



# AWK et les expressions rationnelles

par Philippe Teuwen

# AWK et les expressions rationnelles

## Contenu de la présentation

Introduction

Où et quand AWK peut être utile

Description du langage proprement dit

Exemples

Exercice: rédigez votre propre script

Bibliographie



# AWK et les expressions rationnelles

## Contenu de la présentation

Introduction

Où et quand AWK peut être utile

Description du langage proprement dit

Exemples

Exercice: rédigez votre propre script

Bibliographie



# AWK et les expressions rationnelles

## Introduction

Awk est l'acronyme du nom de ses créateurs:  
**A**ho, **W**einberger et **K**ernighan

Awk est un langage de programmation interprété

De nombreuses implémentations sont disponibles pour pratiquement toutes les plateformes:

UNIX, Linux, Windows (via Cygwin ou seul),  
MAC OS, ...



# AWK et les expressions rationnelles

## Introduction

Le but de cette présentation n'est pas de donner une vue exhaustive du langage AWK même si elle est déjà suffisante pour vous permettre d'écrire de puissants scripts pour toute une série de tâches.

Mon seul but est de vous montrer le bénéfice d'AWK dans vos problèmes de tous les jours.

Pour une vue exhaustive, consultez les références  
(Ch2 de "The Awk language", 44 pages, ça couvre tout!)



# AWK et les expressions rationnelles

## Contenu de la présentation

Introduction

Où et quand AWK peut être utile

Description du langage proprement dit

Exemples

Exercice: rédigez votre propre script

Bibliographie



# **AWK et les expressions rationnelles**

Où et quand AWK peut être utile  
AWK peut faire beaucoup de choses!  
Ses applications les plus évidentes sont:  
Conversion rapide de format de données  
Extraction rapide de logs ou de résultats de tests  
Comptage rapide d'occurrences

Toutes ces petites tâches où le temps de développement d'un programme est plus important que la vitesse d'exécution.  
(qui n'est pas mauvaise pour autant!)



# **AWK et les expressions rationnelles**

Où et quand AWK peut être utile

Mais AWK permet de faire encore plus:

Data processing & data validation

Génération de rapports

Gestion de bases de données relationnelles

Parsing et même compilation de nouveaux langages ou de langages connus

...



# AWK et les expressions rationnelles

## Contenu de la présentation

Introduction

Où et quand AWK peut être utile

Description du langage proprement dit

Exemples

Exercice: rédigez votre propre script

Bibliographie



# AWK et les expressions rationnelles

## Description du langage

Expressions rationnelles

Awk et C

Structure d'un programme AWK

Variables système

Syntaxe de la ligne de commande

Langage

Fonctions



# AWK et les expressions rationnelles

## Description du langage

Expressions rationnelles

Awk et C

Structure d'un programme AWK

Variables système

Syntaxe de la ligne de commande

Langage

Fonctions



## Expressions rationnelles (Regular expressions)

Les expressions rationnelles sont utilisées partout:  
dans Sed, AWK, grep, Perl, emacs, manpages,...  
Les connaître vous aidera bien au-delà de AWK.  
C'est une description théorique de motifs (*patterns*),  
donc universelle.

Une expression régulière est souvent composée d'une valeur  
(variable ou littérale) et d'un opérateur.

Tous les caractères excepté les caractères spéciaux sont  
interprétés comme étant des caractères normaux. (logique...)  
Tout le charme (ou la complexité) d'une expression provient  
de ces caractères spéciaux, appelés *méta-caractères*.



## Expressions rationnelles

### Les caractères spéciaux:

- n'importe quel car., sauf le saut de ligne
- [...] un des car. présents entre les [ ]  
si le 1<sup>er</sup> est "**^**" alors l'ensemble devient (. \ [...])  
un "**-**" peut définir un intervalle.  
intervalles POSIX disponibles: [[ :alnum: ]], ...
- ^** si 1<sup>er</sup> car. d'une expr: début de ligne
- \$** si dernier car. d'une expr: fin de ligne
- \** protection du car. suivant, pour spécifier un méta-car.  
comme un caractère normal

*Attention: les caractères peuvent être aussi interprétés par le shell lui-même, donc utiliser ' et pas "*



## Expressions rationnelles

Les opérateurs (qui sont aussi des car. spéciaux!):

\* 0 à n occurrences du caractère précédent

+ 1 à n occurrences

? 0 ou 1 occurrence

{n} exactement n occurrences

{n, } au moins n occurrences

{n, m} entre n et m occurrences

| OU entre l'expression de gauche et celle de droite

( ) regroupe des expressions

*exemples... (pour les essayer, utiliser **egrep**)*



# AWK et les expressions rationnelles

## Expressions rationnelles

Exemples de motifs: comment les retrouver?

Null, a, aa, aaa, ...

$a^*$

aab, aabab, aababab, ...

$a(ab)^*$

a, b, c, d

$[a-d]$

a, b, c, 1, 2, 3

$[a-c] | [1-3]$

ligne de C qui commence par //comment

$^{^//}$

ligne vide

$^{\$}$

une date au format dd/mm/yyyy entre 1900 & 2099

$([1-9]|([12][0-9]|3[01]))/$   
 $([1-9]|1[0-2])/([19][0-9]\{2\}|20[0-9]\{2\})$



## Expressions rationnelles

Exemple inspiré d'un cas réel :

Il faut chercher dans les codes sources (en assembleur) de tous nos modules après les lignes qui contiennent un type très particulier d'instructions multiples (sur DSP)...

Liste des occurrences possibles de certaines instructions "double parallel move" susceptibles d'être supprimées du jeu d'instructions:

instruction

- + 2 parallel moves VERS la mémoire
- A PARTIR DE certains registres: 1, 3, 5 ou 7

Rem: sur un DSP, on a 2 types de mémoire data: X & Y



## Expressions rationnelles

instr , ... {x|y}ptr ... , \*{x|y}ptr ... = r{1|3|5|7}  
ou      instr , \*{x|y}ptr ... = r{1|3|5|7} , ... {x|y}ptr ...

```
(, .* \* [xy]ptr ([0-9]|1[0-5]).*,  
[ \t]* \* [xy]ptr .*= [ \t]* r[1357]) |  
(,[ \t]* \* [xy]ptr .*= [ \t]* r[1357] [ \t]*,  
.*[xy]ptr)
```



# AWK et les expressions rationnelles

## Description du langage

Expressions rationnelles

Awk et C

Structure d'un programme AWK

Variables système

Syntaxe de la ligne de commande

Langage

Fonctions



# AWK et les expressions rationnelles

## Awk and C

Awk est très simple comparé au C ou à Perl.

Néanmoins il y a quelques similitudes:

Les opérateurs arithmétiques et d'assignement sont les mêmes ( **i++**, **a+=b**, ...)

+ l'opérateur exponentiel en Awk: **^**

Les opérateurs booléens sont les mêmes sauf que les opérateurs bit à bit n'existent pas.

Les opérateurs de correspondance sont **~** et **!~**

Certaines fonctions C sont disponibles comme:

**printf()**, **sprintf()**, qqs fcts math, **rand()**,...  
et même **system()**



# AWK et les expressions rationnelles

## Awk and C

Il y a aussi quelques différences notables:

Il n'y a pas vraiment de notion de type.

Tout est nombre ou chaîne (de caractères).

Si requise, la conversion peut être forcée par:

**string + 0 (or \* 1) -> number**

**number + "" -> string**

Les commentaires commencent par **#**



# AWK et les expressions rationnelles

## Awk and C

Une instruction finit par ; ou \n ou }

**Printf** est plus riche et plus pratique

Tableaux associatifs (*hash tables*)

Allocation de mémoire automatique

Déclaration et initialisation des variables implicite



# AWK et les expressions rationnelles

## Description du langage

Expressions rationnelles

Awk et C

Structure d'un programme AWK

Variables système

Syntaxe de la ligne de commande

Langage

Fonctions



## Structure d'un programme AWK

Structure des données:

Les données peuvent venir de stdin, de fichiers ou processés.

Les données traitées par Awk seront scindées en enregistrements:*records* et en champs:*fields*.

Par défaut, un *record* est une ligne

( séparateur = carriage return )

Par défaut, un *field* est un mot

( séparateur = espace ou tabulation )

Mais vous pouvez spécifier ce que vous voulez comme séparateurs.



## Structure d'un programme AWK

Un programme Awk est composé d'instructions de la forme:

**pattern {action}**

Le motif (*pattern*) peut être une expression, une expression rationnelle, un motif composite, une fourchette (*range*) de motifs ou:

**BEGIN:** exécuté avant le traitement des données

**END:** exécuté après le traitement des données

En bref, chaque ligne de donnée est lue, scindée (*parsed*) et testée vis-à-vis des motifs.  
S'il y a correspondance, l'action conjointe est exécutée.



# AWK et les expressions rationnelles

## Structure d'un programme AWK

Une expression:

**\$1>i**

Une expression rationnelle (*regular expression*):

**/^[Cc]ountry/**

Un composite (*compound*):

Combination of patterns with :**&&**, **||**, **!**, **( )**, **? :**

Une fourchette (*range*):

Le test sera vrai depuis l'instant où **pat1** est vrai jusqu'à inclus le moment où **pat2** devient vrai:

**pat1, pat2**



# AWK et les expressions rationnelles

## Structure d'un programme AWK

Exemple:

```
BEGIN { toto=0; }      (décl. & init. optionnelles)
/toto/ { toto++; }
END { print toto; }
```

Motif par défaut est **TRUE**

= // = /.\*/

Ex: { **print** \$1 }

Action par défaut est **{print}**

= {**print** \$0}

Ex: /toto/



# AWK et les expressions rationnelles

## Description du langage

Expressions rationnelles

Awk et C

Structure d'un programme AWK

Variabes système

Syntaxe de la ligne de commande

Langage

Fonctions



## Variables système

Des variables systèmes sont utilisable avec Awk:

**\$0**: enregistrement (*record*) courant

**\$1, \$2,...**: 1<sup>er</sup> champ, 2<sup>ème</sup> champ,... de l'enregistrement courant

**FS**: *Field Separator* pour *parser* les enregistrements

**OFS**: *Output Field Separator* pour l'impression

**NF**: *Number of Fields* dans l'enregistrement courant

**RS**: *Record Separator* à la lecture des données (souvent \n)

**ORS**: *Output Record Separator* pour l'impression

**NR**: *Number of the current Record*

**FILENAME**: nom du fichier de données

**FNR**: même que NR mais par rapport au fichier courant

**ARGC, ARGV, OFMT,...**



# AWK et les expressions rationnelles

## Description du langage

Expressions rationnelles

Awk et C

Structure d'un programme AWK

Variables système

Syntaxe de la ligne de commande

Langage

Fonctions



# AWK et les expressions rationnelles

## Syntaxe de la ligne de commande

Des options peuvent être passées à Awk: (cf manpage)

```
gawk --posix -f script.awk place=Brussels
```

script:

```
/[0-9]{4}/ {if ($2 ~ place) print;}
```

Il trouvera ce type d'enregistrement:

**1000 Brussels**

Une meilleure alternative pour le script:

```
($1 ~ /[0-9]{4}/)&&($2 ~ place)
```



# AWK et les expressions rationnelles

## Syntaxe de la ligne de commande

Si le script Awk est court, il peut être directement écrit dans la ligne de commande:

```
gawk --posix '$1 ~ /[0-9]{4}/) && ($2 ~ Leuven)'
```

Par défaut, stdin est pris comme entrée & stdout comme sortie  
-> On peut utiliser les redirecteurs: < > |

Rem: **--posix** donne accès aux expressions d'intervalles **{n}...**  
& aux intervalles POSIX **[[:alpha:]]...**



## Description du langage

Expressions rationnelle

Awk et C

Structure d'un programme AWK

Variables système

Syntaxe de la ligne de commande

Langage

Fonctions



# AWK et les expressions rationnelles

## Langage

Awk ne serait pas un langage sans ces éléments:

- instructions conditionnelles:

```
if (expr) instr1 [else instr2]
```

```
if (expr) {instr bloc} [else ...]
```

- boucles:

```
while (expr) instr
```

```
do instr while (expr)
```

```
for (init ; test ; incr) instr
```

```
for (var in array)
```

```
break, continue, exit
```

- l'instruction vide est dénotée par ";" seul



# AWK et les expressions rationnelles

## Langage

- Tableaux:

Pas d'initialisation, pas d'allocation mémoire à prévoir, un tableau est directement accessible par:

**name[index]** index peut être un nombre ou une chaîne!

Pour tester si un certain index est présent dans un tableau:

**index in name**

Ex:

**if("age" in John)...**

Jamais:

**!!! if(John["age"] != "")... !!!**



# AWK et les expressions rationnelles

## Langage

Quelques fonctions bien utiles:

**print**:

```
{print $1,$3,"text"}
```

-> l'*Output Field Separator* (OFS) sera utilisé

```
{print $1 $3}
```

-> les chaînes seront concaténées

Le retour à la ligne est implicite.

**printf**: aussi puissant que **print** mais permet de surcroît les définitions de format tout comme le célèbre **printf** d'ANSI-C.

Dans ce cas **\n** doit être explicite.



# AWK et les expressions rationnelles

## Langage

D'autres fonctions utiles (principalement manipulation de chaînes):

**length(s)**: longueur de la chaîne **s**

**exp(x)**, **log(x)**, **sqrt(x)**, **int(x)**, ...

**split(s1,v,s2)**: scinde **s1** en tabl. **v** avec **s2** pour séparateur.

**substr(s,p,n)**: extrait de **s**, de la position **p**, **n** caractères

**index(s1,s2)**: position de **s1** dans **s2**

**gsub(r,s,t)**: substitue globalement les sous-chaînes correspondant au motif **r** par **s** dans la chaîne **t**, retourne le **#** de substitutions; si **t** est omis, **\$0** est utilisé.

...



# AWK et les expressions rationnelles

## Langage

E/S (I/O):

Les redirecteurs peuvent être utilisés pour rediriger la sortie vers un fichier: > & >>

Ex:

```
$3 > 100 {print $1 , $3 > "bigpop"}  
$3 <= 100 {print $1 , $3 > "smallpop"}
```

Ou même:

```
{print($1,$3) > ($3 > 100? "bigpop" : \  
"smallpop")}
```

Les tubes (*pipes*) " | " peuvent être utilisés pour traiter les données de sortie par un autre programme



# AWK et les expressions rationnelles

## Langage

E/S (*I/O*):

```
print > "c:/temp/test.txt"
print > "c:/temp/test.txt"
...
close("c:/temp/test.txt")
```

Le premier **print** ouvre le fichier.

Idem pour les tubes, la première instance ouvre le tube vers un programme.

Si on veut explicitement fermer un fichier:

Par exemple pour un tube:

```
close ("| external_prog");
```



# AWK et les expressions rationnelles

## Langage

E/S (I/O):

**getline()**: force à lire la prochaine ligne du fichier d'entrée (ou autre fichier/tube) et permet ainsi de se passer de la boucle principale.

**getline**

remplace **\$0** & **NF**, **NR**, **FNR**

**getline var**

utilise **var** & **NR**, **FNR**

**getline < "file"**

remplace **\$0** & **NF**

aussi "*cmd*" | **getline**

**getline var < "file"** utilise **var**

aussi "*cmd*" | **getline var**

Ne jamais utiliser **while (getline < "file")**

car si le fichier n'existe pas, **getline** renvoie -1 = boucle ∞ !!

Utiliser de préférence:

**while (getline < "file" > 0)**



# AWK et les expressions rationnelles

## Langage

**system("cmd")** : exécute la ligne de commande.

Ex.: un fichier avec une liste de noms de fichiers  
(sortie d'une commande "find")

```
{system("cp " $1 " .")}
```

  ^        ^

```
{system("emacs " $1 " &")}
```

  ^        ^



# AWK et les expressions rationnelles

## Description du langage

Expressions rationnelles

Awk et C

Structure d'un programme AWK

Variables système

Syntaxe de la ligne de commande

Langage

Fonctions



# AWK et les expressions rationnelles

## Fonctions

Il est possible de définir ses propres fonctions:

```
function square(number, local_var) {  
    local_var = "apple"  
    global_var = "banana"  
    return number * number  
}
```

Usage: **square(4)**

Seuls les arguments sont considérés comme locaux dans la fonction.  
Donc le seul moyen de déclarer des variables locales est  
d'en faire des arguments optionnels (et non utilisés).  
La convention est de les séparer avec des espaces  
supplémentaires.



# AWK et les expressions rationnelles

## Contenu de la présentation

Introduction

Où et quand AWK peut être utile

Description du langage proprement dit

**Exemples**

Exercice: rédigez votre propre script

Bibliographie



# AWK et les expressions rationnelles

## Exemples

```
# Transformer du texte ASCII en HTML
BEGIN {
    printf("<HTML>\n<HEAD>\n");
    printf("    <TITLE>My Title</TITLE>\n");
    printf("</HEAD>\n<BODY color=white>\n");
}

{ print $0 "<BR>" }

END {printf("</BODY>\n</HTML>\n"); }
```



# AWK et les expressions rationnelles

## Exemples

```
# Transformer du texte ASCII en HTML
BEGIN {
    printf("<HTML>\n<HEAD>\n"
           "      <TITLE>My Title</TITLE>\n"
           "</HEAD>\n<BODY color=white>\n");
}
{ print $0 "<BR>" }
END {printf "</BODY>\n</HTML>\n"); }
```



# **AWK et les expressions rationnelles**

## **Exemples**

Le langage est très flexible, puissant et compact...

Etant donné une liste de noms et de n<sup>os</sup> de téléphone:

John 652

Jack 640

Comment lister les noms?

```
{names = names $1 " "}  
END {print names}
```



# AWK et les expressions rationnelles

## Exemples

Le langage est très flexible, puissant et compact...

Etant donné une liste de noms et de n<sup>os</sup> de téléphone:

John 652

Jack 640

Comment lister les noms?

```
{names[$1] = $2}  
END {for (I in names){  
    print "Le tel. de " I " est " names[I]  
    }  
}
```



# AWK et les expressions rationnelles

## Exemples

Comment lister un fichier dans l'ordre inverse? (cf **tac**)

```
{line[NR]==0}  
END {for (i=NR;i>0;i--) print line[i]}
```



# AWK et les expressions rationnelles

## Exemples

Comment imprimer la ligne 27?

**NR==27**



# AWK et les expressions rationnelles

## Exemples

Comment ôter les lignes vides?

**NF>0**



# AWK et les expressions rationnelles

## Exemples

Comment imprimer le corps d'un fichier HTML?

```
/<BODY/, /<\BODY>/
```

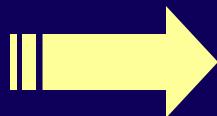


# AWK et les expressions rationnelles

## Exemples

Comment convertir des données ?

John:653:C142  
Jack:620:A145



John;C142;653  
Jack;A145;620

```
BEGIN {FS=":";OFS=","}  
{print $1,$3,$2}
```



# AWK et les expressions rationnelles

## Exemples

Extraction d'information depuis un *profiling result*.

La question posée:

Combien de temps est passé dans un certain type d'instructions?:

les ASIs



Out.txt

```
/asi/{total+=$4}  
END {print total}
```



# AWK et les expressions rationnelles

## Exemples

Extraction d'information depuis un *profiling result*.

La question posée:

Combien de temps est passé dans un certain type d'instructions?:  
les ASIs

Ligne de commande:

```
gawk '/asi/{total+=$4};END {print total}'  
< out.txt
```



# AWK et les expressions rationnelles

## Exemples

Comment écrire un script pour récupérer et mettre en forme des données du Web?

Le but est de connaître qui se cache derrière un numéro de tél.:

**tellwho 02 7429583**

Il faut un site Web: Belgacom

On *sniff* une requête standard en utilisant leur service:

```
GET http://www.belgacom.be/cgi-bin/  
whitePages?L=F&TPL=IP2&BYNUMBER=YES&P1=02&P2=7429583
```

(ça peut aussi être une requête POST  
à convertir en requête GET)



# AWK et les expressions rationnelles

## Exemples

On peut utiliser par exemple LYNX et écrire un script:

```
http_proxy=http://mon_proxy.com:8080/ lynx -dump  
"http://www.belgacom.be/cgi-bin/whitePages?  
L=F&TPL=IP2&BYNUMBER=YES&P1=$1&P2=$2"
```



### Résultats

1 Name

Address

City

Phone nr

Votre recherche:

Téléphone \_\_\_\_\_

Envoyer



# AWK et les expressions rationnelles

## Exemples

On filtre alors le résultat de la requête:

```
/Votre recherche/{exit}  
body==1  
/Résultats/{body=1}
```



# AWK et les expressions rationnelles

## Exemples

Le script complet devient:

```
#!/bin/bash
#Usage:
#>tellwho 02 1234567
http_proxy=http://mon_proxy.com:8080/ lynx -dump \
"http://www.belgacom.be/cgi-bin/whitePages?L=F&TPL=IP2&BYNUMBER=\
YES&P1=$1&P2=$2" | gawk '
/Votre recherche/{exit}
body==1
/Résultats/{body=1}
'
```



# AWK et les expressions rationnelles

## Exemples

Le script complet devient:

```
#!/bin/bash
#Usage:
#>tellwho 02 1234567
http_proxy=http://mon_proxy.com:8080/ lynx -dump \
"http://www.belgacom.be/cgi-bin/whitePages?L=F&TPL=IP2&BYNUMBER=\
YES&P1=$1&P2=$2" | gawk '/Votre recherche/{exit};body==1;/Résultats/\
{body=1}'
```

Une seule (longue) ligne!



# AWK et les expressions rationnelles

## Contenu de la présentation

Introduction

Où et quand AWK peut être utile

Description du langage proprement dit

Exemples

Exercice: rédigez votre propre script

Bibliographie



## Exercice: rédigez votre propre script

Comment compter les lignes de code dans un fichier de sources écrit en C?

En excluant les commentaires et les lignes vides évidemment...

Attention: pas de double-comptage des commentaires imbriqués svp!



# AWK et les expressions rationnelles

## Exercice: solution

Comment compter les lignes de code  
dans un fichier de sources écrit en C?  
En excluant les commentaires et  
les lignes vides évidemment...

```
{com_state=0}
/\/*/,/*//{comment++;com_state=1}
(NF==0||/^\/\//)&&!com_state{comment++}
END {print(NR-comment)}
```



# AWK et les expressions rationnelles

## Exercice: solution

Comment compter les lignes de code  
dans un fichier de sources écrit en C?  
En excluant les commentaires et  
les lignes vides évidemment...

Encore plus court:

```
/\\/*/, /*//{comment++;next}  
NF==0 || /*//{comment++}  
END {print(NR-comment)}
```



# AWK et les expressions rationnelles

## Contenu de la présentation

Introduction

Où et quand AWK peut être utile

Description du langage proprement dit

Exemples

Exercice: rédigez votre propre script

Bibliographie



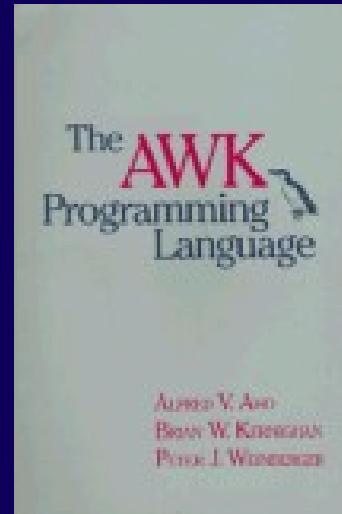
# AWK et les expressions rationnelles

## Bibliographie

**man gawk**

The AWK Programming Language

Aho, Weinberger & Kernighan



- Sed & Awk, advanced programming  
(traduit en français par Marc Vauclair)

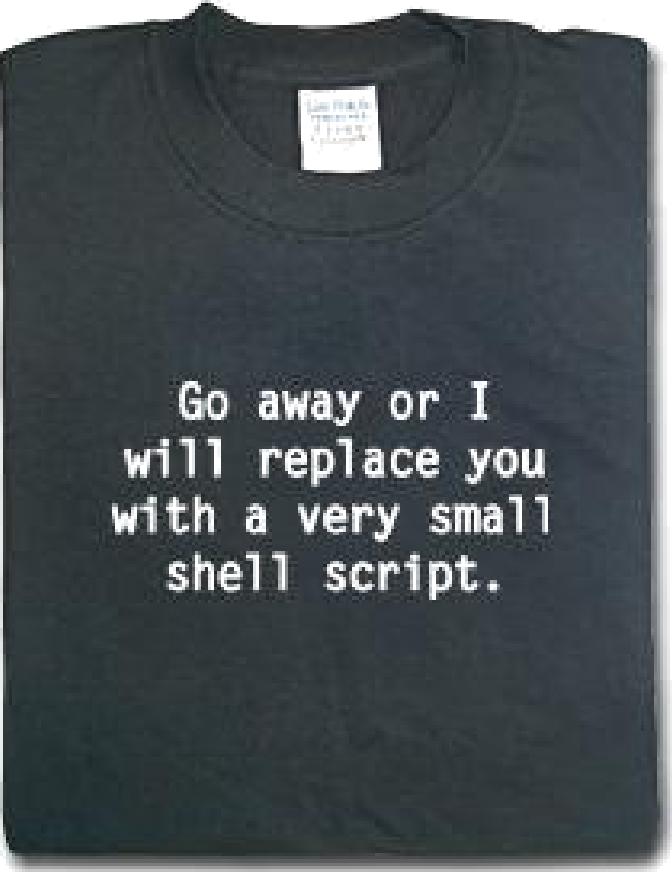
La présentation se basait également sur  
un article d'Eric Dumas pour l'Echo de Linux (Nov 1996)

L'oiseau qui est emblématique d'Awk est l'Auk,  
Grand Pingouin, espèce disparue mais dont le nom est resté...  
- <http://www.birdsofna.org/excerpts/auk.html>



/(\bb|[\^b]{2})/





Go away or I  
will replace you  
with a very small  
shell script.