Introduzione a Cfengine

Cfengine [1] è uno strano sistema di amministrazione di elaboratori Unix, la cui importanza si apprende solo con il tempo e con l'utilizzo. Il suo scopo è quello di facilitare l'amministrazione di tali sistemi operativi, soprattutto quando si dispone di un gruppo eterogeneo di questi su diversi elaboratori. Questi capitoli dedicati a Cfengine non pretendono di esaurire l'argomento, cercando piuttosto di semplificare il suo apprendimento che poi può essere approfondito leggendo la documentazione originale.

A prima vista, si può intendere Cfengine come l'interprete di un linguaggio molto evoluto. In questo capitolo si introduce l'uso specifico dell'eseguibile '**`cfengine`**', il cui scopo è interpretare un file di configurazione, ovvero il suo script, agendo di conseguenza.

Primo approccio con la configurazione

Per funzionare, l'eseguibile '**`cfengine`**' richiede la presenza di un file di configurazione, che eventualmente può essere trasformato in script, se ciò può essere conveniente. La comprensione, anche elementare, del modo in cui si configura questo programma, è la chiave per capire a cosa può servire in generale Cfengine.

Il file di configurazione di '**`cfengine`**' ha una struttura speciale, in cui però si possono inserire commenti, preceduti dal simbolo '`#`', e righe vuote o bianche. In particolare, a proposito dei commenti, se questi si collocano alla fine di una direttiva, devono essere staccati da questa con uno o più spazi orizzontali.

Le direttive del file di configurazione vanno inserite all'interno di sezioni ed eventualmente all'interno di classi. In altri termini, il file di configurazione si articola in sezioni, che possono contenere direttive o scomporsi in classi, che a loro volta contengono le direttive. Come si intende, la suddivisione in classi è facoltativa, ma si tratta comunque di una caratteristica fondamentale di Cfengine, in quanto consente di selezionare le direttive da prendere in considerazione in base all'appartenenza o meno dell'elaboratore alle classi stesse.

Dal momento che il problema non è semplice da esporre, conviene iniziare subito con un esempio che possa essere verificato senza troppi problemi anche da un utente comune:

```
# Esempio di partenza

control:
    actionsequence = ( links )

links:
    /var/tmp/altra -> /tmp
```

Se questo file si chiama '`cfengine.conf`' e si trova nella directory corrente, qualunque essa sia, se non è stata impostata la variabile di ambiente '**`CFINPUTS`**', si può avviare l'interpretazione di tale file semplicemente avviando l'eseguibile '**`cfengine`**':

```
$ cfengine [ Invio ]
```

Quello che si ottiene è soltanto la creazione del collegamento simbolico '`/var/tmp/altra`' che punta in realtà alla directory '`/tmp/`'.

Se il file di configurazione fosse collocato altrove, eventualmente con un'altra denominazione, si potrebbe ottenere lo stesso risultato

con il comando seguente, dove il nome del file viene aggiunto nella riga di comando:

```
$ cfengine -f file_di_configurazione [Invio]
```

Infine, per realizzare uno script dalla configurazione, basta inserire all'inizio una riga simile a quella seguente (ammesso che l'eseguibile si trovi effettivamente in '`/usr/bin/`'):

```
#!/usr/bin/cfengine -f
```

In altri termini, lo script completo dell'esempio precedente sarebbe:

```
#!/usr/bin/cfengine -f
# Esempio di partenza

control:
    actionsequence = ( links )

links:
    /var/tmp/altra -> /tmp
```

La variabile CFINPUTS

La variabile di ambiente '**`CFINPUTS`**' serve per definire un percorso di ricerca per il file di configurazione. In generale, se si utilizza l'opzione '**`-f`**' specificando un percorso assoluto, a partire dalla radice (qualcosa che inizia con '`/`'), si tratta esattamente di quel file, altrimenti, se è disponibile la variabile '**`CFINPUTS`**', questa viene preposta al nome del file indicato. Per esempio, il comando

```
$ cfengine -f prova [Invio]
```

fa riferimento precisamente al file di configurazione '`$CFINPUTS/prova`', ovvero al file '`./prova`' se la variabile '**`CFINPUTS`**' non è disponibile.

Quando non si indica il file di configurazione, si fa implicitamente riferimento al nome '`cfengine.conf`'. In tal caso si tratta precisamente di '`$CFINPUTS/cfengine.conf`', ovvero del file '`./cfengine.conf`' in mancanza della variabile '**`CFINPUTS`**'.

Simulazione

Cfengine è un sistema molto potente, i cui script definiscono operazioni molto complesse con poche direttive. Di fronte a direttive distruttive occorre essere sicuri del risultato che si ottiene effettivamente. Per verificare cosa farebbe Cfengine con la configurazione stabilita, senza eseguire realmente la cosa, si può usare l'opzione '**`-n`**', abbinata a '**`-v`**': la prima simula l'esecuzione; la seconda mostra nel dettaglio cosa succede o cosa dovrebbe succedere.

Finché non si è sicuri del proprio script o della propria configurazione, occorre ricordare di fare tutte le prove utilizzando l'opzione '**`-n`**'.

Realizzando uno script con questo intento, basta modificare la prima riga nel modo seguente:

```
#!/usr/bin/cfengine -n -v -f
```

Sezioni e classi predefinite

Le direttive del file di configurazione vanno inserite all'interno di sezioni, che a loro volta possono suddividersi in classi. Le sezioni rappresentano dei tipi di azione e i loro nomi sono già stabiliti.

```
sezione_ovvero_tipo_di_azione :

    definizione_della_classe : :

        direttiva_o_azione
        ...

    ...
```

Negli esempi visti fino a questo punto, sono state mostrate le sezioni '**`control`**' e '**`links`**'. Nella sezione '**`control`**' è stata inserita la direttiva '**`actionsequence`**', che ha l'aspetto di un assegnamento a una variabile:

```
control:
    actionsequence = ( links )
```

Le direttive, ovvero le istruzioni che possono apparire all'interno di classi o di sezioni non suddivise in classi, possono occupare una o più righe, senza bisogno di simboli di continuazione e senza bisogno di simboli per la conclusione delle istruzioni stesse.

In questo caso particolare, si tratta di assegnare uno o più nomi, che rappresentano altrettante sezioni, alla sequenza di esecuzione. In pratica, la direttiva dell'esempio stabilisce che deve essere eseguita la sezione '**`links`**'. Se non venisse specificata in questo modo, la sezione '**`links`**' non verrebbe presa in considerazione. Pertanto, la configurazione seguente non produrrebbe alcunché:

```
# Non fa nulla

control:
    actionsequence = ( )

links:
    /var/tmp/altra -> /tmp
```

Il prossimo esempio dovrebbe chiarire definitivamente questo particolare. Si osservi il fatto che si vuole eseguire prima la sezione '**`tidy`**' e poi la sezione '**`links`**', anche se l'ordine in cui sono mostrate poi le sezioni è inverso.

```
control:
    actionsequence = ( tidy links )

links:
    /var/tmp/altra -> /tmp

tidy:
    /var/tmp pattern=* age=30 recurse=inf
```

In questo caso, la sezione '**`tidy`**' serve a programmare la cancellazione di file e directory. Per la precisione, la direttiva che si vede cancella tutti i file e le directory a partire da '`/var/tmp/`', purché la data di accesso sia trascorsa da almeno 30 giorni. Si osservi anche l'opzione '**`recurse=inf`**', che richiede una ricorsione infinita nelle sottodirectory. In condizioni normali, questa ricorsione non dovrebbe attraversare i collegamenti simbolici, mentre per ottenere tale comportamento occorrerebbe aggiungere l'opzione '**`-l`**'. Pertanto, anche se dovesse esistere già il collegamento simbolico '`/var/tmp/altra`', che punta a '`/tmp/`', questa directory non verrebbe scandita se non richiesto espressamente.

Le classi sono la caratteristica fondamentale di Cfengine, perché consentono di distinguere le direttive di una sezione in base a una sottoclassificazione che serve a selezionare un gruppo ristretto di elaboratori. In pratica, consente di indicare direttive differenti in base alla «classificazione» a cui appartengono gli elaboratori presi in considerazione. Si osservi l'esempio seguente:

```
control:
    actionsequence = ( links )

links:
    linux_2.2.15::
        /var/tmp/altra -> /tmp
    linux_2.2.16::
        /var/tmp/altre -> /tmp
        /var/tmp/altri -> /tmp
```

Anche se poco significativo, l'esempio è abbastanza semplice e dovrebbe permettere di comprendere il senso della distinzione in classi. In questo caso, la sezione '**`links`**' si articola in due classi, denominate '**`linux_2.2.15`**' e '**`linux_2.2.16`**'. Se viene usato questo file in un elaboratore con un sistema GNU/Linux avente un kernel 2.2.15, si ottiene il collegamento simbolico '`/var/tmp/altra`', mentre con un kernel 2.2.16 si otterrebbero due collegamenti simbolici: '`/var/tmp/altre`' e '`/var/tmp/altri`'. Naturalmente, questa operazione può non avere molto significato in generale, ma l'esempio serve a mostrare la possibilità di indicare direttive diverse in base alla classe a cui appartiene l'elaboratore.

La classe serve principalmente a individuare il sistema operativo (nel caso di GNU/Linux si tratta del nome del kernel), in modo da cambiare azione in funzione delle consuetudini di ogni ambiente. In que-

sto caso, volendo selezionare un sistema GNU/Linux senza specificare la versione del kernel sarebbe stato sufficiente indicare la classe '**linux**'. Tuttavia, come si vede nell'esempio, esistono delle classi più dettagliate che permettono di raggiungere anche altre caratteristiche. Per conoscere quali sono le classi valide nell'elaboratore che si utilizza in un certo momento, basta il comando seguente:

```
$ cfengine -p -v [Invio]
```

A titolo di esempio, ecco cosa potrebbe comparire:

```
GNU Configuration Engine -
cfengine-1.5.3
Free Software Foundation 1995, 1996, 1997
Donated by Mark Burgess, Centre of Science and Technology
Faculty of Engineering, Oslo College, 0254 Oslo, Norway

-------------------------------------------------------------------------

Host name is: dinkel
Operating System Type is linux
Operating System Release is 2.2.15
Architecture = i586


Using internal soft-class linux for host dinkel

The time is now Tue Oct 24 16:11:18 2000


-------------------------------------------------------------------------

Additional hard class defined as: 32_bit
Additional hard class defined as: linux_2.2.15
Additional hard class defined as: linux_i586
Additional hard class defined as: linux_i586_2.2.15
Additional hard class defined as:
 linux_i586_2_2_15__1_Thu_Aug_31_15_55_32_CEST_2000

GNU autoconf class from compile time: linux-gnu

  Careful with this - it might not be correct at run time if you have
  several OS versions with binary compatibility!

Address given by nameserver: 192.168.1.1
dinkel: No preconfiguration file
Accepted domain name: undefined.domain


Defined Classes = ( any debian linux dinkel undefined_domain Tuesday Hr16 Min11
 Min10_15 Day24 October Yr2000 32_bit linux_2_2_15 linux_i586 linux_i586_2_2_15
 linux_i586_2_2_15__1_Thu_Aug_31_15_55_32_CEST_2000 linux_gnu 192_168_1
 192_168_1_1 )

Negated Classes = ( )

Installable classes = ( )

Global expiry time for locks: 120 minutes

Global anti-spam elapse time: 0 minutes

Extensions which should not be directories = ( )
Suspicious filenames to be warned about = ( )
```

Le classi disponibili sono quindi quelle elencate nell'insieme '**Defined Classes**'. Si può osservare che è accessibile anche una classe con il nome della distribuzione GNU/Linux (in questo caso è Debian), oltre agli indirizzi IP abbinati all'interfaccia di rete.

Classi più in dettaglio

Le classi non sono necessariamente nomi singoli; possono essere delle espressioni composte da più nomi di classe, uniti tra loro attraverso operatori booleani opportuni. Prima di arrivare a descrivere questo, è bene riassumere le classi più comuni e vedere come si possono definire delle classi nuove. Una classe elementare può essere:

- la parola chiave '**any**', che rappresenta tutti gli elaboratori;
- il nome del sistema operativo o del kernel, assieme a una serie di varianti che includono altre caratteristiche dell'architettura del sistema;
- il nome finale dell'elaboratore (senza il dominio eventuale a cui appartiene);
- il nome che identifica una componente del tempo (giorno, ora, minuto, ecc.), come si vede nella tabella u56.10;
- il nome di un gruppo di classi definito per comodità dell'utilizzatore;

- il nome di una classe libera definito per comodità dell'utilizzatore.

Tabella u56.10. Elenco delle classi di Cfengine riferite al tempo.

Nome	Descrizione
Monday, Tuesday, Wednesday,...	Giorni della settimana.
Hr00, Hr01,... Hr23	Ore del giorno.
Min00, Min01,... Min59	Minuti di un'ora.
Min00_05, Min05_10,... Min55_00	Intervalli di cinque minuti.
Day1, Day2,... Day31	Giorni del mese.
January, February,... December	Mesi dell'anno.
Yr1999, Yr2000, Yr2001,...	Anni.

Si può definire un gruppo di classi attraverso la sezione '**classes**' o '**groups**', in cui le direttive servono per definire delle classi nuove raggruppando più classi preesistenti:

```
classes: | groups:
    gruppo_di_classi = ( classe_1 classe_2... )
    ...
```

Per esempio, la dichiarazione seguente serve a raggruppare in due classi nuove le ore del mattino e le ore della sera, supponendo che ciò possa avere un significato pratico di qualche tipo:

```
classes:
    OreDelMattino = ( Hr06 Hr07 Hr08 Hr09 )
    OreDellaSera  = ( Hr18 Hr19 Hr20 Hr21 )
```

Inoltre si possono definire delle classi in base al risultato soddisfacente di un programma o di uno script. In altri termini, se un programma restituisce *Vero*, questo fatto può essere preso in considerazione come motivo valido per generare una classe. L'esempio seguente crea la classe '**miashell**' se è presente il file '`/bin/bash`' oppure il file '`/bin/zsh`':

```
classes:
    miashell = ( "/bin/test -f /bin/bash" "/bin/zsh" )
```

Si possono dichiarare anche delle classi fittizie, il cui significato si può comprendere solo in un secondo momento. Queste classi fittizie si dichiarano nella sezione '**control**', con la direttiva '**addclasses**':

```
control:
    ...
    addclasses = ( classe_fittizia... )
    ...
```

L'esempio seguente crea due classi fittizie, denominate '**bianchi**' e '**rossi**':

```
    addclasses = ( bianchi rossi )
```

Avendo più chiaro in mente cosa possa essere una classe elementare, si può iniziare a descrivere la definizione di espressioni legate alle classi. Le espressioni in questione sono booleane, dal momento che le classi, di per sé, rappresentano degli insiemi di elaboratori. In questo senso, la logica booleana si intende correttamente come la logica degli insiemi. Gli operatori di queste espressioni sono elencati nella tabella u56.14.

Tabella u56.14. Operatori logici delle espressioni riferite alle classi di Cfengine.

Operatore	Descrizione
( )	Le parentesi tonde hanno la precedenza nella valutazione.
!	NOT, ovvero insieme complementare.
.	AND, ovvero intersezione.
|	OR, ovvero unione.

Operatore	Descrizione
| |	Modo alternativo di indicare OR.

Per esempio, per indicare una classe complessiva che rappresenta indifferentemente un elaboratore con sistema operativo GNU/Linux o GNU/Hurd, si può usare l'espressione '**linux|hurd**'. In pratica, si scrive così:

```
linux|hurd::
```

Per indicare una classe che rappresenti tutti gli elaboratori che non abbiano un sistema operativo GNU/Linux, si potrebbe usare l'espressione '**!linux**', ovvero:

```
!linux::
```

A questo punto diventa più facile comprendere il senso delle classi fittizie che si possono dichiarare con la direttiva '**addclasses**'. Si osservi l'esempio seguente:

```
control:
    actionsequence = ( links )
    addclasses = ( primo )

links:
    any.primo::
        /var/tmp/altra -> /tmp
```

L'espressione '**any.primo**' si avvera solo quando la classe elementare '**primo**' è stata dichiarata come nell'esempio; infatti, '**any**' è sempre vera. In questo modo, anche se l'esempio non richiederebbe tanta raffinatezza, basterebbe controllare la dichiarazione della direttiva '**addclasses**' per abilitare o meno la classe sottostante. In altri termini, è facile modificare un file di configurazione che richiama in più punti la classe fittizia '**primo**', modificando solo una riga di codice nella sezione '**control**'.

Il controllo sulla definizione di classi fittizie può avvenire anche al di fuori del file di configurazione attraverso le opzioni '**-D*classe_fittizia***' e '**-N*classe_fittizia***'. Nel primo caso, si ottiene la dichiarazione di una classe fittizia, mentre nel secondo si ottiene l'eliminazione di una classe già dichiarata nel file di configurazione. Per esempio, il comando seguente serve ad annullare l'effetto della dichiarazione della classe fittizia '**primo**', dell'esempio precedente.

```
$ cfengine -Nprimo -f prova.conf [Invio]
```

Variabili e stringhe

Cfengine gestisce le variabili di ambiente, oltre ad altre variabili, in modo simile a quanto fanno le shell. Queste variabili vengono espanse usando una delle due notazioni seguenti:

```
$(nome_variabile)
${nome_variabile}
```

Per la precisione, le variabili di Cfengine possono essere state ereditate dall'ambiente, possono essere state definite nella sezione '**control**', oppure possono essere variabili predefinite di Cfengine. L'esempio seguente mostra la dichiarazione della variabile '**percorso**' nella sezione '**control**':

```
control:
    actionsequence = ( tidy links )
    percorso = ( "/var/tmp" )

tidy:
    $(percorso) pattern=* age=30 recurse=inf

links:
    $(percorso)/altra -> /tmp
```

Si intuisce che potrebbe essere più interessante dichiarare la variabile in questione all'interno di classi diverse, in modo da aggiornare automaticamente il percorso di conseguenza. L'esempio seguente mostra due classi inventate, '**bianco**' e '**nero**', che non esistono in realtà:


```
control:
    actionsequence = ( tidy links )
    bianco::
        percorso = ( "/var/tmp" )
    nero::
        percorso = ( "/temp" )

tidy:
    $(percorso) pattern=* age=30 recurse=inf

links:
    $(percorso)/altra -> /tmp
```

Si può osservare in particolare che la direttiva '**actionsequence**', non appartenendo ad alcuna classe, viene presa sempre in considerazione.

Le variabili predefinite di Cfengine sono tali perché sono gestite automaticamente e servono a rendere disponibili delle informazioni, oppure perché servono a definire delle informazioni specifiche. In altri termini, le prime vanno solo lette, mentre le altre vanno impostate opportunamente se richiesto. La tabella u56.20 mostra le variabili destinate alla sola lettura, mentre la tabella u56.21 mostra le variabili da impostare.

Tabella u56.20. Variabili interne di Cfengine, destinate alle sola lettura.

Variabile	Descrizione
`allclasses`	Elenca le classi attive.
`arch`	Architettura in modo dettagliato.
`binserver`	Servente NFS predefinito per dati binari.
`class`	Classe essenziale riferita al sistema operativo.
`date`	La data attuale.
`fqhost`	Il nome a dominio completo.
`ipaddress`	Un indirizzo IP significativo dell'elaboratore.
`year`	L'anno attuale.

Per quanto riguarda la variabile '**domain**', se questa non viene impostata espressamente, occorre considerare che potrebbe trattarsi del dominio che compone il nome dell'elaboratore, ovvero ciò che si legge e si imposta con il comando '**hostname**' dei sistemi Unix. In pratica, se il nome dell'elaboratore è stato impostato senza l'aggiunta del dominio di appartenenza, questa variabile restituisce probabilmente la stringa '**undefined.domain**'. Lo stesso discorso vale per la variabile '**fqhost**': se non si dispone del dominio finale nel nome restituito da '**hostname**', si ottiene una cosa simile a '***nome*.undefined.domain**'.

Tabella u56.21. Variabili interne di Cfengine, modificabili da parte dell'utilizzatore.

Variabile	Descrizione
`domain`	Il dominio, senza il nome iniziale dell'elaboratore.
`faculty` `site`	Nome utilizzabile per definire il luogo.
`maxcfengines`	Numero massimo di processi Cfengine concorrenti.
`repchar`	Carattere usato in sostituzione di '`/`' nei nomi di file.
`split`	Carattere usato per separare gli elenchi nelle variabili.
`sysadm`	Amministratore (nome o indirizzo di posta elettronica).
`checksumdatabase`	File destinato alla raccolta dei codici di controllo.

In generale, i nomi delle variabili sono distinti anche in base all'uso di maiuscole e minuscole; tuttavia, le variabili predefinite possono essere usate con qualunque combinazione di lettere maiuscole e minuscole.

Esiste anche un altro gruppo di variabili speciali, in sola lettura, definite per facilitare l'inserimento di caratteri speciali all'interno di stringhe, quando non è possibile fare altrimenti. Queste variabili sono elencate nella tabella u56.22.

Tabella u56.22. Variabili interne per la rappresentazione di caratteri speciali.

Variabile	Descrizione
`cr`	Ritorno a carrello: <*CR*>.
`dblquote`	Apici doppi: '`"`'.
`dollar`	Dollaro: '`$`'.
`lf`	Avanzamento di riga: <*LF*>.
`n`	Codice di interruzione di riga secondo l'architettura.
`quote`	Apice singolo: '`'`'.
`space`	Spazio singolo: <*SP*>.
`tab`	Tabulazione: <*TAB*>.

Le stringhe sono delimitate indifferentemente attraverso apici doppi e singoli, potendo usare anche gli apici singoli inversi. In pratica, si possono usare le forme seguenti:

```
"stringa"
'stringa'
`stringa`
```

Il significato è lo stesso e l'espansione delle variabili avviene in tutti i casi nello stesso modo. Disponendo di diversi tipi di delimitatori, è più facile includere questi simboli nelle stringhe stesse. In questo senso va considerato il fatto che non esistono sequenze di escape; al massimo si possono usare le variabili predefinite per la rappresentazione di caratteri particolari.

Le stringhe sono utilizzabili solo in contesti particolari, precisamente la definizione di valori da assegnare a una variabile dichiarata nella sezione '**`control`**' e i comandi di shell nella sezione '**`shellcommands`**' (che non è ancora stata mostrata).

Elenchi

Le variabili possono essere intese come contenenti un elenco di sottostringhe. In questi casi, la loro espansione può richiedere una valutazione ulteriore. Tutto ha inizio dalla variabile interna '**`split`**', che normalmente contiene il carattere '`:`'. In questo senso, si osservi l'esempio seguente:

```
control:
    actionsequence = ( tidy )
    elenco = ( "primo:secondo:terzo" )

tidy:
    /var/tmp/$(elenco) pattern=* age=0
```

Assegnando alla variabile '**`elenco`**' la stringa '**`primo:secondo:terzo`**', si ottiene l'indicazione di un elenco di tre sottostringhe: '**`primo`**', '**`secondo`**' e '**`terzo`**'. A questo punto, la direttiva contenuta nella sezione '**`tidy`**', si traduce nella cancellazione dei file '`/var/tmp/primo`', '`/var/tmp/secondo`' e '`/var/tmp/terzo`'. Volendo cambiare il simbolo di separazione delle sottostringhe si agisce nella variabile '**`split`**', come si vede nell'esempio seguente, che ottiene lo stesso risultato.


```
control:
    actionsequence = ( tidy )
    split = ( " " )
    elenco = ( "primo secondo terzo" )

tidy:
    /var/tmp/$(elenco) pattern=* age=0
```

Naturalmente, si può ottenere l'espansione di variabili del genere solo nei contesti in cui questo può avere significato.

Espressioni regolari

In contesti ben determinati, si possono indicare delle espressioni regolari. Cfengine utilizza le espressioni regolari ERE secondo le convenzioni GNU. Sono disponibili gli operatori riassunti nella tabella u56.25.

Tabella u56.25. Elenco degli operatori delle espressioni regolari.

Operatore	Descrizione
`\`	Protegge il carattere seguente da un'interpretazione diversa da quella letterale.
`^`	Ancora dell'inizio di una stringa.
`.`	Corrisponde a un carattere qualunque.
`$`	Ancora della fine di una stringa.
`|`	Indica due possibilità alternative alla sua sinistra e alla sua destra.
`( )`	Definiscono un raggruppamento.
`[ ]`	Definiscono un'espressione tra parentesi quadre.
`[`*`xy...`*`]`	Un elenco di caratteri alternativi.
`[`*`x-y`*`]`	Un intervallo di caratteri alternativi.
`[^...]`	I caratteri che non appartengono all'insieme.
`x``*`	Nessuna o più volte ***x***. Equivalente a '*`x`*`{0,}`'.
`x``?`	Nessuna o al massimo una volta ***x***. Equivalente a '*`x`*`{0,1}`'.
`x``+`	Una o più volte ***x***. Equivalente a '*`x`*`{1,}`'.
`x``{`*`n`*`}`	Esattamente ***n*** volte ***x***.
`x``{`*`n`*`,}`	Almeno ***n*** volte ***x***.
`x``{`*`n`*`,`*`m`*`}`	Da ***n*** a ***m*** volte ***x***.
`\b`	La stringa nulla all'inizio o alla fine di una parola.
`\B`	La stringa nulla interna a una parola.
`\<`	La stringa nulla all'inizio di una parola.
`\>`	La stringa nulla alla fine di una parola.
`\w`	Un carattere di una parola, praticamente '`[[:alnum:]_]`'.
`\W`	L'opposto di '`\w`', praticamente '`[^[:alnum:]_]`'.

Le espressioni regolari GNU includono anche le classi di caratteri (nella forma '`[:`***nome***`:]`', come prescrive lo standard POSIX, mentre mancano i simboli di collazione e le classi di equivalenza..

[1] **CFengine** GNU GPL