Bash Prompt HOWTO

Giles Orr, *giles@interlog.com*

v0.60, 07 gennaio 1999

Viene discusso come creare e controllare il prompt del terminale e di xterm, compresa l'uso delle sequenze di escape standard per ottenere il nome utente, la directory di lavoro corrente, l'ora, ecc. Vengono esposti ulteriori suggerimenti su come modificare le barre di titolo di xterm, usare funzioni esterne per fornire informazioni sul prompt e su come usare i colori ANSI.

Contents

1 Introduzione

1.1 Prerequisiti

Avrete bisogno di Bash. La versione fornita con quasi tutte le distribuzioni Linux è la 1.14.7 (al momento della stesura di questo documento, Novembre 98), che è una shell ben conosciuta e affidabile. Bash è ora disponibile nella versione 2.0+: oramai ho utilizzato Bash 2.0 per qualche tempo, ma quasi tutto il codice qui presentato dovrebbe funzionare con la 1.14.7. Se dovessi essere a conoscenza di un problema, ne farò menzione. Potete controllare la vostra versione di Bash digitando `echo $BASH_VERSION` al prompt. Sulla mia macchina, risponde con `2.02.1(1)-release`.

Sarebbe utile, ma non essenziale, esperienza nella programmazione shell: più ne sapete, più sarete in grado di creare prompt complessi. In questo tutorial presuppongo una conoscenza di base della programmazione shell e delle utility Unix. Comunque, le mie stesse capacità nella programmazione shell sono limitate, così fornisco molti esempi e spiegazioni che possono apparire superflue ad un esperto programmatore shell.

1.2 Come Usare Questo Documento

Includo molti esempi e spiegazioni. Parti diverse saranno utili in varia misura a persone diverse. Questo documento è diventato abbastanza lungo e leggerlo tutto in una volta sarebbe difficile - leggete solamente le sezioni che vi servono, tornate indietro quando necessario.

1.3 Traduzioni

Al momento in cui scrivo (6 gennaio 99), sono in lavorazione traduzioni in giapponese (Akira Endo, *ak-endo@t3.rim.or.jp*) e tedesco (Thomas Keil, *thomas@h-preissler.de*). Molte grazie ad entrambi! Gli URL verranno inclusi quando le traduzioni saranno disponibili.

1.4 Problemi

Questa è una lista di problemi che ho notato nel programmare i prompt. Non iniziate a leggere qui, e non lasciate che questa lista vi scoraggi - questi sono principalmente dettagli minori. Controllate qui se vi scontrate con qualche cosa di strano.

- Molte funzionalità di Bash (come i calcoli matematici all'interno di $(()) fra gli altri) sono opzioni definite in fase di compilazione. Se state usando una distribuzione binaria come quelle fornite con una distribuzione standard di Linux, tutte queste funzionalità dovrebbero essere già definite in fase di compilazione. Ma se state lavorando sul sistema altrui, vale la pena ricordarsene se qualcosa non funziona come vi aspettate. Vi sono alcune note su questo in *Learning the Bash Shell*, p.260-262.
- Lo screen manager "screen" non funziona sempre bene con i colori ANSI. Sfortunatamente non sono un esperto di screen. La mia attuale versione di screen (una molto recente) sembra funzionare bene in tutti i casi, ma ho visto occasioni in cui screen ha ridotto tutti i colori del prompt al colore di primo piano standard negli X terminal. Questo non sembra essere un problema nella consolle.
- I file Xdefault possono reimpostare i colori. Cercate in ~/.Xdefaults linee che fanno riferimento a XTerm*background e XTerm*foreground (o forse XTerm*Background e XTerm*Foreground).
- Uno dei prompt menzionati in questo documento usa l'output di "jobs" - come già discusso, l'output di "jobs" verso un pipe non funziona in Bash 2.02.
- Le sequenze di escape ANSI per il movimento del cursore non sono tutte implementate in tutti gli X terminal. Questo è discusso in una sezione a parte.
- Alcune pseudo-immagini carine possono essere create usando i font VGA piuttosto che i font standard di Linux. Sfortunatamente, questi effetti sono pessimi se non usate font VGA, e non c'è modo di scoprire all'interno di un terminale quali font sta usando.
- Bash 2.0+ è disponibile e include alcune nuove funzionalità e cambia un po' il comportamento. Cose che funzionano sulla 1.14.7 non funzionano necessariamente sulla 2.0+, o vice versa.

1.5 Inviatemi Commenti e Suggerimenti

Questa è anche per me una "esperienza didattica". Sono arrivato a saperne un bel po' su cosa può essere fatto per creare prompt Bash interessanti e utili, ma ho bisogno dei vostri suggerimenti per correggere e migliorare questo documento. Ho provato a controllare i miei suggerimenti con versioni differenti di Bash (principalmente 2.02, che uso, e 1.14.7, che è molto usata), ma fatemi sapere se trovate delle incompatibilità.

L'ultima versione di questo documento dovrebbe essere sempre disponibile su *http://www.interlog.com/~giles/bashprompt.html* . Per favore dategli un'occhiata e lasciate pure un e-mail a *giles@interlog.com*

con suggerimenti.

Uso gli HOWTO del Linux Documentation Project quasi esclusivamente in formato HTML, così quando converto questo documento da SGML, HTML è l'unico formato che controllo interamente. Se ci sono problemi con altri formati, potrei non saperlo e gradirei una nota a riguardo.

1.6 Crediti

Nel produrre questo documento, ho preso in prestito molto dal lavoro del progetto Bashprompt su *http://bash.current.nu/* . Altre fonti usate includono l'*xterm Title mini-HOWTO* di Ric Lister, reperibile su *http://sunsite.unc.edu/LDP/HOWTO/mini/Xterm-Title.html* , *Ansi Prompts* di Keebler, reperibile su *http://www.ncal.verio.com/˜keebler/ansi.html* , *How to make a Bash Prompt Theme* di Stephen Webb, reperibile su *http://bash.current.nu/bash/HOWTO.html* e *X ANSI Fonts* di Stumpy, reperibile su *http://home.earthlink.net/˜us5zahns/enl/ansifont.html* .

Sono state anche di immenso aiuto diverse conversazioni e e-mail di Dan, un collega del Georgia College & State University, le cui conoscenze di UNIX superano di molto le mie. Egli mi ha dato numerosi ed eccellenti suggerimenti, le sue idee hanno portato ad alcuni prompt interessanti.

Tre libri che sono stati utili nel prorammare i prompt sono *Linux in a Nutshell* di Jessica Heckman Perry (O'Reilly, 1997), *Learning the Bash Shell* di Cameron Newham e Bill Rosenblatt (O'Reilly, 2nd. ed., 1998) e *Unix Shell Programming* di Lowell Jay Arthur (Wiley, 1986. Questa è la prima edizione, la quarta è uscita nel 1997).

1.7 Copyright e Liberatoria

2 Bash e i Prompt Bash

2.1 Cosa è Bash

Discendente dalla Bourne Shell, Bash è un progetto GNU, la "**B**ourne **A**gain **SH**ell". È l'interfaccia a linea di comando standard su molte macchine Linux. Eccelle in interattività, nel supportare modifica, completamento e richiamo della linea di comando. Supporta anche prompt configurabili - molti lo sanno, ma non sanno quanto si può fare.

2.2 Quali sono i vantaggi di modificare il prompt?

La maggior parte dei sistemi Linux hanno un prompt predefinito in un colore (di solito grigio) che dice il vostro nome utente, il nome della macchina su cui lavorate e qualche indicazione sulla directory di lavoro corrente. Queste sono tutte informazioni utili, ma si può fare molto di più con il prompt: ogni genere di informazione può essere visualizzata (numero di tty, ora, data, carico della macchina, numero di utenti, uptime ...) e il prompt può utilizzare colori ANSI, per farlo apparire interessante, o per fare sì che certe informazioni siano evidenti. È anche possibile manipolare la barra del titolo di un Xterm per visualizzare alcune di queste informazioni.

2.3 Perché darsi tante noie?

Oltre che essere bello, è spesso utile tenere traccia delle informazioni di sistema. Una idea che so piace a molti è la possibilità di mettere su macchine differenti prompt con colori differenti. Se avete diversi Xterm aperti su macchine differenti, o se tendete a dimenticare su quale macchina state lavorando e a cancellare file sbagliati, troverete questa un ottima maniera per ricordare su che macchina vi trovate.

2.4 Il Primo Passo

L'aspetto del prompt viene controllato dalla variabile della shell PS1. Le continuazioni di un comando sono indicate dalla stringa PS2, che può essere modificata con esattamente gli stessi metodi qui discussi - poiché controllarla è esattamente uguale e non è altrettanto "interessante", modificherò primcipalmente la stringa PS1. (Ci sono anche le stringhe PS3 e PS4. Queste non vengono mai viste dall'utente medio - si veda la pagina di manuale di Bash se si è interessati al loro scopo). Per modificare l'aspetto del prompt, dovete cambiare la variabile PS1. Per fare esperimenti, potete cambiare la stringa PS1 direttamente al prompt e vedere immediatamente i risultati (questo influenza solo la sessione corrente e i cambiamenti spariscono quando fate log-out). Se volete rendere permanente un cambiamento del prompt, modificate il file ~/.bashrc e aggiungete lì la nuova definizione di PS1. Se avete permessi di root, potete guardare in /etc/profile e modificare la linea "**PS1=**". Sappiate che in alcune distribuzioni (almeno la Redhat 5.1) /etc/bashrc resetta le stringhe PS1 e PS2.

Prima di cominciare, è importante ricordare che la stringa PS1 viene salvata nell'ambiente. Se la modificate alla linea di comando, il prompt cambierà di conseguenza. Prima di fare dei cambiamenti, potete salvare il prompt corrente in un'altra variabile d'ambiente:

```
[giles@nikola giles]$ SAVE=$PS1
[giles@nikola giles]$
```

Il prompt più semplice sarebbe un singolo carattere, come:

```
[giles@nikola giles]$ PS1=$
$ls
bin   mail
$
```

Questo dimostra il modo migliore per fare esperimenti con semplici prompt, digitarli alla linea di comando. Si noti che il testo digitato dall'utente appare immediatamente dopo il prompt: io preferisco usare

```
$PS1="$ "
$ ls
bin   mail
$
```

che inserisce uno spazio dopo il prompt, rendendolo più leggibile. Per ripristinare il prompt originale, semplicemente richiamate la variabile che avete salvato:

```
$ PS1=$SAVE
[giles@nikola giles]$
```

2.5 Sequenza di Escape dei Prompt Bash

Vi sono molte sequenze di escape offerte dalla shell Bash da inserire nel prompt. Dalla pagina di manuale di Bash 2.02:

```
Quando eseguita in modalita interattiva, bash mostra il prompt
primario quando e pronta per leggere un comando e il prompt
secondario PS2 quando e necessario ulteriore input per
completare il comando. Bash permette di personalizzare queste stringhe
di prompt inserendo vari caratteri di escape speciali che vengono
interpretati come segue:
       \a     il carattere ASCII beep (07)
       \d     la data nel formato "Giorno-della-settimana Mese Data"
              (e.g., "Tue May 26")
       \e     un carattere di escape ASCII (033)
       \h     l'hostname fino al primo `.'
       \H     l'hostname
       \n     il carattere "newline"
       \r     il carattere "carriage return"
       \s     il nome della shell, il nome base di $0
              (la parte che segue lo slash finale)
       \t     l'ora corrente nel formato 24-ore HH:MM:SS
       \T     l'ora corrente nel formato 12-ore HH:MM:SS
       \@     l'ora corrente nel formato 12-ore am/pm
       \u     lo username dell'utente corrente
       \v     la versione di bash (e.g., 2.00)
       \V     la release di bash, versione + patchlevel
              (e.g., 2.00.0)
       \w     la directory di lavoro corrente
       \W     il nome di base della directory di lavoro corrente
       \!     il numero cronologico (history number) di questo comando
       \#     il numero di questo comando
       \$     se l'UID effettivo e 0, un #, altrimenti un $
       \nnn   il carattere corrispondente al numero ottale nnn
       \\     un backslash
       \[     comuncia una sequenza di caratteri non stampabili, che
              potrebbero essere usati per inserire una sequenza di
              controllo del terminale nel prompt
       \]     termina la sequenza di caratteri non stampabili
```

Continuando da dove avevamo interrotto:

```
[giles@nikola giles]$ PS1="\u@\h \W> "
giles@nikola giles> ls
bin   mail
giles@nikola giles>
```

Questo è simile al prompt predefinito su molte distribuzioni Linux. Volevo un apparenza leggermente differente, così l'ho cambiato a:

```
giles@nikola giles> PS1="[\t][\u@\h:\w]\$ "
[21:52:01][giles@nikola:~]$ ls
bin   mail
[21:52:15][giles@nikola:~]$
```

2.6 Impostare Permanentemente le Stringhe PS?

Varie persone e distribuzioni impostano le loro stringhe PS? in posti diversi. I posti più comuni sono /etc/profile, /etc/bashrc, ~/.bash_profile e ~/.bashrc. Johan Kullstam (*johan19@idt.net*) scrive:

```
La stringa PS1 dovrebbe essere impostata in
.bashrc. questo perche le shell bash non interattive resettano
PS1. La pagina di manuale di bash dice come la presenza o l'assenza di
PS1 e un buon modo di sapere se ci si trova in una sessione
bash interattiva oppure non-interattiva (e.g. uno script).

Il modo in cui mi sono accorto di questo e che startx e
uno script bash. Questo significa che startx annullera il
vostro prompt. Quando impostate PS1 in .profile (o .bash_profile),
fate login alla consolle, fate partire X con startx, il vostro PS1
viene annullato nel processo lasciandovi con il prompt predefinito.

Una soluzione e di lanciare xterm e rxvt con l'opzione -ls per
forzarli a leggere .profile. Ma ogni volta che viene invoca una shell
mediante uno shell-script non interativo PS1 viene perduto. system(3)
usa sh -c che se sh e bash distruggera
PS1. Un modo migliore e mettere la definizione di PS1 in
.bashrc . questo viene letto ogni volta bash parte ed e dove le
cose interattive - come PS1 - dovrebbero restare.

Quindi dovrebbe essere sottolineato che PS1=..blabla.. dovrebbe stare
in .bashrc e non in .profile.
```

Ho provato a duplicare il problema che spiega, e ne ho incontrato uno diverso: la mia variabile PROMPT_COMMAND (che verrà introdotta più tardi) è stata distrutta. Le mie conoscenze in quest'area sono un po' vaghe, così mi associo a quanto dice Johan.

3 Comandi Esterni

3.1 PROMPT_COMMAND

Bash fornisce un'altra variabile d'ambiente chiamata **PROMPT_COMMAND**. Il contenuto di questa variabile viene eseguito come un normale comando Bash appena prima che Bash visualizzi il prompt.

```
[21:55:01][giles@nikola:~] PS1="[\u@\h:\w]\$ "
[giles@nikola:~] PROMPT_COMMAND="date +%H%M"
2155
[giles@nikola:~] d
bin   mail
2156
[giles@nikola:~]
```

Ciò che è accaduto sopra è che ho cambiato PS1 in modo da non includere più la sequenza di escape `\t`, così da visualizzare l'ora in un formato che mi piace di più. Ma l'ora compare in una linea diversa dal prompt. Aggiustando questo con `echo -n ...` (come mostrato sotto) funziona con Bash 2.0+, ma sembra non funzonare con Bash 1.14.7: apparentemente il prompt viene ottenuto in maniera differente e il metodo seguente causa una sovrapposizione del testo.

```
2156
[giles@nikola:~] PROMPT_COMMAND="echo -n [$(date +%H%M)]"
[2156][giles@nikola:~]$
[2156][giles@nikola:~]$ d
bin   mail
[2157][giles@nikola:~]$ unset PROMPT_COMMAND
[giles@nikola:~]
```

`echo -n ...` controlla l'output del comando `date` e sopprime il carattere newline finale, permettendo al prompt di apparire tutto su una riga. Alla fine ho usato il comando `unset` per rimuovere la variabile d'ambiente PROMPT_COMMAND.

Si noti che uso la convenzione $(<comando>) per la sostituzione dei comandi: ovvero:

```
$(date +%H%M)
```

significa "sostituisci qui l'output del comando **date +%H%M**". Questo funziona in Bash 2.0+. In qualche versione più vecchia di Bash, precedente alla 1.14.7, potreste dovere usare i backquote (`` `date +%H%M` ``). I backquote possono essere usati in Bash 2.0+, ma stanno venendo via via rimpiazzati in favore di $(), che si annida meglio. Continuerò ad utilizzare questa convenzione in questo documento. Se state usando una versione precedente di Bash, potete di solito sostituite dei backquote dove vedete $(). Se la sostituzione di comandi è preceduta da "\" (cioè \$(comando)), usate dei backslash davanti ad entrambi i backquote (cioè \'comando\').

3.2 Comandi Esterni nel Prompt

Potete anche usare l'output di normali comandi Linux direttamente nel prompt. Ovviamente, non dovrete inserire molto materiale altrimenti creerà un prompt molto grande. Dovrete anche inserire un comando **veloce**, perché verrà eseguito ogni volta che il prompt appare sullo schemo e ritardi nell'apparire del prompt mentre state lavorando possono essere molto fastidiosi. (Differentemente dall'esempio precedente, a cui assomiglia molto, questo funziona anche con Bash 1.14.7).

```
[21:58:33][giles@nikola:~]$ PS1="[\$(date +%H%M)][\u@\h:\w]\$ "
[2159][giles@nikola:~]$ ls
bin   mail
[2200][giles@nikola:~]$
```

È importante notare il backslash prima del segno dollaro della sostituzione di comando. Senza di esso, il comando esterno viene eseguito esattamente una volta: quando la stringa PS1 viene letta nell'ambiente. Per questo prompt, ciò significherebbe mostrare lo stesso orario indipendentemente da quanto il prompt viene utilizzato. Il backslash protegge il contenuto di $() dall'interpretazione immediata della shell, così "date" viene chiamato ogni volta che il prompt viene generato.

Linux viene fornito con molti piccoli programmi di utilità come **date**, **grep**, o **wc** che consentono di manipolare informazioni. Se vi trovate a creare complesse combinazioni di questi programmi all'interno di un prompt, potrebbe essere più semplice fare voi stessi uno shell script e chiamarlo dal prompt. Per assicurare che le variabili della shell siano espanse al momento giusto negli shell script bash sono spesso necessarie delle sequenze di escape (come visto sopra con il comando `date`): questo viene elevato ad un altro livello all'interno della linea del prompt PS1 ed evitare ciò creando degli shell script è una buona idea.

Un esempio di un piccolo shell script usato all'interno di un prompt viene dato a seguire:

```
#!/bin/bash
#     lsbytesum - somma il numero di byte in un elenco di directory
TotalBytes=0
for Bytes in $(ls -l | grep "^-" | cut -c30-41)
do
    let TotalBytes=$TotalBytes+$Bytes
done
TotalMeg=$(echo -e "scale=3 \n$TotalBytes/1048576 \nquit" | bc)
echo -n "$TotalMeg"
```

A volte l'ho usato come una funzione (molto più efficiente - sfortunatamente, spiegare in dettaglio le funzioni va oltre lo scopo di questo documento), altre come uno shell script nella mia directory ~/bin, che è nel mio path. Usata in un prompt:

```
[2158][giles@nikola:~]$ PS1="[\u@\h:\w (\$(lsbytesum) Mb)]\$ "
[giles@nikola:~ (0 Mb)]$ cd /bin
[giles@nikola:/bin (4.498 Mb)]$
```

3.3 Cosa Mettere nel Prompt

Avrete notato che io metto il nome della macchina, l'ora e la directory corrente nella maggioranza dei miei prompt. Ad eccezione dell'ora queste sono cose molto standard da mettere nel prompt, l'ora è l'aggiunta più comune dopo queste. Ma cosa includere nel prompt è interamente una questione di gusto personale. Questi sono esempi da persone che conosco per aiutare a darvi delle idee.

Il prompt di Dan è minimale ma efficace, particolarmente per via del modo in cui funziona.

```
[giles@nikola:~]$ cur_tty=$(tty | sed -e "s/.*tty\(.*\)/\1/")
[giles@nikola:~]$ echo $cur_tty
p4
[giles@nikola:~]$ PS1="\!,$cur_tty,\$?\$ "
1095,p4,0$
```

A Dan non piace avere la directory corrente che può ridimensionare drasticamente il prompt come ci si muove nell'albero delle directory, così ne tiene traccia a mente (o digita "pwd"). Lui ha imparato Unix con csh e tcsh, così usa molto la command history (qualcosa che molti di noi, viziati da bash, non facciamo), così la prima cosa nel prompt è l'history number. La seconda cosa sono i caratteri significativi del tty (l'output di "tty" viene tagliato con sed), una cosa che può essere utile per gli utilizzatori di "screen". La terza cosa è il valore di uscita dell'ultimo comando/pipeline (si noti che questo viene reso inutile da ogni comando eseguito all'interno del prompt - potreste però ottenerlo salvando il valore in una variabile e rimettendolo poi a posto). In fine, lo "\$" è un carattere dollaro per un normale utente e cambia in un cancelletto ("#") se l'utente è root.

Torben Fjerdingstad ha scritto per dirmi che spesso sospende dei job e poi se ne dimentica, così usa il prompt per ricordarsi dei job sospesi:

```
[giles@nikola:~]$ function jobcount {
```

```
> jobs|wc -l| awk '{print $1}'
> }
[giles@nikola:~]$ export PS1='\W[`jobcount`]# '
giles[0]# man ls &
[1] 4150

[1]+  Stopped (tty output)    man ls
giles[1]#
```

Torben usa **awk** per togliere gli spazi vuoti dall'output di wc, mentre io avrei usato sed oppure tr - non perché siano meglio, ma perché mi sono più familiari. Vi sono probabilmente anche altri modi. Torben delimita la stringa PS1 con apostrofi (single quote), questo evita che Bash interpreti immediatamente i backquote, così lui non deve farli precedere da "\" come ho menzionato.

NOTA: C'è un noto bug in Bash 2.02 che fa sì che il comando **jobs** (integrato nella shell) non restituisca nulla ad un pipe. Se provate quanto sopra con Bash 2.02, otterrete sempre "0" indipendentemente da quanti job avete sospesi. Chet Ramey, uno dei manutentori di Bash, mi dice che questo sarà corretto in v2.03.

3.4 Ambiente e Funzioni Bash

Come menzionato prima, PS1, PS2, PS3, PS4 e PROMPT_COMMAND sono tutti salvati nell'ambiente Bash. Per quelli di noi che hanno precedente esperienza con DOS, l'idea di maneggiare grosse porzioni di codice nell'ambiente è terrificante, perchè l'ambiente DOS era piccolo e proprio non cresceva bene. Vi sono probabilmente limiti pratici su cosa può e cosa dovrebbe essere messo nell'ambiente, ma non so quali siano; stiamo probabilmente parlando di un paio di ordini di grandezza in più di quanto gli utenti DOS siano abituati. Come dice Dan:

"Nella mia shell interattiva ho 62 alias e 25 funzioni. La mia regola generale è che se ho bisogno di qualcosa solamente per uso interattivo e può essere scritta in bash ne faccio una funzione (presumendo che non possa essere espressa facilmente con un alias). Se queste persone hanno problemi di memoria non dovrebbero usare bash. Bash è uno dei programmi più grossi che faccio girare nella mia Linux box (a parte Oracle). Lancia top qualche volta e premi 'M' per ordinare per memoria occupata - vedrai quanto bash è vicino alla cima della lista. Caspita, è più grosso di sendmail! Dì loro di prendere qualcosa come ash".

Credo che quel giorno stesse usando solo la consolle: girando X e le applicazioni X, ho molte cose più grosse di Bash. Ma l'idea è la stessa: l'ambiente è qualcosa da utilizzare senza preoccuparsi di riempirlo troppo.

Rischio la censura da parte dei guru di Unix quando dico questo (per crimine di ipersemplificazione): le funzioni sono in pratica piccoli shell script che vengono caricati nell'ambiente per motivi di efficienza. Citando ancora Dan: "Le funzioni shell sono efficienti quanto più possibile. Approssimativamente è l'equivalente di eseguire uno shell script bash/bourne eccetto che nessun I/O di file è necessario perché la funzione è già in memoria. Le funzioni shell sono tipicamente caricate da .bashrc o .bash_profile a seconda che le si voglia solo nella shell iniziale o anche nelle sottoshell. Se confronti questo con l'esecuzione di uno shell script: la shell fa un fork, il processo figlio apre il file e fa un exec, potenzialmente il path viene esaminato, il kernel apre il file e esamina sufficienti byte da determinare come eseguire il file, nel caso di uno shell script una shell deve essere avviata con il nome dello script come argomento, la shell allora apre il file, legge e esegue i comandi. Diversamente, con una funzione shell, tutto fuorché l'esecuzione dei comandi può essere considerato overhead non necessario".

4 Manipolazioni della Barra del Titolo di Xterm

Posso essere usate dele sequenze di escape non stampabili per produrre effetti interessanti nei prompt. Per usare queste sequenze di escape, dovete includerle fra **\[** e **\]**, dicendo a Bash di ignorare questo materiale nel calcolare la dimensione del prompt. Se non si includono questi delimitatori si fa in modo che il cursore appaia nel posto sbagliato perché la sua effettiva dimensione è sconosciuta. Le sequenze di escape devono anche essere precedute da **\033[** prima della versione 2 di Bash oppure da **\033[** o **\e[** in versioni successive.

Se cercate di cambiare la barra del titolo dell'Xterm con il prompt quando siete alla consolle, produrrete spazzatura. Per evitare questo, testate la variabile d'ambiente XTERM per sapere se il prompt si troverà in un Xterm.

```
function proml
{
case $TERM in
    xterm*)
        local TITLEBAR='\[\033]0;\u@\h:\w\007\]'
        ;;
    *)
        local TITLEBAR=''
        ;;
esac

PS1="${TITLEBAR}\
[\$(date +%H%M)]\
[\u@\h:\w]\
\$ "
PS2='> '
PS4='+ '
}
```

Questa è una funzione che può essere incorporata in ~/.bashrc. Il nome della funzione potrebbe quindi essere chiamato per eseguire la funzione. La funzione, come la stringa PS1, viene salvata nell'ambiente. Una volta che la stringa PS1 viene impostata dalla funzione, potete rimuovere la funzione dall'ambiente con `unset proml`. Dal momento che il prompt non può cambiare da quando sta in un Xterm a quando sta alla consolle, la variabile TERM non viene testata ogni volta che il prompt viene generato. Ho usato i marcatori di continuazione (i backslash) nella definizione del prompt, per consentire di scriverlo su più righe. Questo migliora la leggibilità rendendo più facile modificarlo e fare un debug.

Lo definisco come una funzione perché così è come il pacchetto Bashprompt (discusso più avanti in questo documento) tratta i prompt: non è la sola maniera di farlo, ma funziona bene. Via via che i prompt che utilizzate diventano più complessi, diventa sempre più sconveniente digitarli al prompt e più pratico metterli in qualche sorta di file di testo. In questo caso, per testare questo prompt, salvate quendo sopra come un file di testo chiamato "proml". potete lavorare come segue:

```
[giles@nikola:/bin (4.498 Mb)]$ cd     -> Andate dove volete salvare il prompt
[giles@nikola:~ (0 Mb)]$ vi proml      -> Modificate il file del prompt
...                                    -> Inserite il testo dato sopra
[giles@nikola:~ (0 Mb)]$ source proml  -> Leggete la funzione del prompt
[giles@nikola:~ (0 Mb)]$ proml         -> Eseguite la funzione del prompt
```

Il primo passo nel creare questo prompt è di controllare se la shell che stiamo facendo partire è in un xterm o no: se lo è, la variabile della shell (${TITLEBAR}) viene definita. Essa consiste delle sequenze di escape appropriate e **\u@\h:\w**, che mettono <utente>@<macchina>:<directory di lavoro> nella barra del titolo di Xterm. Questo è particolarmente utile con Xterm minimizzati, rendendoli identificabili più rapidamente. Il resto del materiale in questo prompt dovrebbe essere già noto dai precedenti prompt che abbiamo creato.

Il solo inconveniente di manipolare la barra dell'Xterm in questo modo avviene quando vi loggate in un sistema su cui non avete predisposto il trucchetto della barra del titolo: l'Xterm continuerà a mostrare l'informazione del precedente sistema su cui c'era il trucco.

5 Sequenze di escape ANSI: Colori e Movimenti del Cursore

5.1 Colori

Come menzionato prima, i caratteri di escape non stampabili devono essere racchiusi da **\[\033[** e **\]**. Per le sequenze di escape, devono anche essere seguiti da **m** minuscola.

Se provate i prompt seguenti in un xterm e trovate che non vedete i colori menzionati, controllate nel file ˜/.Xdefaults (e possibilmente gli altri con simile funzione) le linee come "XTerm*Foreground: BlanchedAlmond". Questo può essere commentato mettendo un punto esclamativo ("!") davanti. Ovviamente questo dipenderà anche da che emulatore di terminale state usando. Questo è il posto più probabile in cui i colori del vostro terminale possono essere reimpostati.

Per includere del testo blu nel prompt:

```
PS1="\[\033[34m\][\$(date +%H%M)][\u@\h:\w]$ "
```

Il problema con questo prompt è che il colore blu che inizia con il codice-colore 34 non viene mai cambiato nuovamente al normale colore, così il testo digitato dopo il prompt è ancora nel colore del promot. Questa è una tonalità di blu scura, unendola con il codice **bold** (grassetto) potrebbe aiutare:

```
PS1="\[\033[1;34m\][\$(date +%H%M)][\u@\h:\w]$\[\033[0m\] "
```

Il prompt è ora blu chiaro e termina cambiando il colore nuovamente a nulla (quale che fosse il colore di primo piano che avevate prima)

Questo sono i restanti valori dei colori:

```
Nero           0;30     Grigio Scuro  1;30
Blu            0;34     Blu Chiaro    1;34
Verde          0;32     Verde Chiaro  1;32
Ciano          0;36     Ciano Chiaro  1;36
Rosso          0;31     Rosso Chiaro  1;31
Viola          0;35     Viola Chiaro  1;35
Marrone        0;33     Giallo        1;33
Grigio Chiaro  0;37     Bianco        1;37
```

Potete anche impostare i colori dello sfondo usando 44 per uno sfondo blu, 41 per uno sfondo rosso, ecc. Non ci sono colori dello sfondo in grassetto (bold). Possono essere usate delle combinazioni, come testo Rosso Chiaro su uno sfondo Blu: **\[\033[44;1;31m\]**, sebbene impostare i colori separatamente sembra funzionare meglio (cioè **\[\033[44m\]\[\033[1;31m\]**). Gli altri codici disponibili includono 4: Sottolineatura, 5: Lmpeggiante, 7: Negativo, e 8: Nascosto.

In oltre: Molti (incluso me stesso) si oppongono fermamente all'attributo "blink". Fortunatamente, non funziona in ogni emulatore di terminale che conosco - ma funziona alla consolle. E, se vi state chiedendo (come ho fatto io) "A cosa serve un attributo 'Nascosto'?!" - l'ho visto usato in un esempio di uno shell script (non un prompt) per permettere di digitare una password senza visualizzarla sullo schermo.

Basato su un prompt chiamato "elite2" nel pacchetto Bashprompt (che ho modificato per funzionare meglio su una consolle standard, piuttosto che con i font speciali di xterm richiesti per vedere bene l'originale), questo è un prompt che ho usato molto:

```
function elite
{

local GRAY="\[\033[1;30m\]"
local LIGHT_GRAY="\[\033[0;37m\]"
local CYAN="\[\033[0;36m\]"
local LIGHT_CYAN="\[\033[1;36m\]"

case $TERM in
    xterm*)
        local TITLEBAR='\[\033]0;\u@\h:\w\007\]'
        ;;
    *)
        local TITLEBAR=""
        ;;
esac

local GRAD1=$(tty|cut -d/ -f3)
PS1="$TITLEBAR\
$GRAY-$CYAN-$LIGHT_CYAN(\
$CYAN\u$GRAY@$CYAN\h\
$LIGHT_CYAN)$CYAN-$LIGHT_CYAN(\
$CYAN\#$GRAY/$CYAN$GRAD1\
$LIGHT_CYAN)$CYAN-$LIGHT_CYAN(\
$CYAN\$(date +%H%M)$GRAY/$CYAN\$(date +%d-%b-%y)\
$LIGHT_CYAN)$CYAN-$GRAY-\
$LIGHT_GRAY\n\
$GRAY-$CYAN-$LIGHT_CYAN(\
$CYAN\$$GRAY:$CYAN\w\
$LIGHT_CYAN)$CYAN-$GRAY-$LIGHT_GRAY "
PS2="$LIGHT_CYAN-$CYAN-$GRAY-$LIGHT_GRAY "
}
```

Io definisco i colori come variabili temporanee della shell per leggibilità. È più facile lavorarci. La variablie "GRAD1" è un controllo per determinare in che terminale ci si trova, come il controllo per determinare se ci si trova su un Xterm, è sufficiente farlo una sola volta. Il prompt che vedrete ha questo aspetto, ma a colori:

```
--(giles@nikola)-(75/ttyp7)-(1908/12-Oct-98)--
--($:~/tmp)--
```

Per aiutarmi a ricordare quali colori sono disponibili, ho scritto lo script seguente che visualizza tutti i colori sullo schermo:

```
#!/bin/bash
#
#   Questo file stampa molti codici-colore sul terminale per
```

```
#   mostrare quali sono disponibili. Ogni linea e` un colore su sfondo
#   nero o grigio, con il colore nel mezzo. E` verificato che funziona con
#   sfondo bianco, nero e verde (2 dic 98).
#
echo "  On Light Gray:        On Black:"
echo -e "\033[47m\033[1;37m  White       \033[0m\
 1;37m \
\033[40m\033[1;37m  White       \033[0m"
echo -e "\033[47m\033[37m  Light Gray   \033[0m\
   37m \
\033[40m\033[37m  Light Gray   \033[0m"
echo -e "\033[47m\033[1;30m  Gray        \033[0m\
 1;30m \
\033[40m\033[1;30m  Gray        \033[0m"
echo -e "\033[47m\033[30m  Black       \033[0m\
   30m \
\033[40m\033[30m  Black       \033[0m"
echo -e "\033[47m\033[31m  Red         \033[0m\
   31m \
\033[40m\033[31m  Red         \033[0m"
echo -e "\033[47m\033[1;31m  Light Red    \033[0m\
 1;31m \
\033[40m\033[1;31m  Light Red    \033[0m"
echo -e "\033[47m\033[32m  Green       \033[0m\
   32m \
\033[40m\033[32m  Green       \033[0m"
echo -e "\033[47m\033[1;32m  Light Green  \033[0m\
 1;32m \
\033[40m\033[1;32m  Light Green  \033[0m"
echo -e "\033[47m\033[33m  Brown       \033[0m\
   33m \
\033[40m\033[33m  Brown       \033[0m"
echo -e "\033[47m\033[1;33m  Yellow      \033[0m\
 1;33m \
\033[40m\033[1;33m  Yellow      \033[0m"
echo -e "\033[47m\033[34m  Blue        \033[0m\
   34m \
\033[40m\033[34m  Blue        \033[0m"
echo -e "\033[47m\033[1;34m  Light Blue   \033[0m\
 1;34m \
\033[40m\033[1;34m  Light Blue   \033[0m"
echo -e "\033[47m\033[35m  Purple      \033[0m\
   35m \
\033[40m\033[35m  Purple      \033[0m"
echo -e "\033[47m\033[1;35m  Pink        \033[0m\
 1;35m \
\033[40m\033[1;35m  Pink        \033[0m"
echo -e "\033[47m\033[36m  Cyan        \033[0m\
   36m \
\033[40m\033[36m  Cyan        \033[0m"
```

```
echo -e "\033[47m\033[1;36m  Light Cyan   \033[0m\
 1;36m \
\033[40m\033[1;36m  Light Cyan   \033[0m"
```

5.2 Movimenti del Cursore

Le sequenze di escape ANSI permettono di muovere il cursore a piacere sullo schermo. Questo è più utile per interfacce utente a tutto schermo generate da shell script, ma possono essere usate anche nei prompt. Le sequenze di escape di movimento sono le seguenti:

```
    - Posizione del cursore:
      \033[<L>;<C>H
      mette il cursore alla linea L e colonna C.
    - Muove il cursore su N linee:
      \033[<N>A
    - Muove il cursore giu N linee:
      \033[<N>B
    - Muove il cursore avanti N colonne:
      \033[<N>C
    - Muove il cursore indietro N colonne:
      \033[<N>D

    - Salva la posizione del cursore:
      \033[s
    - Ripristina la posizione del cursore:
      \033[u
```

Gli ultimi due codici NON SONO supportati da molti emulatori di terminale. Gli unici che so che lo fanno sono xterm e nxterm - anche se la maggior parte degli emulatori di terminale sono basati sul codice di xterm. Da quanto ho potuto vedere rxvt, kvt, xiterm, e Eterm non li supportano. Sono supportati alla consolle.

Provate a digitare le seguenti linee di codice al prompt (cosa fa è un po' più chiaro se il prompt è alcune linee giù nel terminale quando lo digitate): `echo -en "\033[7A\033[1;35m BASH \033[7B\033[6D"`. Questo dovrebbe muovere il cursore sette linee su per lo schermo, stampare la parola " BASH ", e poi tornare dov'era per produrre un normale prompt. Questo non è un prompt: è solo una dimostrazione di come muovere il cursore sullo schermo, usando il colore per evidenziare quanto viene fatto.

Salvate questo in un file chiamato "clock":

```
#!/bin/bash

function prompt_command {
let prompt_x=$COLUMNS-5
}

PROMPT_COMMAND=prompt_command

function clock {
local       BLUE="\[\033[0;34m\]"
local        RED="\[\033[0;31m\]"
local  LIGHT_RED="\[\033[1;31m\]"
local      WHITE="\[\033[1;37m\]"
```

```
local  NO_COLOUR="\[\033[0m\]"
case $TERM in
    xterm*)
        TITLEBAR='\[\033]0;\u@\h:\w\007\]'
        ;;
    *)
        TITLEBAR=""
        ;;
esac

PS1="${TITLEBAR}\
\[\033[s\033[1;\$(echo -n \${prompt_x})H\]\
$BLUE[$LIGHT_RED\$(date +%H%M)$BLUE]\[\033[u\033[1A\]
$BLUE[$LIGHT_RED\u@\h:\w$BLUE]\
$WHITE\$$NO_COLOUR "
PS2='> '
PS4='+ '
}
```

Questo prompt è piuttosto semplice, ma tiene un orologio nell'angolo in alto a destra del terminale (anche se il terminale viene ridimensionato). Questo NON funzionerà negli emulatori di terminale che, come ho menzionato, non accettano i codici per salvare e ripristiare la posizione del cursore. Se provate ad usare questo prompt in uno di quegli emulatori di terminale, l'orologio apparirà correttamente, ma il prompt verrà intrappolato nella seconda linea del terminale.

Vedi anche 10.5 (Il Prompt Orologio Inutile ed Elegante) per un uso più esteso di questi codici.

5.3 Muovere il Cursore Con tput

Come molte cose in Unix, c'è più di un modo per raggiungere gli stessi fini. Una utility chiamata "tput" può essere usata anche per spostare il cursore sullo schermo, o per ottenere informazioni sullo stato di un terminale. "tput", per posizionare il cursore, è meno flessibile delle sequenze di escape ANSI: si può muovere il cursore solo ad una posizione assoluta, non si può relativamente alla sua posizione corrente. Non uso "tput", quindi non intendo spiegarlo in dettaglio. Digitate "man tput" e ne saprete quanto me.

6 Caratteri Speciali: Sequenze di Escape Ottali

Oltre i caratteri che si digitano sulla tastiera, ci sono molti altri caratteri che possono essere stampati sullo schermo. Ho creato uno script per permettervi di controllare cosa mette a disposizione il font che state usando. Il comando principale che avrete bisogno di usare è per utilizzare questi caratteri è "echo -e". L'opzione "-e" dice ad echo di abilitare l'interpretazione dei caratteri protetti con backslash. Cosa si vede quando guardate 200-400 ottale è molto diverso con un font VGA da quello che si vede con un font Linux standard. Siate avvisati che queste sequenze di escape hanno effetti strani sul terminale, e non ho tentato di prevenire che facciano quello che fanno. I caratteri linedraw e block (ai quali molti di noi sono diventati familiari con Word Perfect) e che sono usati molto dal progetto Bashprompt, sono fra 260 e 337 ottale.

```
#!/bin/bash

#   Script: escgen
```

```
function usage {
   echo -e "\033[1;34mescgen\033[0m <lower_octal_value> [<higher_octal_value>]"
   echo "   Octal escape sequence generator: print all octal escape sequences"
   echo "   between the lower value and the upper value.  If a second value"
   echo "   isn't supplied, print eight characters."
   echo "   1998 - Giles Orr, no warranty."
   exit 1
}

if [ "$#" -eq "0" ]
then
   echo -e "\033[1;31mPlease supply one or two values.\033[0m"
   usage
fi
let lower_val=${1}
if [ "$#" -eq "1" ]
then
   #   If they don't supply a closing value, give them eight characters.
   upper_val=$(echo -e "obase=8 \n ibase=8 \n $lower_val+10 \n quit" | bc)
else
   let upper_val=${2}
fi
if [ "$#" -gt "2" ]
then
   echo -e "\033[1;31mPlease supply two values.\033[0m"
   echo
   usage
fi
if [ "${lower_val}" -gt "${upper_val}" ]
then
   echo -e "\033[1;31m${lower_val} is larger than ${upper_val}."
   echo
   usage
fi
if [ "${upper_val}" -gt "777" ]
   then
   echo -e "\033[1;31mValues cannot exceed 777.\033[0m"
   echo
   usage
fi

let i=$lower_val
let line_count=1
let limit=$upper_val
while [ "$i" -lt "$limit" ]
do
   octal_escape="\\$i"
   echo -en "$i:'$octal_escape' "
   if [ "$line_count" -gt "7" ]
```

```
   then
      echo
      #   Put a hard return in.
      let line_count=0
   fi
   let i=$(echo -e "obase=8 \n ibase=8 \n $i+1 \n quit" | bc)
   let line_count=$line_count+1
done
echo
```

Potete anche usare **xfd** per mostrare tutti i caratteri un un font X, con il comando "xfd -fn <fontname>". Facendo clic su un carattere vengono date molte informazioni circa quel catattere, incluso il suo valore ottale. Lo script dato sopra sarà utile alla consolle e se non siete sicuri del nome del font corrente.

7 Il Pacchetto Bash Prompt

7.1 Disponibilità

Il pacchetto Bash Prompt è disponibile qui *http://bash.current.nu* , ed è il lavoro di molte persone, coordinate da Rob Current (aka BadLandZ). Il pacchetto è in versione beta, ma offre una maniera semplice di usare prompt multipli (o temi), permette di impostare il prompt per shell di login e per sottoshell (cioè mettere stringhe PS1 in (~/.bash_profile e ~/.bashrc). La maggior parte dei temi usano i set di caratteri estesi VGA, quindi hanno un brutto aspetto a meno che siano usati con font VGA (che non sono presenti su molti sistemi).

7.2 Cambiare il Font di Xterm

Per usare alcuni dei prompt più attraenti nel pacchetto Bash Prompt, dovete prendere e installare installare i font che supportano il set di caratteri richiesto dai prompt. Ci si riferise ad essi come "Font VGA", ma non mi è chiara la distinzione fra essi e i font già distribuiti presenti con Linux - sebbene chiaramente supportano set di caratteri differenti. I font per Xterm standard supportano un alfabeto esteso, inclusi molti caratteri accentati. Nei font VGA, questo materiale viene rimpiazzato da caratteri grafici - blocchi, punti, linee. Se qualcuno può spiegare questo più in dettaglio mi mandi un e-mail e includerò qui una spiegazione.

Ottenere e installare questi font è un processo un po' contorto. Prima, recuperate i/il font. Poi, assicuratevi che siano file .pcf o .pcf.gz. Se sono file .bdf, informatevi sul comando "bdftopcf" (cioè leggete la man page). Mettete i file .pcf o .pcf.gz nella directory /usr/X11R6/lib/X11/fonts/misc (questa è la directory corretta per RedHat 5.1 e Slackware 3.4, potrebbe essere differente su altre distribuzioni). Fate "cd" su quella directory ed eseguite il comando "mkfontdir". Quindi eseguite "xset fp rehash". A volte è una buona idea andare nel file fonts.alias nella stessa directory e creare dei nomi alternativi più corti per i font.

Per usare i nuovi font, lanciate l'Xterm che più vi piace con l'opzione appropriata, che può essere trovata nella man page o usando il parametro "–help" alla linea di comando. Alcuni Xterm comuni dovrebbero essere usati come segue:

```
xterm -font <fontname>

OPPURE

xterm -fn <fontname> -fb <fontname-bold>
Eterm -f <fontname>
rxvt -fn <fontname>
```

I font VGA sono disponibili alla *Stumpy's ANSI Fonts* su *http://home.earthlink.net/~us5zahns/enl/ansifont.html* (da cui ho preso in prestito molto nello scrivere questo documento).

8 Caricare un Prompt Differente

8.1 Caricare un Prompt Differente, Più Tardi

Le spiegazione in questo HOWTO hanno mostrato come creare variabili di ambiente PS1, oppure come incorporare quelle stringhe PS1 e PS2 in funzioni che potrebbero essere create da ~/.bashrc o come un tema dal pacchetto bashprompt.

Usando il pachetto bashprompt, si deve digitare `bashprompt -i` per vedere una lista dei prompt disponibili. Per impostare il prompt in shell di login future (principalmente la consolle, ma anche telnet e Xterm, dipende da come sono configurati i vostri Xterm), si deve digitare `bashprompt -l nometema`. bashprompt quindi modifica il vostro ~/.bash_profile per chiamare il tema richiesto alla partenza. Per impostare il prompt in future sottoshell (solitamente Xterm, rxvt, ecc.), si deve digitare `bashprompt -s nometema`, e bashprompt modifica il vostro file ~/.bashrc per chiamare il tema richiesto alla partenza.

8.2 Caricare un Prompt Differente, Immediatamente

Potete cambiare il prompt nel terminale corrente (usando la funzione esempio "elite" di cui sopra) digitando `source elite` seguito da `elite` (assumendo che il file funzione elite sia nella directory corrente). Questo è un po' poco pratico e vi lascia con un'altra funzione (elite) nel vostro spazio ambiente - se volete ripulire l'ambiente, dovreste digitare anche `unset elite`. Questo sembrerebbe un candidato ideale per un piccolo shell script, ma uno script in questo caso non funziona perché lo script non può modificare l'ambiente della shell corrente: può solo cambiare l'abiente della sottoshell in cui gira. Come lo script termina, la sottoshell sparisce e così i cambiamente fatti all'ambiente. Cosa **può** cambiare le variabili d'ambiente della shell corrente sono le funzioni d'ambiente. Il pacchetto bashprompt mette una funzione chiamato "callbashprompt" nell'ambiente e, sebbene non sia documentata, può essere chiamata per caricare al volo qualsiasi tema bashprompt. Guarda nella directory dei temi che ha installato (il tema che chiamate deve essere lì), interpreta la funzione richiesta, carica la funzione e poi elimina la funzione, mantenendo wuindi l'ambiente in ordine. "callbashprompt" non è pensata per essere usata così e non controlla eventuali errori, ma tenendo questo a mente, funziona piuttosto bene.

9 Caricare il Colori del Prompt Dinamicamente

9.1 Un esempio che dimostra il concetto

Questa è una dimostrazione del concetto piuttosto che un prompt attraente: cambiare i colori all'interno di un prompt dinamicamente. In questo esempio, il colore dell'hostname cambia in funzione del carico (come un avvertimento).

```
#!/bin/bash
#   "hostloadcolour" - 17 ottobre 98, Giles
#
#   Qui l'idea e' di cambiare il colore dell'hostname nel prompt, a
#   seconda del valore del carico.

# THRESHOLD_LOAD e' il valore del carico per un minuto (moltiplicato
```

```
# per cento) al quale volete che il prompt cambi da COLOUR_LOW a
# COLOUR_HIGH

THRESHOLD_LOAD=200
COLOUR_LOW='1;34'
          # blu chiaro
COLOUR_HIGH='1;31'
          # rosso chiaro

function prompt_command {
ONE=$(uptime | sed -e "s/.*load average: \(.*\...\), \(.*\...\), \(.*\...\)/\1/" -e "s/ //g")
#   A quanto pare, "scale" in bc non funziona con la moltiplicazione,
#   ma funziona con la divisionone.
ONEHUNDRED=$(echo -e "scale=0 \n $ONE/0.01 \nquit \n" | bc)
if [ $ONEHUNDRED -gt $THRESHOLD_LOAD ]
then
    HOST_COLOUR=$COLOUR_HIGH
        # rosso chiaro
else
    HOST_COLOUR=$COLOUR_LOW
        # blu chiaro
fi
}

function hostloadcolour {

PROMPT_COMMAND=prompt_command
PS1="[$(date +%H%M)][\u@\[\033[\$(echo -n \$HOST_COLOUR)m\]\h\[\033[0;37m\]:\w]$ "
}
```

Usando il vostro editor preferito, salvate queso in un file chiamato "hostloadcolour". Se avete il pacchetto Bashprompt installato, Questo funzionerà come un tema. Se no, digitate `source hostloadcolour` e poi `hostloadcolour`. In entrambi i modi, "prompt_command" diventa una funzione nel vostro ambiente. Se esaminate il codice, noterete che i colori ($COLOUR_HIGH e $COLOUR_LOW) sono impostati usanto solo un codice del colore parziale, cioè "1;34" invece di "\[\033[1;34m\]", che avrei preferito. Non sono stato in grado di farlo funzionare con il codice completo. Siete pregati di farmi sapere se doveste riuscirvi.

10 Prompt di Esempio

10.1 Un Prompt "Leggero"

```
function proml {
local BLUE="\[\033[0;34m\]"
local RED="\[\033[0;31m\]"
local LIGHT_RED="\[\033[1;31m\]"
local WHITE="\[\033[1;37m\]"
local LIGHT_GRAY="\[\033[0;37m\]"
case $TERM in
    xterm*)
        TITLEBAR='\[\033]0;\u@\h:\w\007\]'
        ;;
    *)
```

```
        TITLEBAR=""
        ;;
esac

PS1="${TITLEBAR}\
$BLUE[$RED\$(date +%H%M)$BLUE]\
$BLUE[$LIGHT_RED\u@\h:\w$BLUE]\
$WHITE\$$LIGHT_GRAY "
PS2='> '
PS4='+ '
}
```

10.2 Elite dai Temi Bashprompt

Si noti che questo necessita di un font VGA.

```
# Creato da KrON da windowmaker su IRC
# Cambiato da Spidey 08/06
function elite {
PS1="\[\033[31m\]\332\304\[\033[34m\](\[\033[31m\]\u\[\033[34m\]@\[\033[31m\]\h\
\[\033[34m\])\[\033[31m\]-\[\033[34m\](\[\033[31m\]\$(date +%I:%M%P)\
\[\033[34m\]-:-\[\033[31m\]\$(date +%m)\[\033[34m\033[31m\]/\$(date +%d)\
\[\033[34m\])\[\033[31m\]\304-\[\033[34m]\\371\[\033[31m\]-\371\371\
\[\033[34m\]\372\n\[\033[31m\]\300\304\[\033[34m\](\[\033[31m\]\W\[\033[34m\])\
\[\033[31m\]\304\371\[\033[34m\]\372\[\033[00m\]"
PS2="> "
}
```

10.3 Un Prompt per il "Power User"

Io in realtà uso questo prompt, ma si ottengono notevoli ritardi quando il prompt appare su una macchina monoutente PII-400, quindi non raccomando di usarlo su un P-100 multiutente o altro... Guardatelo per cercare idee, piuttosto che per usarlo in pratica.

```
#!/bin/bash
#----------------------------------------------------------------------
#       POWER USER PROMPT "pprom2"
#----------------------------------------------------------------------
#
#   Creato nell'agosto 98, Ultima Modifica 9 novembre 98 da Giles
#
#   Problema: quando load va giu', dice "1.35down-.08", eliminare il
#   segno negativo


function prompt_command
{
#   Crea la variabile TotalMeg: somma delle dimensioni dei file
#   visibile nella directory corrente
local TotalBytes=0
```

```
for Bytes in $(ls -l | grep "^-" | cut -c30-41)
do
    let TotalBytes=$TotalBytes+$Bytes
done
TotalMeg=$(echo -e "scale=3 \nx=$TotalBytes/1048576\n if (x<1) {print \"0\"} \n print x \nquit" | bc)

#      Questo viene usato per calcolare il differenziale del valore
#      del carico fornito dal comando "uptime". "uptime" fornisce medie di
#      carico per 1, 5 r 15 minuti.
#
local one=$(uptime | sed -e "s/.*load average: \(.*\...\), \(.*\...\), \(.*\...\)/\1/" -e "s/ //g")
local five=$(uptime | sed -e "s/.*load average: \(.*\...\), \(.*\...\), \(.*\...\).*/\2/" -e "s/ //g")
local diff1_5=$(echo -e "scale = scale ($one) \nx=$one - $five\n if (x>0) {print \"up\"} else {print \"down\"}\n print x \nquit \n" | bc)
loaddiff="$(echo -n "${one}${diff1_5}")"

#   Conta file visibili:
let files=$(ls -l | grep "^-" | wc -l | tr -d " ")
let hiddenfiles=$(ls -l -d .* | grep "^-" | wc -l | tr -d " ")
let executables=$(ls -l | grep ^-..x | wc -l | tr -d " ")
let directories=$(ls -l | grep "^d" | wc -l | tr -d " ")
let hiddendirectories=$(ls -l -d .* | grep "^d" | wc -l | tr -d " ")-2
let linktemp=$(ls -l | grep "^l" | wc -l | tr -d " ")
if [ "$linktemp" -eq "0" ]
then
    links=""
else
    links=" ${linktemp}l"
fi
unset linktemp
let devicetemp=$(ls -l | grep "^[bc]" | wc -l | tr -d " ")
if [ "$devicetemp" -eq "0" ]
then
    devices=""
else
    devices=" ${devicetemp}bc"
fi
unset devicetemp

}

PROMPT_COMMAND=prompt_command

function pprom2 {

local        BLUE="\[\033[0;34m\]"
local  LIGHT_GRAY="\[\033[0;37m\]"
local LIGHT_GREEN="\[\033[1;32m\]"
local  LIGHT_BLUE="\[\033[1;34m\]"
local  LIGHT_CYAN="\[\033[1;36m\]"
local      YELLOW="\[\033[1;33m\]"
```

```
local       WHITE="\[\033[1;37m\]"
local         RED="\[\033[0;31m\]"
local   NO_COLOUR="\[\033[0m\]"

case $TERM in
    xterm*)
        TITLEBAR='\[\033]0;\u@\h:\w\007\]'
        ;;
    *)
        TITLEBAR=""
        ;;
esac

PS1="$TITLEBAR\
$BLUE[$RED\$(date +%H%M)$BLUE]\
$BLUE[$RED\u@\h$BLUE]\
$BLUE[\
$LIGHT_GRAY\${files}.\${hiddenfiles}-\
$LIGHT_GREEN\${executables}x \
$LIGHT_GRAY(\${TotalMeg}Mb) \
$LIGHT_BLUE\${directories}.\
\${hiddendirectories}d\
$LIGHT_CYAN\${links}\
$YELLOW\${devices}\
$BLUE]\
$BLUE[${WHITE}\${loaddiff}$BLUE]\
$BLUE[\
$WHITE\$(ps ax | wc -l | sed -e \"s: ::g\")proc\
$BLUE]\
\n\
$BLUE[$RED\$PWD$BLUE]\
$WHITE\$\
\
$NO_COLOUR "
PS2='> '
PS4='+ '
}
```

10.4 Un Prompt Largo Quanto il Terminale

Un amico si è lamentato perché non gli piaceva che il prompt cambiasse continuamente di dimensione perché c'era $PWD all'interno, così ho scritto questo prompt che adatta la sua dimensione all'esatta larghezza dal terminale.

```
#!/bin/bash

#   termwide prompt
#      Giles - creato il 2 novembre 98
#
#   Qui l'idea e' di avere la linea superiore di questo prompt di due
```

```
#   linee sempre della larghezza del terminale. Questo viene fatto
#   calcolando la larghezza degli elementi di testo e riempendo come
#   appropriato o troncando a destra $PWD.
#

function prompt_command {

TERMWIDTH=${COLUMNS}

#   Calcola la larghezza del prompt:

hostnam=$(echo -n $HOSTNAME | sed -e "s/[\.].*//")
#   "whoami" e "pwd" includono un carattere "carriage return" finale
usernam=$(whoami)
let usersize=$(echo -n $usernam | wc -c | tr -d " ")
newPWD="${PWD}"
let pwdsize=$(echo -n ${newPWD} | wc -c | tr -d " ")
#   Aggiunge tutti gli accessori sotto ...
let promptsize=$(echo -n "--(${usernam}@${hostnam})---(${PWD})--" \
                 | wc -c | tr -d " ")
let fillsize=${TERMWIDTH}-${promptsize}
fill=""
while [ "$fillsize" -gt "0" ]
do
   fill="${fill}-"
   let fillsize=${fillsize}-1
done

if [ "$fillsize" -lt "0" ]
then
   let cut=3-${fillsize}
   sedvar=""
   while [ "$cut" -gt "0" ]
   do
      sedvar="${sedvar}."
      let cut=${cut}-1
   done
   newPWD="...$(echo -n $PWD | sed -e "s/\(^${sedvar}\)\(.*\)/\2/")"
fi
}

PROMPT_COMMAND=prompt_command

function termwide {

local GRAY="\[\033[1;30m\]"
local LIGHT_GRAY="\[\033[0;37m\]"
local WHITE="\[\033[1;37m\]"
local NO_COLOUR="\[\033[0m\]"
```

```
local LIGHT_BLUE="\[\033[1;34m\]"
local YELLOW="\[\033[1;33m\]"

case $TERM in
    xterm*)
        TITLEBAR='\[\033]0;\u@\h:\w\007\]'
        ;;
    *)
        TITLEBAR=""
        ;;
esac

PS1="$TITLEBAR\
$YELLOW-$LIGHT_BLUE-(\
$YELLOW\${usernam}$LIGHT_BLUE@$YELLOW\${hostnam}\
${LIGHT_BLUE})-${YELLOW}-\${fill}${LIGHT_BLUE}-(\
$YELLOW\${newPWD}\
$LIGHT_BLUE)-$YELLOW-\
\n\
$YELLOW-$LIGHT_BLUE-(\
$YELLOW\$(date +%H%M)$LIGHT_BLUE:$YELLOW\$(date \"+%a,%d %b %y\")\
$LIGHT_BLUE:$WHITE\$$LIGHT_BLUE)-\
$YELLOW-\
$NO_COLOUR "

PS2="$LIGHT_BLUE-$YELLOW-$YELLOW-$NO_COLOUR "

}
```

10.5 Il Prompt Orologio Inutile ed Elegante

Questo è probabilmente il più attraente (e inutile) prompt che abbia mai creato. Poiché gli emulatori di terminale non implementano il salvataggio e ripristino della posizione del cursore, l'alternativa per mettere un orologio nell'angolo in alto a destra e di ancorare il prompt nella parte inferiore del terminale. Questo deriva dall'idea del prompt ampio quanto il terminale di cui sopra, disegnando una linea nella parte destra dello schermo dal prompt all'orologio. È richiesto un font VGA.

Nota: qui c'è una sostituzione strana, che potrebbe non venir stampata correttamente quando viene convertita da SGML ad altri formati: ho dovuto sostituire il carattere screen con \304 - normalmente avrei introdotto solo la sequenza "\304", ma in questo caso era necessario per fare questa sostituzione.

```
#!/bin/bash

#   Questo prompt richiede i font VGA. Il prompt e` ancorato in basso
#   al terminale, riempie la larghezza del terminale e disegna una linea
#   sul lato destro del terminale per collegarsi all'orologio
#   nell'angolo in alto a destra del terminale.

function prompt_command {
#   Calcola la larghezza del prompt:
```

```
hostnam=$(echo -n $HOSTNAME | sed -e "s/[\.].*//")
#   "whoami" e "pwd" includono il carattere "carriage return" finale
usernam=$(whoami)
newPWD="${PWD}"
#   Aggiunge tutti gli accessori sotto ...
let promptsize=$(echo -n "--(${usernam}@${hostnam})---(${PWD})-----" \
                 | wc -c | tr -d " ")
#   Trova quanto aggiungere fra user@host e PWD (o quanto rimuovere
#   da PWD)
let fillsize=${COLUMNS}-${promptsize}
fill=""
#   Riempie la linea se il prompt non e` largo quanto il terminale:
while [ "$fillsize" -gt "0" ]
do
   fill="${fill}Ä"
   # La A con la dieresi (apparira` come una lunga linea se state
   # usando un font VGA) e` \304, ma l'ho "tagliata" e poi "incollata"
   # perche' Bash farebbe solo una sostituzione - che in questo caso e`
   # mettere $fill nel prompt.
   let fillsize=${fillsize}-1
done
#   Tronca a destra PWD se il prompt e` piu` largo del terminale:
if [ "$fillsize" -lt "0" ]
then
   let cutt=3-${fillsize}
   sedvar=""
   while [ "$cutt" -gt "0" ]
   do
      sedvar="${sedvar}."
      let cutt=${cutt}-1
   done
   newPWD="...$(echo -n $PWD | sed -e "s/\(^${sedvar}\)\(.*\)/\2/")"
fi
#
#   Crea l'orologio e la barra lungo il lato destro del terminale
#
local LIGHT_BLUE="\033[1;34m"
local     YELLOW="\033[1;33m"
#   Posiziona il cursore per stampare l'orologio:
echo -en "\033[2;$((${COLUMNS}-9))H"
echo -en "$LIGHT_BLUE($YELLOW$(date +%H%M)$LIGHT_BLUE)\304$YELLOW\304\304\277"
local i=${LINES}
echo -en "\033[2;${COLUMNS}H"
#   Stampa linee verticali lungo il lato del terminale:
while [ $i -ge 4 ]
do
   echo -en "\033[$(($i-1));${COLUMNS}H\263"
   let i=$i-1
done
```

```
let prompt_line=${LINES}-1
#   Questo e‘ necessario perche’ facendo \${LINES} all’interno di una
#   espressione matematica Bash (come $(())) sembra non funzionare.
}

PROMPT_COMMAND=prompt_command

function clock3 {
local LIGHT_BLUE="\[\033[1;34m\]"
local     YELLOW="\[\033[1;33m\]"
local      WHITE="\[\033[1;37m\]"
local LIGHT_GRAY="\[\033[0;37m\]"
local  NO_COLOUR="\[\033[0m\]"

case $TERM in
    xterm*)
        TITLEBAR='\[\033]0;\u@\h:\w\007\]'
        ;;
    *)
        TITLEBAR=""
        ;;
esac

PS1="$TITLEBAR\
\[\033[\${prompt_line};0H\]
$YELLOW\332$LIGHT_BLUE\304(\
$YELLOW\${usernam}$LIGHT_BLUE@$YELLOW\${hostnam}\
${LIGHT_BLUE})\304${YELLOW}\304\${fill}${LIGHT_BLUE}\304(\
$YELLOW\${newPWD}\
$LIGHT_BLUE)\304$YELLOW\304\304\304\331\
\n\
$YELLOW\300$LIGHT_BLUE\304(\
$YELLOW\$(date \"+%a,%d %b %y\")\
$LIGHT_BLUE:$WHITE\$$LIGHT_BLUE)\304\
$YELLOW\304\
$LIGHT_GRAY "

PS2="$LIGHT_BLUE\304$YELLOW\304$YELLOW\304$NO_COLOUR "

}
```