awk, le dessein des formes

- Niveau requis :
  débutant, avisé

THÉÉTÈTE

Tu m’en fais souvenir à point : il en reste une en effet.
La première était pour ainsi dire l’image de la pensée dans la parole ;
la seconde, qui vient d’être discutée, la marche vers le tout par la voie des éléments ;
mais la troisième, quelle est-elle, selon toi ?

SOCRATE

C’est juste la définition que la plupart des gens donneraient :
c’est de pouvoir fournir une marque qui distingue l’objet en question de tous les autres.

Platon, *Théétète*, 208c,5-10

Synopsis d'awk

utilisation simple : en ligne de commandes

```
awk 'Programme' Fichier-1 Fichier-2 ..... Fichier-n
```

ou avec un tube :

```
commande | awk 'Programme'
```

utilisation plus complexe : avec un script awk

```
awk -f 'Programme' [Fichier-1 Fichier-2 ..... Fichier-n]
```

Le fonctionnement discriminatoire d'awk

Quand la structure permet l'implicite

- Pré-requis : maîtriser les opérateurs relationnels

Détail du synopsis : "Programme"

On peut définir le “programme” de cette façon.

```
awk 'condition [action] | [condition] action' fichier1 fichier2 ...
```

Que signifie le fait que “condition” tout comme “action” sont facultatives au fonctionnement d'awk ? N'est-ce pas l'utilisation de la structure d'un fichier qui permet à awk de fonctionner tandis que “condition” et “action” font office l'une comme l'autre de *raison suffisante* ?

La structure d'un fichier

- AWK traite un fichier en se basant sur sa structure :

```
<ligne 1> : "Champ 1" "Champ 2" ..... "Champ n"
<ligne 2> : "Champ 1" "Champ 2" ..... "Champ n"
<ligne 3> : "Champ 1" "Champ 2" ..... "Champ n"
.......... .......... .......... ..........
<ligne p> : "Champ 1" "Champ 2" ..... "Champ n"
```

- Soit le fichier “**fichier-awk.txt**”

fichier-awk.txt

```
Constance   20/03/98    Tater        15    14    11    13
Rebecca     09/03/99    Tagou        12    16    13    14
Natanaël    01/08/00    Gouzi        16    11    10    12
Alexis      21/01/02    Gouzi        17    12    13    10
Hélène-Fleur    06/03/05    Verbois        16    18    15    16
Samuel      27/08/08    Verbois        14    16    14    13
```

- Soit le fichier “**fichier2-awk.txt**”

fichier2-awk.txt

```
abricot fraise pomme
tintin mario cendrillon ulysse shadok
27 fleurs
Hélène-Fleur
568 976 123 étoiles
```

Discrimination au moyen de variable prédéfinies

```
Condition {action1; action2; ...; action-n}
```

→ awk exécute l' `action` fichier par fichier, ligne à ligne, séquentiellement.

1. La condition détermine si l'action va être appliquée et comment.
2. L'action est un programme qui est appliqué par awk sur un fichier ou ce qui lui est passé par un pipe.
3. Pour instruire une action on utilise une fonction prédéfinie comme par exemple fonction print.

Par exemple :

```
awk '{print NR,$1,$2,$3,$4,$5,$6,$7}' fichier-awk.txt
```

```
1 Constance 20/03/98 Tater 15 14 11 13
2 Rebecca 09/03/99 Tagou 12 16 13 14
3 Natanaël 01/08/00 Gouzi 16 11 10 12
4 Alexis 21/01/02 Gouzi 17 12 13 10
5 Hélène-Fleur 06/03/05 Verbois 16 18 15 16
6 Samuel 27/08/08 Verbois 14 16 14 13
```

Mais pour comprendre comment fonctionne awk, il est avantageux, avant de s'attacher aux fonctions, de considérer que :

- Si la condition est absente, l'action est exécutée systématiquement (à chaque ligne).
- Si l'action est absente et qu'une condition présente est remplie, un résultat sera imprimée.

→ En fait, l'action standard est toujours l'affichage {print}, conditionné, et conditionné en vue de discriminer quelque chose par rapport à autre chose.

-Cela explique que awk affiche tous les champs quand le programme est constitué par une variable non conditionnée:

```
echo "coucou bonjour salut" | awk '$2'
```

```
coucou bonjour salut
```

> $2 semble alors être un équivalent de $0, ce n'est pas le cas, il en serait de même pour $1, $3 !

-Cela explique aussi qu'une variable conditionnée permette à awk d'afficher le résultat d'une discrimination.
Les exemples ci-dessous approfondiront cette question pour chaque variable prédéfinie.

Avec cet usage d'awk, on peut dire alors que l'action est un programme implicite. De ce point de vue, attention à la différence: `awk ' variable-conditionnée' fichier` ou `cmd | awk 'variable-conditionnée'`

Les champs

Lorsqu'awk lit une ligne dans le fichier qui lui est passé en argument, il la découpe en colonnes.
Une colonne est appelée “**un champ**” (ou “**F**ield” en Anglais).

Le nombre de champs (nombre de colonnes) est stocké dans la variable **NF**.
→ Dans le fichier “**fichier-awk.txt**”, le nombre de champs est constant, il correspond à 7 pour chaque ligne.
(NF == 7)
→ Dans le fichier “**fichier2-awk.txt**”, le nombre de champ varie selon les lignes.

Le nombre de champ(s) d'un fichier est une condition permettant la discrimination, si awk est utilisé sur deux fichiers.

- Exemple 1 :

```
awk 'NF <= 5' fichier2-awk.txt fichier-awk.txt
```

```
abricot fraise pomme
tintin mario cendrillon ulysse shadok
27 fleurs
Hélène-Fleur
568 976 123 étoiles
```

- Exemple 2 :

```
awk 'NF == 7' fichier2-awk.txt fichier-awk.txt
```

```
Constance   20/03/98    Tater        15    14    11    13
Rebecca     09/03/99    Tagou        12    16    13    14
Natanaël    01/08/00    Gouzi        16    11    10    12
Alexis      21/01/02    Gouzi        17    12    13    10
Hélène-Fleur    06/03/05    Verbois        16    18    15    16
Samuel      27/08/08    Verbois        14    16    14    13
```

> L'ordre des arguments n'importe pas dans cet exemple, il y a affichage du fichier dont le nombre de champs correspond à 7.

Les colonnes

Pour UN fichier donné, awk enregistre chaque champ dans une variable prédéfinie, nommée $n ou n est le numéro du champs.
Il y a donc autant de variable, ($1, $2, $3, ...) que de champs, (champs 1, champs 2, champs 3, ...).

→ Le fichier “**fichier-awk.txt**” instruit awk jusqu'à la variable $7
→ Le fichier “**fichier2-awk.txt**” instruit awk jusqu'à la variable $5

Tout fichier comportant plus d'UN champ peut à lui seul servir à la discrimination ;
avec un fichier de plus d'un champ awk pourra utiliser un champs comme condition pour discriminer les lignes du fichier.
C'est là encore une logique discriminatoire.

```
awk '$2 == "fraise"' fichier2-awk.txt
```

```
abricot fraise pomme
```

> Affichage de la ligne de “fichier2-awk.txt” où la chaîne “fraise” correspond au champ 2.

```
awk '$1 == "Hélène-Fleur"' fichier2-awk.txt fichier-awk.txt
```

```
Hélène-Fleur
Hélène-Fleur    06/03/05    Verbois        16    18    15    16
```

> Affichage des lignes comportant la chaîne “Hélène-Fleur” au champ 1 dans l'ordre des fichiers passés en argument.

Remarque sur le caractère $:

- Le caractère “$” est partie constituante du nom des variables, $1, $2, $3, etc.

Ce n'est pas ce caractère qui “appelle” la valeur de la variable comme c'est la cas pour le shell.

- $0 contient une ligne entière (celle relative à une condition).
  Utilisée sans condition $0 affiche toutes les lignes du fichier.
- Il n'en est pas de même pour $NF !

Par exemple, pour imprimer le nombre de champ de la ligne, le premier puis le dernier de chaque ligne prise en entrée

```
awk '"Hélène-Fleur"{print NF,$1,$NF}' fichier-awk.txt
```

```
7 Constance 13
7 Rebecca 14
7 Natanaël 12
7 Alexis 10
7 Hélène-Fleur 16
7 Samuel 13
```

> NF est le nom de la variable qui contient le nombre de champs.
> $NF signifie le dernier $ des champs NF.

Les enregistrements

Une ligne sera appelée “**un enregistrement**” (“**R**ecord” en Anglais).
Le nombre d'enregistrements est stocké dans la variable **NR**
→ Le fichier “**fichier-awk.txt**” comporte 6 lignes.
(NR = 6).
→ Le fichier “**fichier2-awk.txt**” comporte 5 lignes

(NR = 5).

```
awk 'NR > 5' fichier2-awk.txt fichier-awk.txt
```

```
Constance    20/03/98    Tater        15    14    11    13
Rebecca         09/03/99    Tagou        12    16    13    14
Natanaël    01/08/00    Gouzi        16    11    10    12
Alexis      21/01/02    Gouzi        17    12    13    10
Hélène-Fleur    06/03/05    Verbois        16    18    15    16
Samuel 27/08/08 Verbois        14    16    14    13
```

> Le fichier plus petit selon NR d'abord, sinon il affiche les lignes supérieures, c'est-à-dire celles à partir de l'enregistrement 5 pour chaque fichier !

Les autres variables prédéfinies

Celles en mémoire dès le lancement de la commande

Voici les variables internes du langage awk présentes en mémoire dès le lancement de la commande.

Variables	Significations	Valeurs par défaut
ARGC	Nombre d'arguments de la ligne de commande	-
ARGV	Tableau des arguments de la ligne de commande	-
FS	Séparateur de champs en entrée	“ “
NF	Nombre de champs de l'enregistrement courant	-
OFMT	Format de sortie des nombres	”%.6g”
OFS	Séparateur de champs pour la sortie	“ “
ORS	Séparateur d'enregistrement pour la sortie	“\n”
RLENGTH	Longueur de la chaîne trouvée	-
RS	Séparateur d'enregistrement en entrée	“\n”
RSTART	Positionnée par la fonction match :début de la chaîne trouvée	-
SUBSEP	Séparateur des éléments dans un tableau	“\034”

Celles initialisées lors du traitement d'une ligne

Les enregistrements sont traités successivement.
L'enregistrement est automatiquement découpé en champs et un certain nombre de variables internes awk sont alors initialisées.
Les voici :

$0	Valeur de l'enregistrement courant
NF	Nombre de champs de l'enregistrement courant
$1 $2 ... $NF	$1 : champ n°1 $2 champ n°2 $NF : valeur du dernier champ
NR	Indice de l'enregistrement courant (NR vaut 1 quand la première ligne est lu, et s'incrémente à chaque enregistrement.)

FNR	Indice de l'enregistrement courant relatif au fichier en cours de traitement
FILENAME	Nom du fichier en cours de traitement

"Fichier1" "Fichier2"

- Il peut s'agir du même fichier (répétition) :

```
awk '$3 == "Tagou"' fichier-awk.txt fichier2-awk.txt
```

```
Rebecca     09/03/99    Tagou        12    16    13    14
Rebecca     09/03/99    Tagou        12    16    13    14
```

- Ou de deux fichiers différents :

```
awk '"9"' fichier-awk.txt fichier2-awk.txt
```

```
Constance   20/03/98    Tater        15    14    11    13
Rebecca     09/03/99    Tagou        12    16    13    14
Natanaël    01/08/00    Gouzi        16    11    10    12
Alexis      21/01/02    Gouzi        17    12    13    10
Hélène-Fleur    06/03/05    Verbois        16    18    15    16
Samuel      27/08/08    Verbois        14    16    14    13
zouzou  06-34-58-69-01:/home/zouzou
gazou   06-36-96-58-02:/home/gazou
tibou   06-25-76-31-03:/home/tibou
```

Utilisation simple et notion de "programme" awk

Quand un programme explicite est ajouté

```
awk 'Programme' Fichier-1 Fichier-2 ..... Fichier-n
```

Le “programme” :

Jusqu'à présent on a vu que **Programme** pouvait être simplement constitué d'une condition et d'une variable prédéfinie :

```
awk '[condition]' Fichier-1 Fichier-2 ... Fichier-n
```

La plupart du temps awk est utilisé avec une instruction conditionnée ou non.
Le programme devient alors :

```
awk '[condition]{instruction}' Fichier-1 Fichier-2 ... Fichier-n
```

- Le “Programme” peut aussi être constitué par une ou plusieurs instructions.
- Quand il y en a plusieurs, chaque instruction permet une action particulière sur le ou des fichier(s), ou encore sur les données du flux d'entrée.

- Avec awk, on ne parle pas de commandes internes à awk.
  Une instruction peut être constituée par des fonctions prédéfinies, ou créées par l'utilisateur, par des variables prédéfinies ou crées par l'utilisateur.
- Chaque instruction du programme peut être conditionnée.
- Il est possible de conditionner les instructions au moyen d'un test “if-else” et maîtriser le déroulement d'une instruction au moyen des boucles “while” et “for”.

La création de fonctions et de variables, ainsi que l'utilisation des tests et des boucles, relèvent de l'utilisation avancée de awk qui, plus qu'une simple commande, est un langage de programmation.
Cela se fait dans un script awk.

Utilisations simples avec la fonction print

La fonction print permet d'imprimer sur la sortie standard.

Pour afficher le fichier "fichier-awk.txt"

Remarquer la syntaxe
La fonction est entourée de deux guillemets simples autour d'accolades.

```
awk '{print}' fichier-awk.txt
```

```
Constance   20/03/98    Tater        15    14    11    13
Rebecca     09/03/99    Tagou        12    16    13    14
Natanaël    01/08/00    Gouzi        16    11    10    12
Alexis      21/01/02    Gouzi        17    12    13    10
Hélène-Fleur    06/03/05    Verbois        16    18    15    16
Samuel      27/08/08    Verbois        14    16    14    13
```

Pour afficher le nombre de champ du fichier "fichier-awk.txt"

```
awk '{print NF}' fichier-awk.txt
```

```
7
7
7
7
7
```

```
7
```

> awk donne là le nombre de champ(s) par ligne du fichier

Pour afficher une colonne :

```
awk '{print $1}' fichier-awk.txt
```

```
Constance
Rebecca
Natanaël
Alexis
Hélène-Fleur
Samuel
```

Pour afficher une ligne précise entière

```
awk 'NR==6{print $0}' fichier-awk.txt
```

```
Samuel      27/08/08    Verbois        14    16    14    13
```

Pour afficher le champs d'une ligne précise

```
awk 'NR==6{print $1}' fichier-awk.txt
```

```
Samuel
```

Une instruction pour plusieurs variables

```
awk '{print NR,$1,$2}' fichier-awk.txt1
```

```
Constance 20/03/98
2 Rebecca 09/03/99
3 Natanaël 01/08/00
4 Alexis 21/01/02
5 Hélène-Fleur 06/03/05
6 Samuel 27/08/08
```

Plusieurs instructions successives s'appliquent toutes pour chaque ligne

```
awk '{print NR ; print $1 ; print $2}' fichier-awk.txt
```

```
1
Constance
```

```
20/03/98
2
Rebecca
09/03/99
3
Natanaël
01/08/00
4
Alexis
21/01/02
5
Hélène-Fleur
06/03/05
6
Samuel
27/08/08
```

Conclusion sur la syntaxe de l'action en ligne de commande.

Lorsque awk est utilisé en ligne de commandes,

- **programme** est toujours encadré par des guillemets simples et une paire d'accolades.

À l'intérieur des accolades :

- un espace sépare une instruction du programme et une variable prédéfinies ;
- un point virgule sépare plusieurs **instructions** ;
  Rappel : **'{print NR ; print $1 ; print $2}'**
- une virgule (sans espace avant et après) sépare une suite de variables prédéfinies.
  Rappel : **'{print NR,$1,$2,$3}'**

Attention à la localisation d'une variable prédéfinie par rapport aux instructions.
Voir la remarque ci-dessous.

Remarque sur la position d'une variable prédéfinie et une instruction

- imprimer “Bonjour” et “coucou” pour chaque ligne du fichier et numéroter

```
awk '{print NR ; print "Bonjour" ; print "coucou"}' fichier-awk.txt
```

```
1
Bonjour
coucou
2
```

```
Bonjour
coucou
3
Bonjour
coucou
4
Bonjour
coucou
5
Bonjour
coucou
6
Bonjour
coucou
```

- imprimer “Bonjour” et “coucou” pour chaque ligne du fichier et imprimer la numérotation au niveau de “coucou”

```
awk '{print "Bonjour" ; print "coucou" NR}' fichier-awk.txt
```

```
Bonjour
coucou1
Bonjour
coucou2
Bonjour
coucou3
Bonjour
coucou4
Bonjour
coucou5
Bonjour
coucou6
```

- imprimer “Bonjour” et “coucou” pour chaque ligne du fichier et imprimer la numérotation au niveau de “Bonjour”

```
awk '{print "Bonjour" NR ; print "coucou"}' fichier-awk.txt
```

```
Bonjour1
coucou
Bonjour2
coucou
Bonjour3
coucou
Bonjour4
coucou
Bonjour5
coucou
Bonjour6
coucou
```

Détail sur l'utilisation des séparateurs

La frontière entre deux champs est spécifiée par un caractère spécial appelé : “le séparateur”.
Le séparateur est défini dans une variable globale qui s’appelle **FS**.
(Mnémonique : FS signifie “Field Separator” en anglais.)
Sa valeur par défaut est l'espace ou la tabulation[1)].

Mais le séparateur de champs peut être n'importe quel caractère qu'on choisi.
Pour le spécifier, on utilise en ligne de commandes **l'option -F** :

```
awk -F'séparateur'
```

OU on le précise dans un script en modifiant la variable **FS** :

```
BEGIN {FS = "séparateur"}
```

→ Dans “fichier-awk.txt” le séparateur est tabulation (FS = “\t“).
→ Dans “fichier2-awk.txt” le séparateur est espace (FS = espace).

Spécifier un séparateur : option -F

- Soit “fichier2-awk.txt” dont les champs peuvent être différents séparateurs.

```
zouzou  06-34-58-69-01:/home/zouzou
gazou   06-36-96-58-02:/home/gazou
tibou   06-25-76-31-03:/home/tibou
```

```
awk -F':' '{print $2}' fichier2-awk.txt
```

```
/home/zouzou
/home/gazou
/home/tibou
```

```
awk -F'/' '{print $2}' fichier2-awk.txt
```

```
home
home
home
```

```
awk -F'-' '{print $2}' fichier2-awk.txt
```

```
34
36
25
```

Spécifier une classe de séparateurs

On peut aussi spécifier le séparateur en assignant la variable FS.
Il est possible alors de spécifier une classe de séparateurs.

```
awk 'BEGIN {FS = "(-|/)" }; {print $5}' fichier2-awk.txt
```

> En fonction du séparateur /, il n'y a pas de 5ième champ, donc (- ou /), pour le 5ième champs :

```
01:
02:
03:
```

“BEGIN” est un “modèle” (“patterns”). Voir plus bas la notion de “blocs”

Modifier le séparateur de sortie : variable OFS

```
awk '{ OFS="," ; print $1,$4 }' fichier-awk.txt
```

```
Constance,15
Rebecca,12
Natanaël,16
Alexis,17
Hélène-Fleur,16
Samuel,14
```

- Cela ne fonctionne qu'avec la virgule.

À remarquer :

1. les doubles guillemets autour de la virgule.
2. reprise de la virgule entre les champs sans espace.
3. le point virgule (avec espaces avant et après) pour séparer les instructions.

- Mais pour améliorer la sortie, on peut insérer n'importe quel caractère entre les variables comme ceci :

```
awk '{ print $1 ": moyenne math => " $4 }' fichier-awk.txt
```

```
Constance: moyenne math => 15
Rebecca: moyenne math => 12
Natanaël: moyenne math => 16
Alexis: moyenne math => 17
Hélène-Fleur: moyenne math => 16
Samuel: moyenne math => 14
```

Déterminer l'instruction par une condition

Il a été vu :

- qu'un programme awk se présente toujours ainsi :

```
condition {instruction}
condition {instruction}
..........
```

- qu'une **condition** porte exclusivement sur une ligne du fichier d'entrée s'il en est précisé un, ou sur l'entrée standard (stdin), si aucun fichier n'est indiqué.
- qu'une action ne s'exécutera que si la condition est validée.
- à ces sujets, quelques types de condition.

Mais il y a différentes types de conditions.

- On désigne ces différents types de condition de "**modèles de condition**".
- Il y a quatre sorte de modèles de condition

BEGIN	BEGIN	avant que toute entrée soit lue
END	END	après que toute entrée soit lue
expression relationnelle	NF>5	matche les lignes où le nombre de champ est supérieur à 5
expression régulière (ER)	/$1 ~ /[:aplha:]*/	matche les lignes dont la chaîne du champ 1 correspond à minuscule ou majuscule, zéro ou plusieurs fois
composé d'ER	NF>5 && /l/	match les lignes contenant plus de 5 champs et contenant le caractère l
intervalle	NR==5,NR==10	match les lignes de 5 à 10

- Tous ces modèles, bien que distingués les uns des autres, ont une *forme* en commun : qu'elle est-elle ; qu'est-ce qui fait que chacun "modèle de condition" est condition ?

Les différents genres de condition

Une chaîne comme condition

- afficher la date de naissance puis le nom d'élève à condition que la chaîne "Hélène-Fleur" existe :

```
awk '"Hélène-Fleur" {print $2,$1}' fichier-awk.txt
```

```
20/03/98 Constance
```

```
09/03/99 Rebecca
01/08/00 Natanaël
21/01/02 Alexis
06/03/05 Hélène-Fleur
27/08/08 Samuel
```

Une expression relationnelle comme condition

Symboles binaires des expressions relationnelles (ou de comparaison):

```
<  <=  ==  !=  >  >=
```

Pour la correspondance avec ER voir "Les modèles".

- Pour afficher la deuxième colonne à condition que la colonne 1 corresponde à la chaîne “Constance” :

```
awk '$1 == "Constance" {print "date de naissance de Constance:" ; print $2}'
fichier-awk.txt
```

```
date de naissance de Constance:
20/03/98
```

Une intervalle comme condition

- affichage de la ligne dont la colonne 7 comporte 14 jusqu'à la ligne dont la quatrième colonne comporte 17 :

```
awk '$7==14, $4==17 {print $0}' fichier-awk.txt
```

```
Rebecca     09/03/99    Tagou        12    16    13    14
Natanaël    01/08/00    Gouzi        16    11    10    12
Alexis      21/01/02    Gouzi        17    12    13    10
```

- affichage de la ligne dont le champ 1 comporte “Alexis” jusqu'à la ligne n°7

```
awk '$1=="Alexis", NR==7 {print $0}' fichier-awk.txt
```

```
Alexis      21/01/02    Gouzi        17    12    13    10
Hélène-Fleur    06/03/05    Verbois        16    18    15    16
Samuel      27/08/08    Verbois        14    16    14    13
```

Une ER comme condition

Voir les symboles des expressions régulières.

- Pour afficher le ou les enfants nés en mars :

```
awk '/[0-9]\/03\/[0-9]/{print $1}' fichier-awk.txt
```

```
Constance
Rebecca
Hélène-Fleur
```

Une ER en correspondance comme condition

```
awk '$1 ~ /[[:alpha:]]*/{print NR,$1,$2,$3,$4,$5,$6,$7}' fichier-awk.txt
```

> Ici la condition est : **'$1 ~ /[:aplha:]*/** .
> Elle est constituée par l'équivalence (**~**) entre le champ 1 ($1) et une expression rationnelles : la classe **[[:alpha:]]** (minuscule ou majuscule), zéro ou plusieurs fois (*****).
> *(Les expressions rationnelles sont entourées de slashs : /ER/)*

```
1 Constance 20/03/98 Tater 15 14 11 13
2 Rebecca 09/03/99 Tagou 12 16 13 14
3 Natanaël 01/08/00 Gouzi 16 11 10 12
4 Alexis 21/01/02 Gouzi 17 12 13 10
5 Hélène-Fleur 06/03/05 Verbois 16 18 15 16
6 Samuel 27/08/08 Verbois 14 16 14 13
```

Le complément d'une correspondance avec ER comme condition

On appelle complément, ce qui correspond à ce qui est exclu d'une correspondance.

```
awk '$4 !~ /12/{print NR,$1,$2,$3,$4,$5,$6,$7}' fichier-awk.txt
```

```
1 Constance 20/03/98 Tater 15 14 11 13
3 Natanaël 01/08/00 Gouzi 16 11 10 12
4 Alexis 21/01/02 Gouzi 17 12 13 10
5 Hélène-Fleur 06/03/05 Verbois 16 18 15 16
6 Samuel 27/08/08 Verbois 14 16 14 13
```

La correspondance et la non-correspondance peuvent utiliser le | (non).

```
awk '$4 ~/16|17|18|19|20/ {print NR,$1,$4,$5,$6,$7}' fichier-awk.txt
```

> Élèves bon en math, ceux ayant 16 ou 17 ou 18 ou 19 ou 20, bref plus de 16

Composés de ER comme condition

Rappel :

```
|| (ou) && (et) | (non)
```

Pour utiliser les composés de ER, il faut bien tenir compte que c'est awk qui les utilise comme condition pour l'affichage de ligne(s) ou de champ(s).

- le “ou” : **/ER1/ || /ER2/** sélectionne toute(s) ligne(s) replissant l'une ou l'autre condition :

```
awk '/14/ || /18/{print NR,$1,$4,$5,$6,$7}' fichier-awk.txt
```

> Sélection des lignes ayant soit 14 soit 18 parmi ces champs

```
1 Constance 15 14 11 13
2 Rebecca 12 16 13 14
5 Hélène-Fleur 16 18 15 16
6 Samuel 14 16 14 13
```

- le “et” : **/ER1/ && /ER2** sélectionne les lignes ayant les deux conditions :

```
awk '/15/ && /16/{print NR,$1,$4,$5,$6,$7}' fichier-awk.txt
```

> Sélectionne les lignes ayant un 15 et un 16 parmi ces champs

```
5 Hélène-Fleur 16
```

- le | “non” fait **partie de ER1 ou/et de ER2** :

```
awk '/16|18/ && /13|15/ {print NR,$1,$4,$5,$6,$7}' fichier-awk.txt
```

Sélectionne (si elles existent) les lignes ayant un 16 et un 13 ou un 16 et 15, ou un 18 et un 13 ou un 18 et un 15

```
2 Rebecca 12 16 13 14
5 Hélène-Fleur 16 18 15 16
6 Samuel 14 16 14 13
```

À remarquer que :

```
/16|18/ || /13|15/
```

et équivalent à :

```
/16|18|13|15/
```

Un calcul comme condition

Voir "Les opérateurs arithmétiques"

- sélectionner les élèves dont la moyenne générale est inférieure ou égale à 14 :

```
awk '($4+$5+$6+$7)/4 >= 14 {print $1 ; print "peut faire mieux."}' fichier-awk.txt
```

```
Hélène-Fleur
peut faire mieux.
Samuel
peut faire mieux.
```

Penser en terme de condition

Une fois que l'on maîtrise chacun des types de condition ci-dessus, leur signification, leur syntaxe, il faut encore tenir compte de ce que dit à awk la forme ou le genre de condition.

- Une chaîne de caractère dit : “exécute le programme **partout** si la chaîne existe”

```
awk '"13"{print NR,$1,$2,$3,$4,$5,$6,$7}' fichier-awk.txt
```

```
1 Constance 20/03/98 Tater 15 14 11 13
2 Rebecca 09/03/99 Tagou 12 16 13 14
3 Natanaël 01/08/00 Gouzi 16 11 10 12
4 Alexis 21/01/02 Gouzi 17 12 13 10
5 Hélène-Fleur 06/03/05 Verbois 16 18 15 16
6 Samuel 27/08/08 Verbois 14 16 14 13
```

- Une ER dit : “exécute le programme **là où** la ER correspond”

```
awk '/13/{print NR,$1,$2,$3,$4,$5,$6,$7}' fichier-awk.txt
```

```
1 Constance 20/03/98 Tater 15 14 11 13
2 Rebecca 09/03/99 Tagou 12 16 13 14
4 Alexis 21/01/02 Gouzi 17 12 13 10
6 Samuel 27/08/08 Verbois 14 16 14 13
```

- Excepté pour la condition de type “chaîne”[2], toutes les autres conditions explicitées jusqu'ici, déterminent un “lieu” précis pour lequel la condition doit être remplie.
  Il n'y aurait par exemple aucun sens à ce que awk exécute un programme à condition d'un calcul juste en lui-même.

Autrement dit, la *forme* d'une condition est de déterminer l'action en fonction d'une localisation, et cela *a priori* de l'explicitation de la condition.

- “BEGIN” et “END” font partie des “modèles” ; comme les autres “conditions”, ils

contrôlent l'exécution des actions.
Mais contrairement aux autres modèles, ils ne déterminent pas l'exécution du programme en fonction d'une condition qui, par correspondance, détermine l'action relativement à une partie du fichier.
Ils sont “condition” dans le sens où ils “modélisent” l'action. Cette modalité impose que l'action n'ait lieu qu'une seule fois, et que le résultat de cette action s'affiche en lieu particulier par rapport au résultat de l'ensemble du programme.

- BEGIN et END ne se composent pas avec les autres modèles.
- Ils ne sont pas obligatoires.
- Ils s'utilise en général dans des scripts.
  Voir notion de programme awk et utilisation de script rudimentaire.

BEGIN et END

Notion de blocs

Jusque là, seul des programmes simples (une seule paire d'accolades composée d'une ou plusieurs instructions) ont été abordés.
Il est possible d'enchaîner plusieurs séries de programme comme ceci :

```
Condition {
    Action
  }
...
Condition {
    Action
}
```

À ce programme qui est la partie centrale, il est possible d'ajouter un début (BEGIN) et une fin (END) de programme.

```
BEGIN {
    instructions
}
Condition {
    Action
  }
...
Condition {
    Action
}
END {
    instructions
}
```

- “Le programme central” :

Awk rappelle les “actions” du programme central autant de fois qu'il y a d'enregistrement(s).
Par exemple, appliqué à un fichier de 4 lignes, le programme va être appelé quatre fois[3].
À chaque passage, c'est un nouvel enregistrement (une nouvelle ligne) qui est traité.

- “BEGIN” :

Le bloc “BEGIN” est exécuté une fois au début, avant le traitement des données.
Il peut être constitué de plusieurs instructions.

- “END” :

Le bloc “END” est exécuté une fois à la fin, après le traitement des données.
Il peut aussi être constitué de plusieurs instructions.

Le bloc “END” permet aussi d'afficher des résultats du programme central.

BEGIN

- Il peut être utiliser uniquement pour définir le séparateur.

Par exemple dans ce cas, BEGIN est obligatoire pour que le séparateur spécifié soit pris en compte avant la lecture du fichier.

```
awk 'BEGIN {FS = "(-|/)" }; {print $5}' fichier2-awk.txt
```

```
01:
02:
03:
```

- Sans le BEGIN :

```
awk '{FS = "(-|/)" }; {print $5}' fichier2-awk.txt
```

```


02:
03:
```

- Si plusieurs BEGIN apparaissent, ils sont exécutés dans l'ordre où ils apparaissent.

```
BEGIN {
FS = "(-|/)"
}
BEGIN {
```

```
print "deuxième BEGIN exécuté aussi avant le premier enregistrement"
}
{print $5}
```

```
awk -f begin-end1.awk fichier2-awk.txt
```

```
deuxième BEGIN exécuté aussi avant le premier enregistrement
01:
02:
03:
```

Exemple de blocs de début et de fin

- Soit le script “begin-end.awk”

```
#!/usr/bin/awk -f
BEGIN{
print "Élèves très bons en math :"
print ""
}
$4 > 15 {print $1}
END{
print "Il y a " FNR " élèves bons en maths" ; print "Bon résultats"
}
```

> Le programme central est ici la ligne : `$4 > 15 {print $1}`,
> encadré par instructions de “BEGIN” et de “END”.

```
./begin-end.awk fichier-awk.txt
```

```
Élèves très bons en math :

Natanaël
Alexis
Hélène-Fleur
Il y a 6 élèves bons en maths
Bons résultats
```

Notion de "programme" awk et utilisation de scripts rudimentaires

```
awk -f Programme [Fichier-1 Fichier-2 ..... Fichier-n]
```

- **Programme** est dans ce cas le nom d’un fichier contenant la suite d’instructions. Si aucun fichier n’est précisé, le fichier d’entrée sera stdin.

- -f est l'option qui permet d'utiliser un script

Éléments de syntaxe

Les instructions sont placées dans un "fichier script"

“Programme” avec awk en ligne de commandes	“Programme” dans un fichier script

`awk -F ':' '{ print $1 " est " $5 }' /etc/passwd`	Soit le fichier “script.awk”:
	`{ FS = ":"}`
	`{ print $1 " est " $5 }`

Notez que dans le script, on doit saisir **FS = ”:“** et non **FS = ':'** .

Exemple de script

- Soit le script “script-awk” :

```
{print "ÉLÈVE:";
print $1 ;
print "année 2013/2014";
}
```

```
awk -f script-awk fichier-awk.txt
```

```
ÉLÈVE:
Constance
année 2013/2014
ÉLÈVE:
Rebecca
année 2013/2014
ÉLÈVE:
Natanaël
année 2013/2014
ÉLÈVE:
Alexis
année 2013/2014
ÉLÈVE:
Hélène-Fleur
année 2013/2014
ÉLÈVE:
Samuel
année 2013/2014
```

> L'action est appliquée pour chaque enregistrement (chaque ligne du fichier).

Quelques règles de syntaxe :

-Il est impératif que l'accolade de début de section soit placée juste après le mot clef BEGIN.
Pour la lisibilité, un espace peut les séparer.

-Les chaînes de caractères sont à spécifier entre double guillemets.

-Si deux instructions se suivent sur la même ligne, il faut alors les séparer par un point-virgule:

```
print "Nombre d'élèves : " NR ; print "année 2013/2014" ; print
"bons résultats"
```

-À la place d'un point-virgule, on peut aussi les séparer par un retour à la ligne:

```
print "Nombre d'élèves : " NR ; print "année 2013/2014"
print "bons résultats"
```

-Les variables prédéfinies n'ont pas à être séparées des actions par un point-virgule[4], un espace avant et après et suffisant.

Appel du script en ligne de commande

On applique l'action de ses instructions (placées dans un fichier) ainsi :

```
awk -f script.awk /etc/passwd
```

Script awk exécutable

Pour rendre un script awk directement exécutable :

- on insère à la première ligne du fichier script le sha-bang adéquat !

Soit le script “script.awk”

```
#!/usr/bin/awk -f
BEGIN{
print "ÉLÈVES:"
}
{print $1}
END{
print "Nombre d'élèves : " NR ; print "année 2013/2014"
print "bons résultats"
}
```

Rappelons-nous !

```
whereis awk
```

```
awk: /usr/bin/awk
```

- On rend le script exécutable :

```
chmod a+x script.awk
```

- On exécute “script.awk” comme n'importe quel script :

```
./script.awk fichier-awk.txt
```

```
ÉLÈVES:
Constance
Rebecca
Natanaël
Alexis
Hélène-Fleur
Samuel
Nombre d'élèves :  6
année 2013/2014
```

> Il apparaît des blocs de début (BEGIN) et de fin (END) où les actions ne sont pas appliquées par enregistrement.
> Voir "Notion de Blocs"

La fonction printf

On utilise printf pour un contrôle plus précis sur le format de sortie que ce qui est normalement fourni par print.

La fonction printf peut être utilisée pour spécifier la largeur à utiliser pour chaque élément, ainsi que les différents choix de mise en forme (par exemple la base à utiliser, ou le choix de l'exposant à imprimer...).
Ceci est fait en fournissant **une chaîne**, appelée **la chaîne de format**, qui contrôle comment et où imprimer les valeurs.
Une chaîne de format est constituée **d'un contrôleur de format**ou **string**, précédé d'un **%** et éventuellement **d'un modificateur de format**.

À savoir :

-printf crée n'importe quel type de format.

-Le format est une string (chaîne de format) contenant des %

-Chaque % est associé à une valeur de variable: il y a autant de % que de valeur.

-Le printf déclaration n'ajoute pas automatiquement un saut de ligne à sa sortie. Il émet seulement ce que la chaîne de format spécifie. Donc, si un saut de ligne est nécessaire, il faut l'inclure un dans la chaîne de format.

-Les séparateurs de sortie des variables OFS et ORS n'ont aucun effet sur printf déclarations.

Syntaxe

Un simple printf déclaration ressemble à ceci:

```
{printf "format1 format2 formatN", valeur1, valeur2, valeurN}
```

La liste complète des arguments peut éventuellement être mise entre parenthèses :

```
{printf ("format1 format2 formatN", valeur1, valeur2, valeurN)}
```

Exemple expliqué

Titi, Toto, et Lili aiment les sucreries.
Voici leur consommation par semaine pour :

```
awk '{print $0}' printf-txt
```

```
consommateurs   paquet_de_100g_de_bonbon    plaque_de_100g_de_chocolat

Toto        10                1
Titi        3                2
Lili        1                4
```

- Imprimer le nombre total de gramme de sucreries ingurgitées chaque jour pour chacun des consommateurs :

```
awk 'NR==3,NR==5 {printf (" %s\
 bonbon %d plaque de chocolat %d\
 consommation en gramme par jour %2f\n",\
 $1,$2,$3,($2+$3)/0.07)}' printf-txt
```

```
 Toto bonbon 10 plaque de chocolat 1 consommation en gramme par jour
157.142857
 Titi bonbon 3 plaque de chocolat 2 consommation en gramme par jour
71.428571
 Lili bonbon 1 plaque de chocolat 4 consommation en gramme par jour
71.428571
```

- **NR==3,NR==5** : la condition est un intervalle (de la ligne 3 à 5)
- **%s** : le contrôleur de la chaîne de format (“toute chaîne de caractères”)
- **%d** : le contrôleur de la chaîne de format (“nombre décimal, partie entière”)
- **%2f** : le contrôleur de la chaîne de format ('Nombre réel sous la forme [-]ddd.dddddd”) et le modificateur 2
- **\n** : retour à la ligne (qui n'est pas inclut dans les déclarations de type printf.

- explications :

```
'{printf (" %s bonbon %d plaque de chocolat %d consommation en gramme par jour %2f\n", $1,$2,$3,($2+$3)/0.07)}'
```

On place à gauche, la liste des chaînes de format, elle est encadrée de guillemets double, et entre les chaînes de format on peut écrire des chaînes de caractères.
Après le guillemet fermant la liste de chaînes de format, on place une virgule et un espace

À droite (après virgule et espace), on place la liste des variables dans le même ordre que celui des chaîne de format qui leur correspond.
Chaque chaîne de format est associée à la valeur qui lui correspond en fonction de leur ordre d'apparition respectif :
Par exemple pour Toto :

- %s correspond à $1 dont la valeur est Toto
- %d correspond à $2 dont la valeur est 10 (quantité en g de bonbon mangés par Toto)
- %d correspond à $3 dont la valeur est 1 (quantité plaque choco mangée par Toto)
- %2f correspond au format du calcul ($2+$3)/0.07 (pour Toto (10+1)*100/7jours ou (10+1)/(0.07)

printf dans un script awk rudimentaire

Titi, toto et Lili sont des buveurs de vin.

```
awk '{print $0}' printf2.txt
```

```
consommateurs   bouteilles/semaine    vin         prix/bouteille

Titi        1             Cheval_blanc    500
Toto        7             Beaujolais    6
Lili        2             Pauillac    20
```

- Soit le script “printf.awk” :

```
cat printf.awk
```

```
#!/usr/bin/awk -f
NR==3,NR==5 {printf (" %s dépenses en euros par semaine %.f\n", $1,$2*$4)}
```

> On peut aussi choisir le contrôleur **%d** à la place de **%.f**.
> **%.f** : utilise le modificateur point (.) qui retire les six chiffres après la virgule du nombre réel.

```
chmod u+x printf.awk
```

- affichage des dépenses de vin par semaine pour chacun

```
./printf.awk printf2.txt
```

```
 Titi dépenses en euros par semaine 500
 Toto dépenses en euros par semaine 42
 Lili dépenses en euros par semaine 40
```

Synthèse

Récapitulatif des contrôleurs et modificateurs de formats

1)
espaces et tabulations contiguës sont considérées comme un séparateur unique.
2)
prenant pour lieu tout le fichier
3)
Il y a un ensemble de variables qui permettent d'accéder à l'enregistrement (voir plus bas).
4)
Placer une virgule entre action et variable prédéfinie ne constitue pas une erreur de code; mais cela le rend moins lisible, et c'est à éviter.