Christophe Blaess

Expressions rationnelles et outil Grep

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 17

Ecriture de scripts shell

Expressions rationnelles et outil Grep

Christophe Blaess

Utilité des expressions rationnelles

Nous utiliserons le fichier d'exemple `var_log_messages.txt` présent dans le répertoire de travaux pratiques. Il s'agit d'une copie d'un fichier de journalisation des activités du système :

`/var/log/messages`

```
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'
```

Utilisation de Grep

```
grep "user root" < /var/log/messages.txt > messages_1.txt
```

messages_1.txt

```
oct 19 08:23:32 venux su(pam_unix)[4908]: session opened for user root by ccb(ui
oct 19 08:24:15 venux su(pam_unix)[4908]: session closed for user root
oct 19 08:31:49 venux su(pam_unix)[5181]: session opened for user root by ccb(ui
oct 19 08:33:30 venux su(pam_unix)[5181]: session closed for user root
oct 19 17:50:11 venux su(pam_unix)[5967]: session opened for user root by ccb(ui
oct 19 17:51:18 venux su(pam_unix)[5967]: session closed for user root
oct 19 17:51:37 venux su(pam_unix)[6076]: session opened for user root by ccb(ui
oct 19 17:52:39 venux su(pam_unix)[6076]: session closed for user root
oct 19 22:02:25 venux su(pam_unix)[7964]: session opened for user root by ccb(ui
oct 19 22:30:28 venux su(pam_unix)[7964]: session closed for user root
oct 20 13:14:16 venux su(pam_unix)[2398]: session opened for user root by ccb(ui
oct 20 13:25:00 venux su(pam_unix)[1766]: session closed for user root
Oct 20 17:31:06 venux login(pam_unix)[1116]: session opened for user root by LOG
Oct 20 17:33:31 venux login(pam_unix)[1117]: session opened for user root by LOG
Oct 20 17:41:45 venux login(pam_unix)[1118]: session opened for user root by LOG
Oct 20 18:56:13 venux login(pam_unix)[1118]: session closed for user root
Oct 20 18:56:13 venux login(pam_unix)[1116]: session closed for user root
```

Utilisation de Sed

```
sed -e 's/[Oo]ct/10/g'  < messages_1.txt |\
sed -e 's/\([0-9]\):\([0-9]\)/\1 \2/g'  > messages_2.txt
```

messages_2.txt

```
10 19 08 23 32 venux su(pam_unix)[4908]: session opened for user root by ccb(uid
10 19 08 24 15 venux su(pam_unix)[4908]: session closed for user root
10 19 08 31 49 venux su(pam_unix)[5181]: session opened for user root by ccb(uid
10 19 08 33 30 venux su(pam_unix)[5181]: session closed for user root
10 19 17 50 11 venux su(pam_unix)[5967]: session opened for user root by ccb(uid
10 19 17 51 18 venux su(pam_unix)[5967]: session closed for user root
10 19 17 51 37 venux su(pam_unix)[6076]: session opened for user root by ccb(uid
10 19 17 52 39 venux su(pam_unix)[6076]: session closed for user root
10 19 22 02 25 venux su(pam_unix)[7964]: session opened for user root by ccb(uid
10 19 22 30 28 venux su(pam_unix)[7964]: session closed for user root
10 20 13 14 16 venux su(pam_unix)[2398]: session opened for user root by ccb(uid
10 20 13 25 00 venux su(pam_unix)[1766]: session closed for user root
10 20 17 31 06 venux login(pam_unix)[1116]: session opened for user root by LOGI
10 20 17 33 31 venux login(pam_unix)[1117]: session opened for user root by LOGI
10 22 15 01 19 venux su(pam_unix)[8418]: session closed for user root
10 22 15 01 23 venux su(pam_unix)[8596]: session closed for user root
10 20 18 56 13 venux login(pam_unix)[1116]: session closed for user root
10 20 18 56 13 venux login(pam_unix)[1117]: session closed for user root
10 20 18 58 30 venux su(pam_unix)[1228]: session opened for user root by ccb(uid
```

Utilisation de Awk

```
cat messages_2.txt |\
awk '{t[$2] ++} END{for (i in t) print "le " i " :  " t[i] " messages"}'
```

messages_3.txt

```
le 19 :  10 messages
le 20 :  10 messages
le 21 :  8 messages
le 22 :  14 messages
le 23 :  8 messages
```

Utilitaire Grep

L'éditeur *Ed* est disponible sur tous les Unix. Invoqué ainsi : `ed` [*fichier*] il s'agit d'un éditeur *ligne-à-ligne*, contrairement aux éditeurs *pleine page* comme *Vi* ou *Emacs*.

Ed a été écrit pour Unix par Ken Thompson sur PDP 7, en se basant sur l'éditeur *QED* qu'il avait précédement adapté pour *CTSS*, *Gecos* et *Multics*.

Ed fonctionne en appliquant des commandes à la ligne sélectionnée :
- `a` : ajouter du texte *après* la sélection,
- `i` : insérer du texte *avant* la sélection,
- `d` : supprimer la sélection,
- `p` : afficher la sélection,
- `g` : appliquer une commande aux lignes contenant une expression rationnelle,
- ...

En 1973, Ken Thompson extrait les routines de recherche de l'éditeur `ed`

Voir aussi :
Manuel Unix
- `ed`(1)

Utilitaire Grep

`grep` [*options*] *motif* [*fichiers...*]

	Options principales
`-i`	ne pas tenir compte des différences majuscules/minuscules
`-v`	inverser le principe de sélection
`-c`	afficher seulement le nombre de lignes correspondant au motif
`-l`	afficher seulement les noms des fichiers contenant des correspondances
`-h`	n'afficher que les lignes, pas les noms de fichiers
`-n`	afficher les numéros de ligne
`-s`	ne pas afficher de messages d'erreur
`-E`	le motif est une expression rationnelle **étendue**
`-F`	le motif est une chaîne **fixe**

Voir aussi :
Manuel Unix
– `grep`(1)

Principes des expressions rationnelles

Une expression rationnelle (*regular expression*) est un filtre permettant de sélectionner certaines chaînes de caractères.

Il existe des expressions rationnelles **simples** et des expressions **étendues**.

- `ed`, `grep`, `sed` travaillent avec des expressions rationnelles simples,
- `grep -E`, `awk`, `perl` utilisent des expressions rationnelles étendues.

De nos jours, la différence entre expressions simples et étendues tient essentiellement au préfixage de certains caractères spéciaux par un *backslash* \.

Caractères normaux et spéciaux

A vous...

Quelles lignes sont sélectionnées par les commandes suivantes ?

```
$ grep "" var_log_messages.txt
```

```
$ grep "root" var_log_messages.txt
```

```
$ grep "12345" var_log_messages.txt
$ grep "[12345]" var_log_messages.txt
$ grep "\[12345\]" var_log_messages.txt
```

Une expression rationnelle vide ne filtre rien : elle peut être mise en correspondance avec n'importe quelle chaîne.

Dans une expression rationnelle, un caractère **normal** est mis en correspondance avec lui-même.

Dans les expressions rationnelles **simples**, les caractères spéciaux sont :
`\ . ^ $ * [` et `]`.
Dans les expressions rationnelles **étendues**, les caractères suivants sont également spéciaux :
`| + ? ( ) {` et `}`.

Un caractère spécial précédé d'un *backslash* \ perd sa signification spéciale.

A vous. . .

Vous recherchez ce qu'il s'est passé à 15 heures 57. La commande suivante suffit-elle ?

```
$ grep "15:57" var_log_messages.txt
```

A présent affichez les lignes concernant les événements entre 15 heures 50 et 15 heures 59.

```
$ grep "            " var_log_messages.txt
```

Quelles lignes sont sélectionnées par les commandes suivantes ?

```
$ grep "2.4." var_log_messages.txt
```

```
$ grep "2\.4\." var_log_messages.txt
```

Dans une expression rationnelle, le caractère spécial point `.` peut être mis en correspondance avec n'importe quel caractère de la chaîne.

Contrairement au symbole générique du shell (`?`) le point peut être mis en correspondance avec le *slash*.

Pour représenter un point littéral, il faut le préfixer par un *backslash* dans l'expression rationnelle.

A vous. . .

Recherchez les messages correspondant au 20 octobre entre 18 heures et 19 heures.

```
$ grep "             " var_log_messages.txt
```

La commande précédente suffit-elle ? Comment la compléter ?

```
$ grep "             " var_log_messages.txt
```

Quelles lignes sont sélectionnées par les commandes suivantes ?

```
$ grep "eth0" var_log_messages.txt
```

```
$ grep "eth0$" var_log_messages.txt
```

En début d'expression rationnelle, le caractère spécial `^` représente un point d'ancrage pour le début de la chaîne de caractères.

la chaîne `ABCD` peut correspondre à l'expression rationnelle `BCD` mais pas à l'expression rationnelle `^BCD`.

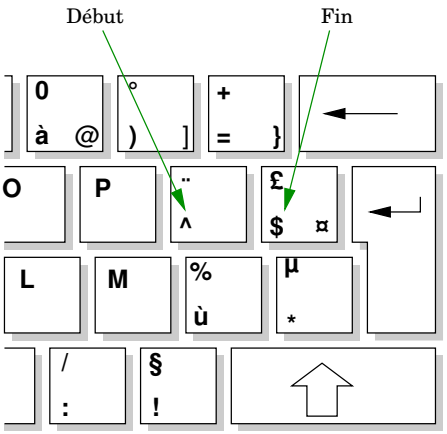
En fin d'expression rationnelle, le caractère spécial `$` représente un point d'ancrage pour la fin de la chaîne de caractères.

la chaîne `ABCD` peut correspondre à l'expression rationnelle `ABC` mais pas à l'expression rationnelle `ABC$`.

L'utilisation, ailleurs qu'en début et fin d'expression rationnelle, des caractères `^` et `$` non protégés par un *backslash* est invalide.

L'expression rationnelle `^$` ne peut être mise en correspondance qu'avec une chaîne vide (ligne blanche)

Moyen mnémotechnique *sur un clavier AZERTY*.



Listes et intervalles

A vous...

Nous recherchons ce qui s'est déroulé entre 06 heures 50 et 06 heures 59.

```
$ grep "            " var_log_messages.txt
```

La commande précédente suffit-elle ? Comment la compléter ?

```
$ grep "            " var_log_messages.txt
```

Dans une expression rationnelle, les crochets `[ ]` encadrent une *liste* de caractères convenant pour une position donnée.

Dans une liste, les caractères spéciaux perdent leurs significations spéciales.

Le symbole `^` en début de liste *inverse* sa signification.

A vous. . .

Quelles lignes sont sélectionnées par les commandes suivantes ?

```
$ grep "06:5[^01234546789]:" var_log_messages.txt
```

```
$ grep "[922]" var_log_messages.txt
$ grep "\[922\]" var_log_messages.txt
```

Pour inclure un '`[`' – ou un '`]`' – littéral dans une liste, il faut le placer en dernière – respectivement en première – position.

A vous. . .

Comment extraire les lignes contenant une lettre entre parenthèses, comme `(C)` ?

```
$ grep "                " var_log_messages.txt
```

Dans une liste, un tiret `-` représente un *intervalle* incluant tous les caractères Ascii entre les deux bornes.

Pour inclure un tiret dans une liste de caractères, il faut le placer en première ou dernière position.

0x00	*Nul*	*Ctrl-A*	*Ctrl-B*	*Ctrl-C*	*Ctrl-D*	*Ctrl-E*	*Ctrl-F*	*Bell*
0x08	*BS*	*HT*	*LF*	*VT*	*FF*	*CR*	*Ctrl-N*	*Ctrl-O*
0x10	*Ctrl-P*	*Ctrl-Q*	*Ctrl-R*	*Ctrl-S*	*Ctrl-T*	*Ctrl-U*	*Ctrl-V*	*Ctrl-W*
0x18	*Ctrl-X*	*Ctrl-Y*	*Ctrl-Z*	*ESC*				
0x20	*Espace*	!	"	#	$	%	&	’
0x28	(	)	*	+	,	-	.	/
0x30	0	1	2	3	4	5	6	7
0x38	8	9	:	;	<	=	>	?
0x40	@	A	B	C	D	E	F	G
0x48	H	I	J	K	L	M	N	O
0x50	P	Q	R	S	T	U	V	W
0x58	X	Y	Z	[	\	]	^	_
0x60	‘	a	b	c	d	e	f	g
0x68	h	i	j	k	l	m	n	o
0x70	p	q	r	s	t	u	v	w
0x78	x	y	z	{	|	}	~	*Del*

Classes de caractères

Pour améliorer la portabilité, la notation `[[:`*classe*`:]]` permet de représenter un caractère de manière qualitative.

`alpha`	Lettres alphabétiques	`[A-Za-zÀÁÂÃÄ...ûüý]` (iso-8859-1)
`digit`	Chiffres décimaux	`[0-9]`
`xdigit`	Chiffres hexadécimaux	`[0-9A-Fa-f]`
`alnum`	Chiffres ou lettres alphabétiques	`[[:alpha:][:digit:]]`
`lower`	Lettres minuscules	`[a-zàáâã...ûüý]` (iso-8859-1)
`upper`	Lettres majuscules	`[A-ZÀÁÂÃÄ...ÛÜÝ]` (iso-8859-1)
`blank`	Caractères blancs	espace, tabulation
`space`	Séparateurs de mots	espace, tab., sauts ligne et page, retour-chariot
`punct`	Signes de ponctuation	`]!"#$%&'()*+,-./:;<=>?@\^_‘{|}~[`
`graph`	Symboles représentables	`[[:alnum:][:punct:]]`
`print`	Symboles imprimables	`[[:graph:] espace]`
`cntrl`	Caractères de contrôle	Ascii < 31 + Ascii 127

0xA0		¡	?	£	?	?	|	§
0xA8	¨	©	?	<<	?	?	?	?
0xB0	?	±	^{2}	^{3}	´	?	¶	·
0xB8	¸	^{1}	?	>>	$\frac{1}{4}$	$\frac{1}{2}$	$\frac{3}{4}$	¿
0xC0	À	Á	Â	Ã	Ä	Å	Æ	Ç
0xC8	È	É	Ê	Ë	Ì	Í	Î	Ï
0xD0	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×
0xD8	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
0xE0	à	á	â	ã	ä	å	æ	ç
0xE8	è	é	ê	ë	ì	í	î	ï
0xF0	∂	ñ	ò	ó	ô	õ	ö	÷
0xF8	ø	ù	ú	û	ü	ý	þ	ÿ

Répétitions

A vous. . .

Recherchez toutes les lignes contenant une chaîne de lettres entre parenthèses

```
$ grep "              " var_log_messages.txt
```

Recherchez les lignes contenant le mot `PCI` et le mot `IRQ` dans cet ordre.

```
$ grep "              " var_log_messages.txt
```

Dans une expression rationnelle, L'astérique `*` signifie *zéro, une ou plusieurs occurrences de l'élément précedent.*

Ne pas confondre l'astérisque des expressions rationnelles avec celui des motifs du shell (qui représente une chaîne de caractère éventuellement vide).

L'expression rationnelle `.*` représente n'importe quelle chaîne de caractères éventuellement vide.

A vous. . .

Recherchez les lignes contenant des adresses IP (une série de quatre nombres décimaux séparés par des points comme `192.168.1.52`.

```
$ grep "                    " var_log_messages.txt
```

Dans une expression rationnelle **étendue**, le symbole plus `+` signifie *une ou plusieurs occurrences de l'élément précedent*. Dans une expression rationnelle **simple**, c'est l'ensemble `\+` qui a cette signification.

L'expression rationnelle étendue `.+`, ou l'expression simple `.\+` représente n'importe quelle chaîne de caractères non vide.

Symétriquement, dans une expression rationnelle **étendue**, le symbole `?` signifie *zéro ou une occurrence de l'élément précedent*. Dans une expression rationnelle **simple**, c'est l'ensemble `\?` qui a cette signification.

On a souvent coutume de lire le point d'interrogation en prononçant *facultatif*.

A vous. . .

Recherchez les lignes contenant une chaîne de 16 nombres hexadécimaux.

```
$ grep "            " var_log_messages.txt
```

Dans une expression rationnelle **étendue**, la chaîne `{m,n}` signifie *au moins m et au plus n occurrences de l'élément précedent*. Dans une expression rationnelle **simple**, c'est l'ensemble `\{m,n\}` qui a cette signification.

- `{n}` signifie *exactement n occurrences*
- `{0,n}` signifie *au plus n occurrences*
- `{m,}` signifie *au moins m occurrences*

Alternatives et groupements

A vous...

Recherchez les lignes contenant la chaîne `ttyS0` ou la chaîne `eth0`.

```
$ grep "            " var_log_messages.txt
```

Dans une expression rationnelle **étendue**, le symbole `|` introduit une alternative entre deux éléments. Dans une expression rationnelle **simple**, c'est l'ensemble `\|` qui a cette signification.

Pour gérer des priorités, on peut introduire des parenthèses `( )` dans les expressions **étendues**, ou préfixées `\( \)` dans les expressions **simples**.

Références arrières

A vous...

Recherchez les lignes horodatées par la même valeur en minutes et secondes (ex : `15:22:22`, `15:57:57:`, `17:30:30`...)

```
$ grep "            " var_log_messages.txt
```

Le symbole `\n` représente le **contenu** du $n^{\text{ième}}$ regroupement entre parenthèses.

L'expression rationnelle simple
`\([[:alpha:]]\)\1\1`
permet de rechercher les triplets de lettres identiques
(*fautes de frapppe*)

Récapitulatif

E.R. Simple	E.R. Etendue	Signification
`\`	`\`	Suppression de la signification des caractères spéciaux
`.`	`.`	N'importe quel caractère
`^`	`^`	Début de chaîne, ou négation en début de liste
`$`	`$`	Fin de chaîne
`[]`	`[]`	Liste, intervalle, classe de caractère
`*`	`*`	Zéro, une, ou plusieurs occurrences
`\+`	`+`	Une ou plusieurs occurrences
`\?`	`?`	Zéro ou une occurrence
`\{\}`	`{}`	Nombre d'occurrences
`|`	`|`	Alternative entre deux éléments
`\(\)`	`()`	Regroupement en sous-expression rationnelle
`\`n	`\`n	Contenu du $n^{\text{ième}}$ regroupement

Association Find et Grep

`find` [*répertoire*] [*option*] [*selection*] [*action*]

L'utilitaire `find` parcours l'arborescence à partir du *répertoire* de départ, filtre les fichiers avec des commandes de *sélection* et leur applique une *action*.

Par défaut, le *répertoire* est '`.`' et l'*action* est `-print`.

Selections courantes :

`-name "*.bak"`	(motif du shell)
`-type f`	« f » fichier régulier, « d » répertoire...
`-size +1k`	taille supérieure à 1 kilo-octet
`-atime -2`	accès (lecture) depuis deux jours
`-mtime +30`	dernière modification il y a plus d'un mois
`-uid 500`	appartient à l'utilisateur UID 500

Voir aussi :
Manuel Unix
– `find`(1)

Actions :

`-print`	(action par défaut)
`-exec commande {} \;`	est remplacé par le nom du fichier
`-ok commande {} \;`	avec confirmation avant exécution

Remarque : `find` ne s'intéresse jamais au contenu des fichiers, uniquement à leur nom, taille, type, etc.

Attention, dans la construction `find ... | grep ...` la recherche avec `grep` a lieu dans les **noms** des fichiers trouvés par `find`, et non pas dans leurs contenus.

Pour exécuter une recherche avec `grep` **dans** les fichiers sélectionnés par `find`, il faut utiliser `xargs` :

```
find ... | xargs grep ...
```

Voir aussi :
Manuel Unix
– `xargs`(1)

Travaux pratiques

Exercice numéro 1

Grep et ses options

Exercice numéro 2

Correspondance d'expressions rationnelles

Exercice numéro 3

Ecriture d'expression simple

Exercice numéro 4

Construction d'expression rationnelle

Exercice numéro 5

Construction d'expression rationnelle complexe

Index