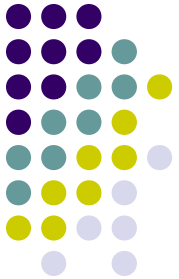
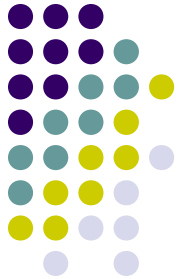


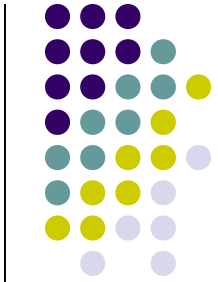
Expressions régulières et utilitaires

Rappels ou révélations



- Une *expression régulière* (ou *expression rationnelle*, ou *ER*, ou encore «*regex*») est une expression qui décrit un langage rationnel
- Un *langage rationnel* sur un alphabet Σ est:
  - Soit le langage vide $\varnothing$
  - Soit un langage de la forme {a}, où a $\in \Sigma$
  - Soit un langage de la forme L*, L + R, ou L·R, où L et R sont des langages rationnels, et L* dénote la fermeture de Kleene de L.
- Par définition:
  - L+R est le langage formé des mots de L et R (l'union), et L·R = { xy : x $\in$ L, $\in$ R } celui formé par concaténation de toutes les mots de L et de R
  - L* est le langage obtenu en concaténant aucun, un, ou plusieurs mots de L

Rappels ou révélations



Exemple: soit Σ = { a,b,c }, L = { a,ab }, et R = { ε,c }.

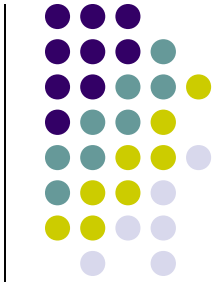
Alors:

- L + R = { ε, a, ab, c }
- L·R = { a, ac, ab, abc }
- L* = { ε, a, ab, aab, aba, aaab, … }
- L et R sont réguliers sur Σ

Autres exemples:

- L'ensemble des numéros minéralogiques français forme un langage rationnel sur Σ = { 0,1,2,3,4,5,6,7,8,9, A, …, Z }
- L'ensemble des entiers naturels forme un langage rationnel sur l'alphabet Σ = { 0,1,2,3,4,5,6,7,8,9 }

Rappels ou révélations



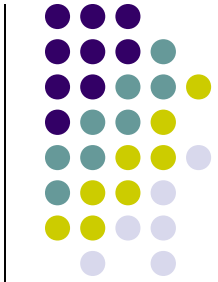
Contre-exemples:

- L'ensemble des expressions algébriques bien parenthèsées de deux variables a et b ne forme pas un langage rationnel sur Σ = { a,b,+,-,(,) }
- L'ensemble des palindromes ne forme pas un langage rationnel sur l'alphabet usuel Σ = { a, b, …, z }

Théorème (Kleene) : tout langage régulier est reconnaissable par un automate à états finis (AEF).

Donc il suffit de trouver un moyen de représenter un AEF - et la reconnaissance du langage régulier suivra… c'est précisément le but des expressions régulières.

Rappels ou révélations



Un automate à états finis est un quintuplet $(Q,\Sigma, \delta, q0, T)$ dans lequel :

- Q est un ensemble dénombrable et fini appelé ensemble des états - très souvent, IN
- Σ est l'alphabet (fini)
- $\delta : Q \times \Sigma \rightarrow Q$ est une fonction appelée fonction de transition
- $q0 \in Q$ est un état appelé état initial
- T est l'ensemble des états terminaux

On représente généralement un AEF par un graphe orienté dans lequel les arcs représentent la fonction de transition, et les nœuds les états. **Chaque transition «consomme» les symboles qui la définissent**

Rappels ou révélations

Exemple: automate reconnaissant une plaque d'immatriculation (modèle simplifié : au moins un chiffre qui n'est pas 0, suivi d'au moins une lettre, suivie de deux chiffres exactement):

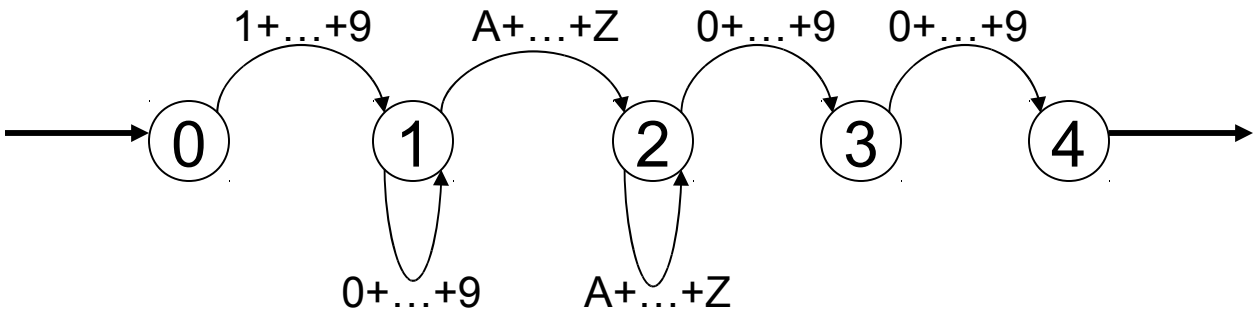
Q={0,1,2,3,4}, q0=0, Σ = { 0,…,9,A,…,Z }, T = {4}

$\delta(0,1)= \delta(0,2)=\ldots= \delta(0,9)=1$; $\delta(0,0)$ indéfini

$\delta(1,A)=\ldots= \delta(1,Z)= 2$

$\delta(1,0)= \delta(1,1)=\ldots= \delta(1,9)=1$

(exercice: compléter la définition de δ)

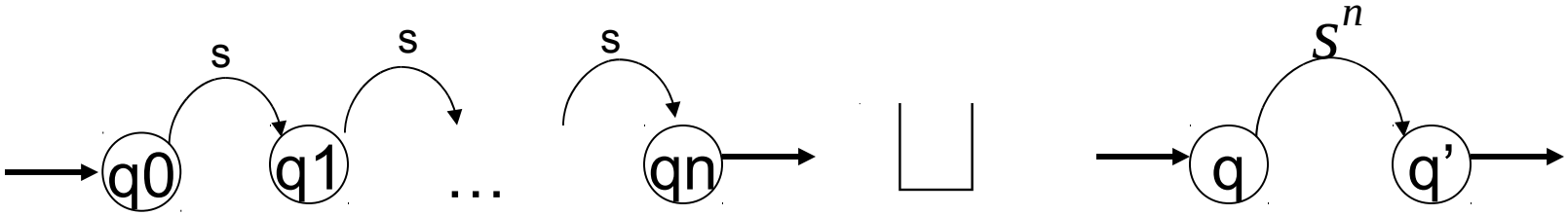


Rappels ou révélations

Remarques:

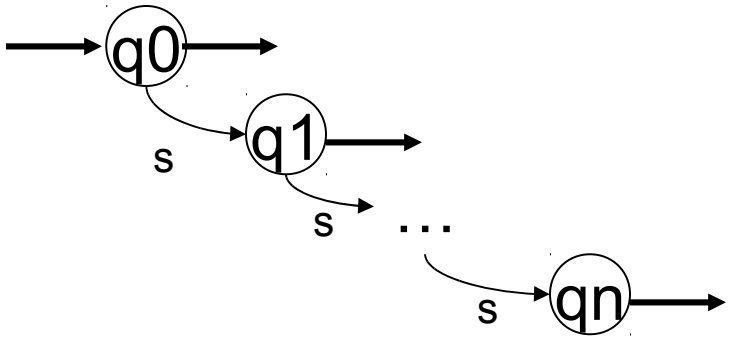
- En toute rigueur, on aurait du utiliser 9 flèches pour la transition état 0 -> état 1 (une par chiffre). Pour alléger les notations, on a recours à l'opérateur '+' pour dénoter le OU logique entre ***mots*** ; de même, l'opérateur '.' (ou produit) dénote le ET logique
- On peut alors étendre δ aux mots sur Σ, et on obtient les représentations suivantes :

1. Répétition n fois (n ≥1) d'un même symbole s

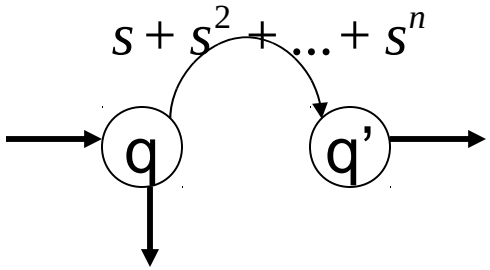


Rappels ou révélations

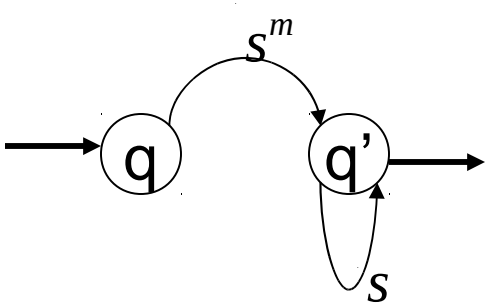
2. Répétition au plus n fois (n ≥1) d'un même symbole s



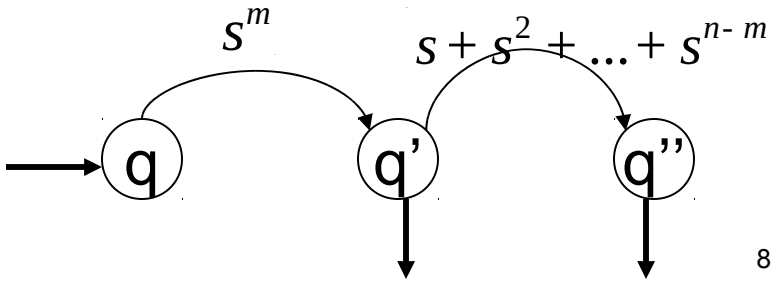
■



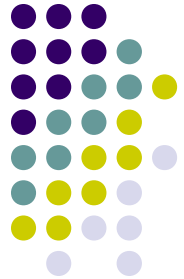
3. Répétition au moins m fois (m ≥1) d'un même symbole s



4. Répétition au moins m et au plus n fois (n > m ≥1) d'un même symbole s

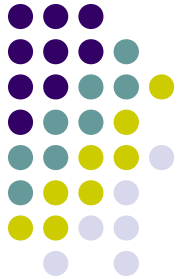


Expressions régulières



- Les expressions régulières (ER) UNIX sont des chaînes de caractères qui permettent de coder l'AEF reconnaissant un LR
- Elles servent généralement à compiler (fonction système C recomp()) puis à exécuter (regexec()) l'AEF de reconnaissance associé
- Deux formes usuelles d'ER définies par le standard IEEE 1003.2:
  - Forme **de base** : les méta-caractères '{','}','(',')' doivent être précédés d'un '\', sinon ils se comportent comme des caractères ordinaires. Les caractères '+', '?', et '|' sont toujours des caractères ordinaires. Les caractères '^' et '$' n'ont de sens qu'en début et en fin d'expression.
  - Forme **étendue** : celle décrite ci-après

ER : Forme étendue



Une ER étendue est un ensemble de sous-expressions régulières (SER) délimitées par des '|'. L'ER est reconnues ssi ***l'une*** des SER est reconnue.

Une SER est une succession d'atomes, chacun pouvant être suivi de '*', '+', '?', ou de bornes {inf,sup}. Une SER est reconnue ssi ***tous*** les atomes sont reconnus dans leur ordre (analyse gauche→droite) :

- Les bornes {inf,sup}, où inf et sup sont des entiers, indiquent que l'atome qui précède doit être reconnu au moins inf fois et au plus sup fois durant la reconnaissance. Il est possible d'omettre l'une ou l'autre des bornes, mais pas les deux.
- '*' est équivalent à '{0,}' - l'atome peut être reconnu un nombre arbitraire de fois, y compris zéro

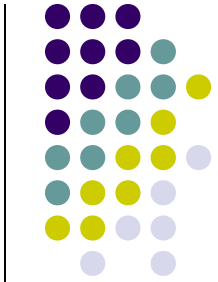
ER : Forme étendue

- ‘+’ est équivalent à ‘{1,}’ - l’atome doit être reconnu au moins une fois.
- ‘?’ est équivalent à {,1} - l’atome qui précède peut être reconnu au plus une fois

Un atome est soit :

- une ER délimitée par ‘(‘ et)’ (y compris la chaîne vide). Si ‘(‘ et ‘)’ sont changées en ‘\(‘,’\)’, l’ER représente la portion de chaîne à extraire de la chaîne candidate lors de la reconnaissance, si celle-ci réussit
- le caractère ‘.’, qui autorise la reconnaissance de n’importe quel caractère

ER : Forme étendue



- une expression crochetée, de la forme ‘[p…p]’, qui est reconnue ssi l’un des ‘p’ est reconnu. Chaque ‘p’ est:
  - soit une plage de la forme *x-y*, où *x* et *y* sont des caractères, et qui signifie « tous les caractères dans la plage x:y »
  - soit un caractère simple
  - soit une plage ou un caractère simple précédé par ‘^’, et qui signifie « tout sauf » la plage ou le caractère qui suit
- le caractère ‘^’, qui force la reconnaissance d’un début de ligne
- le caractère ‘$’, qui force la reconnaissance du mot vide (fin de la chaîne)
- une chaîne de la forme ‘\c’, où c est un caractère qui devra être reconnu comme tel - n’a d’intérêt que pour \^, \$, \[, etc.

ER : Forme étendue

Exemples

1. Expression régulière étendue reconnaissant les chaînes de la forme «Prénom Nom»:

   `[A-Z][a-z]+ [A-Z][a-z]+`

2. ER étendue reconnaissant un prénom éventuellement composé: «Prénom1[-Prénom2]»

   `[A-Z][a-z]+(-[A-Z][a-z]+)?`

3. ER étendue reconnaissant un prénom arbitrairement composé: «Prénom1[-Prénom2[-Prénom3]…]»

   `[A-Z][a-z]+(-[A-Z][a-z]+)*`

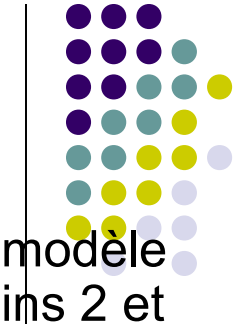
4. ER étendue reconnaissant les chaînes qui ne débutent pas par un ‘A’

   `^[^A]`

5. ER étendue reconnaissant les chaînes ne débutant pas par une lettre majuscule ou se terminant par une lettre minuscule, et comportant 3 caractères exactement:

   `^[^A-Z]..$|^..[a-z]$` ou bien: `^([^A-Z]..|..[a-z])$`

ER : Forme étendue



6. ER étendue reconnaissant une plaque minéralogique française (modèle simplifié: entier sur au plus 4 chiffres, espaces arbitraires, au moins 2 et au plus 3 lettres, espaces arbitraires, 2 chiffres):

   `[1-9][0-9]{,3} *[A-Za-z]{2,3} *[0-9]{2,2}`

Exercice:

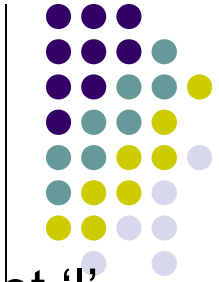
- ER reconnaissant un entier naturel:

- ER reconnaissant un entier relatif:

- ER reconnaissant un nombre flottant (sans notation scientifique ‘E’)

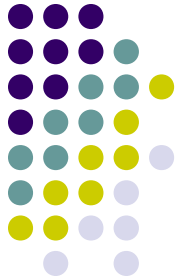
- ER reconnaissant un nombre flottant (avec notation scientifique ‘E’)

ER : Forme de base



- Forme moins puissante que la forme étendue, puisque ‘+’, ‘?’, et ‘|’ sont des caractères ordinaires
- Les fonctions associées à ‘+’ et ‘?’ disparaissent, cependant c+ équivaut à c\{0,\} et c? à c\{0,1\} → transformation toujours possible
- Les métacaractères (,), {, } de la forme étendue conservent leur sens s’ils sont précédés d’un \
- Les méta caractères ^ et $ gardent leur sens en début et fin d’expression sans besoin d’être précédés de \
- En revanche, le choix multiple ‘|’ est perdu, et le caractère ^ perd son sens négatif en milieu d’expression.

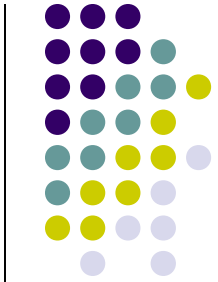
ER : Forme de base



Exemples :

Chaîne	ER de base	Concordance ?
`Alain`	`[A-Za-z]+`	non
`Alain`	`[A-Za-z]\{1,\}`	oui
`Pierre-Alain`	`[A-Za-z]\{1,\}\(-[A-Za-z]\{1,\}\)?`	non
`Pierre-Alain`	`[A-Za-z]\{1,\}\(-[A-Za-z]\{1,\}\)\{0,1\}`	oui
`Alain`	`^Alai$`	non
`Alain`	`^[A-Za-z]*n$`	oui
`Alain`	`^Alain$\|^Eric$`	non

La commande expr



Elle permet de tester la concordance d'une chaîne avec une expression régulière de base, et optionnellement d'extraire certaines parties reconnues

Syntaxe: `expr e1 op e2`

- e1 et e2 sont deux expressions, et op un opérateur (peut être : ou |, &, +, -, …)
- Si op est ':', alors e2 doit être une ER **de base**, et e1 une chaîne candidate. Expr écrit soit le résultat de la reconnaissance de e1 par e2 si e2 comporte des '\(\)', soit la longueur du résultat .

La commande expr

Exemple : reconnaissance d'une chaîne «Nom Prénom»

```
$ expr "Xavier Hilaire" : "[A-Z][a-z]+ [A-Z][a-z]+"
0
```

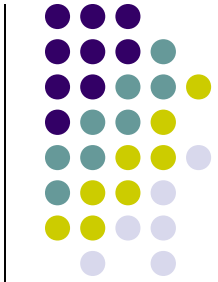
Ne marche pas : e2 doit être une ER **de base**. Donc:

```
$ expr "Xavier Hilaire" : "[A-Z][a-z]\{1,\} [A-Z][a-z]\{1,\}"
14
```

On peut aussi extraire une partie de l'ER reconnue, à condition de la délimiter par '\(' et '\)'; par exemple, pour le nom seul:

```
$ expr "Xavier Hilaire" : "[A-Z][a-z]\{1,\} \([A-Z][a-z]\{1,\}\)"
Hilaire
$
```

La commande expr

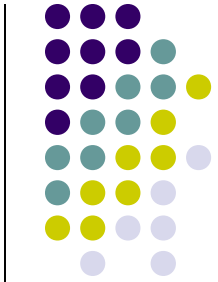


Remarques:

1. Du fait de l'utilisation des ER de base, expr ne peut pas utiliser de parenthésage sans réaliser l'extraction !
2. On utilise plus fréquemment le code de retour de expr que le résultat de la reconnaissance proprement dit:
   - 0 → sortie avec succès
   - 1 → chaîne candidate non reconnue
   - 2 → ER invalide

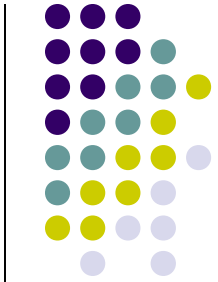
Utilisation de expr (avec extraction) pour les exemples précédents (à vérifier): voir TD

La commande grep



- Elle lit le(s) fichier(s) passés en paramètre, ou à défaut l'entrée standard, et écrit les lignes du fichier pour lesquelles la reconnaissance de l'ER exp réussit.
- Syntaxe: `grep [options] exp [fichiers…]`
- Options utiles:
  - -E : interpréter exp comme une ER étendue (par défaut, c'est une ER de base)
  - -l : écrit les numéros des lignes qui concordent plutôt que les lignes elles-mêmes
  - -L : écrit les noms des fichiers qui concordent (au moins une occurrences) plutôt que les lignes
  - -v : inverse le comportement de grep - écrit les lignes qui ne concordent pas

La commande grep



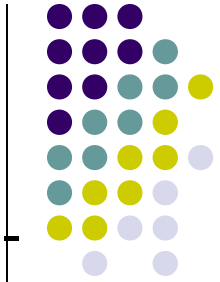
Exemples

```
$ cat /etc/passwd
##
# User Database
##
nobody::-2:-2:Unprivileged User:/:/usr/bin/false
root:*:0:0:System Administrator:/var/root:/bin/sh
daemon:*:1:1:System Services:/var/root:/usr/bin/false
xavier:a293deFZ=/K52:1001:1001:Xavier
   Hilaire:/home/xavier:/usr/local/bin/bash
```

- Elimination des lignes de commentaires (commencent par '#'):

```
$ grep "^[^#]" /etc/passwd Ou: grep -v "^#" passwd
nobody::-2:-2:Unprivileged User:/:/usr/bin/false
root:*:0:0:System Administrator:/var/root:/bin/sh
daemon:*:1:1:System Services:/var/root:/usr/bin/false
xavier:a293deFZ=/K52:1001:1001:Xavier
   Hilaire:/home/xavier:/usr/local/bin/bash
```

La commande grep



- Utilisateurs n'ayant pas de mot de passe (2ème champ vide - situation anormale!):

```
$ grep -E "^[^:]*::" /etc/passwd
nobody::-2:-2:Unprivileged User:/:/usr/bin/false
```

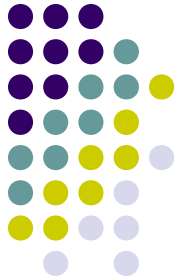
- Utilisateurs ayant un compte « étoilé » (2ème champ est '*'):

```
$ grep -E "^[^:]*:\*:" /etc/passwd
root:*:0:0:System Administrator:/var/root:/bin/sh
daemon:*:1:1:System Services:/var/root:/usr/bin/false
```

- Utilisateurs dont le chemin d'accès au shell (7ème champ) comporte au moins 3 répertoires:

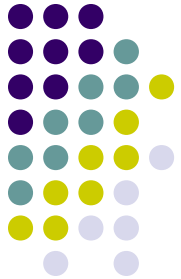
```
$ grep -E "^([^:]*:){6,6}(/[^/]+){4,}" /etc/passwd
xavier:a293deFZ=/K52:1001:1001:Xavier
   Hilaire:/home/xavier:/usr/local/bin/bash
```

La commande sed



- Syntaxe (simplifiée):
  - `sed [-E] com [fichiers …]`
  - `sed [-E] -e com [-e com [-e com …]] [fichiers …]`
  - `sed [-E] -f fcom [fichiers …]`
- La commande sed (*stream editor*) effectue dans l'ordre et cycliquement les opérations suivantes:
  - Lit une ligne à partir du premier fichier spécifié, sinon l'entrée standard
  - Copie cette ligne dans un espace tampon (ET)
  - Applique la fonction com sur l'ET si c'est la seule spécifiée; ou l'ensemble des fonctions com si l'option -e est utilisée, dans l'ordre; ou l'ensemble des fonctions stockées dans le fichier fcom

La commande sed



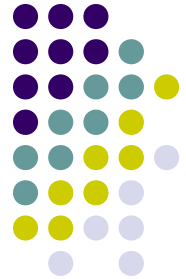
- Ecrit le contenu de l'ET sur la sortie standard
- Libère l'ET
- Passe à la ligne suivante s'il en reste
- Passe au fichier suivant s'il y en a

- Il existe plus d'une vingtaine de fonctions différentes, mais nous ne verrons que la plus utile d'entre elles: 's' (substitute) :

  `s/regex/rempl/[flags]`

- La fonction 's' teste si l'ET concorde avec l'ER regex spécifiée.
- S'il y a concordance, l'ET est substitué par l'expression rempl, la substitution étant paramétrée par la ou les valeurs données dans les drapeaux flags, optionnels:

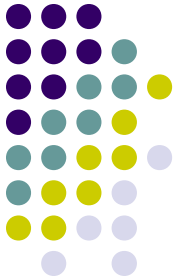
La commande sed



- N : ne remplacer que la Nième occurrence de regex dans l'ET
- g : remplacer toutes les occurrences dans l'ET - par défaut, seule la première occurrence est remplacée
- p : écrit l'ET vers la sortie standard
- w fichier : ajoute l'ET à fichier

- S'il n'y a pas concordance, l'ET est laissé intact.
- L'expression rempl peut contenir:
  - Le caractère &, qui sera remplacé par toute la chaîne consommée lors de la reconnaissance de regex
  - Des séquences de la forme \# ,où # est un chiffre (1-9) désignant la nième des portions d'ER reconnues et délimitées par '\(\)' si regex en comporte
- regex est supposée « de base », sauf si l'option -E a été spécifiée lors de l'appel à sed

La commande sed



Exemples

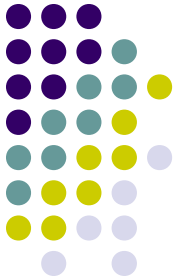
- Supprimer toutes les occurrences de ‘root’ dans le fichier passwd précédent:

```
$ sed "s/root//g" passwd
##
# User Database
##
nobody::-2:-2:Unprivileged User:/:/usr/bin/false
:*:0:0:System Administrator:/var/:/bin/sh
daemon:*:1:1:System Services:/var/:/usr/bin/false
xavier:a293deFZ=/K52:1001:1001:Xavier
   Hilaire:/home/xavier:/usr/local/bin/bash
```

- Ne conserver que le premier champ:

```
$ sed  "s/\([^:]*\):.*/\1/" passwd
##
# User Database
##
nobody
root
daemon
xavier
```

La commande sed



- Supprimer les deux premiers champs:

```
$ sed -E -e "s/[^:]*://1" -e "s/[^:]*://1" passwd
##
# User Database
##
-2:-2:Unprivileged User:/:/usr/bin/false
0:0:System Administrator:/var/root:/bin/sh
1:1:System Services:/var/root:/usr/bin/false

1001:1001:Xavier Hilaire:/home/xavier:/usr/local/bin/bash
```

- Permuter les champs 1 et 3:

```
$ sed "s/\([^:]*\):\([^:]*\):\([^:]*\)/\3:\2:\1/" passwd
##
# User Database
##
-2::nobody:-2:Unprivileged User:/:/usr/bin/false
0:*:root:0:System Administrator:/var/root:/bin/sh
1:*:daemon:1:System Services:/var/root:/usr/bin/false
1001:a293deFZ=/K52:xavier:1001:Xavier
   Hilaire:/home/xavier:/usr/local/bin/bash
```