From Power Up To Bash Prompt

Greg O'Keefe, gcokeefe@postoffice.utas.edu.au v0.9, November 2000

Questa è una breve descrizione di quello che succede in un sistema Linux, dal momento in cui accendete il computer, a quello in cui vi accreditate sul sistema e vi viene presentato il prompt della shell bash. Capirlo vi sarà di grande aiuto quando dovrete risolvere dei problemi, o configurare il vostro sistema. Traduzione di Luca Ferraro (lferraro@NOSPAM.caspur.it, Dicembre 2004). Revisione di Antonio "bombadur" Bracaglia (bombadur@slacky.it, Febbraio 2005).

Indice

1 Introduzione 3

2 L'Hardware 3

2.1 Configurazione 4

2.2 Esercizi 4

2.3 Maggiori Informazioni 4

3 Lilo 5

3.1 Configurazione 5

3.2 Esercizi 5

3.3 Maggiori Informazioni 6

4 Il Kernel Linux 6

4.1 Configurazione 6

4.2 Esercizi 7

4.3 Maggiori Informazioni 7

5 La libreria GNU C 8

5.1 Configurazione 8

5.2 Esercizi 8

5.3 Maggiori Informazioni 9

6 Init 9

6.1 Configurazione 10

6.2 Esercizi 10

6.3 Maggiori Informazioni 10

7 Il Filesystem 10

7.1 Configurazione 11

7.2 Esercizi 11
7.3 Maggiori Informazioni 12

8 I Demoni del Kernel 12
8.1 Configurazione 13
8.2 Esercizi 13
8.3 Maggiori Informazioni 13

9 Il Logger di Sistema 14
9.1 Configurazione 14
9.2 Esercizi 14
9.3 Maggiori Informazioni 14

10 Getty e Login 14
10.1 Configurazione 14
10.2 Esercizi 15

11 Bash 15
11.1 Configurazione 15
11.2 Esercizi 16
11.3 Altre Informazioni 16

12 Comandi 16

13 Conclusioni 16

14 Amministrativa 16
14.1 Copyright 16
14.2 Homepage 17
14.3 Segnalazioni 17
14.4 Ringraziamenti 17
14.5 Storico delle modifiche 18
14.5.1 0.8 -> 0.9 (Novembre 2000) 18
14.5.2 0.7 -> 0.8 (Settembre 2000) 18
14.5.3 0.6 -> 0.7 18
14.5.4 0.5 -> 0.6 18
14.6 DA FARE 19

1 Introduzione

Trovo frustrante che all'interno della mia macchina Linux avvengano delle cose che non comprendo. Se, come me, volete veramente conoscere il vostro sistema invece di saperlo solamente usare, questo documento dovrebbe essere un buon punto di partenza. Questo tipo di conoscenze di basso livello è necessario se volete diventare dei veri esperti nel risolvere i problemi di Linux.

Darò per scontato che abbiate un sistema Linux funzionante e una conoscenza basilare di Unix e dell'hardware di un PC. Se così non fosse, un documento eccellente per iniziare ad imparare è quello di Eric S. Raymond,

The Unix and Internet Fundamentals HOWTO `http://www.linuxdoc.org/HOWTO/Unix-and-Internet-Fundamentals-HOWTO.html`

È breve, facilmente comprensibile, e copre tutte le basi.

L'argomento principale di questo documento è come fa Linux ad avviarsi. Ma cerca anche di essere una risorsa più generale di apprendimento. Per ciascuna sezione ho inserito degli esercizi. Se ne farete qualcuno, imparerete molto più di quanto fareste solamente leggendo.

Spero che alcuni lettori si cimenteranno nell'esercizio migliore che io conosca per Linux, ossia quello di costruirsi un sistema a partire dal codice sorgente. Giambattista Vico, filosofo italiano (1668 - 1744), disse verum ipsum factum, che significa la comprensione viene attraverso le azioni. Ringrazio Alex (vedi 14.4 (Acknowledgements)) per questa citazione.

Se volete farvelo da soli (NdT: l'espressione originale inglese era “roll your own”, come le sigarette arrotolate a mano), dovreste leggervi l'HOWTO di Gerard Beekmans,

Linux From Scratch HOWTO `http://www.linuxfromscratch.org`

(LFS). LFS contiene istruzioni dettagliate sulla cotruzione di un sistema operativo Linux usabile e completo a partire dal codice sorgente. Sul sito di LFS troverete anche una mailing list per le persone che stanno creando il proprio sistema operativo in questo modo. Le istruzioni a riguardo che facevano parte di questo documento si trovano ora in un documento separato Building a Minimal Linux System from Source Code, che potete trovare su

From PowerUp to Bash Prompt home page `http://www.netspace.net.au/~gok/power2bash/` . Spiegano come “giocare” con il sistema, puramente a scopo educativo.

I pacchetti vengono presentati nell'ordine in cui compaiono durante il processo di avvio del sistema. Questo significa che se installate i pacchetti in questo ordine, potrete riavviare dopo ciascuna installazione, e vedere come ogni volta il sistema si avvicina sempre più al prompt finale della shell. Trovo un rassicurante senso di avanzamento in questo.

Vi consiglio di leggere prima il testo principale di ogni sezione, saltando la parte degli esercizi e dei riferimenti. Successivamente scegliete a quale profondità di comprensione volete arrivare, e quanti sforzi siete disposti a fare. Quindi ricominciate a leggere dall'inizio, eseguendo gli esercizi e leggendo i documenti aggiuntivi man mano che procedete.

2 L'Hardware

Quando accendete il computer, questo esegue una sorta di auto-verifica per controllare che tutto funzioni correttamente. Questa fase viene chiamata auto-verifica di accensione (dall'inglese Power on self test). A questo punto, un programma chiamato bootstrap loader, che risiede nella memoria ROM del BIOS, cerca un settore di avvio (boot sector). Il settore di avvio è il primo settore di un disco, che contiene un piccolo programma capace di avviare un sistema operativo. I settori di avvio vengono marcati con il numero magico

0xAA55 = 43603 al byte 0x1FE = 510. Questo corrisponde agli ultimi due byte del settore. È in questo modo che l'hardware è in grado di riconoscere se un settore è di avvio oppure no.

Il bootstrap loader ha una lista di luoghi in cui cercare i possibili settori di avvio. Il mio vecchio computer cerca nel primo floppy drive e quindi nell'hard disk primario. I computer più moderni possono cercare un settore di avvio anche sul CD-ROM (NdT: recentemente anche tramite rete o dispositivi esterni come memory card o penne USB). Se trova un settore di avvio, lo carica in memoria e passa il controllo al programma in esso contenuto, che a sua volta carica il sistema operativo. In un tipico sistema Linux, questo programma corrisponde probabilmente alla prima parte del boot loader LILO (NdT: o di GRUB). Esistono diversi modi di configurare l'avvio del vostro sistema. Date uno sguardo al documento *LILO User's Guide* per maggiori dettagli, e alla sezione 3.3 (LILO) per dei rimandi sull'argomento.

Ovviamente ci sarebbe molto altro da dire su quello che fa l'hardware di un PC. Ma questo non è il posto adatto per dirlo. Leggetevi magari uno dei numerosi libri sull'hardware dei PC.

2.1 Configurazione

Il computer immagazzina alcune informazioni su se stesso nella sua CMOS. Queste informazioni comprendono ad esempio il tipo di hard disk e di RAM presenti nel sistema. Il BIOS del computer contiene un programma che permette di modificare queste impostazioni. Per capire come accedervi, controllate i messaggi che compaiono sullo schermo quando accendete il computer. Sul mio computer, bisogna premere il tasto Canc prima che inizi ad avviarsi il sistema operativo.

2.2 Esercizi

Un ottimo modo per imparare a conoscere l'hardware di un PC è quello di costruirsi una macchina da soli con dei pezzi di seconda mano. Procuratevi un processore, almeno un 386, così da poter poi eseguire un sistema Linux agevolmente. Non vi costerà molto. Chiedete in giro, qualcuno potrebbe darvi alcuni dei pezzi di cui avrete bisogno.

Scaricate e compilate il programma

Unios `http://www.netspace.net.au/~gok/resources`

(avevano una homepage su `http://www.unios.org` , ma è scomparsa) e fatevi con questo un dischetto di avvio. Si tratta semplicemente del programma tipo Ciao mondo! avviabile, che consiste soltanto di un centinaio di righe di codice assembly. Sarebbe bene vederle convertite in un formato comprensibile dal compilatore assembler GNU.

Aprite l'immagine del dischetto di avvio di unios con un editor esadecimale. Questa immagine è grande 512 byte, ossia esattamente un settore. Individuate il numero magico 0xAA55. Fate lo stesso con il settore di boot di un dischetto avviabile del vostro computer. Potete utilizzare il comando `dd` per copiarlo su un file: `dd if=/dev/fd0 of=myboot.sector`. Fate *molta* attenzione a come usate `if` (file di input) e `of` (file di output).

Date uno sguardo al codice sorgente del boot loader LILO.

2.3 Maggiori Informazioni

- *The Unix and Internet Fundamentals HOWTO* `http://www.linuxdoc.org/HOWTO/Unix-and-Internet-Fundamentals-HOWTO.html`

  di Eric S. Raymond, in particolare la sezione numero 3, *What happens when you switch on a computer?*

- Il primo capitolo della guida *The LILO User's Guide* fornisce un'eccellente spiegazione sulle partizioni dei dischi e sul processo di avvio di un PC. Vedete la sezione 3.3 (LILO) per un link.
- *The NEW Peter Norton Programmer's Guide to the IBM PC & PS/2*, di Peter Norton e Richard Wilton, Microsoft Press 1988 Esiste un libro di Norton più recente, che sembra buono, ma non posso permettermelo al momento!
- Uno dei molti libri disponibili sull'aggiornamento di un PC.

3 Lilo

Quando il computer carica il settore di boot su un normale sistema operativo Linux, quello che carica effettivamente è una parte di LILO, detta la prima parte del boot loader (NdT: first stage boot loader). Questo è un piccolo programma il cui unico scopo nella vita è quello di caricare ed avviare la seconda parte del boot loader (NdT: second stage boot loader).

La seconda parte del boot loader vi fornisce un prompt (se installato in modo tale da farlo) e carica il sistema operativo che avrete scelto.

Quando il vostro sistema è attivo e in uso, se eseguite il comando `lilo`, quello che di fatto mandate in esecuzione è l'installatore di mappa (NdT: map installer). Questo legge il file di configurazione `/etc/lilo.conf` e scrive sull'hard disk il boot loader e le informazioni sui sistemi operativi che possono venire avviati.

Esistono molti modi diversi di configurare l'avvio del vostro sistema. Quello che vi ho appena illustrato è il più ovvio e normale, se non altro per un computer il cui sistema operativo principale è Linux. La Guida Utente di LILO (NdT: LILO Users' Guide) tratta vari esempi di avvio. Sarebbe bene leggerli e provarne alcuni.

3.1 Configurazione

Il file di configurazione di lilo è `/etc/lilo.conf`. C'è una pagina di manuale per questo file: digitate `man lilo.conf` in una shell per leggerlo. La parte principale del file `lilo.conf` consiste in una voce per ogni sistema che lilo deve avviare. Una voce per Linux deve includere la posizione del kernel, e quale partizione del disco deve essere montata come filesystem radice. Per gli altri sistemi operativi, l'informazione principale è la partizione da cui eseguire l'avvio.

3.2 Esercizi

ATTENZIONE: fate molta attenzione con questi esercizi. È piuttosto facile che qualcosa vada storto e che venga compromesso il vostro master boot record, rendendo il vostro sistema inutilizzabile. Assicuratevi di avere un disco di recupero funzionante, e di sapere come utilizzarlo per rimettere le cose a posto. Vedete sotto per un link a tomsrtbt, il disco di recupero che uso e vi raccomando. La migliore precauzione è quella di utilizzare una macchina di cui non vi importa molto.

Installate lilo su un dischetto. Non importa se nel floppy ci sarà solo il kernel - incorrerete in un "kernel panic" quando il kernel tenterà di avviare init, ma almeno saprete che lilo sta funzionando.

Se vi piace, potete provare ad andare avanti a stiparvi del materiale, e vedere quanto del sistema operativo riuscite a mettere sul floppy. Questa è sicuramente la seconda miglior maniera per imparare qualcosa su Linux. Vedete il Bootdisk HOWTO (l'url è riportata più sotto), e tomsrtbt (url più sotto) per alcuni suggerimenti utili.

Provate a configurare lilo per avviare unios (vedete la sezione 2.2 (hardware exercises)). Come ulteriore esercizio, provate a farlo da un dischetto.

Create un ciclo di boot (NdT: boot-loop): configurate lilo nel master boot record in modo tale che avvii lilo da uno dei settori di boot delle partizioni primarie, e configurate quest'ultimo in modo tale da avviare quello nel master boot record... Oppure usate il master boot record e tutte e quattro le partizioni primarie per creare un ciclo a cinque punti. Divertente!

3.3 Maggiori Informazioni

- La pagina man di lilo.
- Il pacchetto LILO (*lilo* `ftp://lrcftp.epfl.ch/pub/linux/local/lilo/`), contiene la "LILO User's Guide" `lilo-u-21.ps.gz` (o superiore). Potreste già avere questo documento installato. Controllate in `/usr/doc/lilo`, o comunque in quella zona del filesystem. La versione postscript è migliore di quella in testo semplice, perché contiene diagrammi e tabelle.
- *tomsrtbt* `http://www.toms.net/rb` il miglior floppy Linux esistente. Un potente strumento di recupero.
- *The Bootdisk HOWTO* `http://www.linuxdoc.org/HOWTO/Bootdisk-HOWTO/`

4 Il Kernel Linux

Il kernel assume molti compiti. Credo che un modo semplice per riassumerli sia quello di dire che esso fa in modo che l'hardware soddisfi tutte le richieste dei programmi, in modo semplice ed efficiente.

Il processore può eseguire solo un'istruzione alla volta, ma i sistemi Linux sembrano far girare molte cose contemporaneamente. Il kernel riesce a fare questo passando molto velocemente da un task ad un altro. Riesce a fare il miglior uso del processore, tenendo traccia dei processi che sono pronti per essere eseguiti, e di quelli invece che sono in attesa di qualcosa, come un record da un file sull'hard disk o di un input dalla tastiera. Questa attività del kernel è chiamata scheduling.

Se un programma non sta facendo nulla, allora non è necessario che risieda nella RAM. Anche se un programma sta facendo qualcosa, potrebbe avere delle parti che non stanno facendo nulla. Lo spazio degli indirizzi di memoria di ciascun processo viene diviso in pagine. Il kernel tiene traccia delle pagine di cui i processi stanno facendo maggiore uso. Le pagine che non vengono utilizzate con frequenza possono venire spostate nella partizione di swap. Quando saranno nuovamente necessarie, un'altra pagina non utilizzata potrà essere spostata per fare lo spazio. Questa è la gestione della memoria virtuale.

Se avete compilato qualche volta il vostro kernel, avrete notato che esistono moltissime opzioni per specifici dispositivi. Il kernel contiene una grande quantità di codice specifico per comunicare con diverse tipologie di hardware, e presentarle ai programmi applicativi in modo semplice e uniforme.

Il kernel inoltre gestisce il filesystem, le comunicazioni tra i processi e diverso materiale riguardante il il networking.

Una volta che il kernel è stato caricato, la prima cosa che fa è quella di cercare un programma `init` da eseguire.

4.1 Configurazione

La maggior parte della configurazione del kernel viene fatta durante la sua compilazione, utilizzando il comando `make menuconfig`, o `make xconfig` da dentro la directory `/usr/src/linux/` (o comunque dalla directory contenente i sorgenti del kernel Linux). Potete riconfigurare i valori della modalità video predefinita, il filesystem radice, il dispositivo di swap, e la dimensione del disco RAM, tramite il comando `rdev`. (NdT: dalla versione 0.95 del kernel Linux il comando`rdev -s` per l'indicazione delle partizioni di swap non è più valido, in quanto l'attivazione dello swap viene effettuato tramite la chiamata di sistema `swapon()`). Questi parametri e altri ancora possono essere passati al kernel attraverso lilo. Potete fornire a lilo i parametri da passare al kernel o dal file di configurazione lilo.conf, o direttamente dal prompt di lilo. Per esempio, se voleste utilizzare hda3 come filesystem radice al posto di hda2, potreste digitare il comando

```
LILO: linux root=/dev/hda3
```

Se state costruendo un sistema dai sorgenti, potete rendervi la vita molto più semplice se create un kernel "monolitico". Ossia senza moduli. A questo punto non avrete bisogno di copiare i moduli del kernel sul sistema per cui è stato compilato.

NOTA: il file `System.map` viene utilizzato dal logger del kernel per determinare i nomi dei moduli che generano messaggi. Anche il programma `top` utilizza queste informazioni. Quando copiate il kernel sul nuovo sistema, copiate anche il file `System.map`.

4.2 Esercizi

Riflettete su questo: `/dev/hda3` è un file speciale che descrive una partizione dell'hard disk. Ma risiede su un filesystem, proprio come tutti gli altri file. Il kernel vuole sapere quale partizione deve montare come filesystem radice - ma questa non ha ancora un filesystem. Allora, come può leggere `/dev/hda3` per sapere quale partizione montare?

Se non lo avete ancora fatto: compilatevi un kernel. Leggete tutta la documentazione di aiuto per ciascuna opzione.

Vedete quanto piccolo riuscite a compilare un kernel che ancora funzioni. Potrete imparare molto se lascerete fuori le cose sbagliate!

Leggete "The Linux Kernel" (vedi l'indirizzo URL più sotto) e, mentre lo leggete, individuate le parti di codice sorgente a cui fa riferimento. Il libro (come ho detto) si riferisce alla versione 2.0.33 del kernel, che è piuttosto vecchia. Potrebbe essere più semplice da seguire se scaricaste questa vecchia versione del kernel ed esaminaste quel codice sorgente. Trovo sia entusiasmante trovare pezzi di codice C chiamati "processo" e "pagina".

Hack! Vedete se riuscite a farvi sputare fuori qualche messaggio extra o cose simili.

4.3 Maggiori Informazioni

- `/usr/src/linux/README` ed i contenuti di `/usr/src/linux/Documentation/` (Questi potrebbero trovarsi da qualche altra parte sul vostro sistema).
- *The Kernel HOWTO* `http://mirror.aarnet.edu.au/linux/LDP/HOWTO/Kernel-HOWTO.html`
- Le spiegazioni di aiuto disponibili quando configurate il kernel tramite `make menuconfig` o `make xconfig`
- *The Linux Kernel (and other LDP Guides)* `http://mirror.aarnet.edu.au/linux/LDP/LDP/`
- il codice sorgente, vedete

  Building a Minimal Linux System from Source Code `http://www.netspace.net.au/~gok/power2bash`

  per i link.

5 La libreria GNU C

Il passo successivo all'accensione del computer consiste nel caricamento e nella esecuzione di init. Comunque, init, come la maggior parte dei programmi, utilizza delle funzioni di libreria.

Potreste aver già visto un esempio di programma C come questo:

```
main() {
        printf("Hello World!\n");
}
```

Il programma non contiene nessuna definizione della funzione `printf`, e quindi da dove salta fuori? Questa proviene dalle librerie standard C che, su un sistema GNU/Linux, si chiamano glibc. Se compilate il programma sotto Visual C++, allora quella funzione viene da una implementazione Microsoft delle stesse funzioni standard. Esistono milioni di queste funzioni standard, per matematica, stringhe, allocazione di memoria per data e ora, e così via. Qualsiasi cosa in Unix (incluso Linux) è stata scritta scritta in C, perciò ogni applicazione utilizza queste funzioni.

Se guardate nella directory `/lib` del vostro sistema Linux troverete molti file chiamati `libqualcosa.so` o `libqualcosa.a`, ecc. Sono le librerie di queste funzioni. Glibc è esattamente l'implementazione GNU di queste funzioni.

Ci sono due modi in cui i programmi possono utilizzare queste le funzioni di libreria. Se linkate il programma in modo *statico*, queste funzioni di libreria verranno copiate dentro l'eseguibile che viene creato. Le librerie `libqualcosa.a` servono a questo. Se invece linkate il programma in modo *dinamico* (e questa è la modalità predefinita), quando il programma sarà in esecuzione e avrà bisogno del codice della libreria, questo verrà chiamato dal file `libqualcosa.so`.

Il comando `ldd` farà al caso vostro quando vorrete sapere quali librerie sono necessarie ad un particolare programma. Per esempio, ecco le librerie utilizzate da `bash`:

```
[greg@Curry power2bash]$ ldd /bin/bash
        libtermcap.so.2 => /lib/libtermcap.so.2 (0x40019000)
        libc.so.6 => /lib/libc.so.6 (0x4001d000)
        /lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
```

5.1 Configurazione

Alcune funzioni delle librerie dipendono dal posto in cui vivete. Per esempio, in Australia scriviamo le date nel formato dd/mm/yy (NdT: giorno/mese/anno, come in Italia), ma in America viene usato il formato mm/dd/yy. Esiste un programma che viene fornito con le `glibc`, chiamato `localedef`, che consente di configurare le localizzazioni.

5.2 Esercizi

Usate `ldd` per scoprire quali librerie utilizzano le vostre applicazioni preferite.

Usate `ldd` per scoprire quali librerie sono utilizzate da `init`.

Createvi una libreria giocattolo, con una o due funzioni. Per crearla utilizzate il programma `ar`. La pagina di manuale di `ar` può essere un buon punto di inizio per capire come fare. Scrivete, compilate e linkate un programma che faccia uso di questa vostra libreria.

5.3 Maggiori Informazioni

- il codice sorgente, vedete

  Building a Minimal Linux System from Source Code `http://www.netspace.net.au/~gok/power2bash`

  per ulteriori link

6 Init

Parlerò solamente dello stile “System V” di init, utilizzato dalla maggior parte dei sistemi Linux. Esistono anche altre alternative. Infatti, potete mettere qualsiasi programma vogliate in `/sbin/init`, e il kernel lo eseguirà non appena avrà terminato la sua fase di caricamento.

È compito di `init` mandare in esecuzione qualsiasi cosa nel modo corretto. `init` verifica che i filesystem siano a posto e li monta. Avvia i demoni per registrare i messaggi di sistema, gestisce le funzionalità di rete, rende disponibili le pagine web, gestisce il vostro mouse, e così via. Avvia anche i processi getty che fanno comparire il prompt di login sui vostri terminali virtuali.

Dovrei anche parlare di una faccenda piuttosto complessa riguardo al passaggio tra i vari “run-levels”, ma la salterò quasi del tutto e vi parlerò piuttosto di come si avvia il sistema.

Init legge il file `/etc/inittab` che gli dice cosa fare. La prima cosa che di solito viene richiesta è quella di eseguire uno script di inizializzazione. Il programma che esegue (o interpreta) questo script è `bash`, lo stesso programma che vi fornisce un prompt dei comandi. Nei sistemi Debian, lo script di inizializzazione è `/etc/init.d/rcS`, mentre in Red Hat, è `/etc/rc.d/rc.sysinit`. In questo script vengono verificati e montati i filesystem, viene impostato l’orologio di sistema, abilitato lo spazio di swap, il nome dell’host, ecc.

Successivamente, viene chiamato un altro script che ci porta al run-level predefinito. Questo significa un gruppo di sottosistemi da avviare. Esistono una serie di directory `/etc/rc.d/rc0.d`, `/etc/rc.d/rc1.d`, ..., `/etc/rc.d/rc6.d` in Red Hat, o `/etc/rc0.d`, `/etc/rc1.d`, ..., `/etc/rc6.d` in Debian, che corrispondono ai vari run-level. Se stiamo entrando nel run-level 3 di un sistema Debian, allora lo script eseguirà tutti gli script presenti nella directory `/etc/rc3.d` i cui nomi inizino per ‘S’ (S sta per start). Questi script in realtà sono soltanto dei link simbolici a degli script che si trovano in un’altra directory, generalmente chiamata `init.d`.

Quindi, il nostro script di run-level è stato chiamato da `init` e ora sta cercando all’interno di una directory tutti gli script che cominciano per ‘S’. Potrebbe trovare `S10syslog` per primo. Il numero dice allo script di run-level in quale ordine questi devono venire eseguiti. Perciò in questo caso, dal momento che non ci sono altri script che cominciano per S00 ... S09, `S10syslog` viene eseguito per primo. Ma `S10syslog` è in realtà un link a `/etc/init.d/syslog`, uno script per avviare o fermare il logger di sistema. Dato che il link comincia per ‘S’, lo script di run-level sa che dovrà eseguire lo script `syslog` con il parametro “start”. Esistono anche i corrispettivi link che cominciano per ‘K’ (K sta per kill), che specificano quali servizi fermare e in quale ordine, quando viene lasciato un certo run-level.

Per scegliere quali sottosistemi (servizi) far partire in modo predefinito, dovrete impostare questi collegamenti nella directory `rcN.d`, dove per N si intende il runlevel predefinito, configurato nel vostro `inittab`.

L’ultima cosa importante che fa init è far partire alcune sessioni del programma `getty`. Questi sono di tipo “respawned”, che significa che se si fermano, `init` li fa partire nuovamente. La maggior parte delle distribuzioni vengono configurate con sei terminali virtuali. Potreste volerne di meno per risparmiare memoria, o di più in modo da mandare in esecuzione più cose, e passare rapidamente da una all’altra a seconda delle vostre necessità. Potreste anche voler eseguire `getty` per un terminale testuale o per un modem. In tal caso dovrete modificare il file `inittab`.

6.1 Configurazione

Il file `/etc/inittab` rappresenta il vertice della configurazione di init.

Le directory `rcN.d`, dove N = 0, 1, ..., 6 determinano quali sottosistemi vengono avviati.

Da qualche parte, in uno degli script richiamati da init, verrà dato il comando `mount -a`. Questo significa montare tutti i filesystem che si suppone debbano venire montati. Il file `/etc/fstab` definisce quali filesystem il sistema deve montare. Se volete modificare quello che viene montato quando il il sistema viene avviato, questo è il file che dovrete editare. Esiste anche una pagina di manuale per `fstab`.

6.2 Esercizi

Individuate la directory `rcN.d` del run-level predefinito del vostro sistema ed eseguite un `ls -l` per vedere a cosa puntano link in essa presenti.

Cambiate il numero di sessioni getty che vengono eseguite sul vostro sistema.

Eliminate ogni servizio di cui non avete bisogno dal vostro run-level predefinito.

Provate a verificare il numero minimo di servizi con cui riuscite a partire.

Create un dischetto con lilo, un kernel e un programma ciao mondo! compilato staticamente, chiamato `/sbin/init` e osservate se si avvia ed emette il messaggio di saluto.

Osservate attentamente l'avvio del vostro sistema, e prendete nota di tutto ciò che si sta verificando. Oppure stampate una porzione del file di log di sistema `/var/log/messages` dal momento dell'avvio. A questo punto, partendo da `inittab`, seguite tutti gli script e cercate di capire ogni parte del codice, e che cosa fa. Potete anche inserire dei messaggi extra, come ad esempio

```
echo "Ciao, io sono il file rc.sysinit"
```

Questo è un buon esercizio per imparare anche il linguaggio di scripting della shell Bash, visto che alcuni di questi script sono piuttosto complicati. Procuratevi una buona guida di Bash da tenere sottomano.

6.3 Maggiori Informazioni

- Esistono le pagine di manuale per i file `inittab` e `fstab`. Digitate ad esempio `man inittab` in una shell per vederlo.
- The Linux System Administrators Guide contiene una buona pagina *section* `http://mirror.aarnet.edu.au/linux/LDP/LDP/` su init.
- il codice sorgente, vedete

  Building a Minimal Linux System from Source Code `http://www.netspace.net.au/~gok/power2bash`

  per ulteriori link

7 Il Filesystem

In questa sezione, utilizzerò la parola "filesystem" in due differenti accezioni. Una riferendomi ai filesystem delle partizioni del disco o di altre periferiche, l'altra riferita al filesystem che vi viene presentato da un sistema Linux in esecuzione. In Linux, infatti, voi "montate" il filesystem di un disco all'interno del filesystem di sistema.

Nella sezione precedente ho detto che gli script di avvio verificano e montano i filesystem. I comandi a cui vengono affidati questi compiti sono rispettivamente `fsck` e `mount`.

L'hard disk può essere visto come un grande spazio su cui potete scrivere tanti 1 e 0. Un filesystem impone su di esso una struttura, e fa in modo che i file vengano visualizzati all'interno di directory dentro altre directory... Ogni file viene rappresentato da un inode, che definisce il file a cui fa riferimento, quando è stato creato e dove si trova il suo contenuto. Anche le directory vengono rappresentate dagli inode, ma questi dicono piuttosto dove trovare gli inode dei file in esse contenuti. Se il sistema vuole leggere il file `/home/greg/bigboobs.jpeg`, prima trova l'inode della directory radice `/` nel "superblocco", poi troverà l'inode per la directory `home` tra quelli contenuti in `/`, quindi l'inode per la directory `greg` tra quelli contenuti in `/home`, e infine l'inode per `bigboobs.jpeg`, che indica quali blocchi del disco devono essere letti.

Se aggiungiamo dei dati alla fine di un file, potrebbe accadere che questi dati vengano scritti prima che l'inode sia aggiornato in modo da sapere che i nuovi blocchi appartengono al file, o viceversa. Se venisse a mancare la corrente in questo momento, il filesystem risulterebbe corrotto. Ed è proprio questo genere di cose che `fsck` tenta di verificare e riparare.

Il comando `mount` prende il filesystem di un dispositivo, e lo innesta nella gerarchia che vedete quando usate il vostro sistema. Generalmente, il kernel monta la radice del proprio filesystem in modalità di sola lettura. Il comando `mount` viene utilizzato per rimontarlo in modalità di lettura e scrittura dopo che `fsck` abbia verificato che tutto sia in ordine.

Linux supporta anche altri tipi di filesystem: msdos, vfat, minix, e così via. Le caratteristiche di ogni filesystem vengono descritte in modo astratto attraverso il file system virtuale (VFS). Ma su questo argomento non entrerò nei dettagli. Potete trovarne una trattazione su "The Linux Kernel" (vedete la sessione 4.3 (The Linux Kernel) per la URL).

Un tipo di filesystem completamente differente viene montato su `/proc`. Questo contiene una rappresentazione di quello che accade nel kernel. Qui esiste una directory per ogni processo che si trova in esecuzione sul sistema, con il numero del processo come nome per la directory. Sono presenti anche dei file come `interrupts` o `meminfo`, che vi informano sul modo in cui viene utilizzato l'hardware. Potete imparare molto esplorando `/proc`.

7.1 Configurazione

Sono presenti dei parametri per il comando`mke2fs`, che crea un filesystem ext2. Questi controllano la dimensione dei blocchi, il numero di inode, e così via. Controllate la pagina di manuale di `mke2fs` per maggiori dettagli.

Ciò che viene montato, e dove, sul vostro filesystem, viene controllato dal file `/etc/fstab`. Anche per questo esiste una pagina di manuale.

7.2 Esercizi

Create un piccolissimo filesystem, e guardatelo attraverso un editor esadecimale. Identificate gli inode, i superblocchi e i file in esso contenuti.

Sono sicuro che esistono degli strumenti che vi possano fornire una rappresentazione grafica del filesystem. Trovatene uno, provatelo e mandatemi una email con una bella recensione!

Leggete il codice del filesystem ext2 nel Kernel.

7.3 Maggiori Informazioni

- Il capitolo 9 del libro LDP "The Linux Kernel" contiene una eccellente descrizione dei filesystem. Potete trovarlo nel sito australiano di LDP

  mirror `http://mirror.aarnet.edu.au/linux/LDP/LDP/`

- Il comando `mount` fa parte del pacchetto util-linux, di cui trovate un link su

  Building a Minimal Linux System from Source Code `http://www.netspace.net.au/~gok/power2bash`

- Le pagine man per `mount`, `fstab`, `fsck`, `mke2fs` e `proc`

- Il file `Documentation/proc.txt` nella directory dei sorgenti del Kernel Linux illustra il filesystem `/proc`.

- EXT2 File System Utilities

  ext2fsprogs `http://web.mit.edu/tytso/www/linux/e2fsprogs.html` home page

  ext2fsprogs `ftp://mirror.aarnet.edu.au/pub/linux/metalab/system/filesystems/ext2/` Mirror Australiano. Qui trovate anche un documento su Ext2fs, sebbene un po' datato, e non così leggibile come il capitolo 9 di "The Linux Kernel".

- *Unix File System Standard* `ftp://tsx-11.mit.edu/pub/linux/docs/linux-standards/fsstnd/`

  Un altro link *link* `http://www.pathname.com/fhs/` allo Unix File System Standard. Descrive cosa dovrebbe andare, e dove, in un filesystem Unix, e perché. Contiene anche i requisiti minimi per i contenuti di `/bin`, `/sbin` e così via. Questo è un buon riferimento se il vostro obiettivo è quello di creare un sistema minimo ma completo.

8 I Demoni del Kernel

Se eseguite il comando`ps aux`, vedrete probabilmente qualcosa di simile:

```
USER       PID %CPU %MEM  SIZE   RSS TTY STAT START   TIME COMMAND
root         1  0.1  8.0  1284   536   ? S    07:37   0:04 init [2]
root         2  0.0  0.0     0     0   ? SW   07:37   0:00 (kflushd)
root         3  0.0  0.0     0     0   ? SW   07:37   0:00 (kupdate)
root         4  0.0  0.0     0     0   ? SW   07:37   0:00 (kpiod)
root         5  0.0  0.0     0     0   ? SW   07:37   0:00 (kswapd)
root        52  0.0 10.7  1552   716   ? S    07:38   0:01 syslogd -m 0
root        54  0.0  7.1  1276   480   ? S    07:38   0:00 klogd
root        56  0.3 17.3  2232  1156   1 S    07:38   0:13 -bash
root        57  0.0  7.1  1272   480   2 S    07:38   0:01 /sbin/agetty 38400 tt
root        64  0.1  7.2  1272   484  S1 S    08:16   0:01 /sbin/agetty -L ttyS1
root        70  0.0 10.6  1472   708   1 R   Sep 11   0:01 ps aux
```

Questa è una lista dei processi in esecuzione sul sistema. Le informazioni provengono dal filesystem `/proc` che ho descritto nella sezione precedente.

Notate che `init` è il processo numero uno. I process 2, 3, 4 e 5 sono kflushd, kupdate, kpiod e kswapd. C'è tuttavia qualcosa di strano: notate che in entrambe le colonne del Virtual Storage Size (SIZE) e del Real Storage Size (RSS), questi processi hanno valori nulli. Come può un processo non utilizzare memoria?

Questi processi sono i demoni del kernel. La maggior parte dei kernel non mostra affatto una lista dei processi attivi, e potete capire quanta memoria questi stanno utilizzando solamente sottraendo la memoria disponibile da quella totale del sistema. I demoni del kernel vengono avviati dopo init, e quindi prendono un numero di processo come fanno tutti gli altri processi. Ma il loro codice ed i loro dati risiedono nella parte di memoria riservata al kernel.

Intorno alle voci della colonna dei comandi ci sono le parentesi perché il filesystem `/proc` non contiene informazioni sulla linea di comando per questi processi.

Ma allora a cosa servono questi demoni del kernel? Versioni precedenti di questo documento riportavano una richiesta di aiuto per questo paragrafo, dal momento che non sapevo molto sui demoni del kernel. La storia parziale che segue è stata realizzata combinando le diverse risposte a questa richiesta, per le quali sono veramente grato. Maggiori informazioni, riferimenti e correzioni sono benvenute!

L'input e l'output viene realizzato tramite i *buffer* in memoria. Questo consente di far eseguire le cose più velocemente. Quello che i programmi scrivono può essere tenuto in memoria, in un buffer, e quindi scritto su blocchi di disco in modo molto più efficiente. I demoni `kflushd` e `kupdate` hanno questo compito: `kupdate` viene eseguito periodicamente (5 secondi?) per verificare se vi siano dei buffer utilizzati. Se ci sono, richiama `kflushd` per riversarli sul disco.

Spesso i processi non hanno nulla da fare, e spesso, quelli in esecuzione non hanno bisogno di tenere in memoria tutto il loro codice ed i loro dati. Questo significa che possiamo fare un utilizzo migliore della memoria, spostando le porzioni inutilizzate dei programmi sulla(e) partizione(i) di swap dell'hard disk. Il trasferimento di questi dati da e verso la memoria a seconda della necessità viene gestito di demoni `kpiod` e `kswapd`. Ogni secondo circa, `kswapd` si attiva e verifica lo stato della memoria e, se qualcosa che era stato portato sul disco diventa necessario alla memoria, o se non c'è memoria libera a sufficienza, viene attivato `kpiod`.

Sul vostro sistema potrebbe essere in esecuzione anche il demone `kapmd`, se avete configurato il kernel con la gestione automatica dell'energia.

8.1 Configurazione

Il programma `update` vi consente di configurare `kflushd` e `kswapd`. Provate il comando `update -h` per avere maggiori informazioni.

Lo spazio di swap viene attivato da `swapon`, e disattivato da `swapoff`. Lo script di init (`/etc/rc.sysinit` oppure `/etc/rc.d/rc.sysinit`) generalmente fa una chiamata a `swapon` quando il sistema si avvia. Mi è stato detto che `swapoff` è comodo per il risparmio energetico sui portatili.

8.2 Esercizi

Eseguite un `update -d` (NdT: al momento della traduzione, l'opzione `-d` non viene contemplata dal programma `udate`, e notate cosa viene blaterato nell'ultima riga riguardo al "threshold for buffer fratricide" (NdT: "punto limite per un fratricidio tra buffer"). Ecco un argomento intrigante, provate ad investigare!

Entrate nella directory `/proc/sys/vm` e visualizzate con `cat` i file ivi contenuti. Vedete cosa riuscite a scoprire.

8.3 Maggiori Informazioni

Il documento "The Linux Kernel" su Linux Documentation Project (LDP) (vedete la sessione 4.3 (The Linux Kernel) per la URL)

Il codice sorgente del kernel Linux, se siete abbastanza coraggiosi! Il codice di `kswapd` è in `linux/mm/vmscan.c`, quello di `kflushd` e di `kupdate` sono in `linux/fs/buffer.c`.

9 Il Logger di Sistema

Init avvia i demoni `syslogd` e `klogd`. Questi scrivono dei messaggi nei log. I messaggi del kernel vengono gestiti da `klogd`, mentre `syslogd` gestisce i messaggi di log degli altri processi. Il file di log principale è `/var/log/messages`. Questo è un buon posto in cui investigare se c'è qualcosa che non va nel vostro sistema. Spesso potreste trovarvi degli indizi importanti.

9.1 Configurazione

Il file `/etc/syslog.conf` istruisce il programma che gestisce i log sui messaggi da gestire, e dove riportarli. I messaggi vengono identificati tramite il servizio da cui provengono, e dal loro livello di priorità. Questo file di configurazione consiste di una serie di righe su cui viene indicato che i messaggi provenienti dal servizio x, con priorità y, devono venire inviati su z, dove z può essere un file, una tty, una stampante, un host remoto o altro.

NOTA: il demone Syslog richiede che il file `/etc/services` sia presente sul sistema. Il file services assegna le porte. Non sono sicuro se syslog necessiti di una porta assegnata per poter gestire i log in remoto, o se la stessa gestione locale dei log venga fatta tramite una porta, o se utilizza `/etc/services` soltanto per convertire i nomi dei servizi che avete impostato in `/etc/syslog.conf` in numeri di porta.

9.2 Esercizi

Osservate i vostri log di sistema. Cercate un messaggio che non comprendete, e sforzatevi di capire cosa significhi.

Mandate tutti i vostri messaggi di log su una tty. (poi reimpostateli al modo predefinito)

9.3 Maggiori Informazioni

Australian sysklogd *Mirror* `http://mirror.aarnet.edu.au/pub/linux/metalab/system/daemons/`

10 Getty e Login

Getty è il programma che vi permette di autenticarvi attraverso un dispositivo seriale, come ad esempio un terminale fisico o virtuale, oppure tramite un modem. Getty visualizza il prompt di autenticazione. Una volta inserito il vostro nome utente, getty lo passa a `login`, il quale vi chiederà una password, controllerà la sua validità, e vi fornirà una shell.

Esistono diverse versioni disponibili di getty. Alcune distribuzioni, compresa Red Hat, ne usano una molto piccola chiamata `mingetty`, che funziona solamente con i terminali virtuali.

Il programma `login` fa parte del pacchetto util-linux, che contiene a sua volta una versione di getty chiamata `agetty`, e che funziona molto bene. Questo pacchetto contiene anche `mkswap`, `fdisk`, `passwd`, `kill`, `setterm`, `mount`, `swapon`, `rdev`, `renice`, `more` e molti altri ancora.

10.1 Configurazione

Il messaggio che viene visualizzato sulla parte alta del vostro schermo assieme al prompt di login, proviene da `/etc/issue`. In genere le istanze di Getty vengono avviate da `/etc/inittab`. Login verifica i dati di un utente contenuti in `/etc/passwd`, o in `/etc/shadow` se usate lo shadow delle password.

10.2 Esercizi

Create un file `/etc/passwd` a mano. Potete anche impostare delle password vuote, e modificarle con il comando `passwd` una volta eseguito il login. Leggete la pagina di manuale corrispondente tramite il comando `man 5 passwd`, per avere il manuale del file piuttosto che quello del programma passwd.

11 Bash

Se fornite a `login` una combinazione di username e password valida, questo verificherà nel file `/etc/passwd` quale shell deve mettervi a disposizione. Nella maggior parte dei casi per un sistema Linux, questa sarà la shell `bash`. Spetta a `bash` leggere ed interpretare i vostri comandi. Può essere vista contemporaneamente come un'interfaccia utente, e come un interprete di linguaggio di programmazione.

Come interfaccia utente legge i vostri comandi e, se sono dei comandi "interni" come `cd` li esegue essa stessa, altrimenti cerca ed esegue un programma se sono dei comandi "esterni", come `cp` o `startx`. Bash svolge anche altri compiti diversi, come il mantenimento dello storico dei comandi ed il completamento automatico dei nomi.

Abbiamo già visto `bash` in azione come interprete di linguaggio di programmazione. Gli script che vengono eseguiti da `init` per avviare il sistema solitamente sono degli script di shell, e vengono eseguiti da `bash`. Avere a disposizione un vero e proprio linguaggio di programmazione affianco ai programmi di utilità messi a disposizione dalla linea di comando, costituisce un potente strumento, se sapete usarlo. Ad esempio (permettetemi l'immodestia), l'altro giorno ho dovuto applicare una serie di patch ai file di una directory contenente del codice sorgente. Sono riuscito a farlo con questo semplice, singolo comando:

```
for f in /home/greg/sh-utils-1.16*.patch; do patch -p0 < $f; done;
```

Questo comando cerca nella mia directory personale tutti quei file i cui nomi iniziano per `sh-utils-1.16` e finiscono per `.patch`. Quindi ne prende uno alla volta, lo imposta alla variabile `f`, ed esegue i comandi contenuti tra le istruzioni `do` e `done`. Nel mio caso c'erano 11 file di patch, ma avrebbero potuto facilmente essere anche 3000.

11.1 Configurazione

Il file `/etc/profile` controlla il comportamento globale di bash sul sistema. Quello che scriverete su questo file avrà effetto per tutti gli utenti che utilizzeranno bash sul vostro sistema. Aggiungerà inoltre alcune directory alla variabile di ambiente `PATH`, imposterà la variabile `MAIL`, ecc.

Il comportamento predefinito della tastiera lascia spesso molto a desiderare. Attualmente è la libreria readline che lo gestisce. Readline è un pacchetto separato che gestisce la linea di comando, fornendo lo storico dei comandi ed il completamento automatico dei nomi, così come alcune caratteristiche di editing avanzato da linea di comando. Viene compilata all'interno di bash. In modo predefinito, readline viene configurata attraverso il file `.inputrc`, presente nella vostra directory home. La variabile bash INPUTRC può venire utilizzata per modificare questa impostazione. Per esempio in Red Hat 6, `INPUTRC` viene impostata a `/etc/inputrc` in `/etc/profile`. Questo permette ai tasti backspace, delete, home e end, di funzionare adeguatamente per tutti gli utenti.

Dopo che bash ha letto il file di configurazione globale di sistema, va a leggere il vostro file di configurazione personale. Ne verifica la presenza nella vostra directory home attraverso i nomi di file `.bash_profile`, `.bash_login` e `.profile`. Esegue il primo dei tre che individua. Se volete modificare il comportamento di bash solo per voi, senza cambiare il suo funzionamento per gli altri utenti, fatelo in questi file. Ad esempio, molte applicazioni usano delle variabili di ambiente per impostare il proprio funzionamento. Io ho la variabile

`EDITOR` impostata a `vi`, in modo da poter utilizzare vi all'interno di Midnight Commander (un eccellente file manager testuale), invece del suo editor predefinito.

11.2 Esercizi

I principi base di bash sono facili da imparare. Ma non fermatevi lì: c'è ancora molto materiale per approfondire. Prendete l'abitudine di cercare metodi sempre migliori per svolgere determinati compiti.

Leggete gli script di shell, cercate di capire le cose che non vi sono chiare.

11.3 Altre Informazioni

- Esiste un "Bash Reference Manual", che comprende praticamente tutto, ma di pesante lettura.
- C'è un libro della O'Reilly su Bash, ma non sono sicuro che sia buono.
- Non conosco dei tutorial gratuti, buoni e aggiornati per bash. Se li conoscete, per favore fatemelo sapere e segnalatemi per posta elettronica l'indirizzo URL su cui reperirli.
- il codice sorgente, vedete

  Building a Minimal Linux System from Source Code `http://www.netspace.net.au/~gok/power2bash`

12 Comandi

Gran parte del lavoro che svolgete in bash viene eseguito attraverso dei comandi come `cp`. La maggior parte di questi comandi sono dei piccoli programmi, sebbene alcuni, come `cd`, siano contenuti all'interno della shell.

I comandi vengono distribuiti sotto forma di pacchetti, la maggior parte dei quali dalla Free Software Foundation (o GNU). Piuttosto che mettermi ad elencare qui i pacchetti, vi rimando direttamente a

Linux From Scratch HOWTO `http://www.linuxfromscratch.org` . Contiene una lista completa ed aggiornata dei pacchetti che vengono installati in un sistema Linux, insieme alle istruzioni su come compilarli.

13 Conclusioni

Una delle cose migliori di Linux, secondo la mia modesta opinione, è che potete entrarci dentro e capire davvero come funzioni l'intero sistema. Spero che apprezziate questo come l'ho apprezzato io. E spero che questa piccola guida vi sia stata di aiuto.

14 Amministrativa

14.1 Copyright

14.2 Homepage

Potete trovare la versione aggiornata di questo documento su

From Powerup To Bash Prompt `http://www.netspace.net.au/~gok/power2bash`

insieme al suo complementare “Building a Minimal Linux System from Source Code”.

Esiste una traduzione in Francese su

From Powerup To Bash Prompt `http://www.freenix.fr/unix/linux/HOWTO/From-PowerUp-To-Bash-Prompt-HOWTO.html` grazie a Dominique van den Broeck. La versione giapponese arriverà presto, se non si trova già su

Japanese Documentation and FAQ Project `http://www.linux.or.jp/JF` .

14.3 Segnalazioni

Mi piacerebbe ricevere i vostri commenti, critiche o suggerimenti per migliorare questo documento. Per favore scrivetemi all’indirizzo

Greg O’Keefe `mailto:gcokeefe@postoffice.utas.edu.au`

14.4 Ringraziamenti

I nomi dei prodotti sono marchi registrati dei rispettivi proprietari, e sono perciò da considerarsi come noti.

Ci sono alcune persone che vorrei ringraziare per avermi aiutato a realizzare questa guida.

Michael Emery

Per avermi ricordato di Unios.

Tim Little

Per alcuni buoni suggerimenti su `/etc/passwd`

sPaKr su #linux di efnet

Che mi ha fatto notare come syslogd necessiti di `/etc/services`, e che mi ha fatto conoscere l’espressione “rolling your own” in riferimento alla costruzione di un sistema a partire dal codice sorgente.

Alex Aitkin

Per avermi fatto conoscere Vico e il suo “verum ipsum factum” (la comprensione viene attraverso le azioni).

Dennis Scott

Per aver corretto la mia aritmetica esadecimale.

jdd

Per avermi segnalato alcuni errori di battitura.

David Leadbeater

Per aver contribuito a chiarire alcuni dubbi sui demoni del kernel.

Dominique van den Broeck

Per aver tradotto questo documento in Francese.

Matthieu Peeters

Per alcune preziose informazioni sui demoni del kernel.

John Fremlin

Per alcune preziose informazioni sui demoni del kernel.

Yuji Senda

Per la traduzione in Giapponese del documento.

Antonius de Rozari

Per aver contribuito alla realizzazione di una versione assembler GNU di UNIOS (vedete la sezione “risorse” nella mia home page)

14.5 Storico delle modifiche

14.5.1 0.8 -> 0.9 (Novembre 2000)

- Incorporate alcune informazioni da Matthieu Peeters e John Fremlin sui demoni del kernel e sul filesystem `/proc`.

14.5.2 0.7 -> 0.8 (Settembre 2000)

- Rimosse le istruzioni su come costruire un sistema, e poste in un documento separato. Corretti alcuni link.
- Modificata la homepage da *learning@TasLUG* `http://learning.taslug.org.au/power2bash`

  a *my own webspace* `http://www.netspace.net.au/~gok/power2bash` .
- Completamente fallito il tentativo di inserire molto buon materiale, fornito dai contributi di svariate persone. Forse sarà per una prossima volta :(

14.5.3 0.6 -> 0.7

- maggiore enfasi sulla parte teorica, minore sul metodo di costruzione di un sistema; le informazioni sulla costruzione riunite in una sezione separata, e riordinato il sistema compilato; indirizzati i lettori verso il documento di “Linux From Scratch” di Gerard Beekmans per una compilazione più completa
- aggiunte alcune divagazioni suggerite da David Leadbeater
- corrette un paio di url, aggiunto il link al download di unios a learning.taslug.org.au/resources
- verificati e corretti i link
- riscrittura generale, riordinamento

14.5.4 0.5 -> 0.6

- aggiunto lo storico delle modifiche
- aggiunte alcune voci alla lista DA FARE

14.6 DA FARE

- spiegare i moduli del kernel, i comandi depmod, modprobe, insmod e simili (devo prima provarli!)
- menzionare il filesystem /proc, possono venire proposti esercizi su di esso
- convertire a docbook sgml
- aggiungere altri esercizi, magari una intera sezione dedicata ad esercizi più complessi, come ad esempio la creazione di un filesystem minimale, un file alla volta, attraverso l'installazione di una distribuzione.
- aggiungere miglioramenti al makefile di compilazione di bash.