Systèmes d'exploitation

AWK

Yves Stadler

Codasystem, UPV-M

13 mars 2013

Historique

Vocabulaire

- 1977, langage de programmation, UNIX
- Alfred V. Aho, Peter J. Weinberger et Brian W. Kernighan.
- Créer pour remplacer grep et sed.
- Dispose d'une grande souplesse
- Plus de possibilité "programmative" que sed.

Premiers pas

Appel

- `#!/usr/bin/awk -f`
- `awk -f fscript < finput`
- `ls | awk -f fscript`

Terminologie

Enregistrement

```
Fichier := record \n record ... \n record \n EOF;
```

Champs

```
record := field SEP field ... SEP field SEP field \n;
```

Terminologie

Premières lignes

- Le dièse est commentaire
- `Condition { actions }`
- ; ou `\n` pour séparer les actions
- ```
  Condition {
  action1
  action2; action3
  }
  ```
- ```
  Condition \
  {
  actions
  }
  ```


Le début

BEGIN

- Condition : `BEGIN { actions }`
- S'exécute avant le début de traitement du fichier
- Pratique pour initialiser les variables, ou afficher un message de bienvenue
- `BEGIN { print "Debut du script AWK" }`

END

- Condition : `END { actions }`
- S'exécute après le traitement du fichier
- Pratique pour afficher les résultats, ou un message de fin
- `END { print "Fin du script AWK" }`

Variables

Typage

- Comme bash, awk ne connait pas de type
- `variable_1 = "Bonjour"`
- `variable_1 = 1`
- Le type est interprété en fonction du contexte

Tableau

- Comme en bash : `tab[0] = "Bonjour"`
- `tab["item1"] = 1`

Variables spéciales

Billet vert

- `$0` désigne toute la ligne
- `$i` désigne le ième champ de la ligne
- Séparateur : espace, tabulation (Peut être modifier)
- `n = 2; print $n` affiche le champ 2

Variables globales

- `$NF` nombre de champ sur la ligne (number of fields)
- `$FS` contient l'expression pour les séparateurs de champs (peut être une RE)
- `$NR` nombre de ligne (number of records)
- `$ORS` séparateur de ligne (`\n` par défaut)

Exemples

Exemples

```
BEGIN { print "Le script qui affiche le premier champ" }
{ print $1 }
END { print "Le script est fini" }

BEGIN { print "Le script qui affiche le nombre de
ligne sans utiliser $NR"
total = 0 }
{ total = total + 1 }
END { print "Le total de lignes est " $NR " ou " total }
```


Conditions

Opérateurs

- == != <= >= < >
- && ||
- + - / * % ^
- += -= /= *= %/ ++ --

Motifs

- /regexp/ { actions }
- var ~ /regexep/ { actions }
- var !~ /regexp/ { actions }

Exemples

Script 1

```
BEGIN { print "Le script qui compte le nombre
 de ligne contenant des nombres"
total = 0 }
/[0-9]+/ { total = total + 1 }
END { print "Le total est " total }
```

Script 2

```
BEGIN { print "Le script qui compte le nombre
 de ligne dont le champ 1 est un nombres"
total = 0 }
$1 ~ /[0-9]+/ { total = total + 1 }
END { print "Le total est " total }
```


Fonctions

Arithmetiques

- sqrt(x) racine carrée
- atan2(y,x) arctangente de x/y en (radians)
- cos(x) cosinus (radians)
- sin(x) sinus (radians)
- exp(x) exponentielle e^x
- int(x) valeur entière
- log(x) logarithme
- rand() nombre aléatoire entre 0 et 1
- srand(x) initialisation de rand

Affichage

- print
- printf()

Boucles, etc...

Si le père noel exite alors ...

- `if (condition)`
  `action`
- `if (condition) {`
  `actions`
  `} [else {`
  `actions`
  `}]`

Pour...

- `for (start; condition; step) {`
  `actions`
  `} #C-like`
- `for (var in tableau) {`
  `actions`
  `} #Bash-like`

Introduction

Vocabulaire

```
awk 'BEGIN { print "Mémorisation de votre fichier " FILENAME }
                {memfile [NR] = $0 }
          END   { for ( i = NR ; i >= 1 ; i-- ) {
                    print i ":" memfile[i]
                    }
         print "Fin"
                } ' fichier
```


Tant qu'il y aura des ombres

While

- `while (condition)`
  `{`
  `actions`
  `}`
- `do`
  `{`
  `actions`
  `} while (condition)`

Contrôle

- break ;
- continue ;