awk : traiter des colonnes et des enregistrements

Quand vous travaillez avec des fichiers de logs, des exports CSV ou des données tabulaires, vous avez souvent besoin d'**extraire des colonnes**, de **filtrer des lignes** ou de **calculer des statistiques**. **awk** est un mini-langage de traitement de texte qui lit un fichier **ligne par ligne**, découpe chaque ligne en **champs** ($1, $2...), et permet d'appliquer des **conditions** et des **actions** sur ces champs.

Ce que vous allez apprendre

- Extraire des colonnes spécifiques d'un fichier
- Filtrer des lignes selon des conditions (valeur, regex)
- Effectuer des calculs (somme, moyenne, comptage)
- Utiliser les blocs **BEGIN** et **END** pour structurer vos traitements
- Manipuler des fichiers CSV avec un séparateur personnalisé

La commande awk dans l'écosystème Linux

awk fait partie des **commandes de traitement de texte** sous **Linux**. Chaque outil a sa spécialité :

Commande	Spécialité	Quand l'utiliser
awk	Traiter des données en colonnes	Extraire, filtrer, calculer sur des champs
sed	Éditer des fichiers en flux	Remplacer du texte, supprimer des lign

Commande	Spécialité	Quand l'utiliser
grep	Rechercher par contenu	Filtrer les lignes qui matchent un motif
cut	Extraire des colonnes simples	Découpage basique par délimiteur
sort / **uniq**	Trier et dédupliquer	Classement, comptage d'occurrences
xargs	Exécuter en masse	Passer des lignes à une commande

→ Glissez pour voir

Combinaisons de commandes fréquentes :

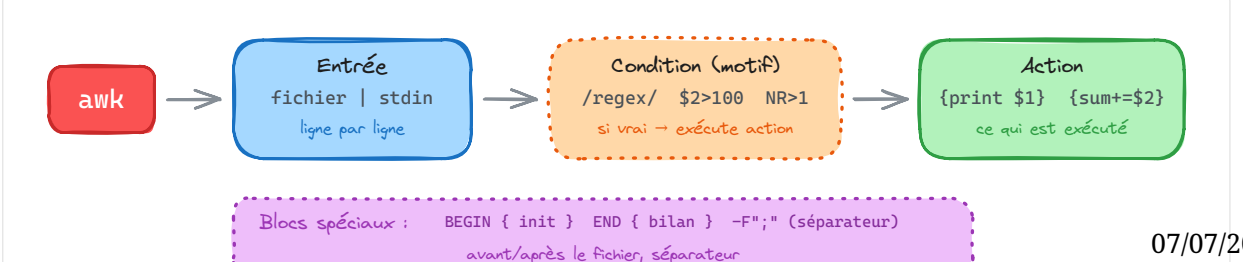
La commande **awk** est souvent utilisée en combinaison avec d'autres outils pour des traitements plus complexes :

```
# awk + sort : trier les résultats d'awk
awk -F';' '{ print $2, $3 }' ventes.csv | sort -k2 -n

# grep + awk : filtrer puis extraire
grep "ERROR" app.log | awk '{ print $1, $NF }'

# awk + xargs : traitement en masse
awk '/pattern/ { print $1 }' fichier | xargs commande
```

Comprendre la commande awk en 2 min

🧠 Modèle mental — Comment fonctionne awk

awk = Pour chaque ligne → Si condition vraie → Exécuter action



POINTS CLÉS

- awk lit le fichier ligne par ligne automatiquement
- Chaque ligne est découpée en champs : $1, $2, $3... ($0 = ligne entière)
- Par défaut, le séparateur est l'espace ou la tabulation
- Sans condition, l'action s'applique à TOUTES les lignes
- BEGIN s'exécute AVANT de lire le fichier (init)
- END s'exécute APRÈS avoir lu tout le fichier (bilan, totaux)

RÈGLES D'OR

1. **Toujours définir -F pour les CSV**

   Par défaut awk sépare sur les espaces. Pour un CSV : awk -F';'...

2. **Tester sur quelques lignes d'abord**

   Utilise head -5 fichier | awk '...' pour vérifier ta commande.

VOCABULAIRE ESSENTIEL

`$0`
La ligne entière

`$1, $2, $N`
Champ 1, 2, N de la ligne

`$NF`
Dernier champ de la ligne

`NR`
Numéro de la ligne courante

`NF`
Nombre de champs dans la ligne

`FS`
Field Separator (séparateur de champs)

> 📚 **Pour aller plus loin — 12 options avancées**

Contrairement à **grep** (qui filtre des lignes) ou **sed** (qui modifie en flux), **awk** est un **vrai**

langage qui permet de traiter chaque ligne comme un enregistrement avec des champs numérotés.

Syntaxe complète

```
awk [options] 'programme' fichier(s)     # prgramme awk ecrit dans la ligne de commande pour les cas simples
awk [options] -f script.awk fichier(s)   # programme awk lu depuis un fichier externe
```

Structure du programme :

```
awk 'BEGIN { init } condition { action } END { final }' fichier
```

Options principales :

Option	Description	Exemple
`-F sep`	Définit le séparateur de champs	**`awk -F';' '...'`** (CSV)
`-v var=val`	Définit une variable avant l'exécution	**`awk -v seuil=100 '$2 > seui`**
`-f script`	Lit le programme depuis un fichier	**`awk -f analyse.awk data.txt`**

→ Glissez pour voir

Formes courantes :

```
# Forme minimale : action sur toutes les lignes
awk '{ print $1 }' fichier

# Avec condition : filtrer les lignes
awk '/pattern/ { print }' fichier
awk '$3 > 100 { print $1, $3 }' fichier

# Avec blocs BEGIN/END : initialisation et finalisation
awk 'BEGIN { FS=";" } { sum += $2 } END { print sum }' fichier

# Depuis un pipe
cat fichier | awk '{ print $NF }'

# Plusieurs fichiers
awk '{ print FILENAME, $0 }' fichier1 fichier2
```

Les 3 parties d'un programme awk

La commande **awk** suit **toujours** la même logique :

Partie	Question	Exemples	O
Entrée	Quel fichier ou flux traiter ?	**`fichier.txt`**, **`stdin`**	⚠️
Condition (motif)	Quelles lignes traiter ?	**`/ERROR/`**, **`$3 > 100`**, **`NR > 1`**	❌
Action	Que faire sur ces lignes ?	**`{ print $1 }`**, **`{ sum += $2 }`**	❌

→ Glissez pour voir

Entrée, **awk** lit le fichier ou le flux **ligne par ligne**. Chaque ligne est automatiquement découpée en champs selon le séparateur (espace par défaut).

Condition *(optionnelle)*, Si vous ne mettez pas de condition, l'action s'applique à **toutes les lignes**. Sinon, seules les lignes qui matchent sont traitées.

Action *(optionnelle)*, Entre accolades **`{ }`**. Sans action explicite, **awk** fait **`{ print }`** (affiche la ligne).

Les variables intégrées : comprendre le découpage

awk découpe automatiquement chaque ligne en **champs** numérotés :

Variable	Signification	Exemple
$0	La ligne entière	**print $0**
$1, **$2**, **$N**	Champ 1, 2, N	**print $1, $3**
$NF	Dernier champ	**print $NF**
NR	Numéro de ligne (global)	**NR > 1** (sauter en-tête)
NF	Nombre de champs	**NF >= 3**
FS	Séparateur d'entrée	**BEGIN { FS=";" }**

→ Glissez pour voir

Exemple concret :

```
# Fichier : "Alice 25 Paris"
# $1=Alice, $2=25, $3=Paris, NF=3
awk '{ print $1, "a", $2, "ans" }' fichier.txt
# Résultat : Alice a 25 ans
```

Définir le séparateur de champs

Par défaut, **awk** sépare sur les **espaces** et **tabulations**. Pour un CSV :

```
# Fichier CSV avec séparateur ;
awk -F';' '{ print $1, $3 }' ventes.csv

# Équivalent avec BEGIN
awk 'BEGIN { FS=";" } { print $1, $3 }' ventes.csv
```

⚠️ **Piège courant** : si **$2** est vide ou incorrect, vérifiez que vous avez bien défini **-F** !

Les filtres : cibler les lignes à traiter

Les conditions permettent de **sélectionner** les lignes sur lesquelles appliquer l'action :

Condition	Ce qu'elle teste	Exemple
$N == "val"	Égalité stricte	**$1 == "admin"**
$N != "val"	Différent de	**$3 != "OK"**
$N > val	Supérieur (numérique)	**$2 > 100**
$N ~ /regex/	Matche la regex	**$1 ~ /^192/**
$N !~ /regex/	Ne matche pas	**$4 !~ /error/**
/pattern/	Pattern sur toute la ligne	**/ERROR/**
NR > 1	À partir de la ligne 2	Sauter l'en-tête CSV

→ Glissez pour voir

Combiner les conditions, ET (&&) / OU (||) / NON (!) :

```
# ET : les deux conditions doivent être vraies
awk '$2 > 100 && $3 == "OK" { print }' fichier.txt

# OU : au moins une condition vraie
awk '$1 == "admin" || $1 == "root" { print }' fichier.txt

# NON : inverser une condition
awk '!($3 ~ /error/) { print }' fichier.txt
```

Les actions : que faire des lignes sélectionnées ?

Une fois les lignes filtrées, vous pouvez agir dessus :

Action	Ce qu'elle fait	Exemple
`{ print }`	Affiche la ligne (défaut)	`{ print }`
`{ print $1, $3 }`	Affiche certains champs	`{ print $1, $3 }`
`{ sum += $2 }`	Accumule dans une variable	Calcul de total
`{ count[$1]++ }`	Tableau associatif	Comptage par clé
`{ next }`	Passe à la ligne suivante	Ignorer cette ligne

→ Glissez pour voir

*Les blocs spéciaux : **BEGIN** et **END***

awk propose deux blocs qui s'exécutent **avant** et **après** le traitement du fichier :

```
awk 'BEGIN { actions_init } condition { actions } END { actions_fin }'
fichier
```

Bloc	Quand s'exécute	Cas d'usage
`BEGIN { }`	**Avant** de lire le fichier	Définir FS, afficher un titre
`END { }`	**Après** avoir lu tout le fichier	Afficher un total, une moyenne

→ Glissez pour voir

Exemple complet, Rapport avec total :

```
awk -F';' '
  BEGIN { print "=== Rapport ===" }
  NR > 1 { total += $3; print $1, $3 }
  END { print "Total:", total }
' ventes.csv
```

Effectuer des calculs

awk excelle pour les **statistiques simples** sur des fichiers :

```
# Somme d'une colonne
awk '{ sum += $2 } END { print "Total:", sum }' fichier.txt

# Moyenne
awk '{ sum += $2; n++ } END { print "Moyenne:", sum/n }' fichier.txt

# Min et Max
awk 'NR == 1 || $2 > max { max = $2 } END { print "Max:", max }'
fichier.txt

# Comptage de lignes selon une condition
awk '$3 == "OK" { count++ } END { print "OK:", count }' fichier.txt
```

Tableaux associatifs : regrouper et compter

Les tableaux associatifs sont **la fonctionnalité la plus puissante** de **awk** pour l'analyse :

```
# Compter les occurrences par valeur
awk '{ count[$1]++ } END { for (k in count) print k, count[k] }' access.log

# Somme par catégorie
awk -F';' '{ ventes[$4] += $2 } END { for (v in ventes) print v, ventes[v]
}' data.csv

# Combiner avec un tri externe
awk '{ count[$1]++ } END { for (k in count) print k, count[k] }' log | sort
-k2 -rn
```

*Formater la sortie avec **printf***

print ajoute automatiquement des espaces et un retour à la ligne. **printf** offre un

contrôle précis :

```
# Alignement et format
awk '{ printf "%-15s %8.2f€\n", $1, $2 }' prix.txt

# Tableau formaté
awk 'BEGIN { printf "%-10s %s\n", "Nom", "Valeur" }
     { printf "%-10s %d\n", $1, $2 }' fichier.txt
```

Format	Signification	Exemple
`%s`	Chaîne	**`printf "%s", $1`**
`%d`	Entier	**`printf "%5d", $2`** (5 caractères)
`%.2f`	Décimal 2 chiffres	**`printf "%.2f", $3`**
`%-10s`	10 caractères, aligné gauche	**`printf "%-10s", $1`**
`\n`	Retour à la ligne	**Obligatoire** avec printf !

→ Glissez pour voir

⚠️ **Piège** : **`printf`** ne met **pas** de retour à la ligne automatiquement. Ajoutez **`\n`** !

Erreurs typiques (et solutions)

Erreur	Cause	Solution
`$2` est vide ou incorrect	Mauvais séparateur	Définir **`-F';'`** pou
Le shell interprète **`$1`**	Doubles guillemets	Utiliser des guille
L'en-tête est inclus dans les calculs	Pas de filtre	Ajouter **`NR > 1`**
`printf` colle tout sur une ligne	Pas de **`\n`**	Ajouter **`\n`** à la fin
Comparaison string au lieu de nombre	"9" > "100" est vrai (string)	Forcer avec **`($2 -`**

→ Glissez pour voir

Les modèles de traitement courants

Maintenant que vous comprenez **awk**, voici dix modèles prêts à l'emploi, du plus simple au plus avancé. Chacun donne sa **formule générale** et un **exemple concret**.

Extraire une colonne

Afficher un champ précis de chaque ligne.

```
awk '{ print $N }' fichier        # formule
awk '{ print $2 }' fichier.txt    # exemple
```

- **`$N`** : numéro du champ (1, 2, 3…)
- **`$0`** : la ligne entière
- **`$NF`** : le dernier champ

Traiter un CSV

Lire un fichier avec un séparateur personnalisé.

```
awk -F'<sep>' '{ print $1, $3 }' fichier.csv   # formule
awk -F';' '{ print $1, $3 }' ventes.csv        # exemple
```

- **`-F`** : définit le séparateur de champs
- **`;`** : séparateur CSV (ou **`,`**, ou **`\t`**)

Filtrer par condition

N'afficher que les lignes qui remplissent une condition.

```
awk '<condition> { action }' fichier            # formule
awk '$3 > 100 { print $1, $3 }' fichier.txt     # exemple
```

- **`$N == "val"`** : égalité stricte
- **`$N > val`** : comparaison numérique
- **`$N ~ /regex/`** : correspondance d'une expression régulière

Sauter l'en-tête

Ignorer la première ligne, les en-têtes d'un fichier CSV.

```
awk 'NR > 1 { action }' fichier             # formule
awk -F';' 'NR > 1 { print $1 }' data.csv    # exemple
```

- **`NR`** : numéro de ligne courant (1, 2, 3...)
- **`NR > 1`** : à partir de la ligne 2

Additionner une colonne

Calculer le total d'une colonne numérique.

```
awk '{ sum += $N } END { print sum }' fichier       # formule
awk '{ sum += $2 } END { print sum }' ventes.txt    # exemple
```

- **`sum += $N`** : accumule la valeur dans une variable
- **`END`** : bloc exécuté après la lecture du fichier

Calculer une moyenne

Moyenne d'une colonne.

```
awk '{ sum += $N; n++ } END { print sum/n }' fichier            #
formule
awk 'NR > 1 { sum += $3; n++ } END { print sum/n }' data.csv    # exemple
```

- **n++** : compte le nombre de lignes traitées
- **sum/n** : moyenne = total ÷ nombre

Compter par groupe

Regrouper et compter les occurrences avec un tableau associatif.

```
awk '{ count[$1]++ } END { for (k in count) print k, count[k] }'
fichier        # formule
awk '{ count[$1]++ } END { for (ip in count) print ip, count[ip] }'
access.log  # exemple
```

- **count[$1]++** : tableau associatif indexé par **$1**
- **for (k in arr)** : parcourt les clés du tableau

Somme par groupe

Additionner une valeur par catégorie.

```
awk '{ total[$1] += $2 } END { for (k in total) print k, total[k] }'
fichier             # formule
awk -F';' '{ vente[$4] += $2 } END { for (v in vente) print v, vente[v] }'
data.csv      # exemple
```

- **total[$1] += $2** : accumule la valeur de **$2** par clé **$1**

Filtrer par expression régulière

Sélectionner les lignes qui correspondent à un motif.

```
awk '/<pattern>/ { action }' fichier      # formule
awk '/ERROR|WARN/ { print }' app.log      # exemple
```

- **`/regex/`** : motif testé sur la ligne entière
- **`$N ~ /regex/`** : motif testé sur un champ précis
- **`!~`** : l'inverse, ne correspond pas

Formater la sortie

Afficher avec un alignement et un format précis.

```
awk '{ printf "%-10s %5d\n", $1, $2 }' fichier        # formule
awk '{ printf "%-15s %8.2f€\n", $1, $2 }' prix.txt    # exemple
```

- **`%-10s`** : chaîne sur 10 caractères, alignée à gauche
- **`%5d`** : entier sur 5 caractères
- **`%.2f`** : décimal avec 2 chiffres après la virgule

Les pièges à éviter

Six erreurs reviennent constamment avec **`awk`**. Pour chacune : la commande piégée, le symptôme, la cause et la correction.

Mauvais séparateur de champs

```
awk '{ print $2 }' fichier.csv   # ❌ affiche la ligne entière
```

Symptôme : **$2** est vide ou ne contient pas ce qu'on attend. **Cause** : le fichier utilise **;** ou **,**, mais **awk** découpe sur les espaces par défaut.

```
awk -F';' '{ print $2 }' fichier.csv   # ✅ séparateur explicite
```

Confusion entre guillemets simples et doubles

```
awk "{ print $1 }" fichier   # ❌ le shell remplace $1 avant awk
```

Symptôme : résultat vide ou inattendu. **Cause** : les guillemets doubles laissent le shell interpréter les **$**.

```
awk '{ print $1 }' fichier   # ✅ guillemets simples : awk reçoit le
programme intact
```

Encadrez **toujours** le programme **awk** de guillemets simples.

En-tête inclus dans les calculs

```
awk -F';' '{ sum += $2 } END { print sum }' data.csv   # ❌ résultat
faux
```

Symptôme : le total ou la moyenne est incorrect. **Cause** : la ligne d'en-tête (du texte) est ajoutée au calcul, où elle vaut 0.

```
awk -F';' 'NR > 1 { sum += $2 } END { print sum }' data.csv   # ✅ en-tête ignoré
```

Comparaison de chaînes au lieu de nombres

```
awk '$2 > 100' fichier   # ❌ "9" > "100" est vrai en comparaison de chaînes
```

Symptôme : des résultats de comparaison incohérents. **Cause** : si le champ ressemble à du texte, **awk** compare comme des chaînes, pas comme des nombres.

```
awk '($2 + 0) > 100' fichier.txt   # ✅ le + 0 force le contexte numérique
```

Condition sans action explicite

```
awk '$3 > 100' fichier   # ⚠️ fonctionne, mais peu lisible
```

Symptôme : la commande affiche des lignes sans action visible. **Cause** : sans **`{ }`**, **awk** applique **`{ print }`** par défaut, un comportement implicite peu clair.

```
awk '$3 > 100 { print }' fichier   # ✅ action explicite
```

`printf` *sans retour à la ligne*

```
awk '{ printf "%s %s", $1, $2 }' fichier   # ❌ toute la sortie sur une seule ligne
```

Symptôme : la sortie est collée, sans retours à la ligne. **Cause** : contrairement à **`print`**, **`printf`** n'ajoute **pas** de saut de ligne automatiquement.

```
awk '{ printf "%s %s\n", $1, $2 }' fichier   # ✅ \n explicite en fin de format
```

Travaux pratiques

Rien ne remplace la pratique. Créez l'environnement de test ci-dessous, puis déroulez les huit étapes. À la fin, nettoyez avec **`rm -rf ~/awk-lab`**.

Préparer le terrain

Ce script crée quatre fichiers de test dans **`~/awk-lab`** : un CSV de ventes, un log **Apache**, un relevé de notes et un fichier d'employés.

```
# Créer le lab
mkdir -p ~/awk-lab
cd ~/awk-lab

# Fichier CSV avec séparateur ;
cat > ventes.csv << 'EOF'
produit;quantite;prix_unitaire;vendeur
laptop;5;899.99;Alice
souris;50;19.99;Bob
clavier;30;49.99;Alice
ecran;10;299.99;Charlie
laptop;3;899.99;Bob
souris;100;19.99;Alice
EOF

# Fichier de logs Apache
cat > access.log << 'EOF'
192.168.1.10 - - [01/Jan/2024:10:00:00] "GET /index.html HTTP/1.1" 200 1234
192.168.1.20 - - [01/Jan/2024:10:01:15] "GET /api/users HTTP/1.1" 200 567
192.168.1.10 - - [01/Jan/2024:10:02:30] "POST /api/login HTTP/1.1" 401 89
192.168.1.30 - - [01/Jan/2024:10:03:45] "GET /index.html HTTP/1.1" 200 1234
192.168.1.20 - - [01/Jan/2024:10:04:00] "GET /api/users HTTP/1.1" 500 45
192.168.1.10 - - [01/Jan/2024:10:05:10] "GET /images/logo.png HTTP/1.1" 200
5678
EOF

# Fichier de notes d'étudiants
cat > notes.txt << 'EOF'
Alice 15 18 12
Bob 10 14 16
Charlie 18 17 19
Diana 12 11 13
Eve 20 19 18
EOF

# Fichier avec espaces (plus complexe)
cat > employes.txt << 'EOF'
Jean Dupont;Développeur;45000
Marie Martin;Chef de projet;55000
Pierre Durand;Administrateur;50000
Sophie Bernard;Développeur;48000
```

```
EOF

echo "✅ Lab créé dans ~/awk-lab"
ls -la ~/awk-lab
```

Les huit étapes

1. **Afficher une colonne spécifique.** Affiche uniquement la première colonne (produit) du fichier CSV.

   ```
   awk -F';' '{ print $1 }' ~/awk-lab/ventes.csv
   ```

   Résultat attendu : la liste des produits (**`laptop`**, **`souris`**, **`clavier`**, **`ecran`**…). **`-F';'`** définit le séparateur, **`$1`** cible le premier champ.

2. **Sauter l'en-tête.** Affiche les produits sans la ligne d'en-tête.

   ```
   awk -F';' 'NR > 1 { print $1 }' ~/awk-lab/ventes.csv
   ```

   `NR` est le numéro de ligne ; **`NR > 1`** saute la première ligne (**`produit`**).

3. **Filtrer par valeur.** Affiche uniquement les ventes réalisées par Alice.

   ```
   awk -F';' '$4 == "Alice" { print }' ~/awk-lab/ventes.csv
   ```

   `$4 == "Alice"` teste si le quatrième champ vaut exactement **`Alice`**, trois lignes correspondent.

4. **Calculer un total.** Calcule le total des quantités vendues.

```
awk -F';' 'NR > 1 { total += $2 } END { print total }' ~/awk-lab/
ventes.csv
```

Résultat attendu : **198** (5+50+30+10+3+100). **`total += $2`** accumule, **`END`** affiche après la lecture.

5. **Calculer une moyenne.** Calcule la moyenne des trois notes de chaque étudiant.

```
awk '{ moy = ($2 + $3 + $4) / 3; print $1, moy }' ~/awk-lab/
notes.txt
```

On effectue le calcul directement dans **`awk`**. Pas besoin de **`-F`** ici : le séparateur par défaut (l'espace) convient.

6. **Compter par catégorie.** Compte le nombre de ventes par vendeur.

```
awk -F';' 'NR > 1 { count[$4]++ } END { for (v in count) print v,
count[v] }' ~/awk-lab/ventes.csv
```

Résultat attendu : **`Alice 3`**, **`Bob 2`**, **`Charlie 1`**. **`count[$4]++`** utilise un tableau associatif indexé par le vendeur.

7. **Filtrer avec une expression régulière.** Affiche les lignes du log contenant une erreur (code HTTP 4xx ou 5xx).

```
awk '$9 ~ /^[45]/ { print }' ~/awk-lab/access.log
```

Le code HTTP est dans le neuvième champ ; **`$9 ~ /^[45]/`** teste s'il commence par 4 ou 5.

8. **Formater la sortie.** Affiche les employés avec leur salaire suivi du symbole €, en colonnes alignées.

```
awk -F';' '{ printf "%-20s %s€\n", $1, $3 }' ~/awk-lab/employes.txt
```

`printf` permet un formatage précis : **`%-20s`** réserve 20 caractères alignés à gauche.

Exercices progressifs

Huit exercices pour valider votre maîtrise, du plus simple au plus complet. Tous se traitent sur les fichiers du lab **`~/awk-lab`** (ou sur **`/etc/passwd`**). Cherchez d'abord par vous-même, puis dépliez la solution pour vous corriger.

Niveau fondations

Exercice 1, Afficher la dernière colonne. Affichez la dernière colonne de chaque ligne de **`/etc/passwd`**. *Indice : **`$NF`** représente le dernier champ.*

> **Voir la solution**

Exercice 2, Filtrer les utilisateurs avec un shell. Affichez les utilisateurs dont le shell est **`/bin/bash`**. *Indice : filtrez sur **`$7`**.*

> **Voir la solution**

Exercice 3, Numéroter les lignes. Affichez chaque ligne du fichier **`notes.txt`** précédée de son numéro. *Indice : **`NR`** contient le numéro de ligne.*

> **Voir la solution**

Niveau composition

Exercice 4, Total d'une colonne CSV. Calculez le total de la colonne **`prix_unitaire`** de **`ventes.csv`** (séparateur **`;`**). *Indice : accumulez avec **`+=`** et affichez dans **`END`**.*

> **Voir la solution**

Exercice 5, Moyenne avec formatage. Calculez la moyenne de la colonne 2 de **`notes.txt`** et affichez-la avec 2 décimales. *Indice : **`printf "%.2f"`**.*

> **Voir la solution**

Niveau industrialisation

Exercice 6, Compter les requêtes par IP. Comptez combien de requêtes chaque IP a faites dans **`access.log`**. *Indice : tableau associatif indexé par **`$1`**.*

> **Voir la solution**

Exercice 7, Filtrer les erreurs HTTP. Affichez uniquement les lignes d'**`access.log`** dont le code HTTP est une erreur (4xx ou 5xx). *Indice : regex sur le champ du code.*

> **Voir la solution**

Niveau synthèse

Exercice 8, Rapport formaté avec totaux. Générez, à partir de **`ventes.csv`**, un rapport avec en-tête, données alignées et total final. *Indice : **`BEGIN`** pour l'en-tête, **`printf`** pour l'alignement, **`END`** pour le total.*

> **Voir la solution**

Dépannage

Quand un programme **`awk`** ne donne pas le résultat attendu, le problème vient presque toujours du **découpage en champs**. Voici comment diagnostiquer.

Méthodes de diagnostic

```
# Voir ce qu'awk reçoit réellement (numéro de ligne + ligne)
awk '{ print NR, $0 }' fichier

# Vérifier le nombre de champs détectés sur chaque ligne
awk '{ print NF, $0 }' fichier

# Afficher chaque champ séparément, numéroté
awk -F';' '{ for (i=1; i<=NF; i++) print i, $i }' fichier

# Tester sur quelques lignes seulement, sans traiter tout le fichier
head -5 fichier | awk '...'
```

Erreurs fréquentes

Message d'erreur	Cause probable
`awk: syntax error at source line 1`	Guillemets ou accolades **`{ }`** mal fermés
`awk: illegal field $()`	Le shell a interprété **`$1`** avant **`awk`**
`awk: division by zero`	Le diviseur vaut 0 (fichier vide, compteur nul)

→ Glissez pour voir

Aide-mémoire awk

Syntaxe générale

Élément	Rôle
`awk [options] '{ programme }' fichier(s)`	Format général de la command
`awk [options] -f script.awk fichier(s)`	Programme lu depuis un fichier
`awk 'BEGIN {…} condition {…} END {…}' fichier`	Structure complète avec blocs

→ Glissez pour voir

Options

Option	Rôle	Exemple
`-F sep`	Définit le séparateur de champs	`awk -F';' '...' data.csv`
`-v var=val`	Définit une variable avant l'exécution	`awk -v seuil=100 '$2 > se`
`-f script.awk`	Lit le programme depuis un fichier	`awk -f analyse.awk data.t`

→ Glissez pour voir

Variables intégrées

Variable	Rôle	Exemple
`$0`	La ligne entière	`print $0`
`$1, $2, $N`	Champ 1, 2, N	`print $1, $3`
`$NF`	Le dernier champ	`print $NF`
`NR`	Numéro de ligne (global)	`NR > 1`
`NF`	Nombre de champs de la ligne	`NF >= 3`
`FS`	Séparateur d'entrée	`BEGIN { FS=";" }`
`OFS`	Séparateur de sortie	`BEGIN { OFS="," }`

→ Glissez pour voir

Conditions

Condition	Rôle	Exemple
`$N == "val"`	Égalité stricte	`$1 == "admin"`

Condition	Rôle	Exemple
`$N ~ /regex/`	Correspond à une regex	`$1 ~ /^[A-Z]/`
`$N !~ /regex/`	Ne correspond pas	`$2 !~ /error/`
`/pattern/`	Correspond sur la ligne entière	`/ERROR/`
`&& ||`	ET / OU logiques	`$2 > 0 && $3 < 100`

→ Glissez pour voir

Blocs, fonctions et affichage

Élément	Rôle	Exemple
`BEGIN { }`	Bloc exécuté avant la lecture	`BEGIN { print "Titre"`
`END { }`	Bloc exécuté après la lecture	`END { print total }`
`length()`	Longueur d'une chaîne	`length($1)`
`substr(s,i,n)`	Extrait une sous-chaîne	`substr($1,1,3)`
`gsub(r,s)`	Remplace toutes les occurrences	`gsub(/old/,"new")`
`split(s,a,fs)`	Découpe une chaîne en tableau	`split($1,parts,"-")`
`tolower()` / `toupper()`	Change la casse	`tolower($1)`
`print`	Affiche (avec saut de ligne)	`print $1, $2`
`printf`	Affichage formaté, sans saut auto	`printf "%-10s", $1`
`%s` / `%d` / `%.2f`	Chaîne / entier / décimal	`printf "%.2f", $3`
`arr[$1]++`	Compteur par clé	`count[$1]++`
`for (k in arr)`	Parcourt les clés d'un tableau	`for (ip in count)`
`delete arr[k]`	Supprime une clé	`delete data["tmp"]`

→ Glissez pour voir

Checklist de maîtrise

Vous maîtrisez **awk** lorsque vous pouvez cocher chacun de ces points :

- ☐ Je sais extraire une colonne avec **$N**
- ☐ Je sais définir un séparateur avec **-F**
- ☐ Je sais filtrer les lignes avec une condition
- ☐ Je sais utiliser **NR**, **NF** et **$0**
- ☐ Je sais calculer une somme avec le bloc **END**
- ☐ Je sais formater une sortie avec **printf**
- ☐ Je sais utiliser les tableaux associatifs
- ☐ Je sais filtrer avec une expression régulière (~)
- ☐ Je sais structurer un traitement avec **BEGIN** et **END**
- ☐ Je sais générer un rapport complet avec en-tête et total

Contrôle de connaissances

Contrôle de connaissances

Validez vos connaissances avec ce quiz interactif

10 questions | **5** min. | **80%** requis

INFORMATIONS

- Le chronomètre démarre au clic sur *Démarrer*
- Questions à choix multiples, vrai/faux et réponses courtes
- Vous pouvez naviguer entre les questions
- Les résultats détaillés sont affichés à la fin

Conclusion

La commande **awk** est un outil fondamental pour quiconque travaille avec des données textuelles. Grâce à son découpage automatique en champs, ses conditions flexibles et ses capacités de calcul, elle permet de **traiter, analyser et transformer** des fichiers structurés en quelques lignes. Que ce soit pour parser des logs, traiter des CSV ou générer des rapports, la maîtrise de **awk** vous fera gagner un temps considérable.

ARTICLES QUI POURRAIENT VOUS INTÉRESSER

- **Éditer des fichiers avec sed**
- **Rechercher avec grep**
- **Rechercher des fichiers avec find**
- **Traitement en masse avec xargs**

FAQ - Questions Fréquemment Posées

Qu'est-ce que la commande awk et à quoi sert-elle ?

La commande awk est un outil en ligne de commande qui permet d'analyser, transformer et...

Comment extraire une colonne spécifique avec awk ?

Pour extraire une colonne, on utilise la syntaxe awk '{ print $n }'. Par exemple, awk '{ print $2 }'...

Comment filtrer des lignes contenant un motif spécifique ?

On peut filtrer en utilisant une expression régulière : awk '/motif/' fichier.txt. Pour filtrer selon un...

Comment faire des calculs sur des colonnes numériques ?

awk permet de calculer des sommes, moyennes ou autres opérations. Exemple : awk '{ total += ...

Comment utiliser des variables shell dans awk ?

Pour passer une variable du shell à awk, utilisez l'option -v. Exemple : awk -v val="$var" '{ print v...

Comment formater la sortie avec printf dans awk ?

La commande printf permet de contrôler le format de sortie. Exemple : awk '{ printf "%-10s |...

Peut-on traiter plusieurs fichiers en même temps ?

Oui, awk peut traiter plusieurs fichiers listés à la suite : awk '{ print FILENAME, $0 }' fichier1...

Comment utiliser des tableaux associatifs avec awk ?

awk prend en charge les tableaux associatifs pour regrouper ou compter des données. Exemple...

Comment écrire un script awk complet dans un fichier ?

Pour des traitements complexes, vous pouvez écrire un script awk dans un fichier (ex : script.aw...

Comment gérer les fichiers CSV avec awk ?

Pour traiter des CSV, spécifiez le séparateur avec -F. Exemple : awk -F',' '{ print $1, $2 }' fichier.csv....

Comment utiliser des expressions régulières dans awk ?

awk permet d'utiliser des regex avec ~ ou !~. Par exemple, awk '$1 ~ /^user/' affiche les lignes...

Quelles sont les principales variables spéciales en awk ?

Parmi les variables intégrées : $0 (ligne entière), $1, $2... (champs), NF (nombre de champs), NR...

LEÇON SUIVANTE **cut** ›