

# Systemes d'exploitation

## Chapitre 7-7

Filtre programmable `nawk` ( 1 )

# Filtres programmable `nawk ( 1 )`

---

- Il s'agit d'un programme UNIX capable d'interpréter un programme utilisateur.
- Le programme doit être écrit en utilisant les instructions légales et selon le format de `nawk ( 1 )`.
- Le concept de programmation est appelé « piloté par données » (*data-driven*).
- On peut utiliser le `nawk ( 1 )` pour:
  - Valider des concepts de programmation
  - Automatiser les tâches de gestion
  - Évaluer rapidement les algorithmes
  - etc.

# Invocation de `nawk` ( 1 )

- Le synopsis de `nawk` ( 1 ) :

`nawk [-F c] [-f prog | 'prog' ] [-v var=valeur...] [fich1 fich2 ...]`

Option et paramètre	Signification
- Fc	Caractère c est le séparateur de champ.
- f pr og	pr og est le nom du fichier contenant le programme n a wk .
' pr ogr amme '	programme <code>nawk</code> donné directement entre apostrophes.
- v var =val eur ...	Initialisation des variables avant l'exécution du programme.
f i chi er 1 ...	Fichier contenant les données à traiter.

Chaque ligne d'entrée est séparée en champs \$0, \$1, \$2, \$3, etc.  
Ces champs n'ont rien à voir avec les \$1, \$2, ... de Bourne shell.

On peut spécifier un programme `nawk(1)` dans un fichier par l'option -f ou l'écrire directement entre apostrophes.

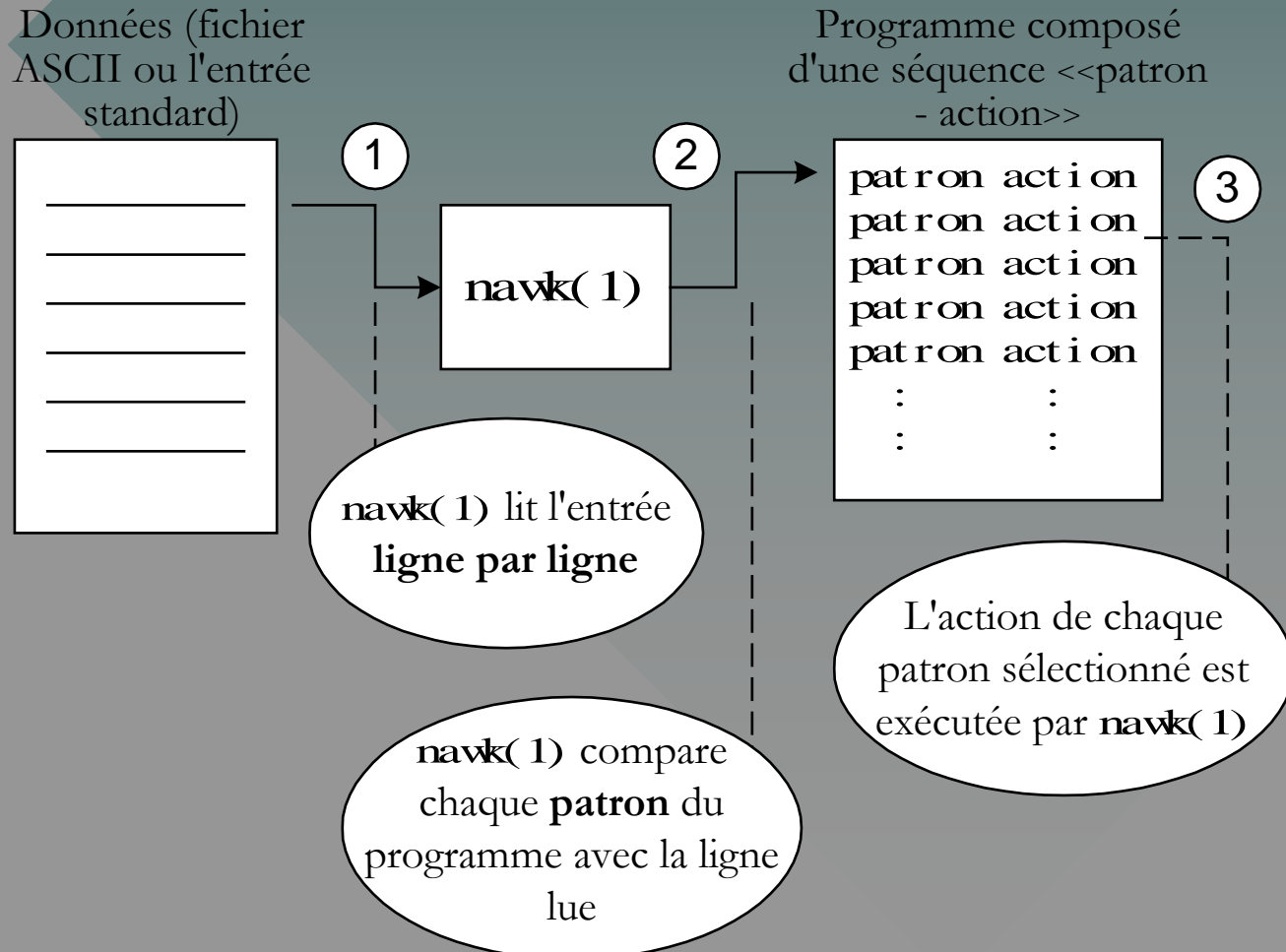
# Invocation de `nawk (1)`

---

- Les données à traiter sont contenues dans les fichiers `fich1`, `fich2`, .... ou acheminées via l'entrée standard.
- Le programme `nawk (1)` proprement dit est une séquence de « patron - action ».
- On peut passer des paramètres à un programme `nawk (1)` par l'option `-v`.
  - Cette option est utile lorsque `nawk (1)` est utilisée à l'intérieur d'un fichier de commande (Bourne shell par exemple).
  - Par exemple, on peut passer la valeur des variables d'un programme Bourne shell à des variables d'un programme `nawk (1)`.

# Filtres programmable `nawk(1)`

- Son principe de fonctionnement:



# Programmation `nawk` ( 1 )

- Voici un premier exemple. Le programme `nawk` ( 1 ) est spécifié directement entre apostrophes.

```
centi 27> nawk 'BEGIN{ print "Premier programme nawk" } \
{print $0} \
END{ print "Fin du programme nawk" }' passwd.dat
Premier programme nawk
si cn1502: hh60XAVSrRPd6: 30573: 112: M t h i e u Si cot t e: / export / hone / exa/ enai l / si cn1502
: / bi n/ csh
l aph2609: 3l 29au7qzUhD3: 30387: 112: H u g o L a p o r t e: / export / hone / exa/ enai l / l aph2609: / b
i n/ csh
j ane1651: opFd1xua3cZI2: 30861: 112: E l s a J a n a u d: / export / hone / exa/ enai l / j ane1651: / bi
n/ csh
: : :
: : :
r obf0208: 1GubPk1NPqDQ 30545: 112: F r a n c o i s R o b i c h a u d: / export / hone / exa/ enai l / r obf0
208: / bi n/ csh
r obcad49: *: 11450: 114: C o m p t e R o b c a d, n a i 95, M r i o T e t r e a u l t : / usr/ peop l e/ exa/ r obc a d
/ r obcad49: / bi n/ csh
Fi n du premier programme nawk
```

# Programmation `nawk` ( 1 )

- Nous pouvons également spécifier le programme `nawk` ( 1 ) dans un fichier.

```
nawk -f mon_prog passwd.dat
```

```
# Programme mon_prog  
BEGIN { print "Premier programme nawk" }  
        {print $0} # affiche la ligne lue  
                # (il n'y a pas patron)  
END { print "Fin du programme nawk" }
```

- L'option `-f` est obligatoire.
- Il est recommandé d'utiliser un fichier source surtout lorsqu'il comporte beaucoup de lignes.
- L'utilisation d'un fichier source facilite la compréhension surtout pour les novices.

# Programmation `nawk` ( 1 )

- ◆ Nous pouvons utiliser `nawk(1)` dans un fichier de commande.

```
1  #!/bin/sh
2  # nawkdemo : montrer l'utilisation de nawk dans un programme Bourne shell
3  #
4
5  # Fonction Bourne shell pour afficher le contenu
6  # d'un fichier passé en paramètre
7  Affiche () {
8
9  nawk '
10 BEGIN { print "Premier programme nawk" }
11       {print $0}
12 END   { print "Fin programme nawk" }
13 ' $1
14
15 }
16
17 # programme principal
18 if [ $# -ne 1 ]
19 then
20     echo "Donner le nom de fichier du fichier à traiter"
21 else
22     Affiche "$1"
23 fi
```



# Structure d'un programme (1)

- ♦ La structure d'un programme `nawk(1)` est une séquence de « patron - action » :

```

nawk '
patron {action}
patron {action}
patron {action}
:
:
'
```

①

- ♦ Le programme est toujours entouré de ' ' s'il est spécifié dans la ligne de commande.
- ♦ Il doit exister au moins un caractère blanc (Espace ou Tab) entre le patron et son action.
- ♦ Les patrons sont évalués dans l'ordre spécifié.

## Structure d'un programme (2)

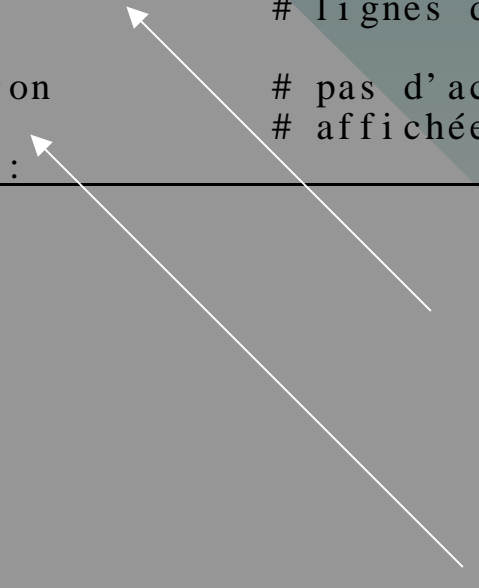
---

- Les déclarations « patron - action » peuvent ne pas contenir de patron OU d'action.
  - Une déclaration sans patron → l'action correspondante est toujours exécutée peu importe la ligne d'entrée.
  - Une déclaration sans action → la ligne d'entrée est affichée à la sortie standard (équivalent à `print $0`) si elle satisfait le critère spécifié par le patron.
- On peut considérer un patron comme un filtre d'où l'appellation « filtre programmable »

# Structure d'un programme (3)

## ◆ Un exemple illustratif:

```
: : :  
patron {action}  
      {action} # pas de patron donc action est toujours enclenchée pour les  
                # lignes d'entrée lues  
  
patron      # pas d'action donc les lignes correspondant au patron seront  
            # affichées  
: : :
```

The diagram consists of two white arrows. One arrow originates from the text 'Une déclaration sans patron (sans filtre)' and points to the line 'patron {action}' in the code block. The other arrow originates from the text 'Une déclaration sans action (activation de l'action par défaut qui est le print \$0)' and points to the line 'patron' in the code block.

Une déclaration sans patron (sans filtre)

Une déclaration sans action (activation de l'action par défaut qui est le print \$0)

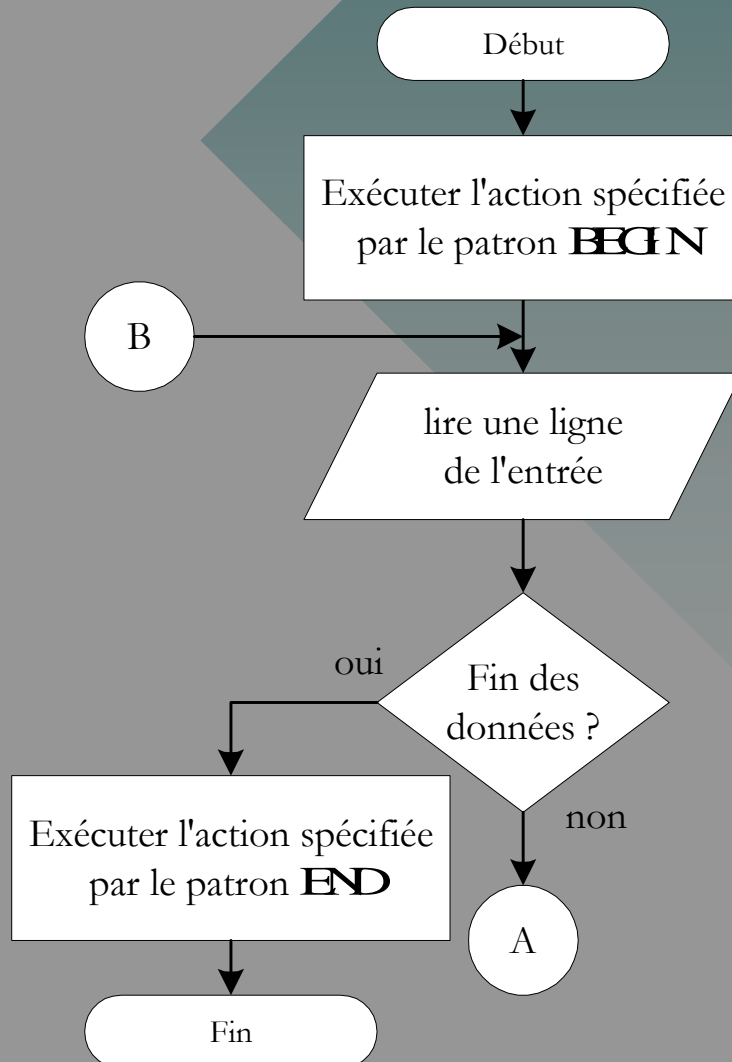
## Structure d'un programme (4)

- ♦ La première accolade est toujours placée sur la même ligne que le patron.
- ♦ On peut utiliser le point-virgule (;) pour séparer les instructions d'une même ligne.

```
: : :  
patron {instr1; instr2; instr3} # instructions séparées par des ;  
patron {          # première accolade doit être située sur la même ligne que le patron  
  instr1  
  instr2          # instructions peuvent s'étendre sur plus d'une ligne  
  instr3}  
: : :
```

Ces deux styles d'écriture sont équivalents.

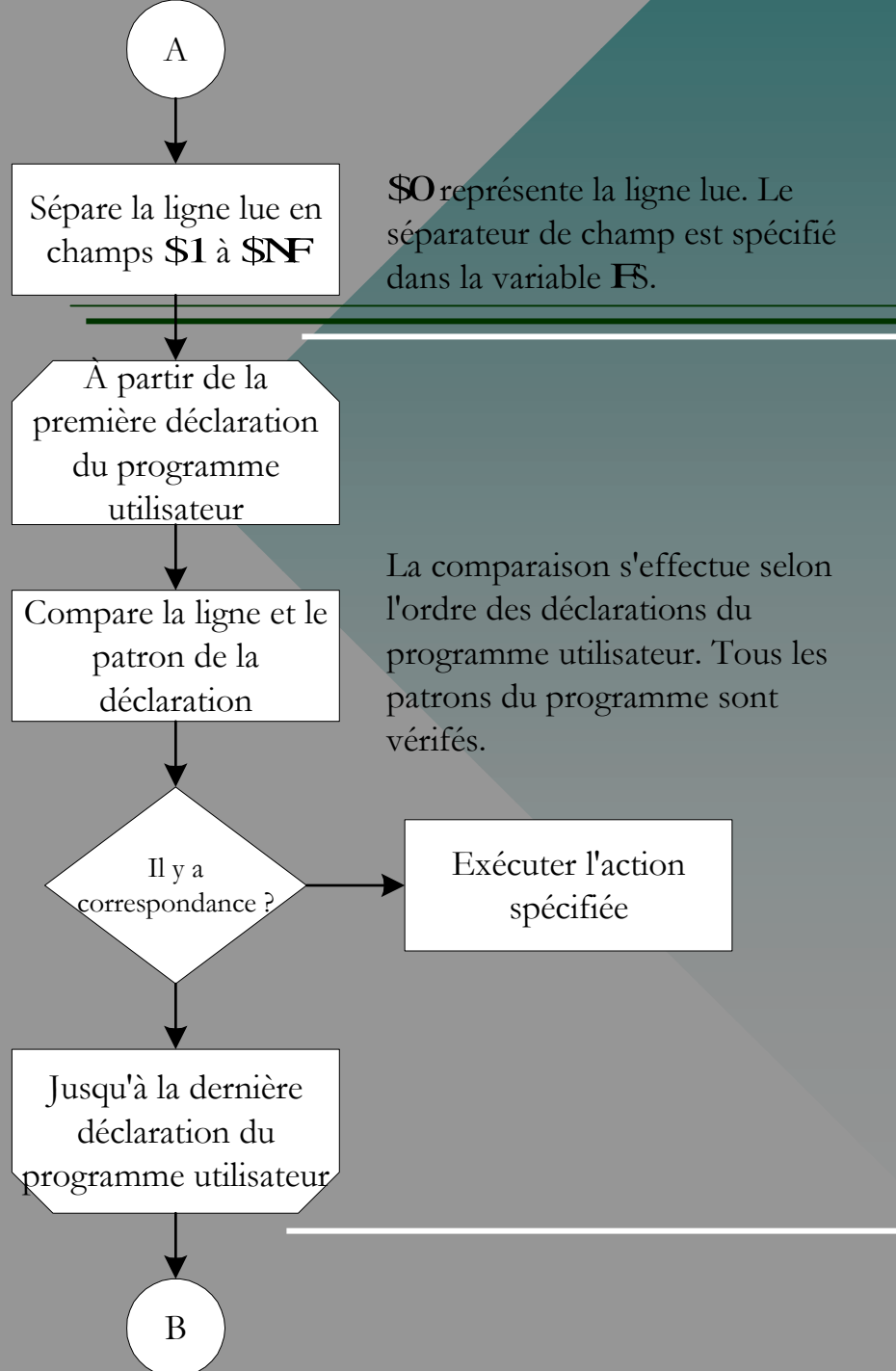
# Activités de traitement (1)



Voici les activités réalisées par `nawk (1)` lors de l'exécution d'un programme utilisateur.

Données peuvent provenir d'un fichier ou par l'entrée standard.

On remarque qu'il y a un ensemble d'opérations effectuées par `nawk (1)` qui facilitent grandement la programmation.



Pour chaque ligne d'entrée (d'un fichier ou de l'entrée standard), `nawk (1)` effectue l'appariement entre les champs de la ligne et chacun des patrons du programme. Donc, ce n'est pas un simple case - esac !!