**L'abc di bc**

Una delle cose più favolose di una distro Gnu/Linux è quella di poter fare qualsiasi operazione dal terminale. Anche i nostri conti!  
 Certo, esistono tanti programmi grafici preposti allo scopo, tuttavia non posso esimermi, qualora non lo conosciate già, dal presentarvi il comando bc! Anche perchè, una volta scoperto, dubito riusciate a farne a meno. Cos'è bc? Bc è una calcolatrice testuale, a riga di comando! Anzi, qualcosa di più! Ma andiamo per gradi. E' già installato, perciò non dobbiamo fare altro che aprire una finestra del terminale e digitare:   
  
$ **bc**   
  
Verrà fuori qualcosa del genere:   
  
Copyright 1991-1994, 1997, 1998, 2000, 2004, 2006 Free Software Foundation, Inc.   
This is free software with ABSOLUTELY NO WARRANTY.   
For details type `warranty'.   
  
E il nostro terminale si trasforma in una potente calcolatrice!!! Il prompt dei comandi è fermo, bc è pronto a ricevere le nostre istruzioni! Iniziamo a fare qualche operazione semplice semplice!   
  
Bisogna fare un po' di attenzione agli operatori aritmetici, che sono un po' diversi dal solito; almeno alcuni! Ad esempio, per la moltiplicazione useremo “**\***”, per la divisione “**/**”, per le potenze “**^**”; mentre per il modulo “**%**”(operazione che ci permette di ricavare il resto della divisione).   
  
  
Per le radici quadrate la formula è la seguente: **sqrt (x)**. Basta sostituire la x con un numero e premere invio! Ad esempio, **sqrt (9)** ci darà come risultato 3.   
  
Andiamo oltre! Finora abbiamo eseguito bc da solo, senza alcuna opzione. E poi ci siamo limitati a dire quasi delle ovvietà. Forse.   
  
Essì, bc è un interprete di linguaggio aritmetico! Certo, non osiamo tanto, nel senso che il nostro intento non è quello di programmare con bc, ma qualcosa di sfizioso la dobbiamo pur dire. Bc lo merita.    
  
**La variabile scale.**   
La finestra del terminale è aperta; abbiamo già digitato bc; è comparso il messaggio di benvenuto; abbiamo eseguito già le nostre prime operazioni. Eppure sembra che qualcosa non funzioni! Proviamo a far compiere a bc la seguente divisione: **3/2**. La shell ci restituirà il seguente risultato: 1   
  
Ma non è possibile! Tre diviso due non fa uno, lo sappiamo benissimo. Cosa sarà successo?   
  
Ma no, non è bc che funziona male! Siamo noi che lo dobbiamo impostare bene per farlo funzionare! Bc di default non si occupa di numeri a virgola mobile! Anche se questo non è del tutto vero. Dobbiamo fare una piccola precisazione!  
  
 Se eseguiamo la seguente moltiplicazione: 3.89\*4.33 possiamo constatare che della virgola, bc, se ne occupa, eccome!  
  
 Il discorso, allora, è che se la virgola è presente, come nella nostra moltiplicazione, bc ne tiene conto, altrimenti no.  
  
 Resta il problema della nostra divisione 3/2. Come ovviare?  
  
 Scende in campo, a questo punto, la variabile "**scale**". La variabile "scale" definisce la quantità di cifre decimale da prendere in considerazione. Di default avremo **scale=0**! Ecco risolto l'arcano mistero. 3/2 dà 1 come risultato perchè alla variabile "scale" è assegnato il valore 0. Di default.  
 Ma se scriviamo **scale=4** e ripetiamo la nostra divisione 3/2 noteremo, con piacere, il ritorno della virgola. A proposito, scale=4 significa quattro cifre decimale dopo la virgola.  
  
Apriamo il terminale, scriviamo bc e poi invio, scriviamo scale=4 e poi premiamo invio, scriviamo 3/2 e poi invio.    
  
 Ovviamente il valore della variabile scale lo possiamo cambiare ogni volta che vogliamo, aumentandolo, diminuendolo o impostandolo a zero. Infatti il valore successivo prenderà il posto di quello precedente.    
  
Anche saper uscire dal programma è importante. Al riguardo, basta digitare la parolina "**quit**" e premere invio. In alternativa possiamo premere la combinazione di tasti Ctrl+D.  
  
**L'opzione -l**   
Siamo usciti dal programma? Bene. Ora però lo dobbiamo riaprire, il tutorial su bc continua! Questa volta, però, non digiteremo bc da solo! Utilizzeremo anche due opzioni: -q e -l. Quindi scriveremo:  
  
 $ **bc -q -l**   
  
L'opzione **-q** ci risparmia del messaggio di benvenuto! Salutare è importante, ma non bisogna neanche esagerare...l'opzione **-l**, invece, è di estrema importanza. Essa abilita, infatti, la **libreria matematica standard** di bc! bc non sarà più lo stesso con questa opzione. Avremo delle funzioni supplementare. Potremo così svolgere delle operazione che senza quella libreria non avremmo mai potuto eseguire, quali  
  
**s(x)**: il seno di x in radianti   
**c(x)**: il coseno di x in radianti   
**a(x**): la tangente inversa di x   
**l(x)**: il logaritmo naturale di x   
  
Basta sostituire la x con il nostro numero e premere invio! Con -l, inoltre, la variabile "scale" avrà un valore pari a 20 (scale=20)   
  
**BC e la programmazione bash**   
Bc è di estrema importanza anche nella programmazione bash; infatti va a colmare una sua grande lacuna, cioè quella di non supportare la virgola mobile.  
Apriamo una finestra del terminale ed impostiamo le righe di comando relative alle più comuni operazioni aritmetiche. Iniziamo dalla divisione. Anzitutto dobbiamo scrivere il nome della variabile che in questo caso chiameremo, per semplicità, "a".  
  
 A seguire, il segno dell'uguale "=". Con l'uguale assegneremo alla variabile "a" il risultato della divisione. E poi impostiamo la nostra formula.   
  
$ **a=$(echo "scale=4; 3 /2" | bc)**  
  
Digitata tutta la riga di comando, non resta che premere Invio. Se la riga di comando è stata scritta  bene, si ritorna al prompt dei comandi. Per vedere il risultato della divisione, basta digitare:   
  
$ **echo $a**   
  
Lo stesso discorso vale per le altre operazioni aritmetiche:   
  
**$ a=$(echo "15+60" | bc)**  
**$ echo $a**  
 **$ a=$(echo "5\*6" | bc)**  
**$ echo $a**  
 **$ a=$(echo "190-50" | bc)**  
**$ echo $a**  
 **$ a=$(echo "9^3" | bc)**  
**$ echo $a**  
 **$ a=$(echo "sqrt (9)" | bc)**  
**$ echo $a**  
 **$ a=$(echo "scale=8; sqrt (10)" | bc)  
$ echo $a**  
 **$ a=$(echo "(15.34+18.13)\*12.2" | bc)**  
**$ echo $a**   
  
Da notare una cosa, ogni volta che digiteremo l'operazione successiva, il valore della variabile $a cambia!   
 **Le variabili ibase e obase.**   
Un'altra caratteristica importante di bc è quella di gestire basi di numerazione diverse da 10 (che è quella che abbiamo usato finora). A tal fine scendono in campo altre due variabili: ‘ibase’ e ‘obase’.   
  
**Ibase** permette di impostare il sistema di numerazione per i numeri che vengono inseriti (input); **obase**, invece, permette di impostare il sistema di numerazione per i numeri in uscita (output). Di default, sia ibase che obase contengono il valore 10.  Quando parliamo di variabili, dobbiamo immaginarci una scatola vuota che va riempita di contenuto. Questa “scatola” viene riempita del suo contenuto attraverso il segno dell'uguale “=”. Perciò quando scrivo **obase=16**, significa che nella variabile “scatola” obase abbiamo messo come contenuto il valore 16. Chiarito questo concetto possiamo fare quale esempio sulle basi di numerazione.  
  
Poniamo il caso che si voglia convertire il numero decimale 100 in **esodecimale**. Fermo restando il valore 10 per ibase, non dobbiamo fare altro che cambiare il valore alla variabile obase. Apriamo il terminale e digitiamo la seguente riga di comando:   
  
**$ echo "obase=16; 100" |bc**   
  
Il risultato sarà 64. Ciò che abbiamo fatto è abbastanza semplice! Qui abbiamo numeri in entrata, nel nostro caso 100, e numeri in uscita, nel nostro caso il risultato dell'operazione. Dato che dobbiamo convertire un numero dal sistemo decimale a quello esodecimale, andremo ad agire sulla variabile obase; infatti è obase la variabile che memorizza il sistema di numerazione in uscita, che, di default, abbiamo detto essere 10 (sistema decimale).  
  
Nel caso si voglia riconvertire 64 esodecimale in decimale, basta fare così   
  
**$ echo "ibase=16; 64" |bc**   
  
In quest'ultimo esempio, fermo restando obase=10 (di default), abbiamo cambiato il valore alla sola variabile ibase: **ibase=16**. A proposito dei numeri esodecimali, essi sono numeri in base 16 e più precisamente sono: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.  
  
Alla luce del fatto che i numeri esodecimali sono rappresentati da caratteri alfanumerici, facciamo un altro esempio di conversione da esodecimale a decimale. Poniamo il caso che si voglia convertire il numero FF in decimale   
  
**$ echo "ibase=16; FF" |bc**   
  
il risultato sarà 255.Facciamo la prova del nove, riconvertiamo il numero 255 in esodecimale:   
  
**$ echo "obase=16; 255" |bc**   
  
il risultato sarà FF. Poniamo ancora il caso che si voglia convertire il numero decimale 100 in binario (altro sistema di numerazione); basta digitare nel terminale la seguente riga di comando:   
  
**$ echo "obase=2; 100" |bc**   
  
il risultato sarà: 1100100. In quest'ultimo esempio, fermo restando ibase=10, abbiamo cambiato valore solo alla variabile obase: obase=2. Nel caso si voglia riconvertire tale risultato in decimale:   
  
**$ echo "ibase=2; 1100100" | bc**  
  
Perchè usare bc al posto di un programma ad interfaccia grafica? Ora la risposta è abbastanza scontata: Con bc avremo velocità, flessibilità e potenza. Basta solo prenderci un pò la mano. Per ulteriori informazioni su bc, basta digitare nel terminale:  
  
**$ man bc**