- « target » est un dossier
  - Déplace « source » dans « target »
- « source » est un dossier
- « target » est un dossier existant
  - Déplacement de « source » dans « target »
- « target » n'existe pas
  - Renomme « source » en « target »

74

sh: Commandes Usuelles

- Suppression

```
% rm [options] {fichiers}
```

- « remove » file
- Destruction définitive de fichiers
  - Exception pour les SGF « journalisés »

[options]

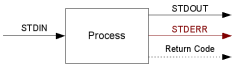
- -r : « recursive » mode
- -f : « force » mode (signifie « ne te pose pas de questions »)
- -i : « interactive » mode

```
# rm -rf /
```

75

sh: Filtre

- Filtre



% cat [options] [file]

- Filtre élémentaire
  - Le plus simple qui soit …
  - Envoie l'entrée standard « stdin » sur la sortie standard « stdout »
  - Comportement standard
    - Visualisation d'un flux de données

76

sh: Visualisation Paginée

- Visualisation de flux d'entrée
  - % more [options] fichier
  - % less [options] fichiers
    - Agissent comme des filtres

- Affichage page/page d'un flux d'entrée
  - Par défaut un fichier
  - more : page à page
  - less : parcours interactif

77

sh: Topologie du S.G.F.

- Visualisation de la structure physique et logique du SGF
  - % df [options] [fichier/partition]
  - Affiche des informations sur la nature d'un fichiers la structure d'un SGF
    - -h : « human readable »

```
Filesystem    Used      Available Capacity ifree %iused      Mounted on
/dev/disk0s2  258372712 229670824      53% 32360587 28708853   53%   /
/dev/hda1     362       362         0    45%      627        0   45%   /dev
/dev/sda2      0         0          0    22%        0        0   22%   /net
map auto_home  0         0          0    34%        0        0   34%   /home
```

78

sh: Usage Disque

- Taille disque utilisée par une entrée logique
  - Fichier, dossier ou partition

```
% du [options] [entrée]
```

[options]

- -h : « human readable »
- -s : affichage de la somme totale uniquement
- -H : déréférencement des liens symboliques

79

sh: Autres Commandes Usuelles

- « Nettoyage » d'une référence du SGF de la partie représentant la branche (dit « préfixe ») et voire de son extension (dit « suffixe »)

```
% basename reference [suffix]
```

•Exemple:

```
% basename /home/usertest/toto.java
toto.java
% basename /home/usertest/toto.java .java
toto
```

80

sh: Autres Commandes Usuelles

- « Nettoyage » d'une référence du SGF de la partie représentant le nom du fichier

```
% dirname reference
```

•Exemple:

```
% dirname /home/usertest/toto.java
/home/usertest
```

81

sh: Autres Commandes Usuelles

% date [options]
- affichage (sous divers format) de la date système

% echo arg_1 arg_2 arg_3 … arg_k
- Envoie sur la sortie standard l'ensemble de ses arguments

% read arg
- Lecture de l'entrée standard et référencement par arg

% expr arg_1 arg_2 arg_3 …
- Evalue une expression arithmétique constituée de ses arguments
- Le résultat est envoyé sur la sortie standard

82

sh: Autres Commandes

- Génération de séquences (sur stdout): **seq**

```
% seq 1 5
  1
  2
  3
  4
  5
 % seq -s' ' 1 5
  1 2 3 4 5
 % seq -s' ' -10 2 10
  -10 -8 -6 -4 -2 0 2 4 6 8 10
```

83

sh: Commandes et Filtres

- Déjà vu:

man	passwd	pwd	ls
cd	mkdir	rmdir	touch
ln	file	chmod	umask
chgrp	chown	cp	mv
rm	cat	more	less
df	du	basename	dirname
date	echo	expr	

84

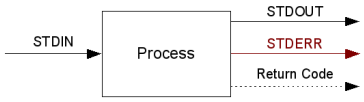
Les entrées/sorties

85

sh: Redirection des Entrées/Sorties

- Redirection des E/S

  % cmd

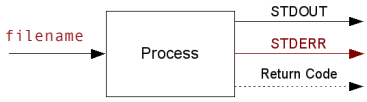


86

sh: Redirection des Entrées/Sorties

- Redirection de l'entrée standard

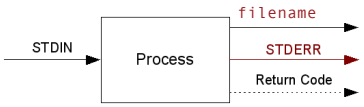
  % cmd < filename



87

sh: Redirection des Entrées/Sorties

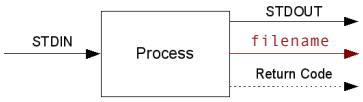
- Redirection de la sortie standard
  % cmd > filename (ecrasement)
  % cmd >> filename (concaténation)



88

sh: Redirection des Entrées/Sorties

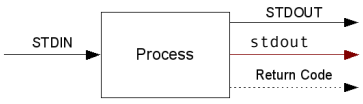
- Redirection de la sortie erreur
  % cmd 2> filename



89

sh: Redirection des Entrées/Sorties

- Redirection de la sortie erreur sur stdout
  % cmd 2>&1

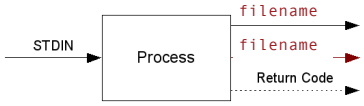


90

sh: Redirection des Entrées/Sorties

- Redirection des deux sorties

```
% cmd >& filename
      ou % cmd &> filename
      ou % cmd > filename 2>&1
```



91

sh: Redirection des Entrées/Sorties

- Redirection « en-place »

```
% cmd << TAG
whatever you write is input of cmd
TAG ## obligatoirement en début de ligne
```

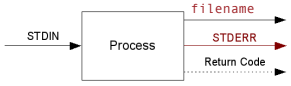
- L'entrée standard est « … whatever you write … »
  - Tout ce qu'il y a entre les deux occurrences de « TAG »

```
% cmd <<- TAG
whatever you write is input of cmd
      TAG ## non nécessairement en début de ligne
```

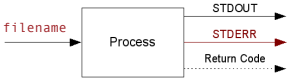
92

sh: filtre

```
% cmd1 > filename
```




```
% cmd2 < filename
```

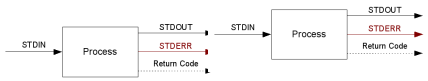


93

sh: les tubes (pipes)

```
% cmd1 > filename ; cmd2 < filename
```


```
% cmd1 | cmd2
```



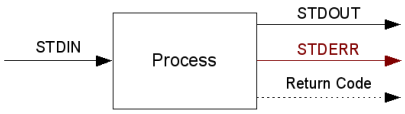
94

LES FILTRES

95

sh: Filtre

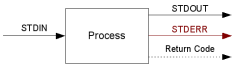
- Filtre



96

sh: Filtre

- Filtre



% head [options] [file]

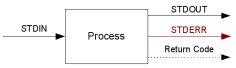
- Filtre élémentaire
  - Envoie sur la sortie standard « stdout » les 10 (valeur par défaut) premières lignes de l'entrée standard

  [options]
  - -n : spécifie le nombre de lignes (du début de flux) à envoyer sur l'entrée standard

97

sh: Filtre

- Filtre



% tail [options] [file]

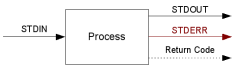
- Filtre élémentaire
  - Envoie sur la sortie standard « stdout » les 10 (valeur par défaut) dernières lignes de l'entrée standard

  [options]
  - -n : spécifie le nombre de lignes (de la fin du flux) à envoyer sur l'entrée standard

98

sh: Filtre

- Filtre



% wc [options] [file]

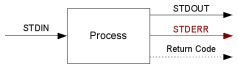
- Filtre
  - Envoie sur la sortie standard « stdout » (suivant les options) le nombre de caractères, de « mots » ou de lignes contenus dans l'entrée standard

  [options]
  - -w : nombre de mots
  - -c : nombre de caractères
  - -l : nombre de lignes

99

sh: Filtre

- Filtre



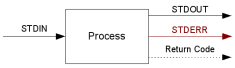
% cut [options] [file]
- Permet le découpage de l'entrée standard

% tr [options] [file]
- Substitution ou suppression de caractères de l'entrée standard

100

sh: Filtre

- Filtre



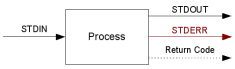
% grep [option] pattern
- Recherche de « pattern » sur l'entrée standard

% sort [options]
- tri de l'entrée standard

101

sh: Filtre

- Filtre



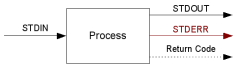
% uniq
- Nettoyage des données dupliquées (et consécutives) de l'entrée standard

% tee
- Idem que « cat ». Envoie de l'entrée standard sur la sortie standard ET dans un ou plusieurs fichiers

102

sh: Filtre

- Filtre



% paste
- Fusion des lignes de 2 fichiers distincts

103

sh: Autres Commandes

- Remplacement des lignes **consécutives** identiques par une seule occurrence
  % cat toto | uniq
- Suppression des lignes **consécutives** identiques :
  % cat toto | uniq -u
- Affichage d'un exemplaire de chaque ligne dupliquée
  % cat toto | uniq –d
- Préfixe les lignes par le nombre d'occurrences
  % cat toto | uniq -c

104

LES SEQUENCES DE COMMANDES

105

sh: Séquences

- ; séquence
  - Évaluée de gauche à droite
  - Résultat : la dernière évaluation
- | tube
  - Séquence potentiellement évaluée en //
  - Redirection stdout (gauche) sur stdin (droite)
  - Résultat : la dernière évaluation
- ( ) concatène les flux de sorties
  - Le flux d'entrée est consommé par la 1ère commande

106

sh: Séquences conditionnelles

- &&
  - séquence si VRAI
    - Evalue la commande droite si retour de gauche est VRAI
    - Arret de la séquence si échec de la commande gauche
- ||
  - séquence si FAUX
    - Evalue la commande droite si retour gauche est FAUX
    - Arret de la séquence dès qu'une commande se déroule avec succès

107

SHELL-SCRIPT

108

sh: shell-script

- Fichier texte ayant le droit d'exécution positionné (par conséquent exécutable)
  - Interprétation par le shell
  - Usage du « shebang »
    `#!/bin/sh [options]`
    •En première ligne d'un shell-script
  - Chaque passage à la ligne est identique à la validation en mode interactif

109

sh: shell-script

- Exécution d'un *shell-script*

  `% monscript`
  - Exécution d'un nouveau processus bash
  - Environnement réinitialisé

  `% . monscript`
  `% source monscript`
  - Inclusion d'un script exécuté par le même processus

110

sh: shell-script

% man bash

111

sh: Variables

- Une variable
  - Une chaine de caractères
  - Pas de typage
    - Elle contient une chaine de caractères
- Affectation :
  `MYVAR="hello world !"`
  - Attention: Pas d'espace avant et après l'opérateur d'affectation '='
- Déréférencement :
  `$MYVAR`
- Affichage
  `echo $MYVAR`

112

sh: Variables

- Une variable
  - Une chaine de caractères
  - Pas de typage
    - Elle contient une chaine de caractères
- Affichage :
  `echo $MYVAR`
- Lecture clavier :
  `read MYVAR`
  - Général

  `read` : lecture d'une ligne de l'entrée standard

113

sh: Opérations arithmétiques et variables

- Une variable
  - Une chaine de caractères
  - Pas de typage
    - Elle contient une chaine de caractères

```
% x=12
  % x=$x+1
  % echo $x
  12+1
```

  - Syntaxe au choix
    - L'outil historique : **expr**
    - La "built-in" commande du bash: **let**
    - Une construction syntaxique du type : **$(())**

114

sh: Variables positionnelles

- Dans le programme principal :
  - arguments d'appel :

```
% mesargs ABC 123 “x y z”
$0=”mesargs”
$1=”ABC”
$2=”123”
$3=”x y z”
$#=3
%
```

- Dans une fonction :
  - arguments de la fonction
  - *Cf. fonctions*

115

sh: variables spéciales

`$IFS` : séparateurs
`$*` : liste des paramètres
- avec les séparateurs “`$*`” équivaut à “`$1c$2c...`” avec c premier caractère de `$IFS`

`$@` : liste des paramètres sans séparateurs
- “`$@`” équivaut à “`$1`” “`$2`” ...

`$#` : nombre de variables positionnelles
- Rmq: au sens de `$*`

`$?` : code retour de la dernière commande
`$$` : PID du shell

116

Quelques variables d'environnement du bash

USER	PATH
SHELL	HOME
PPID	PS1
PWD	PS2
RANDOM	PS3
HOSTNAME	PS4
IFS	...

117

sh: code de Retour

- Valeur de retour de chaque commande
  - Variable d'environnement : $?
  - Correspondance langage C:

```
int main() {
    int v = EXIT_VALUE;
    …
    …
    return(v);
}
```

$? est égale à v

118

sh: code de Retour

- $? : Pour quel usage ?
  - Convention
    - $? est égal à 0 : Succès (ou VRAI logique)
    - $? Est différent de 0 : Code d'erreur (ou FAUX logique)
  - Test de la valeur de retour
    - Conditionnement de l'exécution de commandes
    - Ex: test
      - Evaluation d'une expression booléenne
  - Comment affecter $? :
    - La commande **exit**

119

sh: à propos de la commande test

- *Simulation* de l'évaluation d'une expression booléenne
  - L'expression booléenne est décrite en argument
  - Exemple:

```
% test 1 -eq 2
% echo $?
1
% test 1 -eq 1
% echo $?
0
```

120

sh: à propos de la commande test

- Opérateurs de comparaison « arithmétique »
  - **-ne -eq -le -lt -ge -gt**
- Opérateurs de comparaison sur des chaines de caractères
  - **= !=**
- Identification de propriétés relatives à des fichiers
  - **-e -f -d …**
  - Droits, taille, device, etc.
- Cf. « bash reference card »
- Autre écriture

```
% test 1 -ne 2
% [ 1 -ne 2 ]
```

121

MULTI-INSTRUCTIONS

122

sh: instruction conditionnelle if/elif/else/fi

`expr` est une expression fixant `$?`
`instruction(s)` est une liste de commandes ba

```
if <expr>
then
  <instruction(s)>
elif <expr>
then
  <instruction(s)>
else
  <instruction(s)>
fi
```

```
if <expr>; then
  <instruction(s)>
elif <expr>; then
  <instruction(s)>
else
  <instruction(s)>
fi
```

123

sh: instruction multi-conditions

case/esac

ID est une variable
instruction(s) est une liste de commandes ba
motif* est un « pattern » pouvant contenir des meta-caractères

```
case <ID> in
  <motif 1>) <instruction(s)>  ;;
  <motif 2>) <instruction(s)>  ;;
  …
  <motif n>) <instruction(s)>  ;;
esac
```

124

sh: boucles canoniques

while/until

expr est une expression fixant $?
instruction(s) est une liste de commandes ba

```
while <expr>
do
  <instruction(s)>
Done
```

```
until <expr>
do
  <instruction(s)>
done
```

$? égal à $? dernière instruction

125

sh: itérateur de liste

for

ID est une variable
list est une liste de « mots »
instruction(s) est une liste de commandes ba

```
for ID in list
do
  <instruction(s)>
done
```

```
for ID
do
  <instruction(s)>
done
```

$? égal à $? dernière instruction

126

sh: rupture de flots d'instructions

break/continue

- break [n]
- continue [n]
- « *built-in* » commands
  - Sort ou boucle (prématurément) de n boucles imbriquées
  - Par défaut n = 1 (n ≥ 1)
  - Si n > #boucles imbriquées
    - break : fin du script ($? = 0)
    - continue : Poursuite de la boucle « *la plus externe* »

127

sh: Exercice

- **Génération de vignettes**
  - Soit un répertoire d'images **rep**.
  - On souhaite automatiser la génération de vignettes pour les images contenues dans **rep**
  - L'outil *convert* permet d'obtenir une vignette a partir d'une image **jpeg** en executant la ligne de commande suivante :
    - convert -size 120x120 source.jpg -resize 12x12 +profile " " *vignette.jpg*
    - *Écrire un script permettant d'exécuter cette ligne pour chaque fichier* **JPEG** (ex: source.jpg) du répertoire **rep** de sorte que les nom des vignettes ait la forme **source-thumb.jpg**

128

QUOTATIONS

129

sh: Les différentes façons de quoter

- " ... "
  - Paramètre unique
  - Contenant éventuellement des séparateurs
  - Expansion des variables.
- '...'
  - Paramètre unique
  - Contenant éventuellement des séparateurs
  - Sans expansion des variables.
- ` ... `
  - Exécution d'une commande
  - Expansion par le résultat de la commande

130

FONCTIONS

131

Fonctions

```
function toto ()
{
  instructions
}
```

- Paramètres formels sans déclaration :
  - variables positionnelles ($1...)
  - Masquage des variables positionnelles d'appel du script
- Variables locales précédées du qualifieur `local`

132

Expressions regulières

133

Expressions régulières

- But
  - Identification d'une suite de caractères répondant à certain(s) critère(s) au sein d'une chaine de caractères
    - On parle de « **PATTERN MATCHING** »
  - Utilisation
    - grep, sed, vi, awk, …
    - Pas d'usage d'expressions régulières built-in dans bash
      - Metacaractères

134

Expressions régulières

- But: Identification d'une suite de caractères répondant à certain(s) critère(s) au sein d'une chaine de caractères
  - . caractère quelconque
  - ^ caractère de début de ligne
  - $ caractère de fin de ligne
  - \. caractère point
  - * réplicateur (0 fois ou n fois)
  - + réplicateur non nul (au moins une fois)
  - ? réplicateur d'option (au plus 1 fois)
  - \{n\} réplicateur n fois
  - [] ensemble de caractères
    - [0123456789] [0-9] [[:digit:]]
    - ^ en début d'ensemble : exclusion de caractères
    - [^xyz] [a-w]

135

Expressions régulières : grep

- Recherche des comptes sans /etc/passwd
  ```
  % cat /etc/passwd | grep '::'
  % cat /etc/passwd | grep '^.*::'
  % cat /etc/passwd | grep '^[^:]*::'
  ```
- Recherche des login trop long
  ```
  % cat /etc/passwd | grep '^[^:]\{9,\}:'
  ```
- Recherche des lignes non-vides
  ```
  % grep -v '^$'
  ```

136

Expressions régulières
Sed: « a Stream Editor »

- Comporte un langage de programmation rudimentaire
- Utilisé principalement pour le remplacement de *regexp*

```
% sed 's/regexp/remplacement/'
```

- *Où regexp est une expression régulière à la grep comportant des sous-expressions entre \(\) référencées dans remplacement par \1 \2 …*

```
% echo Abcdef | sed 's/\(.*\)\(b.*\)\(e.*\)/\2\1\3/'
  bcdAef
% sed 's/^\([^:]\)::/\1:x:/'
```

- Autres actions possibles :
  - d delete
  - t translate
  - g plusieurs fois par ligne

137

Expressions Régulières
Exercice

- **Publipostage**
  - Soit une base de données stockée dans 4 fichiers texte nom.txt, adr.txt, cp.txt, ville.txt
  - Chacune des lignes de chacun de ces fichiers texte mises en correspondance correspond à une personne
  - Soit un fichier texte « source » contenant entre autres choses les chaines de caractères suivantes :
    - __NOM__, __ADRESSE__, __CP__, __VILLE__
    - Créer un publipostage de telle sorte que dans les différentes copies du fichier source, les chaînes __X__ ait été remplacées par leur valeurs correspondantes à une ligne de la base de données.

138

PROCESSUS

139

Processus

- Un processus est un programme en cours d'exécution
- Il est caractérisé par :
  - Un numéro unique : le **PID** (Process Identifier)
  - Un contexte propre
  - Un espace d'adressage
  - Une entrée dans la table des processus :
    - Un état
    - Une priorité
- Deux modes d'exécution possibles :
  - Mode utilisateur
  - Mode noyau (lors d'appels système)

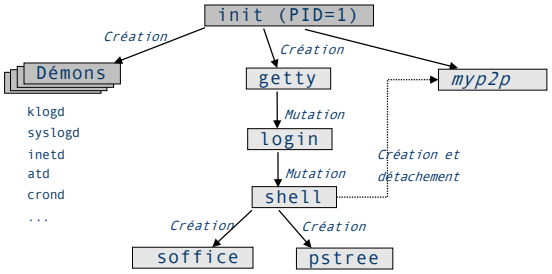
140

Processus
PID et PPID

- PID: Process Identifier :
  - Numéro (entier positif) unique
  - attribué définitivement
- PPID: Parent Process Identifier
  - PID du processus père
  - Tout processus a un père
  - Par défaut, c'est le processus qui l'a créé
- *init*
  - Processus d'initialisation du système
  - PID=1

141

Processus
Arborescence



142

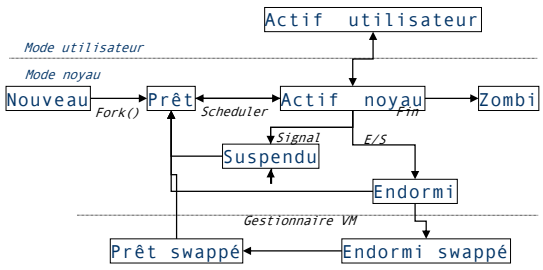
Processus
Création et Mutation

- Création d'un processus par clonage
  - À partir d'init
  - Nouveau PID attribué par le système
  - Arborescence
- Mutation :
  - Chargement d'un nouvel exécutable
  - PID conservé (espace mémoire et contexte aussi)
- Détacher un processus de son père (shell) :
  - Le processus est adopté par init

```
% nohup <commande args>
```

143

Processus
Etats possibles



144

Processus
Fin

- Un processus peut terminer :
  - Quand il a fini son exécution
  - Quand il reçoit un signal
    - de fin (INT...)
    - d'erreur (FPE...)
- La fin d'un processus est signalée à son père
  - Libération des ressources
  - Envoi d'un signel SIGHLD au père
  - Passage à l'état << zombie >> (defunct)
  - Disparition totale à l'accusé de réception du père
- Propagation des signaux d'arrets aux processus fils

145

Processus
les commandes UNIX

- Liste des processus courants :
  `% ps [options]`
- Meme commande, réactualisée à intervalles réguliers
  `% top`
- Arbre des processus :
  `% pstree`
- Envoi d'un signal :
  `% kill -SIG PID`

146

Processus
envoi de signaux

- Différents signaux dont les plus utilisés (man kill):
  - HUP 1 INT 2
  - ABRT 6 FPE 8
  - KILL 9 STOP 15
- Permettent de
  - changer l'état d'un processus (STOP, CONT)
  - Mettre fin à un processus (INT,ABRT, KILL)
  - Réinitialiser un processus (HUP)
  - Récupérer des erreurs graves (FPE...)
  - Communiquer les fins de processus (HLD)
  - ...

147

Processus
cas du shell: les jobs

- Création d'un processus pour chaque commande simple
- Par défaut : en ***foreground***
  - Attend la fin de l'exécution pour relire une commande

```
% cmd args
```

- Lancement en ***background*** (tâche de fond) :

```
% cmd args &
[1]  cmd args
```

- PID : $$ PPID: $PPID

148

Processus
cas du shell: les jobs

- Liste des jobs *(applications lancées par le shell)*:

```
% jobs
[1] Running      simpress Processus.sxi &
[2] Running      myp2p
```

- Relancement en ***foreground*** ou en ***background*** :

```
% fg %2
myp2p
^Z
[2]  Stopped         myp2P
% bg
[2] myp2p &
```

149