

Corso Base

Struttura disordinata del corso

- Definizione e schema di sistema operativo
- Storia di Linux
- Struttura directory e filesystem
- Comandi amministrativi base
- Shell, AWK e sed
- Seminario su VI

il tutto condito da divagazioni ed aneddoti

Il sistema operativo

Collezione di programmi per il coordinamento delle attività e delle componenti funzionali di un calcolatore

Il sistema operativo

Collezione di programmi per il coordinamento delle attività e delle componenti funzionali di un calcolatore

Collezione di programmi

- Nucleo del S.O. (o Kernel)
- Device driver
- Gestore degli accessi
- Gestore dei protocolli di rete
- Programmi di utilità
- Compilatori
- Librerie
- Interfaccia (GUI o testuale)
- ...

Coordinamento delle attività

- Gestione del tempo di processore
- Ripartizione della RAM
- Accesso alla memoria secondaria
- Condivisione d'accesso utente
- Mutua esclusione
- Serializzazione
- Controllo permessi
-

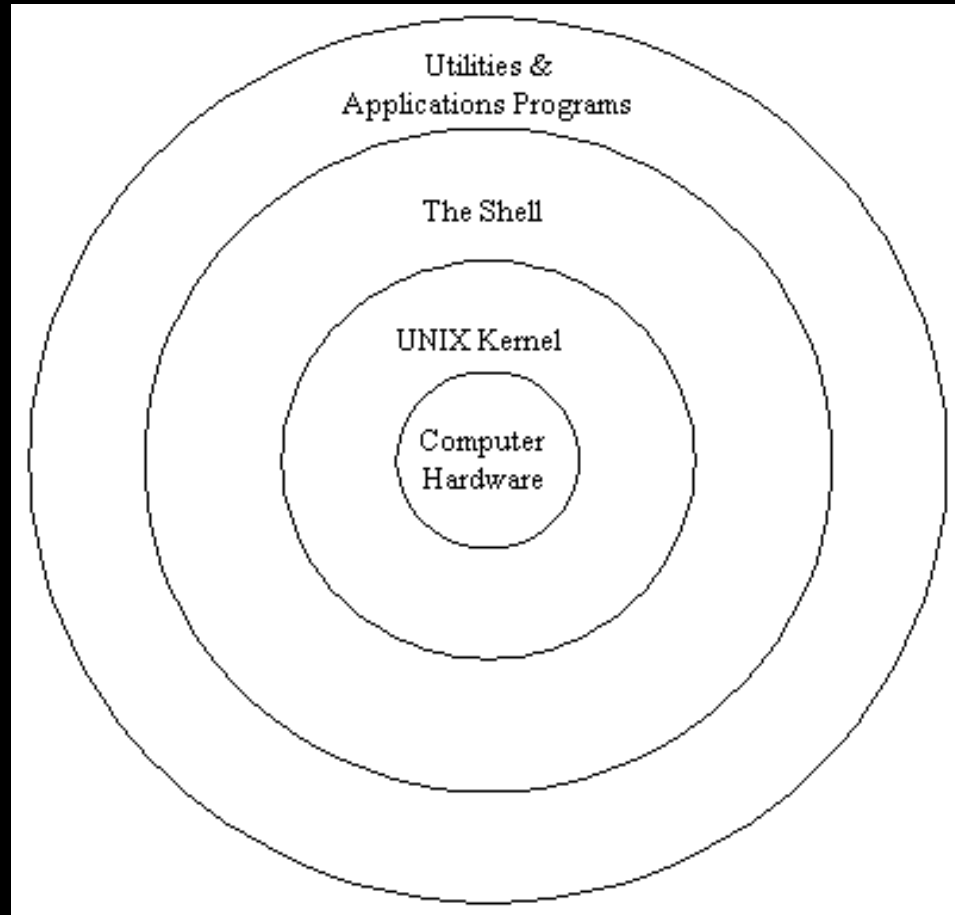
Componenti funzionali

- fruizione contenuti
- elaborazione numerica
- stampe e salvataggi
- scrittura di applicativi
- interazione con applicativi
- attività ludica
- ...

Struttura concettuale

Unix è fatto come una serie di anelli concentrici.

- Più al centro i più privilegiati



Puntualizzazione

Un sistema operativo non è una cosa strettamente necessaria: è “solo” una collezione di routine che istruiscono il processore in che ordine interpretare i byte in RAM, muovere correttamente le testine sull'hard disk ed altre scempiaggini.

Non esiste alcuna ragione teorica per la quale un programmatore sufficientemente determinato non possa reimplementare tutto quanto daccapo

La (lunga) storia di UNIX

- UNIX nasce nel 1970 dalle ceneri del progetto MULTICS. Progetto molto avanzato per l'epoca, proponeva multitasking, portabilità, apertura, memoria virtuale
- L'alternativa più in voga sui server all'epoca era il BESYS. Ogni operazione andava fatta a mano, ogni volta...

La (lunga) storia di UNIX

- Ken Thompson e Dennis Ritchie riaprono il progetto sotto il nome di UNICS su PDP-7



La (lunga) storia di UNIX

Principi di progettazione

1. ogni programma faccia una sola cosa e bene;
2. l'output di un programma diventi l'input di un altro;
3. Il software verrà provato subito: non si esiti a condividere il programma;
4. si usino strumenti appositi nella programmazione e non si cerchi di reinventare la ruota
5. si usi un linguaggio comune alle piattaforme:
viene inventato il C

La (lunga) storia di UNIX

In seguito ad un accordo del 1953, come monopolista delle telecomunicazioni AT&T non poteva commercializzare prodotti informatici.

Quindi AT&T permise che il codice sorgente di Unix venisse distribuito gratuitamente per fini di studio presso le Università di tutto il mondo.

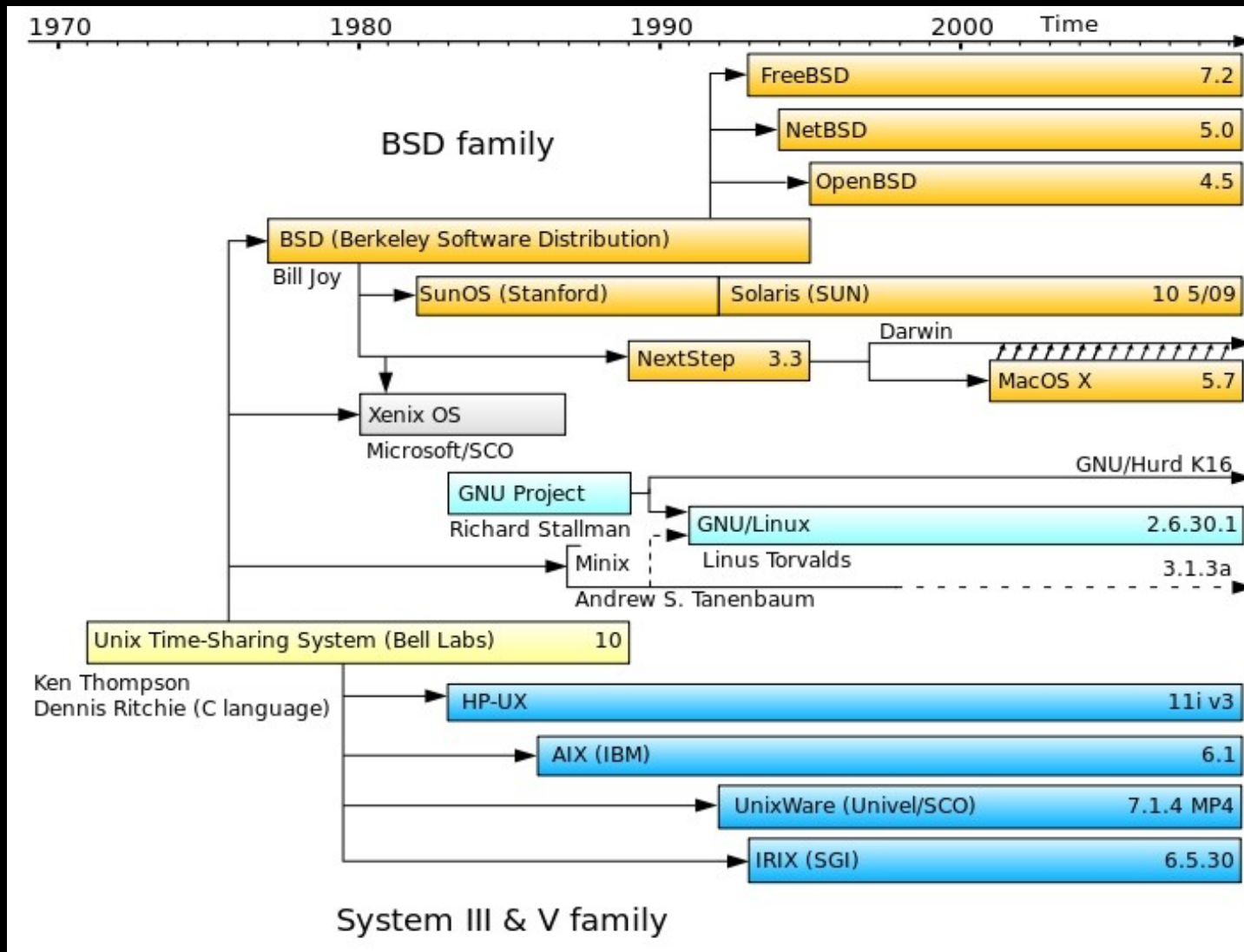
La Berkeley University rilascia BSD

La (lunga) storia di UNIX

Nel 1984 AT&T viene smembrata. Ogni ente in possesso di codice UNIX lo chiude e ne fa una release commerciale (SysIII).

La frammentazione del mercato porta non pochi grattacapi sul fronte de l'interoperatività. Nel 1985 AT&T e l'IEEE rilasciano lo standard POSIX, che detta le linee guida per le interfacce di sistema e di programmazione.

La (lunga) storia di UNIX



La (lunga) storia di UNIX

Nel 1983
Richard
Stallmann
avvia il
progetto
GNU: un
clone di
UNIX libero.

- Non funziona...



La (lunga) storia di UNIX

Nel 1991 uno studentello de l'Università di Helsinki scrive un kernel giocattolo per sistemi Minix e lo chiama *Linux*.

- Lo monta al centro del sistema GNU, il cui kernel HURD faceva acqua e le cui utility erano liberamente disponibili e ricompilabili

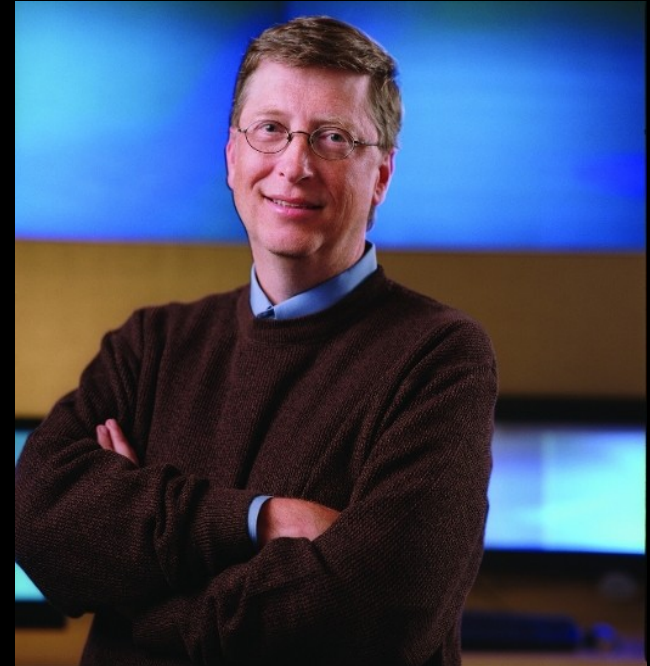


La (lunga) storia di UNIX

postscriptum

La sacra Trinità:

- Stallmann
- Torvalds
- Bill Gates
 - Quest'ultimo ha il merito di aver costruito l'industria dell'home computing, che ha permesso a due squattrinati programmatori di creare il mondo del software libero.



Perchè usare Linux/UNIX?

Una metafora val più di mille parole:
ovvero trapani e OS (by Neal Stephenson)



Il kernel

- **Gestisce:**
 - accesso al filesystem
 - scheduling
 - accesso ai protocolli di rete
- **Fornisce:**
 - API per l'accesso ai dispositivi HW
 - virtualizzazione della memoria
 - strumenti di mutua esclusione

Il filesystem Unix

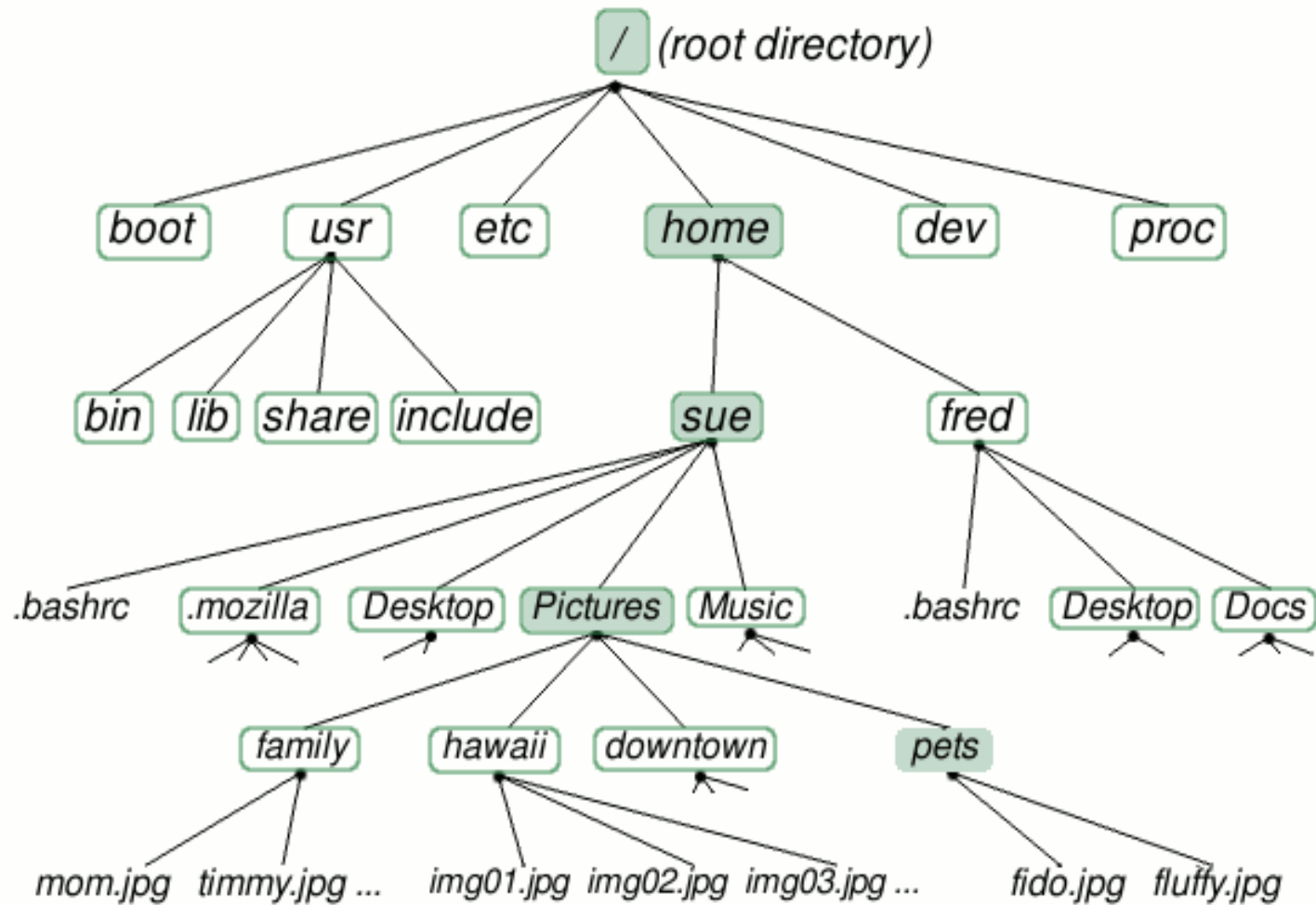
Il filesystem è il complesso sistema che i sistemi operativi moderni utilizzano per organizzare le entità di memorizzazione permanente

- Ogni cosa è un file e i nomi sono case-sensitive
- La struttura logica del FS Unix è un **albero**
- La struttura fisica del FS Unix è fatta di
 - **blocchi**
 - **inode**

Il filesystem Unix inizia da root (/)

- Quello Windows da C:\

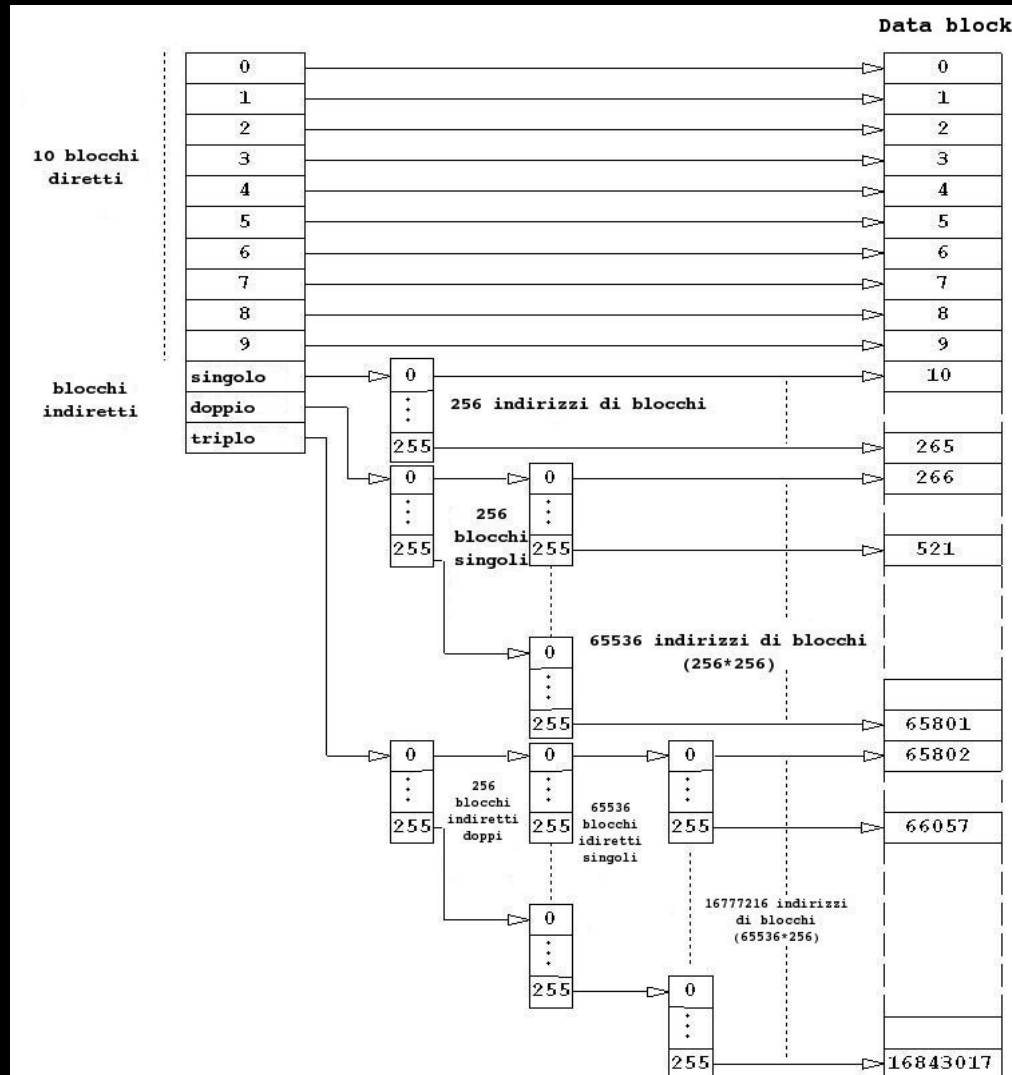
Il filesystem Unix: struttura logica



Il filesystem Unix: blocchi & inode

- Un **blocco** (o **settore**) è un'unità di allocazione indivisibile di dimensione variabile, spesso 512 Bytes.
- Un **inode** è una struttura dati che gestisce, tramite una serie di puntatori gerarchici,
 - la dimensione del file e la sua locazione fisica (se risiede su un dispositivo a blocchi come, ad es., un hard disk)
 - il proprietario e il gruppo di appartenenza
 - le informazioni temporali di modifica (mtime), ultimo accesso (atime) e di cambio di stato (ctime).
 - il numero di collegamenti fisici che referenziano l'inode
 - i permessi d'accesso

Il filesystem Unix: blocchi & inode



Tipi di filesystem e confronti

- XFS
 - FS di IRIX, alte performance su larghi files, con extent, RAID aware, delayed allocation, guaranteed-rate I/O
- ReiserFS (3/4)
 - FS ottimizzato per tanti piccoli file, supporta tail packing, metadata journaling, online resize
- Ext(2/3/4)
 - FS storico di Linux, basato su inode, supporta journaling, delayed allocation ed extents; non richiede deframmentazione
- FAT(16/32/64)
 - FS Microsoft pre-NT, ora standardizzato, usato su flash
- NTFS (6)
 - FS storico di Windows, basato su MFT, journaled, supporta compressione, snapshot, crittografia, transazioni

File ordinari

Rientrano in questa categoria l'80% dei file di uso comune. Possono avere un nome lungo fino a 255 caratteri, che può iniziare e contenere qualunque lettera, numero, spazio, segni di interpunzione.

- Esempio:

`.,asdf. sdfjk,234_ad:`

è un nome di file valido

File ordinari e directory

Gli ordinari si possono creare con:

- ***touch nomefile***
- attraverso ridirezione dell'output
- con un editor (VI, Emacs, OpenOffice =_ =)

Le directory con ***mkdir nomeindir***

Si visualizzano con ***ls (-a per i nascosti)***

```
aijanai@MAGI-6:~/Scaricati$ ls -l
totale 7872
-rw-r--r-- 1 aijanai aijanai 1623502 2010-10-07 11:50 asw120-architetture-software-concetti.pdf
drwxr-xr-x 2 aijanai aijanai 4096 2010-09-27 17:13 CrittografiaTeoriaMegaPack
-rw-r--r-- 1 aijanai aijanai 3150764 2010-09-27 16:56 CrittografiaTeoriaMegaPack.zip
drwxr-xr-x 2 aijanai aijanai 4096 2010-09-27 17:13 Esami_Crittografia
-rw-r--r-- 1 aijanai aijanai 1199935 2010-09-27 16:56 Esami_Crittografia.zip
-rw-r--r-- 1 aijanai aijanai 366381 2010-09-27 18:38 La_bomba.pdf
-rw-r--r-- 1 aijanai aijanai 905363 2010-10-11 15:14 LugRoma3 - Install Fest 2008.pdf
-rw-r--r-- 1 aijanai aijanai 182736 2010-09-27 20:13 PKI.PDF
-rw-r--r-- 1 aijanai aijanai 147352 2010-09-27 18:38 RSA.pdf
-rw-r--r-- 1 aijanai aijanai 395172 2010-09-27 17:22 SmallNetBuilder-WEPCracking[1]...Reloaded.p
df
```

File ordinari: caveats

Occhio solo che:

- il primo carattere non sia un
 - . (nascosto)
 - - (opzione)
- il nome non contenga il
 - ; (fine comando)
 - \$ (variabile d'ambiente)
 - | (system pipe)
 - & (AND logico tra comandi)

File e permessi

Ogni file appartiene a qualcuno, e ha dei permessi d'accesso.

- I permessi sono ordinati per
 - Lettura (r)
 - Scrittura (w)
 - Esecuzione (x)
- e sono specificati per
 - Proprietario (u)
 - Gruppo (g)
 - Resto del mondo (o)

File e permessi

I permessi si leggono con *ls -l*

```
aijanai@MAGI-6:~/Scaricati$ ls -l
totale 7872
-rw-r--r-- 1 aijanai aijanai 1623502 2010-10-07 11:50 asw120-architetture-software-concetti.pdf
drwxr-xr-x 2 aijanai aijanai 4096 2010-09-27 17:13 CrittografiaTeoriaMegaPack
-rw-r--r-- 1 aijanai aijanai 3150764 2010-09-27 16:56 CrittografiaTeoriaMegaPack.zip
drwxr-xr-x 2 aijanai aijanai 4096 2010-09-27 17:13 Esami_Crittografia
-rw-r--r-- 1 aijanai aijanai 1199935 2010-09-27 16:56 Esami_Crittografia.zip
-rw-r--r-- 1 aijanai aijanai 366381 2010-09-27 18:38 La_bomba.pdf
-rw-r--r-- 1 aijanai aijanai 905363 2010-10-11 15:14 LugRoma3 - Install Fest 2008.pdf
-rw-r--r-- 1 aijanai aijanai 182736 2010-09-27 20:13 PKI.PDF
-rw-r--r-- 1 aijanai aijanai 147352 2010-09-27 18:38 RSA.pdf
-rw-r--r-- 1 aijanai aijanai 395172 2010-09-27 17:22 SmallNetBuilder-WEPCracking[1]...Reloaded.p
df
```

-rw-r--r-- 1 aijanai aijanai 1199935 2010-09-27 16:56 Esami_...

- Indica

- rw- lettura, scrittura per il proprietario
- r-- solo lettura per i membri del suo gruppo
- r-- solo lettura per il resto del mondo

File e permessi

Accade che ogni tanto invece della notazione simbolica si incontra la ottale.

I permessi sono espressi come una maschera additiva di bit:

- Lettura (r) 4
- Scrittura (w) 2
- Esecuzione (x) 1

$$rw- r - - r - x = (4+2+0)(4+0+0)(4+0+1)=645$$

File e permessi

- I permessi si cambiano con *chmod*
 - *chmod ug+wr pippo*
 - *chmod 660 pippo*
- É accettata la ricorsione con *-R*
- Nota:
 - le directory per essere accedute hanno bisogno del permesso di esecuzione

File e proprietari

E se volessi cambiare proprietario di un file?

- ***chown*** accetta come parametri il nuovo utente:gruppo
 - eseguibile solo dal proprietario o da root
- É accettata la ricorsione con ***-R***

File speciali: i mount point

- Non abbiamo degli identificatori standard come in DOS per i dispositivi di memorizzazione
- Ma possiamo far corrispondere i dispositivi di memorizzazione con delle directory arbitrarie
- Si dice che il device è *montato* su quella directory

File speciali: i mount point

- Alcuni mount point “famosi” sono in `/etc/fstab`
- Le partizioni
 - `/dev/sdMm`
 - `/dev/srM`
 - `/dev/loopM`

M= major number m=minor number
- Il comando ***mount*** monta e mostra i device montati, assieme ai mount points

File speciali: standard streams

- Facility offerta da UNIX per sgravare il programmatore dalla conoscenza dell'hw
- Sono tubi monodirezionali interconnessi tra il sistema e l'ambiente esterno (tipicamente, un terminale di testo)
 - stdin (standard input)
 - Tutto ciò che inviate al sistema
 - stderr
 - Tutto ciò che è output d'avviso (errori & co.)
 - stdout (standard output)
 - Tutto quello che esce a schermo

Manipolare gli standard streams

- I flussi possono essere redirezionati, sia in entrata che in uscita
 - Una *pipe* | è un tubo che passa l'output di un programma in cascata col l'input di un altro
 - La redirezione > è usata per redirigere l'output in un descrittore di file
 - Usare >> per accodare senza sovrascrivere tutto
 - Con < invece possiamo leggere da un descrittore

Directory di sistema

- Lo standard POSIX prevede la presenza delle seguenti directory di sistema

boot	etc	opt	var	bin	home
sbin	tmp	dev	lib	usr	
sys	proc				

/boot

- Contiene il kernel, la sua configurazione attuale, le mappe dei simboli di sistema e i files del boot manager GRUB

Non cazzeggiate qua dentro

/etc

- Contiene tutti i file di configurazione di sistema
 - Molti programmi memorizzano il file customizzato nella home dell'utente
- Script di avvio e chiusura
- Password (oscurate)

/bin

- Contiene gli eseguibili di base
- cp, rm, mv, ls...
- echo, cat...
- kill, stat...
- Le shell disponibili

/sbin

- /bin per superutenti: Contiene i comandi specifici di pseudo-amministrazione:
 - utility per la rete (setup, probe)
 - utility per i dischi
 - utility per il logging
 - utility per il boot

/usr

- L'equivalente di WINDOWS e Programmi messe insieme. Contiene TUTTI I programmi, in particolare
 - Il sistema grafico X + window manager
 - le utility evolute per la configurazione di:
 - rete
 - stampanti
 - dischi
 - librerie
 - utenti
 - sicurezza

/opt

- Directory nuova, ma poco usata
- Dovrebbe sostituire /usr/local
- Ci vanno programmi che mal si adattano a mettere i propri eseguibili in /bin e le proprie configurazioni in /etc, separatamente.
- Java ha abitato per lungo tempo qui

/tmp

- Una cartella ad accesso universale dove tutti possono andare a scrivere e cancellare
- Tutti i programmi vi scrivono i files temporanei durante la loro esecuzione

/var

- Contiene
 - tutti file di logging
 - i lock dei programmi
 - le cartelle di scaricamento dei gestori di posta

/dev

- Directory dei dispositivi virtuali:
 - dischi e partizioni
 - porte seriali, parallele, ausiliari, . . .
 - frame-buffer, video, ram, . . .
 - terminali seriali e virtuali
 - standard input, output e error
 - zero, null e random

/mnt

- Directory che contiene i mount-point dei dispositivi removibili:
 - floppy
 - cdrom
 - secure digital, pendrive & altre flash card
 - partizioni DOS/Windows :-)

/home

- La propria cartella personale :)
- Il path è
 - quello assoluto
 - /home/nomeutente/
 - oppure quello con ~
 - ~/
- Raggiungibile con il comando **cd** senza parametri o con **cd ~**
- Potete fare *quasi* tutto quello che volete qui

/sys e /proc

- Cartelle generate dinamicamente durante l'attività del kernel che contengono variabili di sistema e notizie sullo stato dei dispositivi e processi
- /sys rimpiazzerebbe /proc, ma anch'essa sta via via sparendo rimpiazzata da debugfs

Comandi per la navigazione

- `cd` (change directory)
 - Vi porta nel path (relativo o assoluto) specificato come parametro
 - Il path `..` porta al livello precedente
 - Il path `.` è un hardlink alla cartella corrente
- `pwd` (present working directory)
 - Vi dice in che cartella vi trovate, stampando il percorso assoluto

Comandi per la manipolazione

- `rm` (remove)
 - Cancella il file specificato come parametro. Con l'opzione `-r` se il target è una dir cancella anche tutto il contenuto, con `-f` non vi chiede neanche la conferma (ottimo per svuotarsi l'intero disco se lanciate `rm -fr /`, fateci attenzione da root)
- `mkdir` (make directory)
 - Crea una directory col nome specificato come parametro; con l'opzione `-p` potete definire più livelli annidati in un colpo solo

Comandi per la manipolazione

- `rmdir` (remove directory)
 - Cancella una directory vuota
- `ln` (link)
 - Crea un collegamento all'inode del file (hardlink). In pratica, esistono due nomi di file che puntano allo stesso file fisico.
Con l'opzione `-s` crea un link simbolico (soft), concettualmente identici ai collegamenti di Windows

Comandi per la manipolazione

- `mv` (move)
 - Sposta un file o lo rinomina
- `cp` (copy)
 - Copia un file
 - `-r` per copiare ricorsivamente; `-a` per copiare preservando tutti gli attributi e tutte le sottocartelle

Comandi per la consultazione

- `cat` (concatenate)
 - Concatena files tra loro e ritorna il risultato su `stdout`
 - Con un solo file, lo mostra e basta su `stdout`
- `less` (e `more`)
 - Pagers che permettono di mostrare un testo troppo lungo per la shell. Potete scorrere su e giù con le frecce, uscire con `q` e cercare con `/` (usare `n` e `N` per next e previous)

Comandi per la consultazione

- **head**
 - Mostra le prime 10 righe di un file (*-n* per un numero arbitrario)
 - Ideale quando vi serve solo l'intestazione di quel log da 10GB che non potete caricare in RAM per leggerlo col blocco note
- **tail**
 - Mostra le ultime 10 righe di un file (*-n* per un numero arbitrario); con *-f* (follow) si “aggancia” al file e pubblica ogni update che viene appeso. Utile per “seguire” l'evoluzione di un file di log

Consultare un manuale

- `man` (manual page)
 - Fondamentale. Vi permette di accedere ad una guida del comando che specificate come parametro in modo molto più dettagliato da quello che otterreste con semplice invocazione del comando con il parametro ***-help***
 - L'output vi viene presentato in un ***less***, quindi avete tutte le funzionalità del comando
- ***“RTFM!” (Read The Fucking Man!)***

Consultare un manuale

- info
 - Una versione ad ipertesti di man; vi permette di saltare da una sezione del manuale all'altra pigiando ENTER sui link contrassegnati da un asterisco
 - Se è presente un link ad un altro man, potete anche saltare su altri manuali, navigando l'intero sistema di manpages come nel web

Cercare un manuale

- Può capitare di cercare una parola, un contesto di applicazione di un comando, un'espressione, un'opzione ma di non sapere o non ricordare quale fosse il comando appropriato
- Tramite *apropos* potete fare una ricerca full text nelle manpages disponibili sul sistema

Esercizio

- Cercate di capire come montare un dispositivo a blocchi tramite ***mount***
- Che cosa fa il comando ***wc*** ?
- Cosa fa ***echo*** ?
- E il comando ***file*** ?
- E ***du*** ?

Troviamo le cose: find e locate

- find
 - Serve per ricercare un file all'interno del filesystem. Cerca discendendo ricorsivamente le directory a partire da quella indicata come parametro. Individua, in questa passeggiata, i file che soddisfano l'espressione immessa come parametro.
 - L'espressione è formata da PRIMARIES ed OPERANDI

Troviamo le cose: find e locate

- PRIMARIES importanti:
 - -delete : cancella tutti i file trovati
 - -empty : ci dice se il file attuale vuoto
 - -iname <pattern> : pattern non case-sensitive
 - -links <n> : ritorna i file che hanno n links
 - -ls : riporta l'output in formato del comando ls -l
 - -perm <nnnn> : riporta i file che hanno il bit dei permessi <nnnn>
 - exec <comando>: esegue un comando su ogni entry

Troviamo le cose: find e locate

La passeggiata di find può essere molto lunga. In un albero molto ramificato come l'intero filesystem può volerci molto!

- locate/slocate/mlocate
 - L'idea è di crearsi un db di tutti i files con il comando **updatedb** (-v per vedere tutta la corsa), che sostanzialmente fa un find in tutte le cartelle;
 - Quindi, si procede a consultare questo db con il comando **locate**, che riporta istantaneamente il risultato

Processi

- Un programma in esecuzione crea un'immagine di sé in memoria, con variabili e parametri d'attivazione associati, chiamata **processo**. Ogni processo ha un identificatore univoco, detto **PID**.
- I processi che girano in background ed esportano servizi si chiamano **daemoni**
- I processi si monitorano e controllano genericamente con apposite utilità:

Controllo dei processi

- `ps` (process' state)
 - Vi mostra lo stato corrente di un processo come PID. Con il parametro *a* vi mostra tutti i processi, anche quelli non vostri; con *u* li mostra in un formato più amichevole; con *x* mostra anche quelli che non hanno una tty (terminale)
- `kill`
 - Manda un segnale ad un processo identificato per PID

Segnali di controllo

- I processi si controllano in maniera generica inviando loro dei segnali.
- Una trattazione completa dei segnali sta nel man di *kill*, i più importanti sono
 - KILL: notifica al processo di fare *harakiri*
 - *CTRL+C*
 - STOP: sospende il processo
 - *CTRL+Z*
 - CONT: riesuma il processo
 - HUP: notifica al processo che la tty è stata chiusa (generalmente da luogo a uscita dei child, a meno di non aver invocato con *nohup*)

Controllo dei processi

- killall
 - Kill a tutti i processi con un certo nome
 - Attenzione se siete su Solaris..!
- top
 - Un task manager per CLI in tutto e per tutto. Con < o > si cicla tra gli ordinamenti per le colonne; con *k* si seleziona un processo da terminare

Controllo dei processi

- jobs
 - Mostra l'elenco di processi sospesi o in bg
- bg (background)
 - Manda un processo sospeso in background, identificandolo per numero job, come si farebbe invocandolo con un &
- fg (foreground)
 - Riporta in primo piano un processo identificandolo per numero job

Moduli

- Il kernel Linux è monoblocco, ma è anche possibile scegliere di compilare certi driver a parte, usabili come fossero “plug-in”. Un ***modulo*** è uno di questi.
- lsmod (list modules)
 - Informazioni sui moduli caricati e dipendenze
- insmod/rmmod (insert/remove module)
 - Inserisce/stacca un modulo nel/dal kernel
- modprobe
 - Util evoluta che fa tutto, con check dipendenze.
Da usare con ***depmod***

Variabili d'ambiente

- La shell è un interprete per un suo linguaggio di programmazione a script, che prevede funzioni e variabili. Il sistema lo sfrutta per memorizzare parametri d'esecuzione dei programmi nelle variabili inizializzabili nel suo linguaggio
- La lista di tutte le variabili si ottiene con il comando *env*
- Un esempio notevole è la variabile *PATH*

Variabili d'ambiente

- La variabile PATH indica tutte le directory dove la shell deve andare a guardare per trovare dei file eseguibili.

```
aijanai@MAGI-6:~$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games
aijanai@MAGI-6:~$
```

- La directory corrente normalmente **NON** viene inclusa nella variabile PATH. È molto insicuro, soprattutto se “in testa” a quelle canoniche. Si può comunque aggiungere in fondo a PATH, o ovviare così (raccomandato):

./nome_eseguibile

Variabili d'ambiente: assegnazione e uso

- Una variabile è definita quando contiene un valore, anche se vuoto.
- L'assegnamento di un valore si ottiene con una dichiarazione del tipo:

```
aijanai@MAGI-6:~$ pluto="cane di Topolino"  
aijanai@MAGI-6:~$ echo $pluto  
cane di Topolino  
aijanai@MAGI-6:~$
```

Variabili d'ambiente: contesti ed esportazione

- Le variabili di shell hanno validità limitata all'ambito della shell stessa: i built-in della shell stessa sono al corrente di queste variazioni, mentre i programmi che vengono avviati non ne risentono.
- Per i programmi esterni le variabili devono essere esportate. L'esportazione delle variabili si ottiene con il comando interno ***export***

Espressioni e filtri

- L'essenza di UNIX è nella linea di comando e nei file di testo ASCII. Per rigirarsi in questo marasma di testi seriali è fondamentale padroneggiare le *espressioni regolari* e i programmi se ne avvalgono per ricercare nei file di testo
 - *Grep*
 - *Sed*
 - *Awk*

grep

- Il comando **grep** stampa, filtrando, solo le linee che soddisfano il pattern
 - -i: non fa differenza fra maiuscole e minuscole
 - -r: ricerca ricorsivamente dentro files e directory
 - -l: mostra solo il nome dell'entità in cui si è trovato il pattern
 - -v: inverte il match (NOT pattern)
 - -c: conta le linee che corrispondono al pattern

Uso base di grep

- Voglio sapere quali utenti dentro `/etc/passwd` utilizzano la shell `/bin/bash`
- `cat /etc/passwd | grep bash`
 - Il comando filtra tutte le linee in cui non compare la parola “bash”

Uso avanzato di grep

- Creiamo un file di testo 'personaggi' con le parole, separate da un a capo “paperone paperino paperoga”
- Lanciamo il comando
`grep ^pap[a-zA-Z]*ne$ personaggi`
- L'***espressione regolare*** usata qua è una stringa di caratteri che matcha più stringhe, in particolare:
 - Cominciano per 'pap'
 - Dopo 'pap' e prima di 'ne' hanno un qualunque numero di caratteri tra 'a' e 'z' e 'A' e 'Z', anche zero
 - Finiscano per 'ne'

Uso avanzato di grep

- Creiamo un altro file di testo con le parole, separate da un a capo “paperoneo epaperone apeperone papirone”
- Lanciamo il comando
grep p.perone\$ personaggi
- Ancora, l'**espressione regolare** usata qua matcha tutte le stringhe che:
 - Hanno da qualche parte una 'p'
 - Seguita da 1 carattere qualunque
 - Il tutto con 'perone' che chiuda la riga

Espressioni regolari

- Un'**espressione regolare** è una stringa che racchiude un intero insieme di stringhe
- È possibile specificare l'insieme tramite operatori matematici (occhio a sintassi “extended” e “basic”):
 - `.` : un carattere qualunque
 - `[...]` : uno dei caratteri inclusi tra parentesi
 - `^` : all'inizio
 - `$` : alla fine
 - `\N` : matcha l'n-esima regexp tra parentesi
 - Es : `'(a)\1'` matcha `'aa'`

Espressioni regolari

- * : il precedente matcha 0 o più volte
- + : il precedente matcha 1 o più volte
- ? : il precedente matcha al più 1 volta
- {n} : il precedente matcha n volte
 - {n,} : n o più volte
 - {,n} : fino ad n volte
 - {n,m}:tra n e m volte

sed

- The Stream editor
- Nell'uso più semplice, nonché ricorrente, si usa per fare search-&-replace su un flusso in input.
 - Ovviamente accetta regexp
- `sed 's/originale/rimpiazzo/g' file > file.new`
 - Non usate mai il file origine come destinazione

awk

- awk è un vero e proprio linguaggio di programmazione per processare testi, fare sostituzioni, conteggi.
- awk legge riga per riga i file ed esegue una o più azioni su tutte le linee che soddisfano certe condizioni.
- Azioni e condizioni sono descritte da un programma, la cui sintassi è simile al C

awk

- Ogni linea è vista come una sequenza di campi separati da tab e/o spazi.
 - Con **-F** potete specificare un separatore di campo diverso.
 - Con **-R** un separatore di linea diverso.
- Il programma in sé compare come input racchiuso da apici singoli
 - Con l'opzione **-f** può essere letto da file

awk -F“ “ -R”\n” 'programma'

awk

- Un *programma* è fatto da 3 sezioni:
BEGIN{code} {main code} END{code}
 - Begin: contiene codice da eseguire prima dell'inizio della scansione; in genere si inizializzano le variabili
 - (*main*): contiene il codice da eseguire su ogni linea (trasformazioni, stampe e/o incrementi contatori)
 - End: codice da eseguire alla fine, come print di statistiche

awk

- Il codice in sé è una lista di
 condizioni → azioni
- Per le azioni, valgono costrutti come ***if***,
while, ***for***, ***break***, ***continue***, ***assegnazioni***
di variabili, ***print*** e ***printf*** ed ***exit***
- Per le variabili, le predefinite comuni sono:
 - \$0 : tutta la linea
 - \$n : n-esimo campo, a partire da \$1
 - NF/NR: numero corrente di campo/riga
- Le user defined non si dichiarano

awk

- **Operatori aritmetici:** +, -, *, /, %, ^, ++, --
- **Operatori logici:** !, &&, ||
- **Operatori di confronto:** <, >, <=, >=, ==, !=
- **Funzioni matematiche:** exp, log, sqrt, sin, . . .
- **Funzioni che operano su stringhe:** length, substr, index, tolower, toupper, . . .
- **Funzioni “bit wise”:** and, or, xor, compl, . . .
- **Funzioni che operano su data/ora:** mktime, strftime

awk

Per ogni linea, stampa il numero, il 1° e il 2° campo, seguiti da '...'; nel frattempo conta le linee e le parole totali e le stampa alla fine

```
BEGIN { print "Scanning file" }  
{  
    printf "line %d: %s %s...\n", NR, $1,$2;  
    lineCount++;  
    wordCount+=NF;  
}  
END {  
    printf "lines=%d, words=%d\n", lineCount, wordCount  
}
```

awk

Stampa il nome file; poi stampa il 1° campo, il 3° e l'ultimo di ogni linea. In fondo c'è la modifica per fargli stampare solo quelli delle linee tra 1 e 4

```
BEGIN { print "Start of file:", FILENAME;}  
{  
    print $1, $3, $NF  
}  
END{ print "End of file" }
```

```
awk 'NR>1 && NR<4 { print NR, $1, $3, $NF }' testo
```

awk

Stampa i campi di ogni linea in ordine inverso

```
{  
    for (i=NF; i>=1; i--)  
        printf "%s", $i;  
    printf "\n";  
}
```

Stampa tutte le linee che hanno una 't' seguita da almeno un carattere ed una 'e'.

Notare che la regexp è racchiusa tra /

```
/t.+e/ { print $0 }
```

awk

Una condizione può essere espressa da 2 espressioni regolari separate da “,”. awk esegue l’azione corrispondente su tutte le linee comprese tra la prima linea che soddisfa la prima espressione alla successiva che soddisfa la seconda espressione. Esempio:

```
/strong/, /clear/ { print $0 }
```

stampa tutte le linee comprese tra la prima linea che contiene la stringa 'strong' e la successiva che contiene la stringa 'clear'.

awk

Il seguente comando fornisce su std output le linee di lunghezza maggiore di 10 caratteri del file prova (length è una funzione predefinita):

```
awk '{ if (length >10) print $0 }' prova
```

Esercizio: Scrivere un comando awk per stampare il numero massimo di campi di una linea in un dato file.

awk

Cosa fa questo programma?

```
{
    nc += length($0) + 1;
    nw += NF;
}
END { print NR, "lines, ", nw, "words, ", nc, "characaters" }
```

awk

'one liners'

```
$3 > 0 { print $1 $2; }
```

```
$2 == "Abrahamson" { print $1 " is nice."; }
```

```
$2 * $3 > 17 { print $5 " is expensive."; }
```

```
NF > 4
```

```
{nf += NF} END { print nf }
```

```
awk 'length($0) > 80'
```

```
{ $2 = ""; print }
```

Per tutto il resto, c'è man awk