GNU/Screen

Da Guide@Debianizzati.Org.

Indice

VERSIONI COMPATIBILI
DEBIAN 7 "WHEEZY"
DEBIAN 8 "JESSIE"
DEBIAN TESTING (NON RECENTE)
DEBIAN "SID" (NON RECENTE)
VERIFICALA CON STRETCH

Introduzione

`GNU/Screen` è un multiplatore di terminale che esegue tanti "schermi" separati su un unico terminale, per questa ragione è paragonabile all'uso delle aree di lavoro (workspace) di un ambiente grafico e quindi possiamo considerarlo in un certo qual modo un Window Manager per applicazioni testuali e ncurses.

Risulta dunque lampante la sua utilità in sistemi privi di ambiente grafico, dove non è appunto possibile avviare N finestre di terminale per eseguire N differenti applicativi/comandi.

Installazione

Come sempre il sistema APT di Debian rende tutto molto semplice. Con privilegi di amministrazione è sufficiente:

```
# apt-get install screen
```

Utilizzo

`GNU/Screen` ha due tipi di utilizzo, tramite interfaccia e tramite riga di comando; premesso che visivamente la differenza è minima si può affermare che la prima modalità risulta prevalentemente utile in tutti quei casi dove è richiesta interazione

con l'utente, mentre la seconda dove ciò non è vero, per esempio nell'esecuzione di script o di servizi come Cron. Concretamente screen una volta avviato non fa altro che duplicare la schermata del terminale, quindi da un punto di vista visivo, ad eccezione del messaggio iniziale, non si noterà alcunché di differente rispetto al classico terminale. Ciò nonostante tutti gli eventuali comandi ed operazioni generiche eseguiti apparterranno esclusivamente al suddetto duplicato del terminale, "lasciando dunque libero" quello di base.
Questo fatto in congiunzione con la possibilità di "abbandonare" il duplicato senza che questo comporti la terminazione di tutte le operazioni in corso permette di eseguire comodamente più operazioni in parallelo, basta infatti creare tanti duplicati quanti ne servono.
In ogni momento è sempre possibile ricollegarsi ad una o più sessioni di `GNU/Screen` e riprendere le varie attività come se niente fosse.

Modalità Interfaccia

Per avviare `GNU/Screen` è sufficiente digitare in un emulatore di terminale o in una tty quanto segue:

```
$ screen
```

Superato il noioso messaggio di benvenuto (che rimuoveremo più avanti) ci si ritrova al solito prompt e nulla sembra cambiato, purtroppo il non mostrare da subito una barra di stato è una delle pecche di screen e la sua stessa configurazione è piuttosto ostica.
Nel seguito si spiegherà come abbandonare e ricollegarsi ad una sessione di `GNU/Screen` già esistente, per il momento ci si limita a sottolineare come ripetendo il precedente comando si crei una nuova sessione senza quindi ricollegarsi ad una già esistente. Ne segue immediatamente che utilizzando scorrettamente `GNU/Screen` si rischia di ritrovarsi con un certo numero di sessioni abbandonate.

NOTA
È possibile creare **m** sessioni di `GNU/Screen` e per ciascuna di esse **n** schermi

Visualizzazione finestre

`GNU/Screen` usa un attivatore per tutte le altre funzioni, ovvero una prima sequenza di tasti che attendono una seconda, questo attivatore è di default `Ctrl-a`, per cui dire `Ctrl-a :` significa premere appunto Ctrl-a, rilasciare e quindi premere i due punti.

Innanzitutto rendiamoci la vita facile con la fantomatica statusbar, all'interno di screen digitate:

```
Ctrl-a :
```

Il cursore si sposterà al fondo dove appariranno i due punti, questo è il prompt dei comandi, per intenderci è simile alla modalità comando di Vim; da qui possiamo

impartire vari comandi che al momento non abbiamo nel file di configurazione o, in altri casi, che vogliamo solo per la sessione in corso.

A questo punto, subito dopo i due punti, scriviamo:

```
hardstatus alwayslastline
```

e premiamo Invio.
Ora abbiamo una barra di stato, configuriamola in modo da avere un riscontro visivo delle finestre, questa volta dal prompt Bash digitiamo:

```
$ screen -X hardstatus string "$(man screen | grep %-L)"
```

Invece del prompt di screen abbiamo usato la riga di comando, sia per mostrarvi un'altra caratteristica sia per la sua versatilità; questi due sistemi non sempre sono uno l'alternativa dell'altro, l'opzione **-X** è particolarmente usata in script o per inviare comandi più complessi, è inoltre utilizzabile dall'esterno della sessione per inviare comandi remoti.

Abbiamo perciò inviato alla sessione il comando "hardstatus string" che richiede un argomento, e per comodità questo argomento è stato pescato direttamente dalla manpage, la stringa si presenta così:

```
%-Lw%{= BW}%50>%n%f* %t%{-}%+Lw%<
```

Scorciatoie e messaggi

Ed ecco che la nostra statusbar, non particolarmente bella ma utile, si presenta come **0-$ bash**, un rettangolino bianco su campo blu, mentre per il colore dell'intera barra viene usato il reverse (il contrario del colore di background e foreground usato dal terminale). La numerazione delle finestre parte da zero, ok è strano e con poco senso ma tant'è, ora apriamo un'altra finestra con un'altra istanza di bash:

```
Ctrl-a c
```

`Ctrl-a` è l'attivatore e **c** la scorciatoia per il comando `screen`, perché all'interno della sessione, *screen* apre una finestra che di default prende il nome dalla variabile d'ambiente SHELL.
Ora abbiamo due voci nella barra, la porzione blu indica la finestra attualmente in focus e anche il suo aspetto "testuale" è diverso:

```
0$ bash  1-$* bash
```

Per passare da una finestra all'altra ci sono diversi metodi:

- finestra specificata da [numero]: **Ctrl-a [numero]**
- finestra successiva: **Ctrl-a n** (oppure Ctrl-a space)
- finestra precedente: **Ctrl-a p** (oppure Ctrl-a backspace)
- finestra visualizzata in precedenza: **Ctrl-a Ctrl-a**
- lista selezionabile di finestre: **Ctrl-a "**

I comandi di uso comune hanno quasi tutti una scorciatoia ed è decisamente più comodo ma tutti sono comunque utilizzabili tramite il prompt, la manpage elenca tutte le scorciatoie predefinite e ogni singolo comando; leggetevelo con calma e, come vedremo più avanti, createvi nuove scorciatoie, anche sostituendo quelle predefinite che non ritenete utili.
In questa guida ne verranno illustrate solo alcune con qualche consiglio ed esempio su come migliorare l'usabilità; alcuni comandi hanno due combinazioni per cui è piuttosto facile eliminarne una e associarla ad altro, fate attenzione perché sono tutte case-sensitive.

Una breve legenda con le eventuali combinazioni create e quindi presenti nel file di configurazione si ottiene con:

```
Ctrl-a ?
```

Ogni finestra è rinominabile a piacimento con:

```
Ctrl-a A
```

Al prompt dei comandi apparirà la scritta "Set window's title to: bash", basta cancellare il nome attuale e scrivere il desiderato.

E se vogliamo rinumerarla:

```
Ctrl-a :number
```

seguito da uno spazio e il nuovo numero.

Per chiudere una singola finestra è possibile digitare:

```
$ exit
```

oppure **solo** in caso di problemi quali congelamento dello schermo o altro è possibile forzarne la chiusura (e uccidere il processo) con:

```
Ctrl-a k
```

Al prompt dei comandi apparirà il messaggio "Really kill this window [y/n]".

Visualizza data, hostname e carico del sistema (load average, quello solitamente mostrato dal comando uptime):

```
Ctrl-a t
```

Il messaggio apparirà in basso, sopra la statusbar, per alcuni secondi; per visualizzare l'ultimo messaggio di attività, qualunque esso sia, si usa:

```
Ctrl-a m
```

I messaggi di attività sono avvisi inviati da altre finestre in varie occasioni, se ad esempio abbiamo aptitude che lavora nella finestra zero e nel frattempo passiamo alla finestra uno per fare altro, quando aptitude finirà screen ci avvisa con "Activity in window %n", dove *%n* è sostituito dal numero della finestra.
Il tempo di visualizzazione è impostato a 5 secondi ed è configurabile, possono sembrare pochi ma si scoprirà che talvolta sono anche troppi in quanto durante tale periodo il processo contenuto nella finestra corrente viene in un certo modo bloccato.

Chiusura

"Non mi piace, è complicato, mi sono impantanato e voglio uscire da questo casino!", ovvero come chiudere screen.
La risposta è che è impossibile chiudere screen, non è stato concepito per essere terminato, perché usarlo altrimenti? ;)
Ma se proprio non potete farne a meno:

```
Ctrl-a C-\
```

Questa è nuova, che diamine è quel "C-\"? Come dicevo prima ci sono alcuni comandi che hanno una doppia scorciatoia, solitamente per renderla più difficile da digitare per errore, in questo caso significa premere `Ctrl-a Ctrl-\` ma se volete davvero evitare brutte sorprese consiglio vivamente di eliminarla e usare il prompt quando occorre:

```
Ctrl-a :quit
```

Chiudere screen equivale ovviamente a chiudere tutte le finestre e terminare i processi della sessione, anche qui è consigliabile terminare i processi in modo pulito, almeno quelli importanti o che devono scrivere qualcosa su disco. In alternativa, terminato l'ultimo processo e chiusa l'ultima finestra terminerà anche screen e al prompt apparirà "[screen is terminating]".

Detach e sessioni

Come anticipato inizialmente GNU/Screen deve restare attivo e insieme a lui tutto ciò che contiene anche nei seguenti casi:

- logout (locale o via SSH) dell'utente;
- riavvio di X o chiusura del terminale (se si dispone di un ambiente grafico);
- altre operazioni esotiche eccettuato il riavvio della macchina.

Tutto ciò è reso possibile grazie alla funzione di *detach*, ovvero la possibilità di "staccare" GNU/Screen da quello che è di fatto il suo contenitore, Xterm o tty che sia:

```
Ctrl-a d
```

Per ricollegarsi ("riattaccarsi") alla specifica finestra di una certa sessione da cui ci si è precedentemente staccati è sufficiente digitare:

```
$ screen -r nome_sessione -p nome_finestra
```

dove `nome_sessione` è appunto il nome della nostra sessione e `nome_finestra` quello della finestra desiderata (al posto del nome è anche possibile utilizzare il numero della finestra, ricordando a tal proposito che la numerazione parte dal valore 0). Qualora sia presente un'unica sessione ed un'unica finestra è possibile omettere sia `nome_sessione` sia `-p nome_finestra`. In caso contrario l'omissione di tali parametri innesca la stampa a video delle sessioni disponibili, per esempio:

```
$ screen -r
There are several suitable screens on:
        4322.pts-17.jackinthebox        (15/10/2011 10:06:17)   (Detached)
        4088.pts-15.jackinthebox        (15/10/2011 10:00:17)   (Detached)
        4656.tty        (08/10/2011 01:00:48)   (Attached)
        1981.fugu       (07/10/2011 20:45:07)   (Attached)
Type "screen [-d] -r [pid.]tty.host" to resume one of them.
```

I primi due nomi di sessioni sono stati generati automaticamente, infatti i nomi predefiniti in GNU/Screen hanno la seguente struttura:

```
numero.pts-numero.hostname
```

L'ultimo nome è frutto invece della scelta dell'utente, almeno per quanto riguarda la parte successiva al punto.
Si noti che l'omissione del parametro `-p`, ovvero del nome finestra, viene interpretato da GNU/Screen come una richiesta di collegarsi all'ultima finestra utilizzata.
Supponendo dunque di voler richiamare la prima sessione è possibile digitare uno tra i seguenti tre comandi:

```
$ screen -r 4322.pts-17.jackinthebox
$ screen -r 4322
$ screen -r pts-17
```

Similmente per l'ultima sessione sarà possibile ricollegarvisi digitando:

```
$ screen -r fugu
```

Per personalizzarne il nome di una sessione direttamente in fase d'avvio è sufficiente digitare:

```
$ screen -S fugu
```

Se invece volessimo attaccare in un altro terminale o tty una sessione già attaccata useremo:

```
$ screen -x
```

Questo può essere utile se ci si collega da un'altra macchina o si vuole condividere la sessione con un altro utente, sebbene per quest'ultima ipotesi ci siano dei comandi appositi.

ATTENZIONE
Prestare cautela nell'eseguire all'interno di sessioni screen l'utilizzo dell'opzione "-x".

La variabile d'ambiente **STY** permette di verificare se Screen è in esecuzione. Se ad esempio lo si volesse avviare al login è possibile inserire nel file `~/.bashrc` quanto segue:

```
if [ -z "$STY" ]; then
   screen -R
fi
```

Che significa: se già esiste una sessione attaccala, altrimenti creane una.

Escape

Utilizzando `Ctrl-a` per screen non sarà più possibile usare tale combinazione in Bash per andare ad inizio riga (beginning-of-line) o in Vim o altrove; in realtà non è usabile direttamente ma con:

```
Ctrl-a a
```

Il che, a seconda delle esigenze potrebbe essere scomodo, per cui possiamo eventualmente modificare l'attivatore ad esempio con un tasto nelle vicinanze come **q** oppure **s** che sono associati al flow control e non ci interessa assolutamente avere, a patto che non interferisca con altre applicazioni. A scelta si può avviare screen con tale opzione:

```
$ screen -e ^Ss
```

o inserire **`escape ^Ss`** nel file di configurazione.

Split

Una delle caratteristiche principali di screen è la possibilità di dividere la finestra in più porzioni proprio come un Window Manager tiling o gli split di Vim, queste prendono il nome di regioni.
Per dividere orizzontalmente si usa:

```
Ctrl-a S
```

Mentre per lo split verticale:

```
Ctrl-a |
```

A questo punto la regione in focus è trattata come una finestra, perché questo è lo scopo degli split: avere più finestre a vista contemporaneamente. Le scorciatoie per cambiare finestra sono le stesse ma valgono solo per quella regione, per muoversi

invece tra una regione e l'altra useremo la combinazione:

```
Ctrl-a TAB
```

Di default abbiamo solo questa e le regioni vengono selezionate in senso orario ma possiamo associare altri comandi per muoverci in senso antiorario e spostarsi alla regione in alto o in basso; ma è forse necessaria qualche piccola considerazione. screen è un programma con una certa età e il suo sviluppo è piuttosto stagnante, nonostante sia abbastanza attivo su Git non esce una nuova versione da anni. La divisione orizzontale è di serie solo su Debian e derivate, per tutti gli altri è necessario applicare una patch o compilarselo da Git, tutto sommato è una feature piuttosto recente e porta alla conseguenza che alcuni comandi non sono stati adattati allo scopo. Ad esempio non ci si può spostare a destra e sinistra (non facilmente almeno) perché esistono solo i comandi *up* e *down*.
Da parecchio ormai esiste questo tipico confronto tra applicazioni simili per scoprire quale sia meglio o peggio, la verità è che una non sostituisce l'altra o se lo fa dipende dalle esigenze del singolo; in particolare sto parlando di "screen vs tmux" e "irssi vs weechat". Visto che sono programmi che necessitano di impegno nel capirne il funzionamento, ciò che salta subito all'occhio in tmux e weechat è la ricchezza di feature abilitate in modo predefinito, questo non è necessariamente un pregio ma nemmeno un difetto, provateli entrambi e valutate.

Notare che lo split prende il focus ma non crea o sceglie alcuna finestra, solo lo spazio dove contenere qualcosa.

In caso volessimo rimuovere la regione in focus useremo:

```
Ctrl-a X
```

Se invece decidiamo di rimuoverle tutte tranne quella in focus:

```
Ctrl-a Q
```

La rimozione di una regione non implica eliminare la finestra o chiudere il programma ma solamente eliminare lo split.

Ogni regione è ridimensionabile in base all'orientamento dello split:

```
Ctrl-a :resize valore
```

dove "valore" è la quantità di righe o colonne desiderata o una percentuale dello spazio totale.

- incrementa regione in focus: **+valore**
- diminuisce regione in focus: **-valore**
- imposta tutte le finestre alla stessa dimensione: **=**
- massimizza regione in focus: **max**
- minimizza regione in focus: **min**

Quando facciamo il detach della sessione e la riattacchiamo, gli split spariscono, questo salverà la disposizione delle regioni per la sessione corrente:

```
Ctrl-a :layout save nome_a_piacere
```

Una volta salvato il layout il ripristino è automatico ma questo è comunque il comando per ripristinarlo:

```
Ctrl-a :layout attach nome_usato_per_salvare
```

Possiamo avere e salvare più layout ed elencarli con:

```
Ctrl-a :layout show
```

Eventualmente rinominarli (senza argomento mostra il nome di quello in uso):

```
Ctrl-a :layout title nome_usato_per_salvare
```

E naturalmente rimuoverlo:

```
Ctrl-a :layout remove nome_usato_per_salvare
```

La versione di Debian è compilata da Git e ha varie opzioni non solitamente presenti, a questa manca ancora `layout dump` che permette di salvare la disposizione su file di configurazione e renderlo disponibile anche dopo la chiusura della sessione. Inoltre non sono comandi documentati, per cui date un'occhiata ai sorgenti (http://git.savannah.gnu.org/cgit/screen.git/tree/src/doc/screen.1#n2262) .

Copia e incolla

La gestione del testo è una delle caratteristiche più importanti e potenti, sebbene sia usabile in un emulatore di terminale e in X, ricordiamoci che screen nasce come applicazione mouseless ed è quindi provvista di tutto (o quasi) ciò che serve per renderla funzionale senza click.

Per copiare del testo si entra in modalità copia:

```
Ctrl-a [
```

Quindi ci si muove con le frecce o meglio ancora con i tasti tipici di Vim; vi rimando alla manpage per il comando **copy** che elenca e spiega il tutto e riporto solo qualche piccolo esempio.

Supponiamo si voglia copiare la frase precedente. Attiviamo la modalità copia quindi scorriamo nel testo con **h, j, k, l** spostando il cursore sulla **Q** di "Quindi" e premiamo il tasto **Space**; ora sempre con i tasti di movimento raggiungiamo la fine della frase (**w, b** scorrono parola per parola e velocizzano l'operazione), possiamo vedere che il testo viene selezionato. Una volta che il cursore raggiungerà **o.** di "esempio" premiamo di nuovo **Space** che, come è intuibile, fa da marcatore per inizio e fine; sulla statusbar apparirà il messaggio "Copied 195 characters into buffer". Ora abbiamo la nostra frase memorizzata nella selezione, premiamo **Esc** o altro tasto (qualsiasi altro tasto che non faccia parte di quelli elencati per le operazioni) per

uscire dalla modalità copia, ci spostiamo su un'altra finestra, che sia un prompt Bash o un editor, e incolliamo con:

```
Ctrl-a ]
```

È un po' macchinoso ma davvero potente, in particolare quando si deve copiare una quantità corposa di testo.
Possiamo salvare la nostra selezione su file senza dover aprire prima un editor.
Ripetiamo il procedimento di prima per la copia ma invece di spostarsi altrove e incollare usiamo questa combinazione:

```
Ctrl-a >
```

Apparirà il messaggio "Copybuffer written to "/tmp/screen-exchange"" e in tale file avremo la nostra frase.
Il percorso del file `/tmp/screen-exchange` è modificabile tramite il comando **`bufferfile`**, sia per salvare la selezione altrove sia per incollare da altra fonte; ora vediamo infatti come sfruttare questo file di scambio o attingere da uno differente.
Apriamo Vim (o altro editor) e mettiamolo in modalità inserimento, ora facciamo leggere il file a screen con:

```
Ctrl-a <
```

Apparirà il messaggio "Slurped 195 characters into buffer". Quindi incolliamo con **`Ctrl-a ]`** ottenendo il testo contenuto in `/tmp/screen-exchange`.
Adesso vogliamo invece incollare nell'editor il contenuto di un file differente, impostiamo quindi il percorso come argomento del comando **`bufferfile`**, lo segnaliamo a screen e incolliamo:

```
C-a :bufferfile /etc/screenrc
C-a < C-a ]
C-a :bufferfile
```

L'ultima riga è **davvero importante** perché riattiviamo il file predefinito per lo scambio `/tmp/screen-exchange`, altrimenti utilizzando successivamente `Ctrl-a >` verrà usato `/etc/screenrc` e, sebbene non avremo i permessi per scriverci, di certo non lo vogliamo; se il file fosse un altro e ne avessimo accesso in scrittura, questo verrebbe sovrascritto. Usate questo sistema con cautela.

Oltre a questo possiamo fare un dump di ciò che è contenuto nello schermo (la parte visibile del buffer) senza bisogno di scroll:

```
Ctrl-a h
```

Apparirà il messaggio "Screen image written to hardcopy.n"", dove **n** sarà il numero della finestra; il file è creato in PWD, il path da dove è stato avviato screen. Se l'opzione **hardcopy_append on** non è specificata, il file viene sovrascritto ad ogni utilizzo.

Modalità riga di Comando

Come inizialmente anticipato è possibile sia inviare comandi sia eseguire applicativi in ciascuno dei vari schermi di GNU/Screen precedentemente creati senza per questo essere obbligati a riportare in primo piano ciascuno di essi. Si noti che terminata l'esecuzione del comando ci si ritroverà ancora davanti al proprio terminale "base" oppure in una finestra GNU/Screen appartenente ad una differente sessione.

Creare una sessione di nome *sessione1*

```
$ screen -m -d -S sessione1
```

Selezionare una finestra (accetta sia il numero sia il nome):

```
$ screen -X select 1
```

Rinumerare finestra (-p accetta sia il numero sia il nome):

```
$ screen -p 9 -X number 11
```

Rinominare finestra:

```
$ screen -p 9 -X title zut
```

Eseguire un comando generico all'interno di una finestra usando il comando **stuff**, il quale concretamente invia una stringa come input. Esempi:

Elencare il contenuto di una cartella tramite **ls** (^M si ottiene premendo ctrl-v e subito dopo Invio o Return che dir si voglia):

```
$ screen -S nome_sessione -p nome_finestra -X stuff 'ls ^M'
```

(si veda il paragrafo *Detach e Sessioni* per il significato dei parametri nome_sessione e nome_finestra). Alternativamente (utile per situazioni in cui *^M* non è applicabile):

```
$ screen -S nome_sessione -p nome_finestra -X stuff $'ls\n'
```

Nel caso in cui GNU/Screen debba essere richiamato all'interno di script o per l'utilizzo in congiunzione con CRON è necessario usare eval; per esempio:

```
$ screen -X eval 'stuff comando "parametri \012"'
$ screen -X eval 'stuff ./mio_script.sh "parametri_script \012"'
```

Eseguire un comando generico all'interno di più finestre usando il comando **at**. Esempi.

Elencare il contenuto di una directory tramite **ls** in tutte le finestre chiamate bash:

```
$ screen -X at bash# stuff 'ls ^M'
```

Elencare il contenuto di una directory tramite **ls** in tutte le finestre chiamate `bash` e alla finestra `l10n`:

```
$ screen -X eval 'at bash# stuff "ls ^M"' 'at l10n# stuff "ls ^M"'
```

Elencare il contenuto di una directory tramite il comando **ls** nelle finestre zero e nove:

```
$ screen -X eval 'at 0# stuff "ls ^M"' 'at 9# stuff "ls ^M"'
```

Elencare il contenuto di una directory tramite **ls** in tutte le finestre:

```
$ screen -X at \# stuff 'ls ^M'
```

Possiamo inviare una sequenza ad un programma contenuto in una finestra (^Q si ottiene premendo `ctrl-v` e `ctrl-q`):

```
$ screen -p 7 -X stuff ^Q
```

nella finestra 7 ho solitamente `rtorrent`, gli verrà quindi inviato `Ctrl-q` che chiuderà il programma in modo pulito, utilizzabile eventualmente in coppia con il comando `sleep` per programmarne la chiusura.

Configurazione

Se l'utente lo desidera è possibile personalizzare `GNU/Screen` tramite il file di configurazione `~/.screenrc` o altro percorso se specificato con l'opzione **-c**.
Di seguito se ne riporta uno d'esempio con i relativi commenti, pensato per sfruttare combinazioni esistenti e non al fine di rendere piacevole e veloce l'uso quotidiano.

```
# utilizzando alcune applicazioni, come ad esempio Vim, otterremo nuovamente la visualizzazione di ciò che c
# come accade normalmente in un emulatore di terminale, senza questa opzione si tornerà al prompt ma Vim o c
# rimarrà nello schermo, come invece accade normalmente in una tty
 altscreen on

# imposto la variabile d'ambiente TERM, necessario se si usa un terminale con 256 colori, se non specificato
# uguale a ''screen'' e si avranno problemi vari; d'altro canto se usato in tty causerà altri problemi, in c
# uso un file di configurazione apposito
 term screen-256color

# disabilito il flow control
# nella maggior parte delle occasioni l'uso del prefisso ''def'' applica un certo comando in modo globale
# controllare la manpage per i dettagli
 defflow off

# gestisce i login degli utenti in ogni pseudo terminale attraverso il file /run/utmp
 deflogin on

# abilito il monitoraggio di attività delle finestre
# (i messaggi di attività spiegati al capitolo 3.1.2: Scorciatoie e messaggi)
 defmonitor on

# la quantità di righe salvate nel buffer
 defscrollback 7777

# UTF-8 everywhere!
 defutf8 on
```

```
# normalmente quando un processo si blocca, screen attende che questo resusciti e nel frattempo non accetta
# pensate ad una connessione SSH che cade e il terminale non accetta più comandi, è decisamente noioso; ques
# sì che la tal finestra venga considerata bloccata dopo un secondo e si possa continuare ad usare la sessio
 defnonblock on

# il tempo di visualizzazione dei messaggi di attività è impostato a un secondo
 msgwait 1

# disabilito il noioso messaggio di benvenuto in apertura
 startup_message off

# disabilito quegli orribili flash in reverse color della visual bell
 vbell off

# quando si usa "Ctrl-a h" appende nuovo testo al file hardcopy.n anziché sovrascriverlo
 hardcopy_append on

# salvo la disposizione degli split per la sessione
 layout save fugu

# se si utilizza un emulatore di terminale con falsa trasparenza, la applica alle finestre e ai programmi co
# (opportunatamente configurati)
 #defbce on

# e di conseguenza la variabile d'ambiente TERM adatta
 #term screen-256color-bce

# blocco l'accesso a screen dopo 15 minuti di inattività, equivale a "Ctrl-a x"
# l'unico output visibile sarà
# Screen used by skizzHG <skizzhg> on jackinthebox.
# Password:
 idle 900 lockscreen

# messaggio di attività personalizzato
 activity "someone is squeaking in  #%n %t"

# voglio che i colori del testo nel prompt, i messaggi di attività, la selezione del testo in copy mode e la
# verticale dello split (quest'ultima solo per quanto riguarda il background, quindi nero) siano blu su camp
 sorendition Bk

# l'aspetto delle barre che appaiono con uno o più split: nera, con testo blu per la finestra in focus e bia
# per quella non in focus, testo centrato
 caption splitonly "%{= kw}%47=%?%F%{= kB}%?%t"

# l'aspetto della statusbar: nera con testo blu per la finestra in focus, racchiuso tra parentesi quadre ver
# e foreground del terminale per le altre, ora e data
# spiegarlo è troppo complicato, la sezione STRING ESCAPES del manuale spiega tutto
# questo non significa sia comprensibile o semplice ma ci sono un sacco di esempi in rete dai quali prendere
 hardstatus alwayslastline "%{= kB}%{= 9}%?%-w%?%{G}[%{B}%n %t%?(%u)%?%{G}]%{d}%+w%?%?%= %{g}%c %D %d-%m-%y"

# le scorciatoie che non hanno un argomento vengono eliminate, disfiamoci di roba fastidiosa che potrebbe
# chiudere, staccare, sospendere o uccidere per sbaglio
 bind \\
 bind d
 bind k
 bind K
 bind ^k
 bind z
 bind ^z

# voglio il focus delle regioni in senso antiorario
 bind ^I focus up

# uso i tasti solitamente associati al flow control per muovermi tra le regioni
 bind q focus top
 bind s focus bottom
# avendo per esempio tre regioni, una grossa in basso e due affiancate in alto, con queste tre combinazioni
# posso muovermi in due sensi; con "Ctrl-a TAB" vado in quello in alto a destra, mentre con
# "Ctrl-a q" e "Ctrl-a s" mi sposto tra quello in alto a sinistra e quello in basso

# ridimensionamento delle regioni
 bind + resize +3
 bind - resize -3
 bind = resize =
```

```
bind * resize max
bind / resize 79%

# oltre ad appendere al file, come specificato prima "hardcopy_append on", voglio salvare in un file diversc
# con '''-h''' salvo l'intero buffer invece del solo schermo
# c'è da notare che screen calcola come buffer l'intero ammontare di righe definite con "defscrollback"
# per cui se questo non è pieno si avrà un tot di spazio vuoto all'inizio
bind h hardcopy -h $HOME/Desktop/screenbuffa

# se si usa "Ctrl-a numero" per spostarsi tra le finestre, di default ce ne sono solo dieci (da 0 a 9)
# in questo modo creo un attivatore aggiuntivo, il punto, e aggiungo altre dieci finestre
# quindi con "Ctrl-a . 1" andrò alla finestra 11
bind -c xwin 0 select 10
bind -c xwin 1 select 11
bind -c xwin 2 select 12
bind -c xwin 3 select 13
bind -c xwin 4 select 14
bind -c xwin 5 select 15
bind -c xwin 6 select 16
bind -c xwin 7 select 17
bind -c xwin 8 select 18
bind -c xwin 9 select 19
bind . command -c xwin

# i programmi che uso più spesso sono associati a comode scorciatoie
# la sintassi è la stessa usabile al prompt bash, con ''screen'' dico di aprire una finestra contenente una
# il numero indica che voglio aprirla con quel dato ordine (questione di muscle memory :)
# ''-t'' e la parola che segue assegnano un titolo alla finestra, dove non c'è viene usato il nome del proce
bind E screen 4 mutt -F $HOME/.config/mailman/mutt/muttrc
bind H screen htop
bind I screen 3 irssi
bind M screen -t muzik 6 mplayer -slave -input file=$HOME/.mplayer/muzikfifo -shuffle -playlist $HOME/allmu
bind T screen -t rTorr 7 rtorrent
bind V screen 5 vim
bind W screen w3m https://www.google.com

# automatizzo certe operazioni all'apertura di screen, consideratelo come il file di startup di un DE/WM
# apro due shell e il file manager
screen 0 bash
screen 1 bash
screen 2 mc -x
# divido lo schermo in due regioni, quella superiore prende il focus e in essa ci apro un'altra shell
split
screen -t l10n 9
# mi sposto nella regione in basso, la ridimensiono e in essa vi seleziono la finestra zero
focus bottom
resize 79%
select 0
```

Dopo aver fatto modifiche al file di configurazione possiamo ricaricarlo senza chiudere la sessione:

```
Ctrl-a :source .screenrc
```

Risorse

- http://www.softpanorama.org/Utilities/Screen/screenrc_examples.shtml
- http://aperiodic.net/screen/

Guida scritta da: skizzhg ven 14 ott 2011, 19.34.46, CEST

Estesa da:

Verificata da:

Debianized
20%

Verificare ed estendere la guida | Cos'è una guida Debianized

Estratto da "http://guide.debianizzati.org/index.php/GNU/Screen"
Categorie: Shell | Window Manager