Christophe Blaess

Sed et Awk

Ecriture de scripts shell

Ce document a été réalisé sur système Linux avec les logiciels libres :

- **Formation+** pour préparer le cours complet,
- **LaTeX** pour produire les transparents et le support de cours imprimé,
- **Xfig** pour les graphiques vectoriels,
- **The Gimp** pour les images et les photos.

Support de cours : version 15

Ecriture de scripts shell

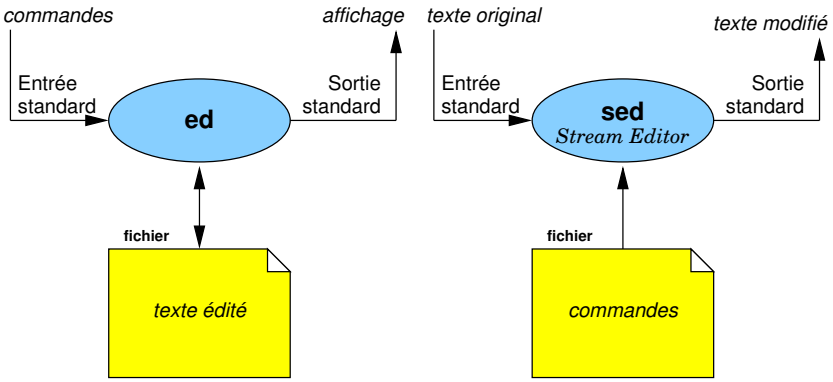
Sed et Awk

Christophe Blaess

– **Langage Sed 3**
  – Introduction à Sed 3
  – Commandes essentielles de Sed 6
  – Commandes supplémentaires de Sed 17
  – Commandes complexes de Sed 18
  – Travaux pratiques 19
– **Introduction à Awk 20**
  – Présentation 20
  – Essentiel de Awk 22
  – Structures de contrôle 30
  – Opérateurs 31
  – Fonctions 32
  – Exemple complet 34

Introduction à Sed

En 1974, Lee McMahon modifie l'éditeur *Ed* pour qu'il travaille sur son flux d'entrée standard et soit contrôlé par des scripts.



Voir aussi :

Articles

– `McMahon_1978.txt` : Lee McMahon – "*SED - A Non-Interactive Text Editor*"

F.A.Q.

– `editor-faq/sed` : "*The Sed FAQ*"

`sed` [*options*] [*fichier*]

`-e` *commande*	ajouter la commande à la liste à exécuter
`-f` *script*	ajouter le fichier aux commandes à exécuter
`-n`	ne pas faire d'affichage automatique des lignes

On peut enchaîner plusieurs options `-e` ou `-f` successives.

Si aucun nom de fichier n'est fourni, *Sed* travaille depuis son entrée standard

Lignes *shebang* pour les scripts *Sed* :
```
#! /bin/sed -f
```
ou
```
#! /bin/sed -nf
```

Voir aussi :
Manuel Unix
– `sed`(1)

Le fonctionnement de *Sed* est le suivant :
- `sed` lit une ligne depuis l'entrée standard,
- il lui applique toutes les commandes qui la concernent,
- si l'option `-n` n'est **pas** demandée, `sed` affiche la ligne modifiée sur la sortie standard,
- puis passe à la ligne suivante.

Les commandes de *Sed* sont toutes représentées par une lettre unique :
`abcdDgGhHilnNpPqrstwxy=`

Une commande peut être précédée d'une *adresse* sélectionnant les lignes sur lesquelles elle s'applique.

Commandes essentielles de Sed

Sur les vingt-trois commandes de *Sed*, il en existe trois essentielles que l'on retrouve fréquemment dans les tâches courantes.

- `d` : éliminer du flux de sortie la ligne sélectionnée
- `p` : affiche la ligne sélectionnée sur la sortie standard
- `s` : appliquer une substitution sur la ligne sélectionnée

Il existe de nombreuses autres commandes, mais elles sont peu utilisées en raison de la mauvaise lisibilité des scripts *Sed*.

La commande `s` (substitute)

La commande `s/`*motif*`/`*remplacement*`/`*options* substitue aux occurences du *motif* (expression rationnelle **simple**) le texte de *remplacement*.

La chaîne de *remplacement* peut contenir les caractères spéciaux suivants :

- `&` : remplacé par l'ensemble de la chaîne mise en correspondance avec le *motif*,
- `\`n : remplacé par le contenu du $n^{\text{ième}}$ regroupement entre parenthèses du *motif*.

Options :

- `g` : effectuer toutes les substitutions possibles sur la ligne,
- n : n'effectuer que la $n^{\text{ième}}$ substitution possible,
- `p` : afficher la ligne si une substitution a lieu.

A vous. . .

Reprenez le fichier `var_log_messages.txt` et remplacez les abbréviations de mois `Oct` et `oct` par la valeur 10.

```
$ sed -e "s/          /      /" < var_log_messages.txt
```

Remplacez les deux-points dans les horodatages par des espaces.

```
$ sed -e "s/          /      /" < var_log_messages.txt
```

A vous. . .

Qu'affichent les commandes suivantes ?

```
$ sed -e "" var_log_messages.txt
$ sed -ne "" var_log_messages.txt
```

```
$ sed -ne "p" var_log_messages.txt
$ sed -e "p" var_log_messages.txt
```

```
$ sed -ne "100p" var_log_messages.txt
$ sed -ne "1880,$p" var_log_messages.txt
$ sed -ne '1880,$p' var_log_messages.txt
```

```
$ sed -ne '/eth0$/p' var_log_messages.txt
```

La commande `p` (print)

La commande `p` affiche la ligne sélectionnée sur la sortie standard.

L'adresse d'une commande peut être :
- un numéro de ligne (commençant à 1),
- une expression rationnelle **simple**,
- le symbole `$` indiquant la dernière ligne.

> Ne pas confondre le symbole `$` indiquant la dernière ligne avec le symbole `$` *dans une expression rationnelle*, qui indique l'ancrage de fin de chaîne.

Un intervalle du type *adresse*`,`*adresse* correspond à toutes les lignes depuis la première correspondant à la première adresse, jusqu'à la première correspondant à la seconde adresses, incluses.

A vous. . .

Qu'affichent les commandes suivantes ?

```
$ sed -ne '100,99p' var_log_messages.txt
$ sed -ne '1880,5000p' var_log_messages.txt
```

```
$ sed -ne '/apmd\[/p' var_log_messages.txt
$ sed -ne '/Suspend/,/Resume/p' var_log_messages.txt
```

```
$ sed -ne '4,1879!p' var_log_messages.txt
```

Un intervalle dont les lignes sont inversées ne correspond qu'à la première ligne.

Un intervalle peut s'étendre au-delà du fichier sans que cela ne soit une erreur.

Les adresses spécifiant un intervalle peuvent être des expressions rationnelles.

Une adresse ou un intervalle suivis d'un caractère `!` sélectionnent les lignes *ne correspondant pas* à l'adresse ou l'intervalle.

La commande `d` (delete)

La commande `d` supprime la ligne en cours du flux de sortie. Il n'y a pas de véritable suppression dans le fichier d'entrée.

Après suppression, *Sed* passe à la ligne suivante du flux d'entrée et reprend son exécution au début de la liste des commandes.

`sed -ne '/`*sélection à conserver*`/p'`
ou
`sed -e '/`*sélection à rejeter*`/d'`
fonctionnent de manière complémentaire, pour parvenir au même résultat.

Utilisez le fichier `mail.txt` dans le répertoire de travaux pratiques.

mail.txt

```
Received: from prof.poudlard.org [12.34.56.78] (user23731)
        by dumble.poudlard.org with SMTP
        id 20htv3-0105nZ-01; Fri, 23 Mar 2001 08:38:27 -0800
Received: from scom.poudlard.ord (salle_commune) [87.65.43.21]
        by prof.poudlard.org with esmtp (Exim 3.12 #1 (Debian))
        id 14gUZv-0004mk-00; Fri, 23 Mar 2001 08:38:19 -0800
Received: from gryfondor by salle_commune with with ESMTP
        id a08tc1-0499cv-31; Fri, 23 Mar 2001 08:38:18 -0800
To: hagrid@poudlard.org
Subject: Paquet recu
Cc: hermione@poudlard.org
Message-Id: <E20htv3-0105hZ-01@touffu>
From: harry@poudlard.org
Date: Fri, 23 Mar 2001 08:38:18 -0800
Status: RO

Edwige m'a bien transmis ton paquet.

A bientôt,

--
Harry
```

A vous. . .

Qu'affichent les commandes suivantes ?

```
$ sed -ne '1,/^$/p' mail.txt
$ sed -e '1,/^$/d' mail.txt
$ sed -e '1,/^$/d;  /^-- $/,$d' mail.txt
```

Il est possible d'enchaîner plusieurs commandes en les séparant par un point-virgule

Exemple de script Sed complet

Les formats d'adresse *e-mail* valides sont : `aa@bb.cc` `AA <aa@bb.cc>` `aa@bb.cc (AA)` `AA : aa@bb.cc`

extrait_adresse.sed

```
#! /bin/sed -nf
# Ne traiter que l'en-tête du mail
1,/^$/ {
        # Travailler sur la ligne From:
        /^From: */ {
                # Supprimer la portion "From:"
                s/^From: *//
                # Supprimer le nom entre parenthèses
                s/ *(.*)//
                # Supprimer tout ce qui suit l'adresse (après >)
                s/>.*//
                # Supprimer tout ce qui précède l'adresse
                s/.*[:<] *//
                # afficher ce qui reste
                p
        }
}
```

```
$ echo "From: aa@bb.cc" | ./extrait_emetteur.sed
aa@bb.cc
$ echo "From: AA <aa@bb.cc>" | ./extrait_emetteur.sed
aa@bb.cc
$ echo "From: aa@bb.cc (AA)" | ./extrait_emetteur.sed
aa@bb.cc
$ echo "From: AA : aa@bb.cc" | ./extrait_emetteur.sed
aa@bb.cc
$ ./extrait_emetteur.sed < mail.txt
harry@poudlard.org
$
```

Commandes supplémentaires de Sed

[*adresse*[,*adresse*]]`l`

Affiche les caractères spéciaux (tab...) sous forme symbolique.

[*adresse*]`=`

Affiche le numéro de la ligne sur la sortie standard.

[*adresse*[,*adresse*]]`n`

Passe à la ligne suivante et continue la liste de commandes.

[*adresse*[,*adresse*]]`y/`*jeu_1*`/`*jeu_2*`/`

Remplace les caractères dans le *jeu_1* par ceux du *jeu_2*

[*adresse*]`q`

Met fin au script après avoir traité la ligne.

[*adresse*]`a`*texte*

Ajoute le *texte* après la ligne sélectionnée.

[*adresse*]`i`*texte*

Insère le *texte* avant la ligne sélectionnée.

[*adresse*[,*adresse*]]`c`*texte*

Remplace les lignes sélectionnées par le *texte*.

[*adresse*[,*adresse*]]`w` *fichier*

Ajoute la ligne en cours dans le *fichier*.

[*adresse*]`r` *fichier*

Affiche le contenu du *fichier* après la ligne indiquée.

Commandes complexes de Sed

[*adresse*[,*adresse*]]`b`*etiquette*

Saute de manière inconditionnelle à l'*étiquette* indiquée.

[*adresse*[,*adresse*]]`t`*etiquette*

Saute à l'*étiquette* si la ligne en cours a été modifiée.

[*adresse*[,*adresse*]]`N`

Charge une nouvelle ligne en mémoire à la suite de celle en cours.

[*adresse*[,*adresse*]]`D`

Efface le contenu de la mémoire jusqu'au premier saut-de-ligne.

[*adresse*[,*adresse*]]`P`

Affiche le contenu de la mémoire jusqu'au premier saut-de-ligne.

[*adresse*[,*adresse*]]`x`

Echange les deux mémoires de *Sed*.

[*adresse*[,*adresse*]]`g`

Copie la mémoire secondaire dans la principale.

[*adresse*[,*adresse*]]`G`

Ajoute la mémoire secondaire à la principale.

[*adresse*[,*adresse*]]`h`

Copie la mémoire principale dans la secondaire.

[*adresse*[,*adresse*]]`H`

Ajoute la mémoire principale à la secondaire.

Travaux pratiques

Exercice numéro 1

Remplacement de chaîne dans plusieurs fichiers

Exercice numéro 2

Extraction de portions de fichier

Exercice numéro 3

Extraction d'une ligne donnée

Exercice numéro 4

Remplacements de motifs complexes

Exercice numéro 5

Elimination de balises HTML

Introduction à Awk

Présentation

En 1977, Alfred Aho, Peter Weinberger, et Brian Kernighan conçoivent chez AT&T Bell un langage de traitement de fichiers de données plus simple que le langage C.

Le langage est nommé *AWK* (leurs initiales).

Awk original n'est plus utilisé, on emploie Nawk (*New Awk*).

Le langage est normalisé par *SUSv3*.

Voir aussi :
F.A.Q.
– `computer-lang/awk/faq` : " *The comp.lang.awk FAQ*"

Invocation de `awk`

`awk` [*options*] [*commandes...*]

Option	Signification
`-f` *script*	Lire les commandes à exécuter dans le *script*
`-v` *variable*=*valeur*	Initialiser la *variable* avant de démarrer les commandes
`-F` *valeur*	Initialiser la variable `FS` avant de démarrer les commandes

Ligne *shebang* des scripts *Awk* :

```
#! /bin/awk -f
```

L'interpréteur `awk` est parfois situé dans `/usr/bin`, mais en principe un lien symbolique est disponible depuis `/bin`.

Chaque implémentation de *Awk* ajoute ses propres options. Celles ci-dessus sont normalisées par *SUSv3*.

Voir aussi :
Manuel Unix
– `awk`(1)

Essentiel de Awk

Awk lit sur son entrée standard une succession d'*enregistrements*, et leur applique des commandes.

Par défaut les enregistrements sont des lignes de texte (séparées par des sauts-de-ligne).

Les enregistrements sont découpés en *champs* placés dans les variables `$1`, `$2`...`$NF`. La variable `$0` accueille l'enegistrement complet.

Par défaut le découpage se fait sur les blancs (espaces, tabulation, retour-chariot).

La variable `NF` (*Number of Fields*) contient le nombre de champs extraits ; la variable `NR` (*Number of Records*) le nombre d'enregistrements déjà lus.

On peut utiliser des notations comme `$NF`, `$(NF-1)`, `$(NF/2)`,etc.

Les commandes *Awk* se présentent sous forme d'un *motif* sélectionnant les enregistrements et d'*actions* à leur appliquer. On écrit en général :

motif `{`
    action
`}`

Une *action* précédée d'un *motif* vide s'applique à tous les enregistrements.

Le *motif* spécial `BEGIN` correspond à une *action* à exécuter avant de lire l'entrée standard.

Le *motif* spécial `END` correspond à une *action* à exécuter après avoir atteint la fin de l'entrée standard.

L'*action* `print` affiche ses arguments. Sans argument elle affiche le contenu de la variable `$0`.

A vous. . .

Que produisent les commandes suivantes ?

```
$ ls -l | awk "{print}"
```

```
$ ls -l | awk "{print $9}"
```

```
$ ls -l | awk '{print $9}'
```

```
$ ls -l | awk '{print $9  " : "  $5}'
```

```
$ awk 'BEGIN{print "hello"}'
$ awk 'BEGIN{print "hello"}  END{print "world"}'
```

Les enregistrements sont séparés en utilisant le contenu de la variable `RS` (*Record Separator*).

- `RS` vide : les enregistrements sont des paragraphes séparés par des lignes blanches.
- `RS` contient un caractère : c'est le séparateur (par défaut le saut de ligne `\n`).
- `RS` contient plusieurs caractères : c'est une expression rationnelle décrivant les séparations.

Les champs sont extraits en utilisant le contenu de la variable `FS` (*Field Separator*).

- `FS` vide : chaque caractère de l'enregistrement est un champ indépendant.
- `FS` contient un caractère : c'est le séparateur (par défaut l'espace).
- `FS` contient plusieurs caractères : c'est une expression rationnelle décrivant les séparations.

A vous. . .

Le fichier `/etc/passwd` contient des enregistrements dont les champs sont séparés par des deux-points : *login*: *password*: *UID*: *GID*: *nom*: *répertoire*: *shell*

Affichez le nom de *login* suivi de l'*UID* de chaque utilisateur

```
$ awk '                                ' < /etc/passwd
```

Sélection

Les motifs de sélection peuvent être :
- des expressions rationnelles
- des conditions portant sur :
  - des comparaisons numériques (numéro d'enregistrement, nombre de champs...)
  - des comparaisons de chaînes (sur un champ)
  - des expressions rationnelles (sur un champ)

A vous...

Quelles lignes sont sélectionnées par les commandes suivantes :

```
$ awk '(NR==14){print}' < /etc/passwd
$ awk '/a.*m/{print}' < /etc/passwd
$ awk -F':' '($1~/a.*m/){print}' < /etc/passwd
```

Variables

La modification d'une variable spéciale entre `$1` et `$NF` met à jour la variable `$0`

Lire une variable spéciale `$i` avec `i>NF` renvoie une chaine vide.

Ecrire dans une variable spéciale `$i` avec `i>NF` augmente NF et met à jour `$0`.

Augmenter `NF` crée des champs supplémentaires vides.

Diminuer `NF` supprime les champs concernés et met à jour `$0`.

Les variables de *Awk* n'ont pas besoin de déclaration préalable, elles peuvent contenir des nombres entiers, réels ou des chaînes, et ne nécessitent pas de préfixe `$`.

A vous. . .

Affichez le pourcentage **cumulé** d'utilisation du processeur par un utilisateur (par tous ses processus).

```
$ ps aux | awk '                    '
```

Affichez le total des tailles des fichiers *réguliers* présents dans un répertoire.

```
$ ls -l | awk '                    '
```

Structures de contrôle

```
function nom (arg1, arg2)
{
  ...
}

nom("abc", 14)
```

```
if (i < 10) {
    print i
} else {
    print "Erreur !"
}
```

```
while (i<n) {
    print i
    i += 1
}
```

```
do {
    print i
    i +=1
} while (i<n)
```

```
for (i=1; i<=NF; i++) {
    print $i
}
```

```
break
continue
exit
```

```
for (i in tableau) {
    print tableau[i]
}
```

```
if (i in tableau) {
    print tableau[i]
}
```

Opérateurs

variable = *expression*	*expression* !~ *expression*	*expression* / *expression*
variable -= *expression*	*expression* == *expression*	*expression* % *expression*
variable += *expression*	*expression* != *expression*	- *expression*
variable /= *expression*	*expression* <= *expression*	! *expression*
variable *= *expression*	*expression* >= *expression*	*expression* ^ *expression*
variable %= *expression*	*expression* < *expression*	*variable* ++
condition ? *expression*	*expression* > *expression*	++ *variable*
expression || *expression*	*expression* + *expression*	*variable* --
expression && *expression*	*expression* - *expression*	-- *variable*
expression ~ *expression*	*expression* * *expression*	(*expression*)

Fonctions

`printf` *format* [*arguments...*]

`print` *arguments* `>` *fichier*
`print` *arguments* `>>` *fichier*
`print` *arguments* `|` `"`*commande*`"`

`close` *fichier*
`close` `"`*commande*`"`

`getline` *variable*
`getline` *variable* `<` *fichier*
`"`*commande*`"` `|` `getline` *variable*

`length` (*chaîne*)
`substr` (*chaîne*, *début*, *longueur*)
`index` (*chaîne*, *sous-chaîne*)
`match` (*chaîne*, *expression*)
`gsub` (*expression*, *remplacement*, *chaîne*)
`split` (*chaîne*, *tableau*, *séparateur*)
`sprintf` (*formats*, *arguments...*)
`tolower` (*chaîne*)
`toupper` (*chaîne*)

`sqrt` (*expression*)
`atan2` (*x*, *y*)
`cos` (*expression*)
`sin` (*expression*)
`exp` (*expression*)
`log` (*expression*)
`int` (*expression*)

Exemple complet

Reprenons le fichier `var_log_messages.txt` nous ayant servi à étudier les expressions rationnelles, `grep` et `sed`.

`/var/log/messages.txt`

```
Oct 19 08:05:07 venux syslogd 1.4.1: restart.
Oct 19 08:23:28 venux login(pam_unix)[4874]: session opened for user ccb by (ui
Oct 19 08:23:28 venux  -- ccb[4874]: LOGIN ON pts/1 BY ccb FROM 192.1.1.52
oct 19 08:23:32 venux su(pam_unix)[4908]: session opened for user root by ccb(u
oct 19 08:24:15 venux su(pam_unix)[4908]: session closed for user root
Oct 19 08:24:15 venux login(pam_unix)[4874]: session closed for user ccb
Oct 19 08:26:45 venux login(pam_unix)[5079]: session opened for user ccb by (ui
Oct 19 08:26:45 venux  -- ccb[5079]: LOGIN ON pts/1 BY ccb FROM 192.168.1.52
oct 19 08:31:42 venux su(pam_unix)[5176]: authentication failure; logname=ccb u
Oct 19 11:44:34 venux cardmgr[965]: socket 0: ATA/IDE Fixed Disk
Oct 19 11:44:35 venux kernel: hde: SunDisk SDCFB-8, CFA DISK drive
Oct 19 11:44:38 venux kernel: ide2 at 0x100-0x107,0x10e on irq 3
Oct 19 11:44:38 venux kernel: hde: attached ide-disk driver.
Oct 19 11:44:38 venux kernel: hde: 15680 sectors (8 MB) w/1KiB Cache, CHS=245/2
Oct 19 11:44:38 venux kernel:  hde: hde1
Oct 19 11:44:38 venux kernel: ide_cs: hde: Vcc = 3.3, Vpp = 0.0
Oct 19 11:44:38 venux cardmgr[965]: executing: './ide start hde'
```

A vous. . .

Appliquez les commandes suivantes pour le modifier :

```
$ sed -e 's/^.ct/10/g;' <var_log_messages >msg1
$ sed -e 's/\([0-9]\):\([0-9]\)/\1 \2/g' <msg1 >msg2
```

Résultat attendu :

msg2

```
10 19 08 05 07 venux syslogd 1.4.1: restart.
10 19 08 09 38 venux login(pam_unix)[3502]: session closed for user ccb
10 19 08 09 39 venux login(pam_unix)[3753]: session closed for user ccb
10 19 08 23 28 venux login(pam_unix)[4874]: session opened for user ccb by (uid
10 19 08 23 28 venux  -- ccb[4874]: LOGIN ON pts/1 BY ccb FROM 192.1.1.52
10 19 08 23 32 venux su(pam_unix)[4908]: session opened for user root by ccb(ui
10 19 08 24 15 venux su(pam_unix)[4908]: session closed for user root
10 19 08 24 15 venux login(pam_unix)[4874]: session closed for user ccb
10 19 08 26 45 venux login(pam_unix)[5079]: session opened for user ccb by (uid
10 19 08 26 45 venux  -- ccb[5079]: LOGIN ON pts/1 BY ccb FROM 192.168.1.52
```

statistiques.awk

```
#! /bin/awk -f

BEGIN {
        for (i = 1; i <= 12; i ++)
                evt_par_mois [i] = 0
        for (i = 1; i <= 31; i ++)
                evt_par_jour [i] = 0
        for (i = 0; i <= 23; i ++)
                evt_par_heure [i] = 0;
}

{       evt_par_mois [$1]++
        evt_par_jour [$2]++
        evt_par_heure[$3]++
}

END {   printf "Nombre d'évènements par mois :\n"
        affiche_repartition(evt_par_mois, 1, 12, 10)
        printf "Nombre d'évènements par jour :\n"
        affiche_repartition(evt_par_jour, 1, 31, 10)
        printf "Nombre d'évènements par heure :\n"
        affiche_repartition(evt_par_heure, 0, 23, 10)
}

function affiche_repartition (tableau, debut, fin, nb_lignes,
        max, i, j, quotient)
{
        max = 0
        for (i = debut; i <= fin; i ++)
                if (tableau [i] > max)
                        max = tableau [i]
        quotient = max / nb_lignes
        for (j = nb_lignes; j > 0; j --) {
                printf "%6d ", j * quotient
                for (i = debut; i <= fin; i ++)
                        if (j * quotient <= tableau [i])
                                printf "##"
                        else
                                printf "  "
                printf "\n"
        }
        printf "       "
        for (i = debut; i <= fin; i ++)
                if ((i - debut) % 2 == 0)
                        printf "%02d  ", i
        printf "\n\n"
}
```

```
Nombre d'évènements par mois :
 568   ##              ##
 511 ####              ##
 454 ######  ##        ##
 397 ######  ####      ##
 340 ############      ##
 284 ##############    ##
 227 ##############    ##
 170 ####################
 113 ####################
  56 ####################
     01  03  05  07  09  11

Nombre d'évènements par jour :
 623                 ##
 560                 ##
 498                 ##
 436       ##        ##            ##  ##           ##
 373   ##  ##        ##            ##  ##         ####
 311   ##  ####      ####    ##    ######  ##    ########
 249   ##  ######    ####  ####    ##########    ########
 186   ##########    ############  ##########    ##########
 124 ##############  ############  ########################
  62 ########################################################
     01  03  05  07  09  11  13  15  17  19  21  23  25  27  29  31

Nombre d'évènements par heure :
 279                               ##    ##  ##
 251                               ####  ######
 223                   ####        ####  ######    ##
 195                   ######      ####  ########  ####
 167                 ########  ################  ####
 139                 ########  ################  ####
 111                 ########  ################  ####
  83             ##############################  ####
  55   ##        ##############################  ####
  27 ######    ######################################
     00  02  04  06  08  10  12  14  16  18  20  22
```

Index

$1 (variable) 22, 24, 27, 28
/etc/passwd (fichier) 26
Aho (Alfred) 20
atan2 (fonction Awk) 33
awk(1) 21, 24, 26, 27, 29
BEGIN (motif) 23, 23, 24, 24, 26
champs 22, 25
close (fonction Awk) 32
commande d (sed) 6, 13, 15
commande p (sed) 6, 9, 11, 15, 16
commande s (sed) 6, 7, 8, 16
cos (fonction Awk) 33
enregistrements 22, 25
exp (fonction Awk) 33
for (mot-clé Awk) 30
FS (variable) 25, 26
function (mot-clé Awk) 30
getline (fonction Awk) 32
gsub (fonction Awk) 33
if (mot-clé Awk) 30
index (fonction Awk) 33
int (fonction Awk) 33
Kernighan (Brian) 20
length (fonction Awk) 33
log (fonction Awk) 33
ls(1) 24, 29
match (fonction Awk) 33
McMahon (Lee) 3
NF (variable) 22, 28
NR (variable) 22, 27
option -e (sed) 4
option -f (sed) 4
option -n (sed) 4
print (action) 23, 24
print (fonction Awk) 32
printf (fonction Awk) 32
ps(1) 29
RS (variable) 25
sed(1) 3, 4, 35
sin (fonction Awk) 33
split (fonction Awk) 33
sprintf (fonction Awk) 33
sqrt (fonction Awk) 33
substr (fonction Awk) 33
SUSv3 (norme) 20, 21
tolower (fonction Awk) 33
toupper (fonction Awk) 33
Weinberger (Peter) 20
while (mot-clé Awk) 30