Manuale studente
stage invernale 2005

“Il Web Dinamico”

Dael Maselli
&
Mario Masciarelli

1 - Indice

2 - Introduzione

Lo scopo di questo manuale, insieme ad una attività di tutoraggio e laboratorio pratico, è quello di insegnare le basi della programmazione di pagine web dinamiche. Una pagina web dinamica si distingue da una statica in quanto viene generata al momento del suo caricamento secondo dati variabili.

Essendo appunto dinamica è necessario l'utilizzo di un linguaggio di programmazione capace di rendere automatica la generazione di codice **HTML** che, interpretato da un qualsiasi browser (navigatore) web, ne permette la visualizzazione.

Innanzi tutto essendo il web costruito sulle reti **TCP/IP** si imparerà a conoscere questo standard che consente la comunicazione tra computer in reti locali nonché estese come Internet.

Di seguito verrà introdotto il sistema operativo **Unix**, essendo questo uno dei pilastri degli ambienti di rete, tanto da risultare tutt'oggi nettamente in maggior numero per quanto riguarda i server dislocati in Internet, in particolare per quanto riguarda **GNU/Linux**.

Procederemo poi con l'approfondimento del sistema **WWW**, ovvero il protocollo **HTTP** di comunicazione del browser web con i server.

Si imparerà quindi a scrivere codice in linguaggio **HTML**, ed infine si studieranno i linguaggi **PHP** e **Perl** per la generazione dinamica delle pagine.

Nota: Questo manuale è stato compilato in parte copiando completamente paragrafi di altri manuali e documentazioni dal web, nella sezione Bibliografia troverete quindi anche questi testi.

Dael Maselli

3 - Le reti di computer ed il TCP/IP

Prima di iniziare a vedere le particolarità delle reti TCP/IP, conviene introdurre alcuni concetti generali.

In questo contesto, il termine rete si riferisce idealmente a una maglia di collegamenti. In pratica indica un insieme di componenti collegati tra loro in qualche modo a formare un sistema. Ogni nodo di questa rete corrisponde generalmente a un elaboratore, che spesso viene definito *host*

3.1. Estensione

Una rete può essere più o meno estesa; in tal senso si usano degli acronimi standard:

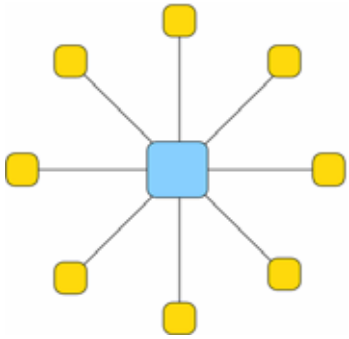
- **LAN, *Local area network*, rete locale**
  quando la rete è contenuta nell'ambito di un edificio, o di un piccolo gruppo di edifici adiacenti;
- **MAN, *Metropolitan area network*, rete metropolitana**
  quando la rete è composta dall'unione di più LAN nell'ambito della stessa area metropolitana, in altri termini si tratta di una rete estesa sul territorio di una città;
- **WAN, *Wide area network*, rete geografica**
  quando la rete è composta dall'unione di più MAN ed eventualmente anche di LAN, estendendosi geograficamente oltre l'ambito di una città singola.
  Evidentemente, Internet è una rete WAN.

3.2. Topologia

Le strutture fondamentali delle reti (si parla in questo caso di ***topologia di rete***) sono di tre tipi:

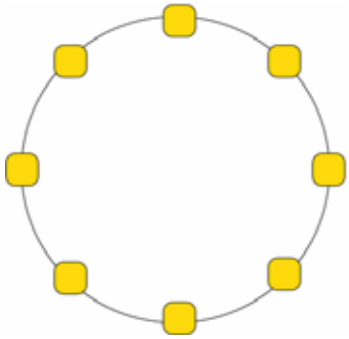
- stella;
- anello;
- bus.

- Si ha una rete a stella quando tutti i nodi periferici sono connessi a un nodo principale in modo indipendente dagli altri. Così, tutte le comunicazioni passano per il nodo centrale e in pratica sono gestite completamente da questo. Rientra in questa categoria il collegamento ***punto-punto***, o *point-to-point*, in cui sono collegati solo due nodi.



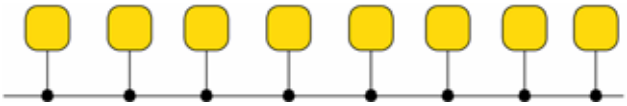
Topologia a stella.

- Si ha una rete ad anello quando tutti i nodi sono connessi tra loro in sequenza, in modo da formare un anello ideale, dove ognuno ha un contatto diretto solo con il precedente e il successivo. In questo modo, la comunicazione avviene (almeno in teoria) a senso unico e ogni nodo ritrasmette al successivo i dati che non sono destinati allo stesso.



Topologia ad anello.

- Si ha una rete a bus quando la connessione dei nodi è condivisa da tutti, per cui i dati trasmessi da un nodo sono intercettabili da tutti gli altri. In questa situazione la trasmissione simultanea da parte di due nodi genera un collisione e la perdita del messaggio trasmesso.



Topologia a bus.

3.3. Pacchetto

I dati viaggiano nella rete in forma di ***pacchetti***. Il termine è appropriato perché si tratta di una sorta di confezionamento delle informazioni attraverso cui si definisce il mittente e il destinatario dei dati trasmessi.

Il confezionamento e le dimensioni dei pacchetti dipendono dal tipo di rete fisica utilizzata.

I dati sono un materiale duttile che può essere suddiviso e aggregato in vari modi. Ciò significa che, durante il loro tragitto, i dati possono essere scomposti e ricomposti più volte e in modi differenti. Per esempio, per attraversare un segmento di una rete particolare, potrebbe essere necessario suddividere dei pacchetti troppo grandi in pacchetti più piccoli, oppure potrebbe essere utile il contrario.

3.4. Protocollo

I pacchetti di dati vengono trasmessi e ricevuti in base a delle regole definite da un ***protocollo di comunicazione***.

A qualunque livello dell'esistenza umana è necessario un protocollo per comunicare: in un colloquio tra due persone, chi parla invia un messaggio all'altra che, per riceverlo, deve ascoltare. Volendo proseguire con questo esempio, si può anche considerare il problema dell'inizio e della conclusione della comunicazione: la persona con cui si vuole comunicare oralmente deve essere raggiunta e si deve ottenere la sua attenzione, per esempio con un saluto; alla fine della comunicazione occorre un modo per definire che il contatto è terminato, con una qualche forma di commiato.

Quanto appena visto è solo una delle tante situazioni possibili. Si può immaginare cosa accada in un'assemblea o in una classe durante una lezione.

La distinzione più importante tra i protocolli è quella che li divide in connessi e non connessi. Il protocollo non connesso, o datagramma, funziona in modo simile all'invio di una cartolina, o di una lettera, che contiene l'indicazione del destinatario ma non il mittente. In tal caso, il protocollo non fornisce il mezzo per determinare se il messaggio è giunto o meno a destinazione. Il protocollo connesso prevede la conferma dell'invio di un messaggio, la ritrasmissione in caso di errore e la ricomposizione dell'ordine dei pacchetti.

3.5. Modello ISO-OSI

La gestione della comunicazione in una rete è un problema complesso; in passato, questo è stato alla base delle maggiori incompatibilità tra i vari sistemi, a cominciare dalle differenze legate all'hardware.

Il modello OSI (*Open system interconnection*), diventato parte degli standard ISO, scompone la gestione della rete in livelli, o strati (*layer*). Questo modello non definisce uno standard tecnologico, ma un riferimento comune ai concetti che riguardano le reti.

I livelli del modello ISO-OSI sono sette e, per tradizione, vanno visti nel modo indicato nell'elenco seguente, dove il primo livello è quello più basso ed è a contatto del supporto fisico di trasmissione, mentre l'ultimo è quello più alto ed è a contatto delle applicazioni utilizzate dall'utente.

Livello	Definizione	Contesto
7	Applicazione	Interfaccia di comunicazione con i programmi (*Application program interface*).
6	Presentazione	Formattazione e trasformazione dei dati a vario titolo, compresa la cifratura e decifratura.
5	Sessione	Instaurazione, mantenimento e conclusione delle sessioni di comunicazione.
4	Trasporto	Invio e ricezione di dati in modo da controllare e, possibilmente, correggere gli errori.
3	Rete	Definizione dei pacchetti, dell'indirizzamento e dell'instradamento in modo astratto rispetto al tipo fisico di comunicazione.
2	Collegamento dati (*data link*)	Definizione delle trame (*frame*) e dell'indirizzamento in funzione del tipo fisico di comunicazione.
1	Fisico	Trasmissione dei dati lungo il supporto fisico di comunicazione.

Modello ISO-OSI

I dati da trasmettere attraverso la rete, vengono prodotti al livello più alto del modello, quindi, con una serie di trasformazioni e aggiungendo le informazioni necessarie, vengono passati di livello in livello fino a raggiungere il primo, quello del collegamento fisico. Nello stesso modo, quando i dati vengono ricevuti dal livello fisico, vengono passati e trasformati da un livello al successivo, fino a raggiungere l'ultimo.

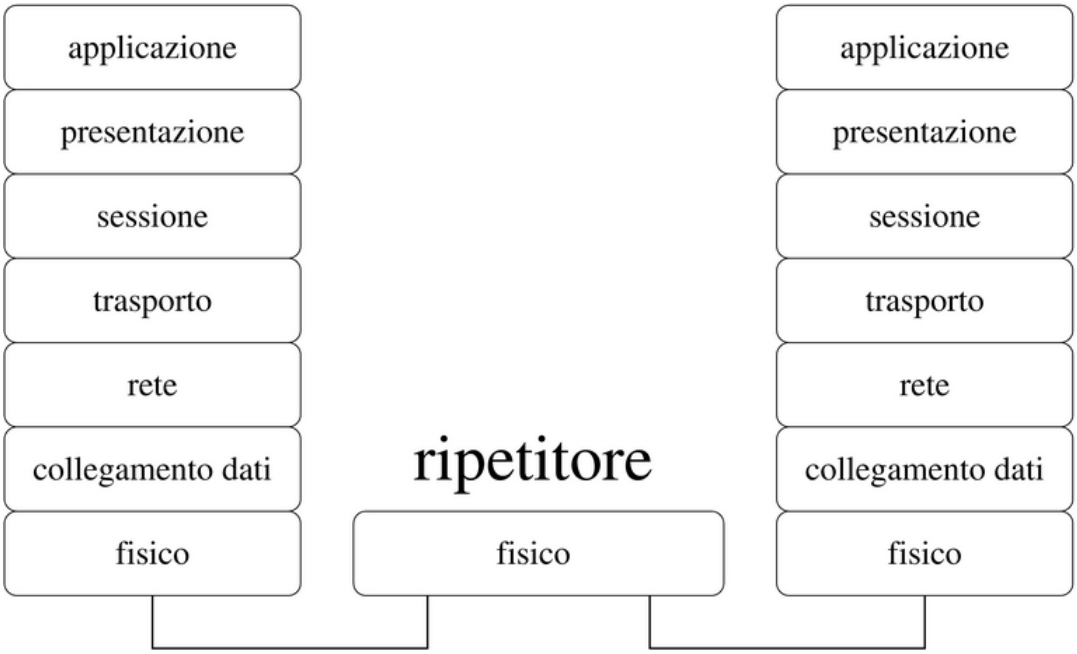
In questo modo, si può dire che a ogni passaggio verso il basso i pacchetti vengano imbustati in pacchetti (più grandi) del livello inferiore, mentre, a ogni passaggio verso l'alto, i pacchetti vengono estratti dalla busta di livello inferiore.

Nel passaggio da un livello a quello inferiore, l'imbustamento implica un aumento delle dimensioni del pacchetto.

3.6. Interconnessione tra le reti

In precedenza sono stati visti i tipi elementari di topologia di rete. Quando si vogliono unire due reti per formarne una sola più grande, si devono utilizzare dei nodi speciali connessi simultaneamente a entrambe le reti da collegare. A seconda del livello su cui intervengono per effettuare questo collegamento, si parla di ripetitore, bridge, router o gateway.

Il ***ripetitore*** è un componente che collega due reti fisiche intervenendo al primo livello ISO-OSI. In questo senso, il ripetitore non filtra in alcun caso i pacchetti, ma rappresenta semplicemente un modo per allungare un tratto di rete che per ragioni tecniche non potrebbe esserlo diversamente. Il ripetitore tipico è l'HUB, ovvero il concentratore di rete.



Il ripetitore permette di allungare una rete,
intervenendo al primo livello del modello ISO-OSI.

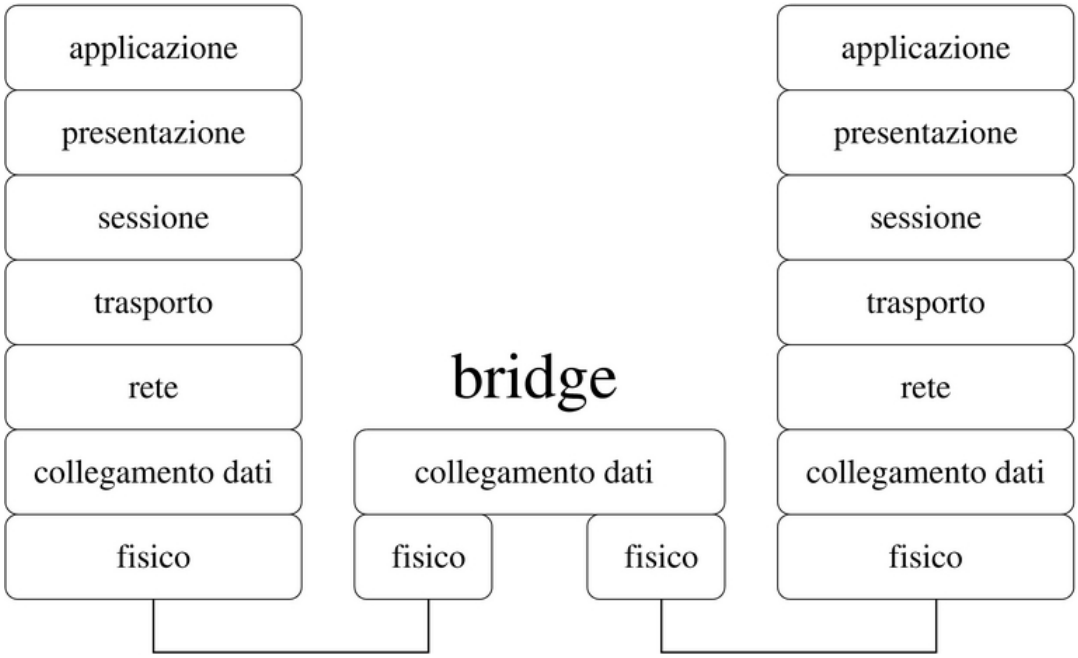
Il ***bridge*** mette in connessione due (o più) reti limitandosi a intervenire nei primi due livelli del modello ISO-OSI. Di conseguenza, il bridge è in grado di connettere tra loro solo reti fisiche dello stesso tipo.

In altri termini, si può dire che il bridge sia in grado di connettere reti separate che hanno uno schema di indirizzamento compatibile.

Il bridge più semplice duplica ogni pacchetto, del secondo livello ISO-OSI, nelle altre reti a cui è connesso; il bridge più sofisticato è in grado di determinare gli indirizzi dei nodi connessi nelle varie reti, in modo da trasferire solo i pacchetti che necessitano questo attraversamento.

Dal momento che il bridge opera al secondo livello ISO-OSI, non è in grado di distinguere i pacchetti in base ai protocolli di rete del terzo livello (TCP/IP, IPX/SPX, ecc.) e quindi trasferisce indifferentemente tali pacchetti.

Teoricamente, possono esistere bridge in grado di gestire connessioni con collegamenti ridondanti, in modo da determinare automaticamente l'itinerario migliore per i pacchetti e da bilanciare il carico di utilizzo tra diverse connessioni alternative. Tuttavia, questo compito viene svolto preferibilmente dai router.

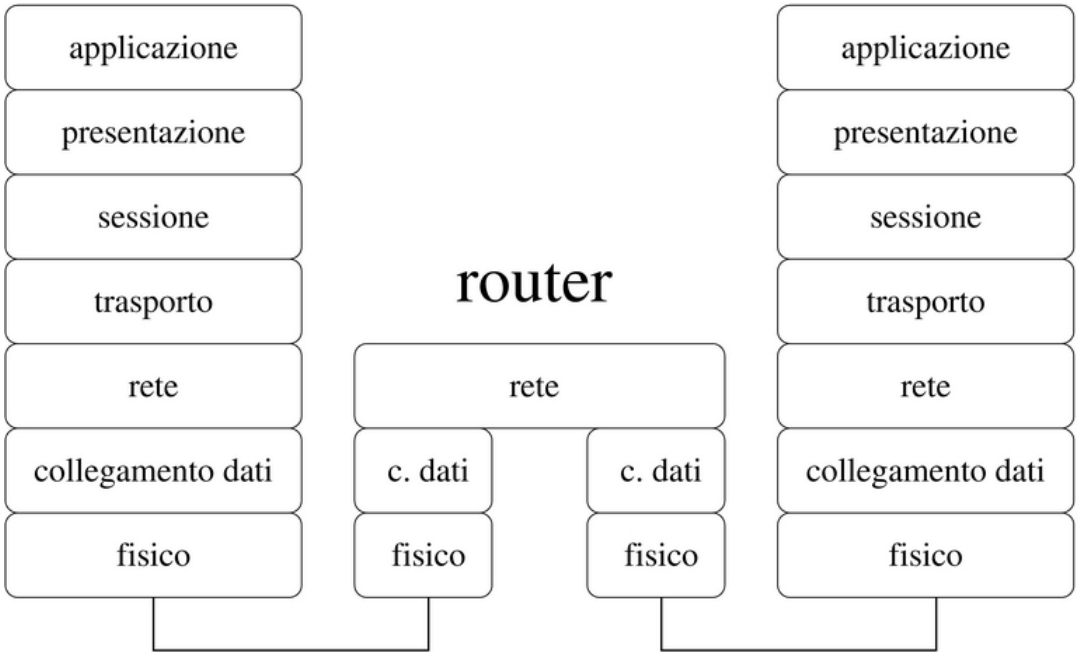


Il bridge trasferisce PDU di secondo livello; in pratica trasferisce tutti i tipi di pacchetto riferiti al tipo di rete fisica a cui è connesso.

Il ***router*** mette in connessione due (o più) reti intervenendo al terzo livello del modello ISO-OSI. Di conseguenza, il router è in grado di trasferire solo i pacchetti di un tipo di protocollo di rete determinato (TCP/IP, IPX/SPX, ecc.), indipendentemente dal tipo di reti fisiche connesse effettivamente.

In altri termini, si può dire che il router sia in grado di connettere reti separate che hanno schemi di indirizzamento differenti, ma che utilizzano lo stesso tipo di protocollo di rete al terzo livello ISO-OSI.

L'instradamento dei pacchetti attraverso le reti connesse al router avviene in base a una tabella di instradamento che può anche essere determinata in modo dinamico, in presenza di connessioni ridondanti, come già accennato per il caso dei bridge.



Il router trasferisce PDU di terzo livello.

Il ***gateway*** mette in connessione due (o più) reti intervenendo ad uno degli ultimi quattro livelli della pila ISO-OSI, ed esegue sempre una conversione di protocollo.

N.B. Erroneamente talvolta si usa il termine *gateway* per indicare il *router*.

3.7. TCP/IP e il modello ISO-OSI

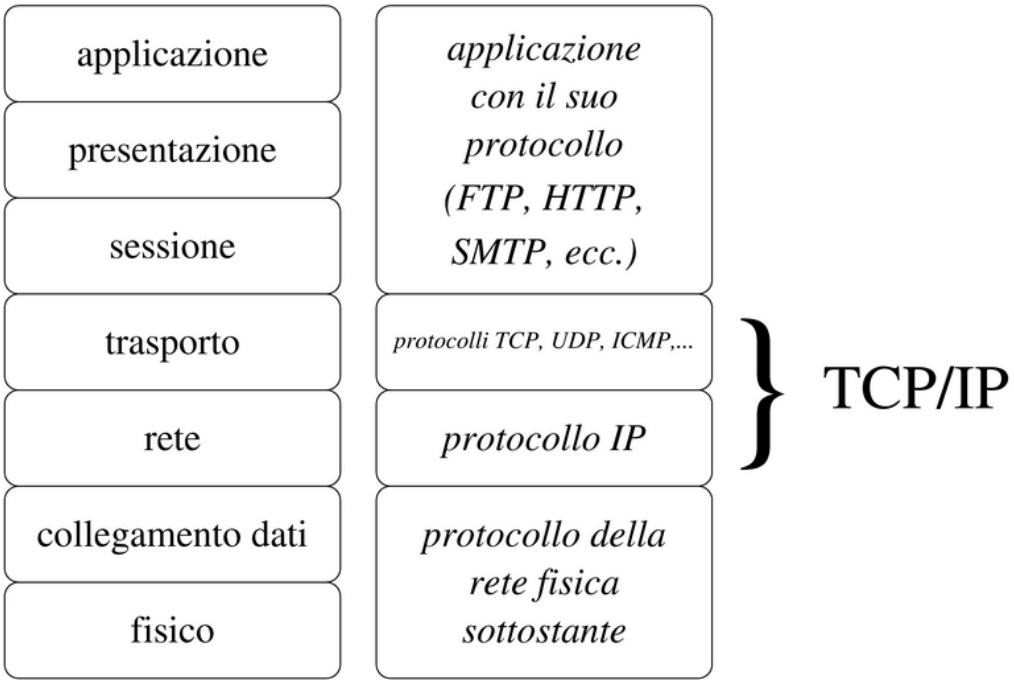
Il nome TCP/IP rappresenta un sistema di protocolli di comunicazione basati su IP. In pratica, il protocollo IP si colloca al terzo livello ISO-OSI, mentre TCP si colloca al di sopra di questo e utilizza IP al livello inferiore. In realtà, il TCP/IP annovera anche un altro protocollo importante: UDP.

I vari aspetti del sistema di protocolli TCP/IP si possono apprendere mano a mano che si studiano gli indirizzamenti e i servizi di rete che vengono resi disponibili. In questa fase conviene rivedere il modello ISO-OSI in abbinamento al TCP/IP.

Livello	Definizione	Descrizione
7	Applicazione	Applicazioni.
6	Presentazione	Definizione standard del formato dei dati utilizzati.
5	Sessione	Protocolli dei servizi (FTP, HTTP, SMTP, RPC, ecc.).
4	Trasporto	Protocolli TCP, UDP e ICMP.
3	Rete	Protocollo IP.
2	Collegamento dati	Trasmissione e ricezione dati dipendente dal tipo di hardware.
1	Fisico	Hardware.

Modello ISO-OSI di suddivisione delle competenze di un sistema TCP/IP.

A parte la descrizione che si fa nel seguito, il TCP/IP vede in pratica solo quattro livelli, che in alcuni casi incorporano più livelli del modello tradizionale.



Esemplificzione dell'abbinamento ISO-OSI e TCP/IP

Questo comunque non significa che gli strati del modello tradizionale non esistono. Piuttosto possono essere svolti all'interno di una sola applicazione, oppure sono al di fuori della competenza del protocollo TCP/IP.

Livello	Definizione	Descrizione
1	fisico	Perché si possa avere una connessione con altri nodi, è necessario inizialmente un supporto fisico, composto solitamente da un cavo e da interfacce di comunicazione. La connessione tipica in una rete locale è fatta utilizzando hardware Ethernet. Il cavo o i cavi e le schede Ethernet appartengono a questo primo livello.
2	collegamento dei dati	Il tipo di hardware utilizzato nel primo livello determina il modo in cui avviene effettivamente la comunicazione. Nel caso dell'hardware Ethernet, ogni scheda ha un proprio indirizzo univoco (stabilito dal fabbricante) composto da 48 bit e rappresentato solitamente in forma esadecimale, come nell'esempio seguente: 00:A0:24:77:49:97
3	rete	Per poter avere un tipo di comunicazione indipendente dal supporto fisico utilizzato, è necessaria un'astrazione che riguarda il modo di inviare blocchi di dati, l'indirizzamento di questi e il loro instradamento. Per quanto riguarda il TCP/IP, questo è il livello del protocollo IP, attraverso il quale vengono definiti gli indirizzi e gli instradamenti relativi. Quando un pacchetto è più grande della dimensione massima trasmissibile in quel tipo di rete fisica utilizzata, è il protocollo IP che si deve prendere cura di scomporlo in segmenti più piccoli e di ricombinarli correttamente alla destinazione.
4	trasporto	A questo livello appartengono i protocolli di comunicazione che si occupano di frammentare e ricomporre i dati, di correggere gli errori e di prevenire intasamenti della rete. I protocolli principali di questo livello sono TCP (*Transmission control protocol*) e UDP (*User datagram protocol*). Il protocollo TCP, in qualità di protocollo connesso, oltre alla scomposizione e ricomposizione dei dati, si occupa di verificare e riordinare i dati all'arrivo: i pacchetti perduti o errati vengono ritrasmessi e i dati finali vengono ricomposti. Il protocollo UDP, essendo un protocollo non connesso, non esegue alcun controllo. A questo livello si introduce, a fianco dell'indirizzo IP, il numero di porta. Il percorso di un pacchetto ha un'origine, identificata dal numero IP e da una porta, e una destinazione identificata da un altro numero IP e dalla porta relativa. Le porte identificano convenzionalmente dei servizi concessi o richiesti e la gestione di questi riguarda il livello successivo.

5	sessione	Ogni servizio di rete (condivisione del file system, posta elettronica, FTP, ecc.) ha un proprio protocollo, porte di servizio e un meccanismo di trasporto (quelli definiti nel livello inferiore). Ogni sistema può stabilire le proprie regole, anche se in generale è opportuno che i nodi che intendono comunicare utilizzino le stesse porte e gli stessi tipi di trasporto. Questi elementi sono stabiliti dal file /etc/services. Segue una riga di questo file dove si può osservare che il servizio www (HTTP) utilizza la porta 80 per comunicare e il protocollo di trasporto è il TCP: www 80/tcp Quando si avvia una comunicazione a questo livello, si parla di sessione. Quindi, si apre o si chiude una sessione.
6	presentazione	I dati che vengono inviati utilizzando le sessioni del livello inferiore devono essere uniformi, indipendentemente dalle caratteristiche fisiche delle macchine che li elaborano. A questo livello si inseriscono normalmente delle librerie in grado di gestire un'eventuale conversione dei dati tra l'applicazione e la sessione di comunicazione.
7	applicazione	L'ultimo livello è quello dell'applicazione che utilizza le risorse di rete. Con la suddivisione delle competenze in così tanti livelli, l'applicazione non ha la necessità di occuparsi della comunicazione; così, in molti casi, anche l'utente può non rendersi conto della sua presenza.

Abbinamento tra il modello ISO-OSI e la semplicità dei protocolli TCP/IP.

3.8. ARP

A livello elementare, la comunicazione attraverso la rete deve avvenire in un modo compatibile con le caratteristiche fisiche di questa. In pratica, le connessioni devono avere una forma di attuazione al secondo livello del modello appena presentato (collegamento dati); i livelli superiori sono solo astrazioni della realtà che c'è effettivamente sotto. Per poter utilizzare un protocollo che si ponga al terzo livello, come nel caso di IP che viene descritto più avanti, occorre un modo per definire un abbinamento tra gli indirizzi di questo protocollo superiore e gli indirizzi fisici delle interfacce utilizzate effettivamente, secondo le specifiche del livello inferiore.

Volendo esprimere la cosa in modo pratico, si può pensare alle interfacce Ethernet, che hanno un sistema di indirizzamento composto da 48 bit. Quando con un protocollo di livello 3 (rete) si vuole contattare un nodo identificato in maniera diversa da quanto previsto al livello 2, se non si conosce l'indirizzo Ethernet, ma ammettendo che tale nodo si trovi nella rete fisica locale, viene inviata una richiesta circolare secondo il protocollo ARP (*Address resolution protocol*).

La richiesta ARP dovrebbe essere ascoltata da tutte le interfacce connesse fisicamente a quella rete fisica e ogni nodo dovrebbe passare tale richiesta al livello 3, in modo da verificare se l'indirizzo richiesto corrisponde al proprio. In questo modo, il nodo che ritiene di essere quello che si sta cercando dovrebbe rispondere, rivelando il proprio indirizzo Ethernet.

Ogni nodo dovrebbe essere in grado di conservare per un certo tempo le corrispondenze tra gli indirizzi di livello 2 con quelli di livello 3, ottenuti durante il funzionamento. Questo viene fatto nella tabella ARP, che comunque va verificata a intervalli regolari.

3.9. Indirizzi IPv4

Come è stato visto nelle sezioni precedenti, al di sopra dei primi due livelli strettamente fisici di comunicazione, si inserisce la rete: un insieme di nodi, spesso definiti *host*, identificati da un indirizzo IP. Di questi ne esistono almeno due versioni: IPv4 e IPv6. Il primo è quello ancora ufficialmente in uso, ma a causa del rapido esaurimento degli indirizzi disponibili nella comunità Internet, è in corso di introduzione il secondo.

Gli indirizzi IP versione 4, cioè quelli tradizionali, sono composti da una sequenza di 32 bit, suddivisi convenzionalmente in quattro gruppetti di 8 bit, rappresentati in modo decimale separati da un punto. Questo tipo di rappresentazione è definito come: ***notazione decimale puntata***. L'esempio seguente corrisponde al codice 1.2.3.4:

```
00000001.00000010.00000011.00000100
```

All'interno di un indirizzo del genere si distinguono due parti: l'indirizzo di rete e l'indirizzo del nodo particolare. Il meccanismo è simile a quello del numero telefonico in cui la prima parte del numero, il prefisso, definisce la zona ovvero il distretto telefonico, mentre il resto identifica l'apparecchio telefonico specifico di quella zona. In pratica, quando viene richiesto un indirizzo IP, si ottiene un indirizzo di rete in funzione della quantità di nodi che si devono connettere. In questo indirizzo una certa quantità di bit nella parte finale sono azzerati: ciò significa che quella parte finale può essere utilizzata per gli indirizzi specifici dei nodi. Per esempio, l'indirizzo di rete potrebbe essere:

```
00000001.00000010.00000011.00000000
```

In tal caso, si potrebbero utilizzare gli ultimi 8 bit per gli indirizzi dei vari nodi.

L'indirizzo di rete, non può identificare un nodo. Quindi, tornando all'esempio, l'indirizzo seguente non può essere usato per identificare anche un nodo:

```
00000001.00000010.00000011.00000000
```

Inoltre, un indirizzo in cui i bit finali lasciati per identificare i nodi siano tutti a uno, identifica un indirizzo ***broadcast***, cioè un indirizzo per la trasmissione a tutti i nodi di quella rete:

```
00000001.00000010.00000011.11111111
```

In pratica, rappresenta simultaneamente tutti gli indirizzi che iniziano con:

`00000001.00000010.00000011.`

Di conseguenza, un indirizzo broadcast non può essere utilizzato per identificare un nodo.

Naturalmente, i bit che seguono l'indirizzo di rete possono anche essere utilizzati per suddividere la rete in sottoreti. Nel caso di prima, volendo creare due sottoreti utilizzando i primi 2 bit che seguono l'indirizzo di rete originario:

xxxxxxxx.*xxxxxxxx*.*xxxxxxxx*.00000000	indirizzo di rete;
xxxxxxxx.*xxxxxxxx*.*xxxxxxxx*.01000000	indirizzo della prima sottorete;
xxxxxxxx.*xxxxxxxx*.*xxxxxxxx*.10000000	indirizzo della seconda sottorete;
xxxxxxxx.*xxxxxxxx*.*xxxxxxxx*.11111111	indirizzo broadcast.

In questo esempio, per ogni sottorete, resterebbero 6 bit a disposizione per identificare i nodi: da 000001_2 a 111110_2.

Il meccanismo utilizzato per distinguere la parte dell'indirizzo che identifica la rete è quello della ***maschera di rete*** o *netmask*. La maschera di rete è un indirizzo che viene abbinato all'indirizzo da analizzare con l'operatore booleano AND, per filtrare la parte di bit che interessano. Prima di vedere come funziona il meccanismo, la seguente tabella può essere utile per ripassare rapidamente le tabelline della verità degli operatori logici principali.

A	B	A AND B	A OR B	NOT A
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0

Riassunto del funzionamento degli operatori logici principali.

Una maschera di rete che consenta di classificare i primi 24 bit come indirizzo di rete è quella seguente, che coincide con il ben più noto codice 255.255.255.0:

`11111111.11111111.11111111.00000000`

Utilizzando l'esempio visto in precedenza, abbinando questa maschera di rete si ottiene l'indirizzo di rete:

00000001.00000010.00000011.00000100	nodo (1.2.3.4)
11111111.11111111.11111111.00000000	maschera di rete (255.255.255.0)
00000001.00000010.00000011.00000000	indirizzo di rete (1.2.3.0).

L'indirizzo che si ottiene abbinando l'indirizzo di un nodo e la sua maschera di rete **invertita** (attraverso l'operatore NOT) con l'operatore AND è l'indirizzo del nodo relativo alla propria rete. Esempio:

00000001.00000010.00000011.00000100	nodo (1.2.3.4)
00000000.00000000.00000000.11111111	Masch. di rete invertita (0.0.0.255)
00000000.00000000.00000000.00000100	indirizzo relativo (0.0.0.4)

Ottetto binario	Ottetto esadecimale	Ottetto decimale
11111111_2	FF_{16}	255_{10}
11111110_2	FE_{16}	254_{10}
11111100_2	FC_{16}	252_{10}
11111000_2	$F8_{16}$	248_{10}
11110000_2	$F0_{16}$	240_{10}
11100000_2	$E0_{16}$	224_{10}
11000000_2	$C0_{16}$	192_{10}
10000000_2	80_{16}	128_{10}
00000000_2	00_{16}	0_{10}

3.9.1. Classi di indirizzi

Gli indirizzi IP sono stati classificati in cinque gruppi, a partire dalla lettera «A» fino alla lettera «E».

Classe A. Gli indirizzi di classe A hanno il primo bit a zero, utilizzano i sette bit successivi per identificare l'indirizzo di rete e lasciano i restanti 24 bit per identificare i nodi.

	Binario	Notazione decimale puntata
modello	0rrrrrrr.hhhhhhhh.hhhhhhhh.hhhhhhhh	
da	00000001._____.____.____	1.___.___.___
a	01111110._____.____.____	126.___.___.___

Classe A

Classe B. Gli indirizzi di classe B hanno il primo bit a uno e il secondo a zero, utilizzano i 14 bit successivi per identificare l'indirizzo di rete e lasciano i restanti 16 bit per identificare i nodi.

	Binario	Notazione decimale puntata
modello	10rrrrrr.rrrrrrrr.hhhhhhhh.hhhhhhhh	
da	10000000.00000001._____.____	128.1.___.___
a	10111111.11111110._____.____	191.254.___.___

Classe B

Classe C. Gli indirizzi di classe C hanno il primo e il secondo bit a uno e il terzo bit a zero, utilizzano i 21 bit successivi per identificare l'indirizzo di rete e lasciano i restanti 8 bit per identificare i nodi.

	Binario	Notazione decimale puntata
modello	110rrrrr.rrrrrrrr.rrrrrrrr.hhhhhhhh	
da	11000000.00000000.00000001._____	192.0.1.___
a	11011111.11111111.11111110._____	223.255.254.___

Classe C

Classe D. Gli indirizzi di classe D hanno i primi tre bit a uno e il quarto a zero. Si tratta di una classe destinata a usi speciali.

	Binario	Notazione decimale puntata
modello	1110xxxx.xxxxxxxx.xxxxxxxx.xxxxxxxx	
da		224.___.___.___
a		239.___.___.___

Classe D

Classe E. Gli indirizzi di classe E hanno i primi quattro bit a uno e il quinto a zero. Si tratta di una classe destinata a usi speciali.

	Binario	Notazione decimale puntata
modello	11110xxx.xxxxxxxx.xxxxxxxx.xxxxxxxx	
da		240.___.___.___
a		247.___.___.___

Classe E

3.9.2. Indirizzi speciali

Lo spazio lasciato libero tra la classe A e la classe B, ovvero gli indirizzi `127.*.*.*`, sono riservati per identificare una rete virtuale interna al nodo stesso. All'interno di questa rete si trova un'interfaccia di rete immaginaria connessa su questa stessa rete, corrispondente all'indirizzo 127.0.0.1, mentre gli altri indirizzi di questo gruppo non vengono mai utilizzati.

Per identificare questi indirizzi si parla di *loopback*, anche se questo termine viene usato ancora in altri contesti con significati differenti.

All'interno di ogni nodo, quindi, l'indirizzo 127.0.0.1 corrisponde a se stesso. Serve in particolare per non disturbare la rete quando un programma (che usa la rete) deve fare riferimento a se stesso.

L'indirizzo speciale 0.0.0.0, conosciuto come *default route* è il percorso, o la strada predefinita per l'instradamento dei pacchetti. Si usa spesso la parola chiave `defaultroute` per fare riferimento automaticamente a questo indirizzo particolare.

3.9.3. Indirizzi riservati per le reti private

Se non si ha la necessità di rendere accessibili i nodi della propria rete locale alla rete globale Internet, si possono utilizzare alcuni gruppi di indirizzi che sono stati riservati a questo scopo e che non corrispondono a nessun nodo raggiungibile attraverso Internet.

Classe		Notazione decimale puntata	Binario
A	da	10.0.0.0	00001010.00000000.00000000.00000000
A	a	10.255.255.255	00001010.11111111.11111111.11111111
B	da	172.16.0.0	10101100.00010000.00000000.00000000
B	a	172.31.255.255	10101100.00011111.11111111.11111111
C	da	192.168.0.0	11000000.10101000.00000000.00000000
C	a	192.168.255.255	11000000.10101000.11111111.11111111

Indirizzi riservati alle reti private.

3.9.4. Sottoreti e instradamento

Quando si scompone la propria rete locale in sottoreti, di solito lo si fa per non intasarla. Infatti è probabile che si possano raggruppare i nodi in base alle attività che essi condividono. Le sottoreti possono essere immaginate come raggruppamenti di nodi separati che di tanto in tanto hanno la necessità di accedere a nodi situati al di fuori del loro gruppo. Per collegare due sottoreti occorre un nodo con due interfacce di rete, ognuno connesso con una delle due reti, configurato in modo da lasciare passare i pacchetti destinati all'altra rete. Questo è un router, che in pratica svolge l'attività di instradamento dei pacchetti.

3.9.5. Maschere IP e maschere di rete

Il modo normale di rappresentare una maschera degli schemi di indirizzamento di IPv4 è quello della notazione decimale puntata a ottetti, come visto fino a questo punto. Tuttavia, considerato che le maschere servono prevalentemente per definire dei gruppi di indirizzi IP, cioè delle reti (o sottoreti), tali maschere hanno una forma piuttosto semplice: una serie continua di bit a uno e la parte restante di bit a zero. Pertanto, quando si tratta di definire una maschera di rete, potrebbe essere conveniente indicare semplicemente il numero di bit da porre a uno. Per esempio, la classica maschera di rete di classe C, 255.255.255.0, equivale a dire che i primi 24 bit devono essere posti a uno.

La possibilità di rappresentare le maschere di rete in questo modo è apparsa solo in tempi recenti per quanto riguarda IPv4. Quindi, dipende dai programmi di servizio utilizzati effettivamente, il fatto che si possa usare o meno questa forma. In ogni caso, il modo normale di esprimerla è quello di indicare il numero IP seguito da una barra obliqua normale e dal numero di bit a uno della maschera, come per esempio 192.168.1.1/24.

Indirizzo iniziale	Indirizzo finale	Impiego
0.0.0.0	--	*Default route*
1.*.*.*	126.*.*.*	Classe A
10.*.*.*	10.*.*.*	Classe A riservata per reti private
127.*.*.*	127.*.*.*	Rete *loopback*
127.0.0.1	--	Indirizzo del nodo locale
128.*.*.*	191.*.*.*	Classe B
172.16.*.*	172.31.*.*	Classe B riservata per reti private
192.*.*.*	223.*.*.*	Classe C
192.168.*.*	192.168.*.*	Classe C riservata per reti private
224.*.*.*	239.*.*.*	Classe D
240.*.*.*	247.*.*.*	Classe E

Riepilogo degli indirizzi IPv4.

3.9.6. Sottoreti particolari in classe C

A causa della penuria di indirizzi IPv4, recentemente si tende a utilizzare la classe C in modo da ottenere il maggior numero di sottoreti possibili. Nella sezione 1.9 è stato mostrato un esempio di suddivisione in sottoreti, in cui si utilizzano 2 bit per ottenere due reti, che possono raggiungere un massimo di 62 nodi per rete, mentre se si trattasse di una rete unica sarebbe possibile raggiungere 254 nodi.

Se si parte dal presupposto che ogni sottorete abbia il proprio indirizzo broadcast, nel senso che non esiste più un indirizzo broadcast generale, si può fare di meglio (anche se la cosa non è consigliabile in generale).

Maschera di rete a 25 bit, pari a 255.255.255.128, per due sottoreti con 126 nodi ognuna:

```
rrrrrrrr.rrrrrrrr.rrrrrrrr.shhhhhhh
```

Rete	IP iniziale	IP finale	Broadcast
x.x.x.0	x.x.x.1	x.x.x.126	x.x.x.127
x.x.x.128	x.x.x.129	x.x.x.254	x.x.x.255

```
rrrrrrrr.rrrrrrrr.rrrrrrrr.sshhhhhh
```

Maschera di rete a 26 bit, pari a 255.255.255.192, per quattro sottoreti con 62 nodi ognuna:

Rete	IP iniziale	IP finale	Broadcast
x.x.x.0	x.x.x.1	x.x.x.62	x.x.x.63
x.x.x.64	x.x.x.65	x.x.x.126	x.x.x.127
x.x.x.128	x.x.x.129	x.x.x.190	x.x.x.191
x.x.x.192	x.x.x.193	x.x.x.254	x.x.x.255

3.10. Routing (instradamento)

Il routing, o instradamento, definisce appunto la strada che devono prendere i pacchetti di livello 3 (rete), secondo il modello ISO-OSI, a partire dal nodo a cui si fa riferimento.

Cominciamo col parlare del routing all'interno della stessa rete, ovvero quello che non si serve del router.

In una rete elementare, in cui ogni elaboratore ha una sola scheda di rete e tutte le schede sono connesse con lo stesso cavo, potrebbe sembrare strana la necessità di dover stabilire un percorso per l'instradamento dei dati sulla rete. Ma in una rete IPv4 non è così: per qualunque connessione possibile è necessario stabilire il percorso, anche quando si tratta di connettersi con l'interfaccia locale immaginaria (*loopback*).

Ogni elaboratore che utilizza la rete ha una sola necessità: quella di sapere quali percorsi di partenza siano possibili, in funzione degli indirizzi utilizzati. Gli eventuali percorsi successivi, vengono definiti da altri elaboratori nella rete. Si tratta di costruire la cosiddetta **tabella di instradamento**, attraverso la quale, ogni elaboratore sa quale strada deve prendere un pacchetto a partire da quella posizione.

La tabella viene definita conoscendo i seguenti dati:

- Rete di destinazione
- Netmask della rete di destinazione
- Router di default per l'instradamento dei pacchetti destinati alla rete
- Interfaccia logica dalla quale far partire i pacchetti per raggiungere il router.

Quando si definisce l'instradamento per la rete della quale fa parte lo stesso nodo non è ovviamente necessario la definizione del router poiché questa sarà raggiungibile attraverso il protocollo di livello 2.

E' inoltre da tenere a mente che il router deve essere raggiungibile a livello 2.

Immaginiamo un nodo con indirizzo `192.168.123.15` con netmask `255.255.255.0`, ne risulta dunque una rete con indirizzo `192.168.123.0` e broadcast `192.168.123.255`.

Supponiamo anche che sulla stessa rete sia presente un router con indirizzo `192.168.123.1` che permetta ai nodi della stessa network di raggiungere computer su altre reti.

La tabella di instradamento configurata sulla macchina in oggetto dovrebbe dunque essere del tipo:

Destinazione	Maschera di rete	Router	Interfaccia
192.168.123.0	255.255.255.0	--	eth0
127.0.0.1	255.255.255.255	--	lo
0.0.0.0	0.0.0.0	192.168.123.1	eth0

Quando il nodo deve inviare un pacchetto ad un altro, viene controllato che la destinazione sia definita nella tabella di instradamento, come ultimo controllo in ordine temporale viene controllato la voce `0.0.0.0` netmask `0.0.0.0` che secondo quanto spiegato in precedenza risulta in `tutti gli indirizzi IPv4`.

Prendendo quindi come esempio un pacchetto con destinazione all'interno della rete `192.168.123.0`, leggendo la tabella di routing il nodo sa che per instradarlo è sufficiente inviarlo attraverso l'interfaccia `eth0` senza che sia necessario che passi per un router; quando invece un pacchetto fosse destinato ad un indirizzo del tipo `206.94.61.185`, questo non risultando all'interno di nessuna definizione tranne l'ultima, verrà inviato al router con indirizzo `192.168.123.1` attraverso l'interfaccia `eth0`.

3.11. Protocolli di trasporto livello: TCP, UDP, ICMP

3.11.1. ICMP

Si tratta di un meccanismo attraverso il quale i router e gli utenti comunicano per sondare eventuali problemi o comportamenti anomali verificatisi in rete. Ricordiamo che il protocollo IP, di per sé, non contiene nessuno strumento per poter riscontrare, da parte della stazione sorgente né destinazione, la perdita di un pacchetto o il collasso di una rete.

L'ICMP consente una comunicazione straordinaria tra router ed host permettendo lo scambio di segnali di errore o di controllo attraverso le interfacce software dell'internet, senza però arrivare su fino al livello degli applicativi; esso è una parte necessaria ed integrante dell'IP ed è contenuto nell'area dati di un datagramma IP.

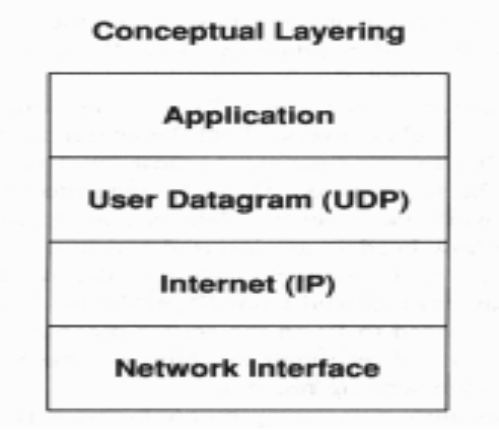
L'ICMP include messaggi di *source quench*, che ritardano il rate di trasmissione, messaggi di *redirect*, che richiedono ad un host di cambiare la propria tabella di routing, e messaggi di *echo request/reply*, che l'host può usare per determinare se la destinazione può essere raggiunta (**ping**).

Un ICMP ha tre campi di lunghezza fissa alla testa del messaggio: il *type field* (8 bit), che identifica il messaggio, il *code field* (8 bit), che contiene informazioni circa il tipo del messaggio, ed il *checksum field* (16 bit). I restanti campi del formato ICMP variano in base al tipo di messaggio.

3.11.2. UDP

Nel complesso del protocollo TCP/IP, l'**User Datagram Protocol** (UDP) fornisce un servizio di recapito dei datagrammi connectionless ed inaffidabile, usando l'IP per trasportare messaggi da una macchina ad un'altra; prevede delle porte di protocollo usate per distinguere tra più programmi in esecuzione (o *processi*) su una singola macchina.

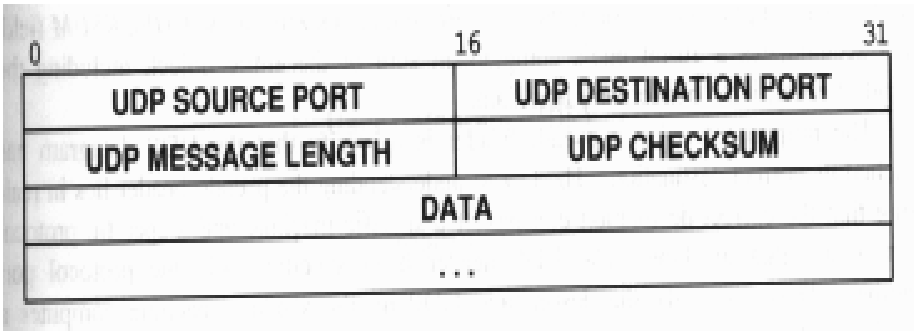
E' un protocollo di trasporto che si colloca sopra l'Internet Protocol Layer (IP):



E' simile all'IP, ma oltre ai dati spediti, ciascun messaggio UDP contiene sia il numero di porta di destinazione che quello di origine, rendendo possibile al software UDP di destinazione di recapitare il messaggio al corretto ricevente (programma od utente), ed a quest'ultimo di inviare una replica.

L'inaffidabilità è quella propria dell'IP, in quanto l'UDP non prevede nessun protocollo per il controllo dell'errore, a differenza del TCP: non usa acknowledgement per assicurare al mittente che i messaggi siano arrivati, non dispone le sequenze di datagrammi in ordine e non fornisce una retroazione per il controllo del rate del flusso di informazioni tra macchine. Perciò i messaggi UDP possono essere persi, duplicati oppure arrivare fuori dall'ordine; inoltre i datagrammi possono arrivare più velocemente di quanto il ricevente sia in grado di processarli.

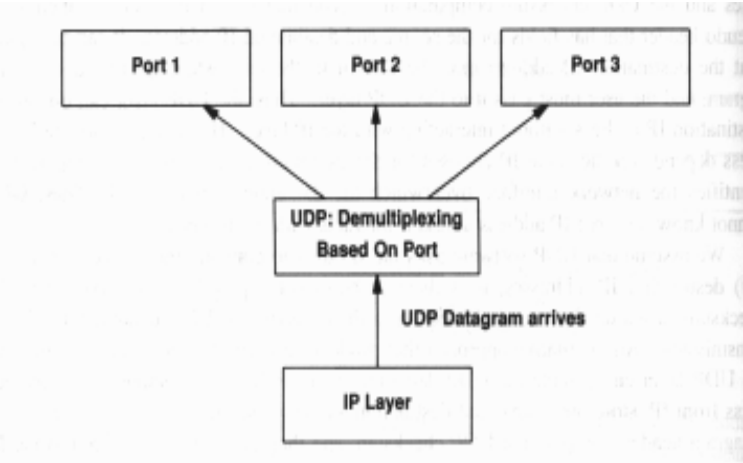
Il protocollo UDP lavora bene in una rete locale, ma potrebbe fallire se usato in una rete di maggiori dimensioni Esso realizza un datagramma più utile di quello IP per i collegamenti fra utenti; infatti, per fare un esempio, su tale base è stato costruito l'applicativo Network File System (NFS) largamente diffuso nelle reti locali. Ciascun messaggio UDP è detto *user datagram* ed è costituito di due parti: UDP header ed UDP data area, che verranno incapsulate nell'IP datagram. Come mostra la figura, l'header dell'UDP è simile a quello del TCP, ma risulta essere più corto non avendo nessun numero di sequenza; è diviso in quattro campi di 16 bit, che specificano il numero di porta da cui il messaggio è stato spedito (opzionale), quello della porta di destinazione (usata per *demultiplexare* i datagrammi tra i processi che aspettano di riceverli), la lunghezza del messaggio ed il checksum.



Concettualmente, tutte le operazioni di ***multiplexing*** e ***demultiplexing*** tra l'UDP ed i programmi applicativi avvengono attraverso il meccanismo delle porte; in pratica, ciascun

programma applicativo deve negoziare con il sistema operativo per ottenere una porta di protocollo e l'associato numero prima che sia spedito un *UDP datagram*.

Assegnata la porta, ogni datagramma che il programma applicativo spedisce attraverso essa avrà quel numero di porta nel suo *UDP source port field*. L'UDP accetta i datagrammi provenienti dal software IP e li *demultiplexa* in base alla *UDP destination port*, come mostra la figura:



3.11.3. Porte UDP

I sistemi operativi della maggior parte dei computer, permettono a più programmi applicativi di essere in esecuzione contemporaneamente (*processes*. *tasks*, *application programs*); tali sistemi sono detti **multitasking systems**.

Potrebbe sembrare naturale che un processo su una particolare macchina sia l'ultima destinazione di un messaggio, ma è più opportuno immaginare che ciascuna macchina contenga un set di punti di destinazione astratti, detti **protocol ports**, identificati ciascuno da un intero positivo. Il sistema operativo locale fornisce un meccanismo di interfaccia che i processi usano per specificare una porta o accedere ad essa.

In generale, le porte sono **bufferizzate**, in modo che i dati arrivati prima che il processo sia pronto, non vengano perduti. Per permettere il buffering, il software di protocollo del sistema operativo posiziona in una **coda** (finita) i pacchetti che arrivano per una particolare porta, finchè il processo non li estrae.

Per comunicare con una porta esterna, il mittente deve conoscere sia l'indirizzo IP che il numero di porta di protocollo della macchina di destinazione. Ciascun messaggio deve contenere sia il numero della **Destination Port** della macchina da cui il messaggio è spedito che il numero della **Source Port** della macchina mittente, a cui la replica dovrà essere indirizzata, rendendo possibile, per ogni processo, il colloquio tra mittente e destinatario.

Ci sono due fondamentali approcci per l'assegnazione delle porte, usando: **Central Authority:** due computer che devono interoperare tra di loro, si accordano per permettere ad un'autorità centrale di assegnare i numeri di porta (***Well-known ports***) che necessitano e di pubblicare la lista di tutte le assegnazioni (*Universal assignment*) il software che gestisce le porte sarà realizzato in base a tale lista.

Dynamic Binding: in questo approccio le porte non sono universalmente conosciute; infatti, se un programma necessita di una porta, è il software di rete ad assegnargliela. Per

sapere la porta corrente assegnata su un altro computer, è necessario inviargli una richiesta del numero di porta assegnata al servizio di interesse.

I progettisti del TCP/IP usano un approccio ibrido che assegna alcuni numeri di porta a priori (*Low values*) e lascia altri disponibili per siti locali o programmi applicativi (*High values*).

La tabella seguente contiene alcune tra le più significative **UDP well-known ports**:

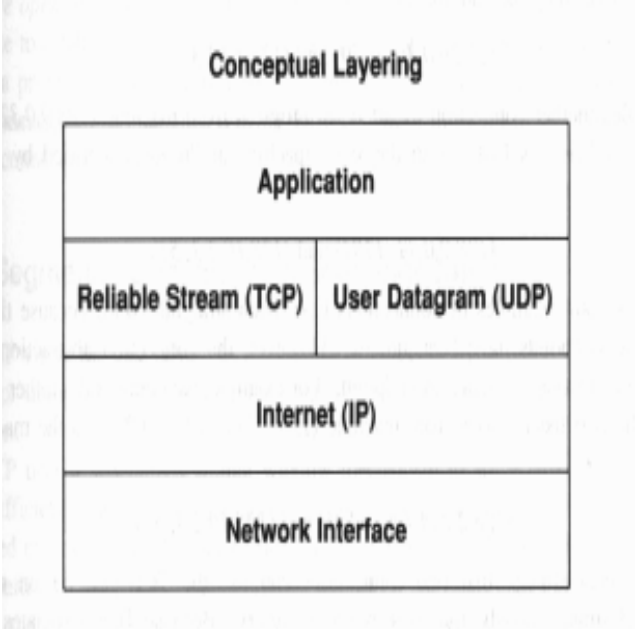
Decimal	Keyword	UNIX Keyword	Description
7	ECHO	echo	Echo
9	DISCARD	discard	Discard
11	USERS	systat	Active Users
42	NAMESERVER	name	Host Name Server
43	NICNAME	whois	Who is
53	DOMAIN	nameserver	Domain Name Server
69	TFTP	tftp	Trivial File Transfer

3.11.4. TCP

Il **Transmission Control Protocol** (TCP), si assume la responsabilità di instaurare un collegamento tra due utenti, di rendere affidabile il trasferimento di dati e comandi tra essi ed infine di chiudere la connessione. Esso è capace di trasferire un flusso continuo di dati fra due utenti in entrambe le direzioni (*full-duplex*), decidendo quando bloccare o continuare le operazioni a suo piacimento.

Poiché il TCP fa veramente poche assunzioni riguardo l'hardware sottostante, è possibile implementarlo sia su una singola rete come una **ethernet** sia su un complesso variegato quale l'**internet**.

Tale protocollo, come l'UDP, si colloca, nel modello a strati, sopra l'Internet Protocol Layer (IP), che gestisce il trasferimento e l'instradamento del singolo pacchetto fino a destinazione, ma, come ulteriore funzionalità, tiene una traccia di ciò che è stato trasmesso ed eventualmente ritrasmette quella parte di informazione che è andata perduta lungo il tragitto.



Come l'UDP, il TCP permette a più programmi applicativi su una stessa macchina di comunicare contemporaneamente, e *demultiplexa* il traffico dei pacchetti in ingresso a tali programmi; usa i numeri di porta per identificare la destinazione finale all'interno di una macchina. La fondamentale differenza con l'UDP è che il TCP garantisce un servizio di trasporto **affidabile** (*Reliable Delivery Service*), ponendo rimedio alle cause di inaffidabilità proprie dell'IP (duplicazione e perdita di dati, caduta di rete, ritardi, pacchetti ricevuti fuori ordine, etc.), anche se ciò comporta una implementazione più complessa. L'importanza dell'affidabilità del flusso permessa da tale protocollo è il motivo per cui il complesso del protocollo TCP/IP ha tale nome.

L'affidabilità di questo servizio è caratterizzata da cinque proprietà:

Stream Orientation: quando due programmi applicativi trasferiscono dati (*stream of bits*), il flusso nella macchina di destinazione passa al ricevente esattamente come è stato originto nella macchina sorgente.

Virtual Circuit Connection: dal punto di vista del programmatore e dell'utente, il servizio che il TCP fornisce è analogo a fornire una connessione dedicata.

Buffered Trasfer: i routers interessati dal trasferimento sono provvisti di buffers per rendere più efficiente il trasferimento e minimizzare il traffico di rete.

Unstructured Stream: il TCP/IP stream service non adotta un flusso di dati strutturato; ovvero non c'è modo di distinguere i records che costituiscono il flusso dati.

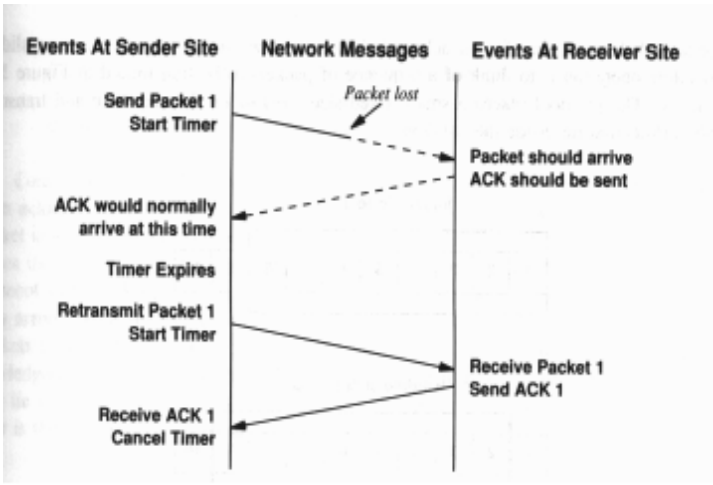
Full-duplex Connection: la connessione fornita dal TCP/IP stream service permette un trasferimento di flusso contemporaneo ed indipendente in entrambe le direzioni, senza apparente interazione.

Se un qualunque messaggio è troppo grande per un singolo pacchetto TCP (gli standard consigliano una dimensione di **576 byte** compreso l'header del IP) si procede a dividerlo in segmenti di lunghezza fissa e poi, arrivato a destinazione, si controlla che siano in ordine e si riassemblano, in modo che tale operazione risulti del tutto invisibile ai due utenti.

Poiché queste funzionalità sono necessarie per molte applicazioni, sono state messe tutte insieme in questo protocollo piuttosto che inserirle, come parte del programma, in ogni applicativo che ne ha bisogno.

L'affidabilità è garantita da una tecnica di fondamentale importanza nota come **acknowledgement with retransmission** (riscontro con ritrasmissione). Tale tecnica prevede che il destinatario invii un messaggio di acknowledgement (**ACK**) al mittente, una volta ricevuto un pacchetto. Il mittente mantiene una copia di ciascun pacchetto spedito e la rimuove dal buffer di trasmissione solo dopo aver ricevuto l'ACK relativo ad essa.

Nella configurazione più banale e meno efficiente l'utente sorgente, dopo aver trasmesso un pacchetto, aspetta di ricevere il suo ACK prima di spedire il successivo; inoltre fa anche partire un "cronometro" per il **timeout**, allo scadere del quale, se non ha ricevuto risposta, ritrasmette quello stesso pacchetto:

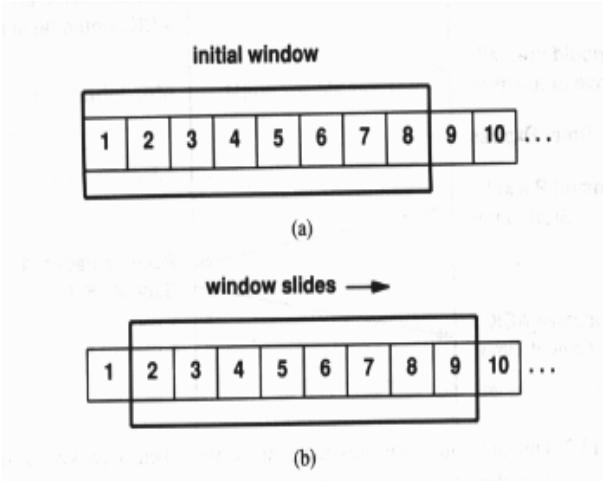


Un semplice protocollo del tipo "***stop and wait***" come questo abbassa notevolmente le prestazioni della rete, sprecando gran parte della banda disponibile nell'attesa dell'ACK relativo al pacchetto precedente; infatti un canale *full-duplex* è utilizzato come se fosse un *half-duplex*.

Tale problema è accentuato se i pacchetti devono attraversare lungo il cammino molti componenti quali **bridge**, **router**, **repeater** che, ovviamente, introducono un ritardo fisso di elaborazione, più una componente dovuta al traffico in rete.

Sliding Windows

L'introduzione del protocollo **Sliding Windows** (finestre scorrevoli) rende molto più efficiente la trasmissione e quindi l'utilizzo della banda, perchè permette al mittente di trasmettere tutti i pacchetti nella finestra senza dover aspettare l'ACK; via via che arrivano i vari ACK, il TCP fa slittare la finestra in avanti trasmettendo dei nuovi pacchetti dinamicamente, come rappresentato in figura:



Le dimensioni della finestra possono variare fino ad un massimo di **64 Kbyte**. Una finestra di ampiezza opportuna riuscirebbe quasi, ipotizzando di non perdere pacchetti, a saturare completamente la banda di trasmissione.

Infatti appena il primo pacchetto della finestra arriva a destinazione, parte subito un ACK: se il **round-trip** di quel collegamento è abbastanza piccolo, o la finestra sufficientemente grande, in modo che l'ACK arrivi prima che il trasmettitore abbia esaurito la finestra, allora il flusso di dati è continuo e pari alle potenzialità massime della rete.

Al contrario, se il round-trip è lento, si può avere la cosiddetta "**silly window syndrome**", che consiste in un comportamento anomalo del TCP. Il trasmettitore spedisce i pacchetti nella finestra e poi perde molto tempo aspettando i relativi ACK prima di passare ai dati successivi, lavorando quindi con una generazione **impulsiva** del carico in rete.

Il meccanismo di acknowledgment può essere un "**go back N**" o un "**go back N selective repeat**", nel senso che le disposizioni dei reference sono molto labili, specificando solo che ogni pacchetto deve essere riconosciuto in qualche modo. Gli standard specificano invece chiaramente che tale meccanismo deve essere **cumulativo**, nel senso che un ACK è relativo ad un certo numero di byte della finestra, non ad un pacchetto, e che gli ACK successivi riconosceranno altri byte in modo cumulativo.

Ad esempio, supposto di lavorare con ACK da 500 byte, il primo ACK conferma i primi 500 byte, il secondo assicura che sono stati ricevuti i primi 1000 byte, il terzo garantisce fino a 1500 byte e così via. Siccome però la dimensione dei pacchetti non è fissata, non è detto che un ACK corrisponda ad un solo pacchetto TCP.

Possiamo allora definire la "***finestra di acknowledgment***" (da non confondere con quella di trasmissione) come il numero di byte, ovvero il numero di pacchetti TCP, una volta fissata la loro dimensione, riconosciuti da un singolo ACK. Avere una finestra di acknowledgment ampia limita il traffico in rete generato dagli ack, poiché lo stesso numero di byte trasmesso viene riconosciuto valido con meno pacchetti ACK.

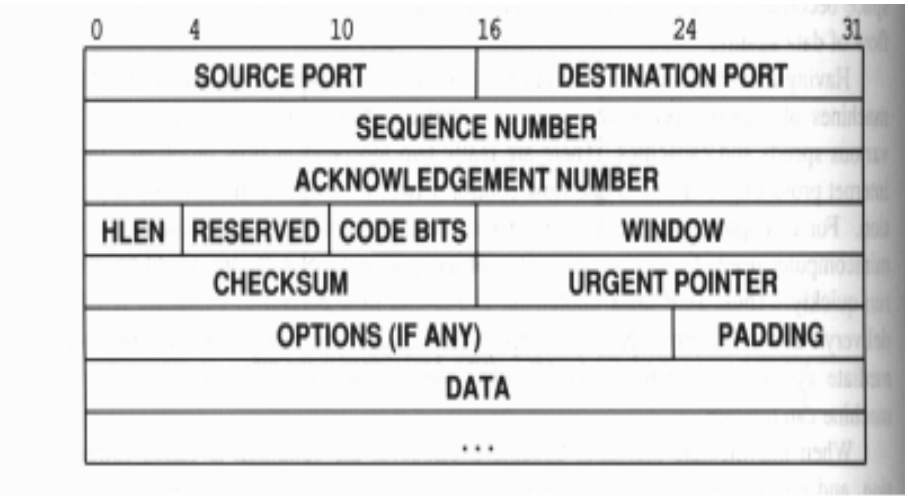
Il meccanismo della finestra è molto importante anche perché fornisce al ricevente un mezzo per governare la mole di dati spediti dall'utente sorgente. Infatti, nell'**header** dei pacchetti TCP esiste un campo specifico, detto "***window***", tramite il quale il ricevente indica al trasmittente la dimensione in byte della finestra che è disposto attualmente a ricevere (**finestra del ricevente**). Questo accordo avviene quando il ricevente spedisce un ACK (che è anche esso un pacchetto TCP) nel quale specifica innanzitutto l'ultima posizione riconosciuta valida e, a partire da questa, il numero di byte che attualmente può accettare.

Headers

L'unità di trasporto tra i software TCP di due macchine è detto **segment**. I segmenti sono scambiati per stabilire connessioni, trasferimenti di dati, inviare ACK, comunicare la dimensione della Sliding Windows e chiudere le connessioni.

Poiché il TCP usa il *piggybacking* (trasmissione contemporanea di dati in entrambe le direzioni), un ACK che viaggia da una macchina A ad una macchina B potrebbe viaggiare in uno stesso segmento in cui viaggiano i dati tra A e B, sebbene l'ACK sia riferito ai dati spediti tra B ed A.

La figura mostra il formato del segmento TCP:



Ciascun segmento è diviso in due parti: un **TCP header** ed un **TCP data**. Un header ha una lunghezza di almeno **20 byte** e comprende molti campi; i più importanti sono sicuramente il "**port number**" e il "**sequence number**", sia della sorgente che della destinazione.

Il numero di porta serve per distinguere fra loro dei trasferimenti che avvengono contemporaneamente; ovviamente devo conoscere anche i numeri di porta degli altri tre nodi.

Il *numero di sequenza* identifica la posizione dei byte dati nel flusso spedito all'interno del segmento; serve per ordinare i pacchetti in ricezione e per verificare di non averne perso nessuno; da notare che tale numerazione riguarda i byte non i pacchetti, nel senso che se si usano pacchetti da **500** byte, il primo è numerato 500, il secondo 1000, il terzo 1500 e così via.

Un altro campo è il "**acknowledgment number**"; anche esso, come il "sequence number", è cumulativo e conta i byte anziché i pacchetti.

Il campo da 2 byte "**window**" è quello che consente al ricevente di indicare al trasmittente la dimensione della *finestra da usare* per il trasferimento in corso; da notare che due alla sedici fa 64 K, cioè la dimensione massima della finestra.
Gli ultimi due campi sono il "**checksum**" dell'header e un "**urgent pointer**" per alcuni casi particolari.

Ovviamente, in ricezione il livello TCP ritaglia l'header TCP, il livello IP ritaglia l'header IP, il livello di rete ritaglia l'header e il checksum relativo ad esso.

3.11.5. Porte TCP

Le porte del TCP sono molto più complesse rispetto a quelle dell'UDP, perchè un dato numero di porta non corrisponde ad un singolo oggetto. Infatti nel TCP gli oggetti da identificare sono delle connessioni di circuito virtuali tra due programmi applicativi, e non delle particolari porte.

Il TCP usa la connessione, e non la porta di protocollo, come sua fondamentale astrazione; le connessioni sono identificate da una coppia di **end points**, ognuno dei quali è costituito da due interi **host,port**, dove l'*host* è l'indirizzo IP dell'host e *port* è il numero di porta TCP su quell'host (per esempio: l'end point **128.10.2.3,25** specifica la porta 25 sulla macchina di indirizzo 128.10.2.3).

Poichè il TCP identifica una connessione con una coppia di valori, uno dato numero di porta può essere condiviso da più connessioni su una stessa macchina, senza che si crei ambiguità. Perciò la macchina identificata da **128.10.2.3,53** può comunicare simultaneamente con le macchine identificate da **128.2.254.139,1184** e **128.9.0.32,1184**.

Si possono così creare servizi concorrenti con connessioni multiple simultanee, senza dover riservare un numero di porta locale per ogni connessione. Per esempio, alcuni sistemi forniscono un accesso concorrente al loro servizio di posta elettronica, permettendo a più utenti di spedire un E-mail contemporaneamente.

Ci sono due fondamentali approcci per l'assegnazione delle porte, usando:

Central Authority: due computer che devono interoperare tra di loro, si accordano per permettere ad un'autorità centrale di assegnare i numeri di porta (***Well-known ports***) che necessitano e di pubblicare la lista di tutte le assegnazioni (*Universal assignment*) il software che gestisce le porte sarà realizzato in base a tale lista.

Dynamic Binding: in questo approccio le porte non sono universalmente conosciute; infatti, se un programma necessita di una porta, è il software di rete ad assegnargliela. Per sapere la porta corrente assegnata su un altro computer, è necessario inviargli una richiesta del numero di porta assegnata al servizio di interesse.

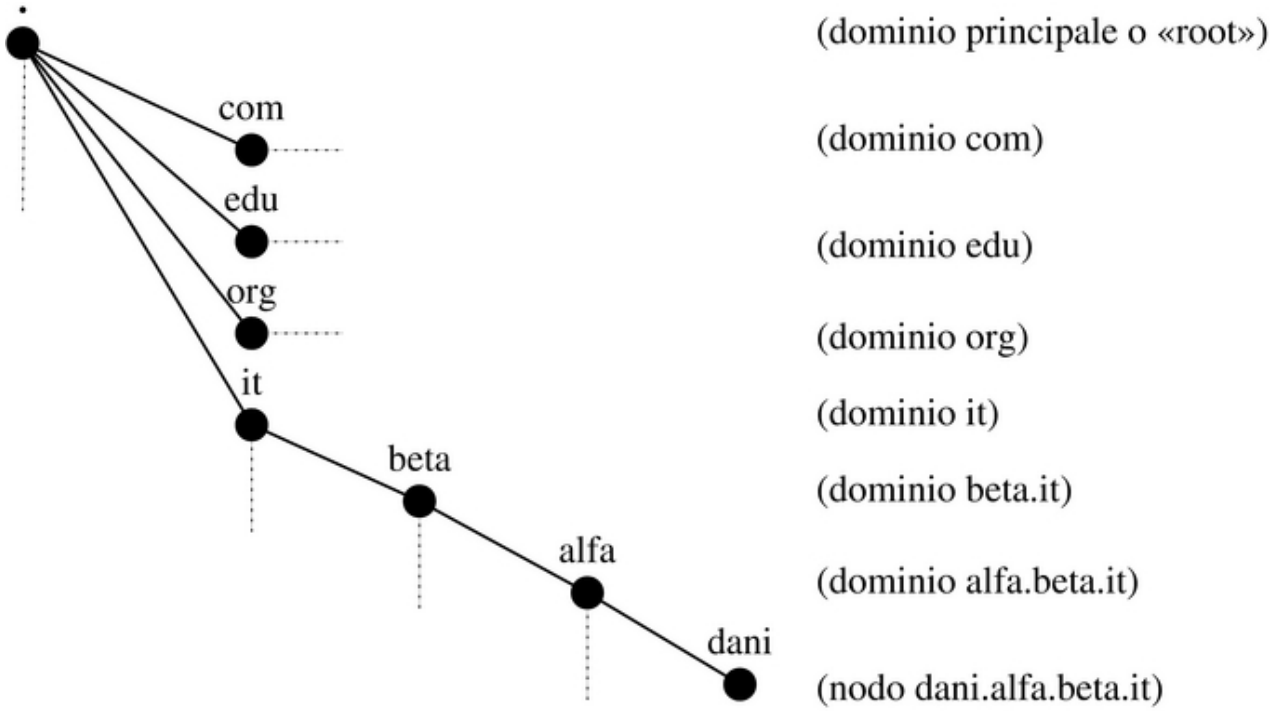
I progettisti del TCP/IP usano un approccio ibrido che assegna alcuni numeri di porta a priori (*Low values*) e lascia altri disponibili per siti locali o programmi applicativi (*High values*).

La tabella seguente contiene alcune tra le più significative **TCP well-known ports**:

Decimal	Keyword	UNIX Keyword	Description
7	ECHO	echo	Echo
9	DISCARD	discard	Discard
11	USERS	systat	Active Users
20	FTP-DATA	ftp-data	File Transfer Protocol (data)
21	FTP	ftp	File Transfer Protocol
23	TELNET	telnet	Terminal connection
25	SMTP	smtp	Simple Mail Transport Protocol
42	NAMESERVER	name	Host Name Server
43	NICNAME	whois	Who is
53	DOMAIN	nameserver	Domain Name Server

3.12. Nomi di dominio

La gestione diretta degli indirizzi IP è piuttosto faticosa dal punto di vista umano. Per questo motivo si preferisce associare un nome agli indirizzi numerici. Il sistema utilizzato è il DNS (*Domain Name System*), ovvero il sistema dei nomi di dominio. Gli indirizzi della rete Internet sono organizzati ad albero in domini, sottodomini (altri sottodomini di livello inferiore, ecc.), fino ad arrivare a identificare il nodo desiderato.



Struttura dei nomi di dominio.

Non esiste una regola per stabilire quante debbano essere le suddivisioni, di conseguenza, di fronte a un nome del genere non si può sapere a priori se si tratta di un indirizzo finale, riferito a un nodo singolo, o a un gruppo di questi.

Con il termine ***nome di dominio***, si può fare riferimento sia al nome completo di un nodo particolare, sia a una parte iniziale di questo, nel lato destro. Dipende dal contesto stabilire cosa si intende veramente. Per fare un esempio che dovrebbe essere più comprensibile, è come parlare di un percorso all'interno di un file system: può trattarsi di una directory, oppure può essere il percorso assoluto che identifica precisamente un file.

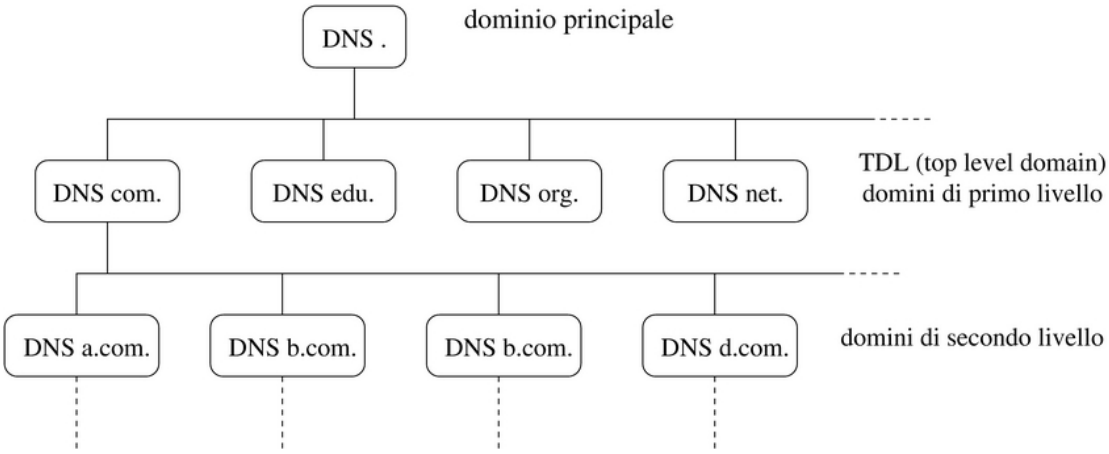
Spesso, all'interno della propria rete locale, è possibile identificare un nodo attraverso il solo nome finale (a sinistra), senza la parte iniziale del dominio di appartenenza. Per esempio, se la rete in cui si opera corrisponde al dominio `brot.dg`, il nodo `roggen` viene inteso essere `roggen.brot.dg`. Quando un nome di dominio contiene tutti gli elementi necessari a identificare un nodo, si parla precisamente di FQDN o *Fully Qualified Domain Name*, quindi `roggen.brot.dg` dell'esempio precedente è un FQDN.

Quando si realizza una rete locale con indirizzi IP non raggiungibili attraverso Internet, è opportuno abbinare nomi di dominio sicuramente inesistenti. Ciò aiuta anche a comprendere immediatamente che non si tratta di un dominio accessibile dall'esterno.

3.12.1. Servizio di risoluzione dei nomi di dominio

In un sistema di nomi di dominio (DNS), il problema più grande è quello di organizzare i *name server* ovvero i ***servizi di risoluzione dei nomi*** (servizi DNS). Ciò è attuato da nodi che si occupano di risolvere, ovvero trasformare, gli indirizzi mnemonici dei nomi di dominio in indirizzi numerici IP e viceversa. A livello del dominio principale (*root*), si trovano alcuni server che si occupano di fornire gli indirizzi per raggiungere i domini successivi, cioè `com`, `edu`, `org`, `net`, `it`,... A livello di questi domini ci sono alcuni server (ogni dominio ha i suoi) che si occupano di fornire gli indirizzi per raggiungere i domini

inferiori, e così via, fino a raggiungere il nodo finale. Di conseguenza, un servizio di risoluzione dei nomi, per poter ottenere l'indirizzo di un nodo che si trova in un dominio al di fuori della sua portata, deve interpellare quelli del livello principale e mano a mano quelli di livello inferiore, fino a ottenere l'indirizzo cercato. Per determinare l'indirizzo IP di un nodo si rischia di dover accedere a una notevole quantità di servizi di risoluzione dei nomi; pertanto, per ridurre il traffico di richieste, ognuno di questi è in grado di conservare autonomamente nella propria cache una certa quantità di indirizzi che sono stati richiesti nell'ultimo periodo. Queste informazioni vengono tenute in memoria per un certo periodo di tempo, questo viene deciso dal server "autoritativo" della rete alla quale appartiene l'host di cui si e' chiesta la risoluzione.



Suddivisione delle competenze tra i vari servizi di risoluzione dei nomi.

In pratica, per poter utilizzare la notazione degli indirizzi suddivisa in domini, è necessario che il sistema locale sul quale si opera possa accedere al suo servizio di risoluzione dei nomi più vicino, oppure gestisca questo servizio per conto suo.

4 - Sistemi Unix

Questo capitolo introduttivo è rivolto a tutti quelli che non hanno avuto esperienze con Unix, ma anche chi ha già una conoscenza di Unix farebbe bene a darci un'occhiata.

Quando si fa riferimento a sistemi Unix, in generale si intende qualcosa che riguarda anche i sistemi GNU/Linux.

4.1. Distinzione tra lettere maiuscole e lettere minuscole

Nei sistemi operativi Unix i nomi dei file sono sensibili alla differenza tra le lettere maiuscole e minuscole. La differenza è sostanziale, per cui gli ipotetici file denominati: `Ciao`, `cIao`, `CIAO`, ecc. sono tutti diversi.

Non bisogna confondere questa caratteristica con quello che può succedere in altri ambienti, come per esempio MS-Windows 95/98/NT/2000, che preservano l'indicazione delle lettere maiuscole o minuscole, ma che poi non fanno differenza quando si vuole fare riferimento a quei file.

Quando in un contesto si fa differenza tra maiuscole e minuscole, capita spesso di vederlo definito come *case sensitive*, mentre per converso, quando non si fa differenza, come *case insensitive*.

4.2. Root

Negli ambienti Unix si fa spesso riferimento al termine ***root*** in vari contesti e con significati differenti. *Root* è la radice, o l'origine, senza altri significati. A seconda del contesto, ne rappresenta l'origine, o il punto iniziale. Per esempio, si può avere:

- una directory *root*, che è la directory principale di un file system, ovvero la directory radice;
- un file system *root*, che è il file system principale di un gruppo che si unisce insieme;
- un utente *root*, che è l'amministratore;
- un dominio *root*, che è il dominio principale;
- una finestra *root* che è quella principale, ovvero la superficie grafica (*desktop*) su cui si appoggiano le altre finestre del sistema grafico X.

Le situazioni in cui si presenta questa definizione possono essere molte di più. L'importante, per ora, è avere chiara l'estensione del significato di questa parola.

4.3. Utenti

Generalmente, i sistemi Unix sono multiutente. La multiutenza implica una distinzione tra i vari utenti. Fondamentalmente si distingue tra l'amministratore del sistema, o *superuser*, e gli altri utenti.

L'amministratore del sistema è quell'utente che può fare tutto ciò che vuole, soprattutto rischia di produrre gravi danni anche solo per piccole disattenzioni.

L'utente comune è quello che utilizza il sistema senza pretendere di organizzarlo e non gli è possibile avviare programmi o accedere a dati che non lo riguardano.

4.3.1. Registrazione dell'utenza

Per poter utilizzare un sistema di questo tipo, occorre essere stati registrati, ovvero, è necessario avere ottenuto un ***account***.

Dal punto di vista dell'utente, l'*account* è un nome abbinato a una parola d'ordine che gli permette di essere riconosciuto e quindi di poter accedere. Oltre a questo, l'*account* stabilisce l'appartenenza a un gruppo di utenti.

Il nome dell'amministratore è sempre `root`, quello degli altri utenti viene deciso di volta in volta.

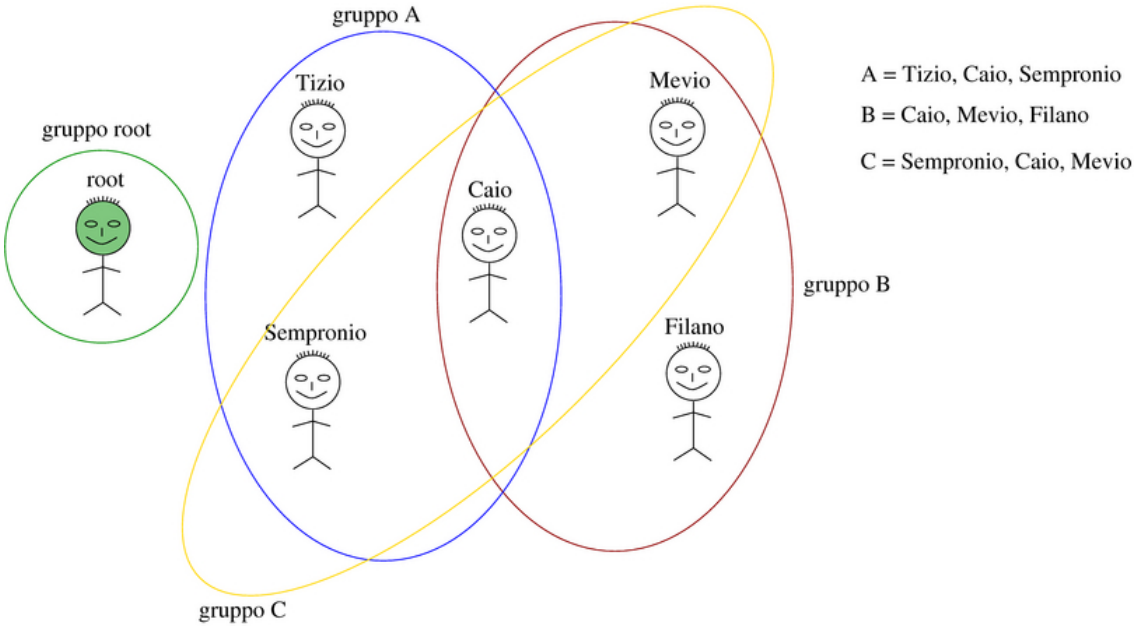
4.3.2. Monoutenza

I sistemi Unix e i programmi che su questi sistemi possono essere utilizzati, non sono predisposti per un utilizzo distratto: gli ordini non vengono discussi. Molti piccoli errori possono essere disastrosi se sono compiuti dall'utente `root`.

È molto importante evitare il più possibile di utilizzare il sistema in qualità di utente amministratore (`root`) anche quando si è l'unico utilizzatore del proprio elaboratore.

4.3.3. Utenti e gruppi

Tutti gli utenti di un sistema Unix sono associati a uno o più gruppi. Un utente ottiene dei privilegi in quanto tale, oppure in quanto appartenente a un certo gruppo.



Utenti e gruppi.

4.4. Composizione

I sistemi Unix sono composti essenzialmente da:

- un sistema di avvio o *boot*;
- un kernel;
- un file system;
- un sistema di inizializzazione e gestione dei processi in esecuzione;
- un sistema di gestione della rete;
- un sistema di gestione delle stampe;
- un sistema di registrazione e controllo degli accessi;
- una shell (interprete dei comandi);
- alcuni programmi di servizio (*utility*) per la gestione del sistema;
- strumenti di sviluppo software (C/C++).

4.4.1. Avvio

Il ***boot*** è il modo con cui un sistema operativo può essere avviato quando l'elaboratore viene acceso. Di solito, il software registrato su ROM degli elaboratori basati sull'uso di dischi, è fatto in modo da eseguire le istruzioni contenute nel primo settore di un dischetto, oppure, in sua mancanza, del cosiddetto MBR (*Master boot record*) che è il primo settore del primo disco fisso. Il codice contenuto nel settore di avvio di un dischetto o del disco fisso, provvede all'esecuzione del kernel (lo avvia).

La parola *boot* è il risultato dell'abbreviazione di *bootstrap*. La parola, usata normalmente al plurale, si riferisce alle linguette degli stivali che servono a calzarli (calzastivali), senza bisogno di aiuto da parte di una seconda persona. Questo fa sì che il termine venga usato in inglese anche per esprimere il concetto di «cavarsela da soli», che spiega il motivo della scelta nel contesto informatico.

In base all'analogia, anche se il suo utilizzo è desueto, è il caso di osservare che il *bootstrap* è il programma o il codice che si prende cura del caricamento del sistema operativo. Pertanto, sarebbe come dire che programmi come LILO e GRUB, installano il *bootstrap*.

Con un sistema GNU installato in un elaboratore i386, la configurazione e la gestione del sistema di avvio viene fatta principalmente attraverso tre modi possibili:

- LILO, che è in grado di predisporre un settore di avvio su un dischetto, sull'MBR o sul primo settore della partizione contenente un sistema GNU/Linux;
- GRUB, che è funzionalmente simile a LILO, ma è adatto anche per GNU/Hurd;
- Loadlin, che permette di avviare l'esecuzione di un kernel Linux da una sessione Dos.

4.4.2. Kernel

Il kernel, come suggerisce il nome, è il nocciolo del sistema operativo. I programmi utilizzano il kernel per le loro attività e in questa maniera sono sollevati dall'agire direttamente con la CPU. Di solito, è costituito da un file unico, il cui nome potrebbe essere `vmlinuz` (oppure `zImage`, `bzImage` e altri), ma può comprendere anche moduli

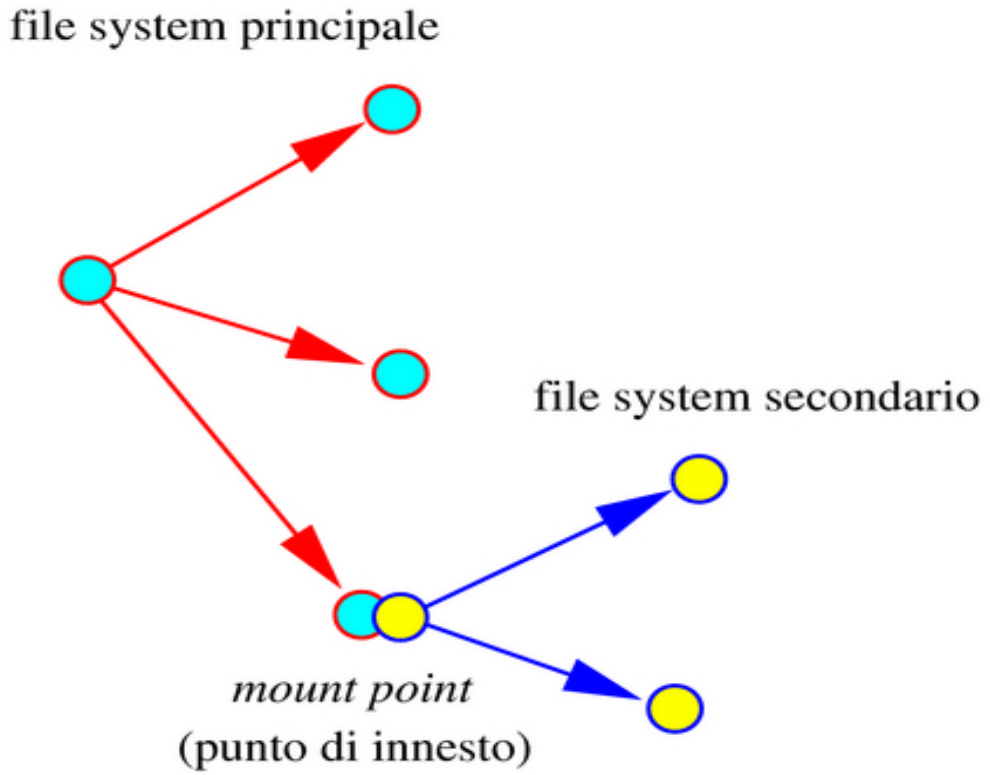
aggiuntivi, per la gestione di componenti hardware specifici che devono poter essere attivati e disattivati durante il funzionamento del sistema.

Quando il kernel viene avviato (attraverso il sistema di avvio), esegue una serie di controlli diagnostici in base ai tipi di dispositivi (componenti hardware) per il quale è stato predisposto, quindi monta (*mount*) il file system principale (*root*) e infine avvia la procedura di inizializzazione del sistema (init).

4.4.3. File system

Il file system è il modo con cui sono organizzati i dati all'interno di un disco o di una sua partizione. Nei sistemi operativi Unix non esiste la possibilità di distinguere tra un'unità di memorizzazione e un'altra, come avviene nel Dos, in cui ogni disco o partizione sono contrassegnati da una lettera dell'alfabeto (`A:`, `B:`, `C:`). Nei sistemi Unix, tutti i file system cui si vuole poter accedere devono essere concatenati assieme, in modo da formare un solo file system globale.

Quando un sistema Unix viene avviato, si attiva il file system principale, o *root*, quindi possono essere collegati a questo altri file system a partire da una directory o sottodirectory di quella principale. Dal momento che per accedere ai dati di un file system diverso da quello principale occorre che questo sia collegato, nello stesso modo, per poter rimuovere l'unità di memorizzazione contenente questo file system, occorre interrompere il collegamento. Ciò significa che, nei sistemi Unix, non si può inserire un dischetto, accedervi immediatamente e toglierlo quando si vuole: occorre dire al sistema di collegare il file system del dischetto, quindi lo si può usare come parte dell'unico file system globale. Al termine si deve interrompere questo collegamento e solo allora si può rimuovere il dischetto.



Collegamento di un file system secondario in corrispondenza di un punto di innesto (o mount point*).*

L'operazione con cui si collega un file system secondario nel file system globale viene detta *mount*, per cui si utilizza normalmente il verbo ***montare*** con questo significato; l'operazione inversa viene detta *unmount* e conseguentemente si utilizza il verbo ***smontare***. La directory a partire dalla quale si inserisce un altro file system è il *mount point*, che potrebbe essere definito come il ***punto di innesto***.

4.4.4. Inizializzazione e gestione dei processi

I sistemi Unix funzionano generalmente in multiprogrammazione, ovvero *multitasking*, pertanto sono in grado di eseguire diversi programmi, o processi elaborativi, contemporaneamente. Per poter realizzare questo, esiste un gestore dei processi elaborativi: init, realizzato in pratica dall'eseguibile `init`, che viene avviato subito dopo l'attivazione del file system principale, allo scopo di occuparsi di eseguire la procedura di inizializzazione del sistema. In pratica, esegue una serie di istruzioni necessarie alla configurazione corretta del sistema particolare che si sta avviando.

4.4.5. Demone

Molti servizi sono svolti da programmi che vengono avviati durante la fase di inizializzazione del sistema e quindi compiono silenziosamente la loro attività. Questi programmi sono detti ***demoni*** (*daemon*).

4.4.6. Gestione dei servizi di rete

Nei sistemi Unix la gestione della rete è un elemento essenziale e normalmente presente. I servizi di rete vengono svolti da una serie di demoni attivati in fase di inizializzazione del sistema. Nei sistemi GNU, i servizi di rete sono controllati fondamentalmente da tre programmi:

il supervisore dei servizi di rete, costituito normalmente dal demone `inetd`, che si occupa di attivare di volta in volta, quando necessario, alcuni demoni che poi gestiscono servizi specifici;

il TCP wrapper, costituito normalmente dal programma eseguibile `tcpd`, che si occupa di controllare e filtrare l'utilizzazione dei servizi offerti dal proprio sistema contro gli accessi indesiderati;

Portmapper, costituito normalmente dal demone `rpc.portmap`, oppure solo `portmap`, che si occupa del protocollo RPC (*Remote procedure call*).

Un servizio molto importante nelle reti locali consente di condividere porzioni di file system da e verso altri elaboratori connessi. Questo si ottiene normalmente con il protocollo NFS che permette quindi di realizzare dei file system di rete.

4.4.7. Registrazione e controllo degli accessi

I sistemi Unix, oltre che essere in multiprogrammazione sono anche multiutente, cioè possono essere usati da più utenti contemporaneamente. La multiutenza dei sistemi Unix è da considerare nel modo più ampio possibile, nel senso che si può accedere all'utilizzo dell'elaboratore attraverso la console, terminali locali connessi attraverso porte seriali (eventualmente anche attraverso la mediazione di modem), terminali locali o remoti connessi attraverso una rete.

In queste condizioni, il controllo dell'utilizzazione del sistema è essenziale. Per questo, ogni utente che accede deve essere stato registrato precedentemente, con un nome e una parola d'ordine, o *password*.

La fase in cui un utente viene riconosciuto e quindi gli viene consentito di agire, è detta ***login***. Così, la conclusione dell'attività da parte di un utente è detta ***logout***.

4.4.8. Shell: interprete dei comandi

Ciò che permette a un utente di interagire con un sistema operativo è la shell, che si occupa di interpretare ed eseguire i comandi dati dall'utente.

Dal punto di vista pratico, il funzionamento di un sistema Unix dipende molto dalla shell utilizzata, di conseguenza, la scelta della shell è molto importante. La shell tipica dei sistemi Unix è una shell derivata da quella di Bourne, possibilmente aderente allo standard POSIX: la shell POSIX. Nei sistemi GNU la shell tipica è Bash (il programma eseguibile `bash`), che è conforme allo standard POSIX.

Una shell Unix normale svolge i compiti seguenti:

- mostra l'invito, o *prompt*, all'inserimento dei comandi;
- interpreta la riga di comando data dall'utente;
- esegue delle sostituzioni, in base ai caratteri jolly e alle variabili di ambiente;
- mette a disposizione alcuni comandi interni;
- mette in esecuzione i programmi;
- gestisce la ridirezione dell'input e dell'output;
- è in grado di interpretare ed eseguire dei file script di shell.

4.4.9. Programmi di servizio per la gestione del sistema

I comandi interni di una shell non bastano per svolgere tutte le attività di amministrazione del sistema. I programmi di servizio sono quelli che di solito hanno piccole dimensioni, sono destinati a scopi specifici di amministrazione del sistema o anche solo di uso comune.

I programmi di servizio di uso comune sono contenuti solitamente all'interno delle directory `/bin/` e `/usr/bin/`. Quelli riservati all'uso da parte dell'amministratore del sistema, l'utente `root`, sono contenuti normalmente in `/sbin/` e `/usr/sbin/` dove la lettera `"s"` iniziale, sta per *superuser*, con un chiaro riferimento all'amministratore.

4.4.10. Strumenti di sviluppo software

Tutti i sistemi operativi devono avere un mezzo per produrre del software. In particolare, un sistema operativo Unix deve essere in grado di compilare programmi scritti in linguaggio C/C++. Gli strumenti di sviluppo dei sistemi Unix, composti da un compilatore in linguaggio C/C++ e da altri programmi di contorno, sono indispensabili per poter installare del software distribuito in forma sorgente non compilata.

4.5. Arresto o riavvio del sistema

Qualunque sistema operativo in multiprogrammazione, tanto più se anche multiutente, deve prevedere una procedura di arresto del sistema che si occupi di chiudere tutte le attività in corso prima di consentire lo spegnimento fisico dell'elaboratore.

Normalmente, in un sistema Unix solo l'utente `root` può avviare la procedura di arresto del sistema con il comando seguente:

```
# shutdown -h now[Invio]
```

Per richiedere il riavvio del sistema:

```
# shutdown -r now[Invio]
```

Con un sistema GNU/Linux installato su un elaboratore con architettura i386, di solito è possibile usare la combinazione di tasti `[Ctrl Alt Canc]` per riavviare il sistema. In questo modo, anche un utente comune ha modo di spegnere il sistema senza creare danni.

4.6. Dispositivi

I vari componenti hardware di un elaboratore, sono rappresentati in un sistema Unix come file di dispositivo, contenuti normalmente nella directory `/dev/` (*device*). Quando si vuole accedere direttamente a un dispositivo, lo si fa utilizzando il nome del file di dispositivo corrispondente.

4.6.1. Tipi

Esistono due categorie fondamentali di file di dispositivo:

- a carattere, cioè in grado di gestire i dati in blocchetti di un solo byte per volta;
- a blocchi, cioè in grado di gestire i dati solo in blocchi (settori) di una dimensione fissa.

Il dispositivo a caratteri tipico è la console o la porta seriale, mentre il dispositivo a blocchi tipico è un'unità a disco. A titolo di esempio, la seguente tabella mostra l'elenco di alcuni nomi di file di dispositivo di un sistema GNU/Linux.

Dispositivo	Descrizione
`/dev/fd0`	la prima unità a dischetti
`/dev/fd0u1440`	unità a dischetti con l'indicazione esplicita del formato: 1 440 Kibyte
`/dev/hda`	il primo disco fisso ATA (IDE)
`/dev/hda1`	la prima partizione del primo disco fisso ATA (IDE)
`/dev/hdb`	il secondo disco fisso ATA (IDE)
`/dev/sda`	il primo disco SCSI
`/dev/sda1`	la prima partizione del primo disco SCSI
`/dev/lp0`	la prima porta parallela dal punto di vista di GNU/Linux
`/dev/lp1`	la seconda porta parallela dal punto di vista di GNU/Linux
`/dev/ttyS0`	la prima porta seriale

Alcuni nomi di dispositivo utilizzati da GNU/Linux.

Alcuni file di dispositivo non fanno riferimento a componenti hardware veri e propri. Il più noto di questi è `/dev/null` utilizzato come fonte per il «nulla» o come pattumiera senza fondo.

4.6.2. Nomi

I nomi utilizzati per distinguere i file di dispositivo, sono stati scelti in base a qualche criterio mnemonico e all'uso più frequente. Tuttavia non è detto che un dispositivo debba chiamarsi in un modo rispetto a un altro.

Sotto questo aspetto, le distribuzioni GNU/Linux non sono tutte uguali: ognuna interpreta in qualche modo questi nomi. Per fare un esempio, il dispositivo corrispondente all'unità a dischetti da 1 440 KByte, può corrispondere a questi nomi differenti:

`/dev/fd0H1440` per la distribuzione Red Hat;

`/dev/fd0u1440` per le distribuzioni Slackware, Debian e SuSE (è anche la sigla indicata nei sorgenti del kernel).

Le cose si complicano ancora di più quando si ha a che fare con sistemi Unix differenti. Quindi, attenzione.

4.7. Dischi

Le unità di memorizzazione a dischi sono dispositivi come gli altri, ma possono essere trattati in due modi diversi a seconda delle circostanze: i dischi, o le partizioni, possono essere visti come dei file enormi o come contenitori di file (file system).

Questa distinzione è importante perché capita spesso di utilizzare dischetti che non hanno alcuna struttura di dati essendo stati usati come se si trattasse di un file unico. Il caso più comune è dato dai dischetti di avvio contenenti solo il kernel Linux: non si tratta di dischetti all'interno dei quali è stato copiato il file del kernel, ma si tratta di dischetti che **sono** il kernel.

La visione che normalmente si ha delle unità di memorizzazione contenenti file e directory è un'astrazione gestita automaticamente dal sistema operativo. Questa astrazione si chiama file system.

4.8. Organizzazione di un file system Unix

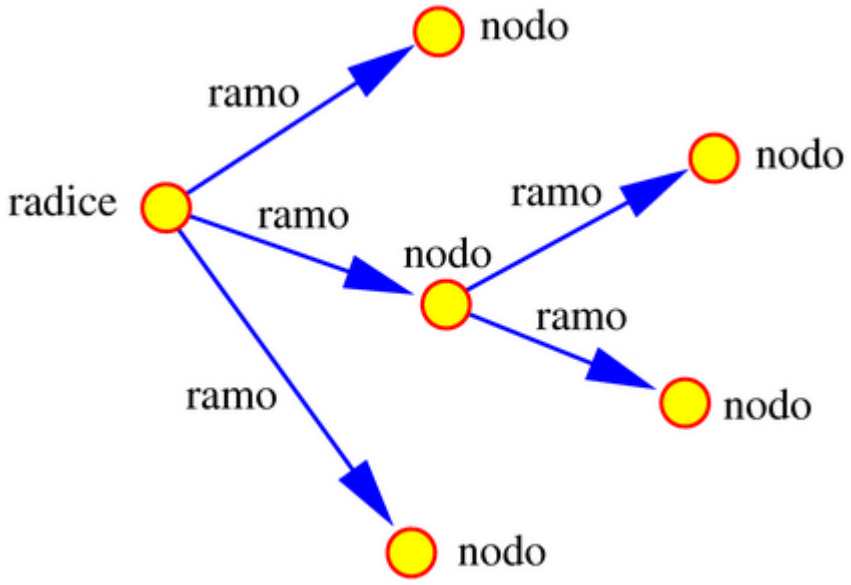
Tutto ciò che è contenuto in un file system Unix è in forma di file: anche una directory è un file.

4.8.1. File normale

Quando si vuole fare riferimento a un file nel senso stretto del termine, ovvero un archivio di dati, se si vuole evitare qualunque ambiguità si utilizza il termine file normale, o *regular file*.

4.8.2. Directory

Una directory è un file speciale contenente riferimenti ad altri file. I dati contenuti in un file system sono organizzati in forma gerarchica schematizzabile attraverso un ***albero***, ovvero un tipo particolare di ***grafo orientato*** che parte da una radice e si sviluppa in rami e nodi. La seguente figura mostra uno schema di un albero:



Albero

La radice è il nodo principale di questo grafo orientato, i rami rappresentano il collegamento (la discendenza) dei nodi successivi con quello di origine (il genitore). La radice corrisponde a una directory, mentre i nodi successivi possono essere directory, file di dati o file di altro genere.

Per identificare un nodo (file o directory) all'interno di questa gerarchia, si definisce il percorso (*path*). Il percorso è espresso da una sequenza di nomi di nodi che devono essere attraversati, separati da una barra obliqua (`/`). Il percorso `idrogeno/carbonio/ossigeno` rappresenta un attraversamento dei nodi `idrogeno`, `carbonio` e `ossigeno`.

Dal momento che il grafo di un sistema del genere ha un nodo di origine corrispondente alla radice, si distinguono due tipi di percorsi: relativo e assoluto.

Percorso relativo

Un percorso è relativo quando parte dalla posizione corrente (o attuale) del grafo per raggiungere la destinazione desiderata. Nel caso dell'esempio precedente,

`idrogeno/carbonio/ossigeno` indica di attraversare il nodo `idrogeno` inteso come discendente della posizione corrente e quindi gli altri.

Percorso assoluto

Un percorso è assoluto quando parte dalla radice.

Il nodo della radice non ha un nome come gli altri: viene rappresentato con una sola barra obliqua (`/`), di conseguenza, un percorso che inizia con tale simbolo, è un percorso assoluto. Per esempio, `/cloro/sodio` indica un percorso assoluto che parte dalla radice per poi attraversare `cloro` e quindi raggiungere `sodio`.

Un albero è un grafo orientato, nel senso che i rami hanno una direzione (archi orientati), ovvero ogni nodo ha un genitore e può avere dei discendenti e il nodo radice rappresenta l'origine. Quando in un percorso si vuole tornare indietro verso il nodo genitore, non si usa il nome di questo, ma un simbolo speciale rappresentato da due punti in sequenza (`..`). Per esempio, `../potassio` rappresenta un percorso relativo in cui si raggiunge il nodo finale, `potassio`, passando prima per il nodo genitore della posizione corrente.

In alcuni casi, per evitare equivoci, può essere utile poter identificare il nodo della posizione corrente. Il simbolo utilizzato è un punto singolo (`.`). Per cui, il percorso `idrogeno/carbonio/ossigeno` è esattamente uguale a `./idrogeno/carbonio/ossigeno`.

4.8.3. Collegamenti

Un albero è tale purché esista uno e un solo percorso dalla radice a un qualunque altro nodo. Nei file system Unix non è necessariamente così; pertanto sono schematizzabili attraverso grafi orientati, ma non necessariamente degli alberi. Infatti è possibile inserire dei collegamenti aggiuntivi, o *link*, che permettono l'utilizzo di percorsi alternativi. Si distinguono due tipi di questi collegamenti: simbolici e fisici (*hard*).

Collegamenti fisici*, *hard link

Un collegamento fisico, o *hard link*, è un collegamento che una volta creato ha lo stesso livello di importanza di quelli originali e non è distinguibile da quelli.

Collegamento simbolico*, *link simbolico*, *symlink

Il collegamento simbolico, o *link* simbolico, è un file speciale contenente un riferimento a un altro percorso e quindi a un altro nodo del grafo di directory e file.

In generale si preferisce l'uso di collegamenti simbolici per poter distinguere la realtà (o meglio l'origine) dalla finzione. Utilizzando un collegamento simbolico si dichiara apertamente che si sta indicando un'alternativa e non si perde di vista il percorso originale.

4.8.4. Nomi dei file

Non esiste una regola generale precisa che stabilisca quali siano i caratteri che possono essere usati per nominare un file. Esiste solo un modo per cercare di stare fuori dai guai: il simbolo `/` non deve essere utilizzato essendo il modo con cui si separano i nomi all'interno di un percorso; inoltre conviene limitarsi all'uso delle lettere dell'alfabeto inglese non accentate, dei numeri, del punto e del trattino basso.

Per convenzione, nei sistemi Unix i file che iniziano con un punto sono classificati come nascosti, perché vengono mostrati e utilizzati solo quando sono richiesti espressamente.

I file che iniziano con un punto sono nascosti per una buona ragione: si vuole evitare che utilizzando i caratteri jolly si faccia riferimento alla directory stessa (`.`) e alla directory genitrice (`..`). Per tale ragione, si deve fare molta attenzione quando si vuole fare riferimento a questi file nascosti. Il comando `rm -r .*` non si limita a eliminare i file e le directory che iniziano con un solo punto iniziale, ma elimina anche `.`, ma soprattutto `..`, cioè tutto il contenuto della directory genitrice!

4.8.5. Permessi

I file di un file system Unix appartengono simultaneamente a un utente e a un gruppo di utenti. Per questo si parla di utente e gruppo proprietari, oppure semplicemente di proprietario e di gruppo.

L'utente proprietario può modificare i permessi di accesso ai suoi file, limitando questi anche per se stesso. Si distinguono tre tipi di accesso: lettura, scrittura, esecuzione. Il significato del tipo di accesso dipende dal file cui questo si intende applicare.

Per i file normali:

- l'accesso in lettura permette di leggerne il contenuto;
- l'accesso in scrittura permette di modificarne il contenuto;
- l'accesso in esecuzione permette di eseguirlo, se si tratta di un eseguibile binario o di uno script di qualunque tipo.

Per le directory:

- l'accesso in lettura permette di leggerne il contenuto, ovvero di poter conoscere l'elenco dei file in esse contenuti (di qualunque tipo essi siano);
- l'accesso in scrittura permette di modificarne il contenuto, ovvero di creare, eliminare e rinominare dei file;
- l'accesso in esecuzione permette di attraversare una directory.

I permessi di un file permettono di attribuire privilegi differenti per gli utenti, a seconda che si tratti del proprietario del file, di utenti appartenenti al gruppo proprietario[(3)], oppure si tratti di utenti diversi. Così, per ogni file, un utente può ricadere in una di queste tre categorie: proprietario, gruppo o utente diverso.

I permessi si possono esprimere in due forme diverse: attraverso una stringa alfabetica o un numero.

I permessi possono essere rappresentati attraverso una stringa di nove caratteri in cui possono apparire le lettere `r`, `w`, `x`, oppure un trattino (`-`). La presenza della lettera `r` indica un permesso di lettura, la lettera `w` indica un permesso di scrittura, la lettera `x` indica un permesso di esecuzione.

I primi tre caratteri della stringa rappresentano i privilegi concessi al proprietario stesso, il gruppetto di tre caratteri successivo rappresenta i privilegi degli utenti appartenenti al gruppo, il gruppetto finale di tre caratteri rappresenta i privilegi concessi agli altri utenti.

I permessi possono essere rappresentati attraverso una serie di tre cifre numeriche, in cui la prima rappresenta i privilegi dell'utente proprietario, la seconda quelli del gruppo e la

terza quelli degli altri utenti. Il permesso di lettura corrisponde al numero quattro, il permesso di scrittura corrisponde al numero due, il permesso di esecuzione corrisponde al numero uno. Il numero che rappresenta il permesso attribuito a un tipo di utente, si ottiene sommando i numeri corrispondenti ai privilegi che si vogliono concedere.

Stringa	Numero	Descrizione
`rw-r--r--`	`644`	L'utente proprietario può accedervi in lettura e scrittura (4+2), mentre sia gli appartenenti al gruppo, sia gli altri utenti, possono solo accedervi in lettura.
`rwxr-x---`	`750`	L'utente proprietario può accedervi in lettura, scrittura ed esecuzione (4+2+1); gli utenti appartenenti al gruppo possono accedervi in lettura e in esecuzione (4+1); gli altri utenti non possono accedervi in alcun modo.
`rw-------`	`600`	L'utente proprietario può accedervi in lettura e scrittura (4+2), mentre tutti gli altri non possono accedervi affatto.

Esempi di rappresentazione di permessi.

4.8.6. Permessi speciali: S-bit

I permessi dei file sono memorizzati in una sequenza di 9 bit, dove ogni gruppetto di tre rappresenta i permessi per una categoria di utenti (il proprietario, il gruppo, gli altri).

Assieme a questi 9 bit ne esistono altri tre, posti all'inizio, che permettono di indicare altrettante modalità: SUID (*Set user identifier*), SGID (*Set group identifier*) e Sticky (*Save text image*). Si tratta di attributi speciali che riguardano prevalentemente i file eseguibili. Solitamente non vengono usati e per lo più gli utenti comuni ignorano che esistano.

Tutto questo serve adesso per sapere il motivo per il quale spesso i permessi espressi in forma numerica (ottale) sono di quattro cifre, con la prima che normalmente è azzerata.

Per esempio, la modalità `0644` rappresenta il permesso per l'utente proprietario di accedervi in lettura e scrittura, mentre agli altri utenti si concede di accedervi in sola lettura.

L'indicazione della presenza di questi bit attivati può essere vista anche nelle rappresentazioni in forma di stringa. L'elenco seguente mostra il numero ottale e la sigla corrispondente.

```
SUID   = 4 = --s------
SGID   = 2 = -----s---
Sticky = 1 = --------t
```

Come si può osservare, questa indicazione prende il posto del permesso di esecuzione. Nel caso in cui il permesso di esecuzione corrispondente non sia attivato, la lettera (`s` o `t`) appare maiuscola.

4.8.7. Date

Tra gli attributi di un file ci sono anche tre indicazioni data-orario:

- la data e l'ora di creazione: viene modificata in particolare quando si cambia lo stato del file (permessi e proprietà) e si riferisce precisamente al cambiamento di inode (che viene descritto più avanti);
- la data e l'ora di modifica: viene modificata quando si modifica il contenuto del file;

- la data e l'ora di accesso: viene modificata quando si accede al file anche solo in lettura.

4.8.8. Utenza e accesso

Una volta avviato un sistema Unix, prima che sia disponibile l'invito della shell, ovvero il *prompt*, occorre che l'utente sia riconosciuto dal sistema, attraverso la procedura di accesso (*login*). Quello che viene chiesto è l'inserimento del nome dell'utente (così come è stato registrato) e subito dopo la parola d'ordine (*password*) abbinata a quell'utente. Eccezionalmente può trattarsi di un utente senza parola d'ordine, così come avviene per i mini sistemi a dischetti fatti per consentire le operazioni di manutenzione eccezionale.

Si distingue solo tra due tipi di utenti: l'amministratore, il cui nome è `root`, e gli altri utenti comuni. L'utente `root` non ha alcun limite di azione, gli altri utenti dipendono dai permessi attribuiti ai file (e alle directory) oltre che dai vincoli posti direttamente da alcuni programmi.

In teoria, è possibile usare un elaboratore personale solo utilizzando i privilegi dell'utente `root`. In pratica, questo non conviene perché si perde di vista il significato della gestione dei permessi sui file e sulle directory, ma soprattutto si rendono vani i sistemi di sicurezza predefiniti contro gli errori. Chi ha usato un sistema Dos può comprendere meglio questo concetto se pensa a cosa succede quando si esegue un comando come quello seguente:

```
C:\> DEL *.*[Invio]
```

Prima di iniziare la cancellazione, il Dos chiede una conferma ulteriore, proprio perché non esiste alcun tipo di controllo. In un sistema Unix, di solito ciò non avviene: la cancellazione inizia immediatamente senza richiesta di conferme. Se i permessi consentono la cancellazione dei file solo all'utente `root`, un utente registrato in modo diverso non può fare alcun danno.

In conclusione, l'utente `root` deve stare molto attento a quello che fa proprio perché può accedere a qualunque funzione o file del sistema, inoltre il sistema non pone alcuna obiezione al suo comportamento. Invece, un utente comune è vincolato dai permessi sui file e dai programmi che possono impedirgli di eseguire certe attività, di conseguenza, è possibile lavorare con meno attenzione.

4.8.9. File globbing

Il ***glob*** (o *globbing*) è il metodo attraverso il quale, tramite un modello simbolico, è possibile indicare un gruppo di nomi di file. Corrisponde all'uso dei caratteri jolly del Dos, con la differenza fondamentale che è la shell a occuparsi della loro sostituzione e non i programmi. Di solito, si possono utilizzare i simboli seguenti:

*	l'asterisco rappresenta un gruppo qualsiasi di caratteri, compreso il punto, purché questo punto non si trovi all'inizio del nome;
?	il punto interrogativo rappresenta un carattere qualsiasi, compreso il punto, purché questo punto non si trovi all'inizio del nome;
`[...]`	le parentesi quadre permettono di rappresentare un carattere qualsiasi tra quelli contenuti al loro interno, o un intervallo di caratteri possibili.

Dal momento che è la shell a eseguire la sostituzione dei caratteri jolly, la sintassi tipica di un programma di servizio è la seguente:

```
programma [opzioni] [file...]
```

Nei sistemi Dos si usa spesso la convenzione inversa, secondo cui l'indicazione dei file avviene prima delle opzioni. Da un punto di vista puramente logico, potrebbe sembrare più giusto l'approccio del Dos: si indica l'oggetto su cui agire e quindi si indica il modo. Facendo così si ottengono però una serie di svantaggi:

ogni programma deve essere in grado di espandere i caratteri jolly per conto proprio;

non è possibile utilizzare l'espansione delle variabili di ambiente e nemmeno di altri tipi;

se si vogliono indicare elenchi di file che non possono essere espressi con i caratteri jolly, occorre che il programma sia in grado di gestire questa possibilità, di solito attraverso la lettura di un file esterno.

In pratica, il tipo di semplificazione utilizzato dal Dos è poi la fonte di una serie di complicazioni per i programmatori e per gli utilizzatori.

4.8.10. Tilde

Di solito, la shell si occupa di eseguire la sostituzione del carattere tilde (~). Nei sistemi Unix, ogni utente ha una directory personale, conosciuta comunemente come directory *home*. Il simbolo ~ da solo viene sostituito dalla shell con la directory personale dell'utente che sta utilizzando il sistema, mentre un nominativo-utente preceduto dal simbolo ~, viene sostituito dalla shell con la directory personale dell'utente indicato.

4.9. Variabili di ambiente

Le variabili di ambiente sono gestite dalla shell e costituiscono uno dei modi attraverso cui si configura un sistema. I programmi possono leggere alcune variabili di loro interesse e modificare il proprio comportamento in base al loro contenuto.

Una riga di comando può fare riferimento a una variabile di ambiente: la shell provvede a sostituirne l'indicazione con il suo contenuto.

4.10. Ridirezione e pipeline

I programmi, quando vengono eseguiti, hanno a disposizione alcuni canali standard per il flusso dei dati (input/output). Questi sono: standard input, standard output e standard error.

- ***Standard input***

Lo standard input viene utilizzato come fonte standard per i dati in ingresso (input) nel programma.

- ***Standard output***

Lo standard output viene utilizzato come destinazione standard per i dati in uscita (output) dal programma.

- ***Standard error***

Lo standard error, viene utilizzato come destinazione standard per i dati in uscita dal programma derivati da situazioni anomale.

Lo standard input è rappresentato di norma dai dati provenienti dalla tastiera del terminale. Lo standard output e lo standard error sono emessi normalmente attraverso lo schermo del terminale.

Per mezzo della shell si possono eseguire delle ridirezioni di questi flussi di dati, per esempio facendo in modo che lo standard output di un programma sia inserito come standard input di un altro, creando così una pipeline.

4.10.1. Ridirezione dello standard input

`programma < file_di_dati`

Si ridirige lo standard input utilizzando il simbolo minore (**<**) seguito dalla fonte alternativa di dati. Il programma a sinistra del simbolo **<** riceve come standard input il contenuto del file indicato a destra.

L'esempio seguente visualizza il contenuto del file `elenco.txt` dopo averlo riordinato:

```
$ sort < elenco.txt[Invio]
```

4.10.2. Ridirezione dello standard output

`programma > file_di_dati`

Si ridirige lo standard output utilizzando il simbolo maggiore (**>**) seguito dalla destinazione alternativa dei dati. Il programma a sinistra del simbolo **>** emette il suo standard output all'interno del file indicato a destra che viene creato per l'occasione.

Lo standard output può essere aggiunto a un file preesistente; in tal caso si utilizza il simbolo **>>**.

Segue la descrizione di alcuni esempi.

```
$ ls > elenco.txt[Invio]
```

Genera il file `elenco.txt` con il risultato dell'esecuzione di `ls`.

```
$ ls >> elenco.txt[Invio]
```

Aggiunge al file `elenco.txt` il risultato dell'esecuzione di `ls`.

4.10.3. Ridirezione dello standard error

`programma 2> file_di_dati`

Si ridirige lo standard error utilizzando il simbolo `2>` seguito dalla destinazione alternativa dei dati. Il programma a sinistra del simbolo `2>` emette il suo standard error all'interno del file indicato a destra che viene creato per l'occasione.

Lo standard error può essere aggiunto a un file preesistente; in tal caso si utilizza il simbolo `2>>`.

Segue la descrizione di alcuni esempi.

```
$ controlla 2> errori.txt[Invio]
```

Genera il file `errori.txt` con il risultato dell'esecuzione dell'ipotetico programma `controlla`.

```
$ controlla 2>> errori.txt[Invio]
```

Aggiunge al file `errori.txt` il risultato dell'esecuzione dell'ipotetico programma `controlla`.

4.10.4. Pipeline

`programma1 | programma2` `[` **`| programma3...`**`]`

Si ridirige lo standard output di un programma nello standard input di un altro, utilizzando il simbolo barra verticale (`|`). Il programma a sinistra del simbolo `|` emette il suo standard output nello standard input di quello che sta a destra.

Nella rappresentazione schematica delle sintassi dei programmi, questo simbolo ha normalmente il significato di una scelta alternativa tra opzioni diverse, parole chiave o altri argomenti. In questo caso fa proprio parte della costruzione di una pipeline.

Segue la descrizione di alcuni esempi.

```
$ ls | sort[Invio]
```

Riordina il risultato del comando `ls`.

```
$ ls | sort | less[Invio]
```

Riordina il risultato del comando `ls` e quindi lo fa scorrere sullo schermo con l'aiuto del programma `less`.

4.11. Comandi e programmi di servizio di uso comune

In linea di principio, con il termine ***comando*** ci si dovrebbe riferire ai comandi interni di una shell, mentre con il termine *utility*, o semplicemente programma, si dovrebbe fare riferimento a programmi eseguibili esterni alla shell. Di fatto però, dal momento che si mette in esecuzione un programma impartendo un comando alla shell, con questo termine (comando) si fa spesso riferimento in maniera indistinta a comandi interni di shell o (in mancanza) a comandi esterni o programmi di servizio.

Naturalmente, questo ragionamento vale fino a quando si tratta di programmi di servizio di uso comune, non troppo complessi, che usano un sistema di input/output elementare. Sarebbe un po' difficile definire comando un programma di scrittura o un navigatore di Internet.

4.11.1. Interpretazione della sintassi

La sintassi per l'avvio di un programma o per l'esecuzione di un comando segue delle regole molto semplici.

Le *metavariabili*, scritte in questo modo, descrivono l'informazione che deve essere inserita al loro posto.

Le altre parole rappresentano dei termini chiave che, se usati, devono essere indicati così come appaiono nello schema sintattico.

Quello che appare racchiuso tra parentesi quadre rappresenta una scelta facoltativa: può essere utilizzato o meno.

La barra verticale (**|**) rappresenta la possibilità di scelta tra due possibilità alternative: quello che sta alla sua sinistra e quello che sta alla sua destra. Per esempio, `uno | due` rappresenta la possibilità di scegliere una tra le parole `uno` e `due`.

Quello che appare racchiuso tra parentesi graffe rappresenta una scelta obbligatoria e serve in particolare per evitare equivoci nell'interpretazione quando si hanno più scelte alternative, separate attraverso il simbolo **|**. Seguono alcuni esempi:

`{uno | due | tre}`

rappresenta la scelta obbligatoria di una tra le parole chiave uno, due e tre;

{-f *file* | --file=*file*}

rappresenta la scelta obbligatoria di una tra due opzioni equivalenti.

I puntini di sospensione rappresentano la possibilità di aggiungere altri elementi dello stesso tipo di quello che li precede. Per esempio, ***file...*** rappresenta la metavariabile «file» che può essere seguita da altri valori dello stesso tipo rappresentato dalla metavariabile stessa.

Naturalmente, può capitare che i simboli utilizzati per rappresentare la sintassi, servano negli argomenti di un comando o di un programma. I casi più evidenti sono:

le pipeline che utilizzano la barra verticale per indicare il flusso di dati tra un programma e il successivo;

le parentesi graffe usate in alcuni linguaggi di programmazione.

Quando ciò accade, occorre fare attenzione al contesto per poter interpretare correttamente il significato di una sintassi, osservando, se ci sono, gli esempi proposti.

4.11.2. Organizzazione tipica

Il programma di servizio tipico ha la sintassi seguente:

```
programma [opzioni] [file...]
```

In questo caso, il nome del programma è proprio `programma`.

Normalmente vengono accettate una o più opzioni facoltative, espresse attraverso una lettera dell'alfabeto preceduta da un trattino (`-a`, `-b`,...). Queste possono essere usate separatamente oppure, spesso si possono raggruppare con un solo trattino seguito da tutte le lettere delle opzioni che si intendono selezionare. Quindi, normalmente, i due comandi seguenti sono equivalenti:

```
programma -a -b[Invio]
programma -ab[Invio]
```

I programmi più recenti includono opzioni descrittive formate da un nome preceduto da due trattini. In presenza di questi tipi di opzioni, non si possono fare aggregazioni nel modo appena visto.

A volte si incontrano opzioni che richiedono l'indicazione aggiuntiva di un altro argomento.

La maggior parte dei programmi di servizio esegue delle elaborazioni su file, generando un risultato che viene emesso normalmente attraverso lo standard output. Spesso, quando non vengono indicati file negli argomenti, l'input per l'elaborazione viene ottenuto dallo standard input.

Alcuni programmi permettono l'utilizzo del trattino (`-`) in sostituzione dell'indicazione di file in ingresso o in uscita, allo scopo di fare riferimento, rispettivamente, allo standard input e allo standard output.

4.12. Programma o eseguibile

In generale, quando si usa il termine «programma» non si chiarisce quale sia la sua estensione reale. Si può usare questo termine per identificare qualcosa che si compone di un solo file eseguibile, oppure un piccolo sistema composto da più componenti, che vengono comandate da un solo sistema frontale.

Spesso, in particolare all'interno di questo documento, quando si vuole fare riferimento a un programma inteso come un insieme di componenti, oppure come qualcosa di astratto per il quale nel contesto non conta il modo in cui viene avviato, lo si indica con un nome che non ha enfatizzazioni particolari e generalmente ha l'iniziale maiuscola. Per esempio, questo è il caso della shell Bash, a cui si è accennato, il cui eseguibile è in realtà `bash`.

Per evitare ambiguità, quando si vuole essere certi di fare riferimento a un programma eseguibile, si specifica proprio che si tratta di questo, cioè di un «eseguibile», mostrandolo attraverso enfatizzazioni di tipo dattilografico, scrivendo il nome esattamente nel modo in cui ciò va fatto per avviarlo.

4.13. Comparazione tra alcuni comandi Dos e GNU/Linux

Dos	GNU/Linux
DIR	ls -l
DIR /W	ls
MD PIPPO	mkdir pippo
CD PIPPO	cd pippo
RD PIPPO	rmdir pippo
COPY *.* \PROVA	cp * /prova
XCOPY *.* \PROVA /E /S	cp -dpR * /prova
REN ARTICOLO LETTERA	mv articolo lettera
MOVE *.* \PROVA	mv * /prova
DEL ARTICOLO	rm articolo
DELTREE TEMP	rm -R temp
TYPE LETTERA	cat lettera
TYPE LETTERA | MORE	cat lettera | more
HELP DIR	man ls
FORMAT A: /N:18 /T:80	fdformat /dev/fd0u1440
FORMAT A: /N:9 /T:80	fdformat /dev/fd0u720
DISKCOPY A: B:	cp /dev/fd0 /dev/fd1
DISKCOPY A: A:	cp /dev/fd0 /tmp/pippo ; cp /tmp/pippo /dev/fd0
KEYB IT	loadkeys it
CLS	clear
BACKUP C:\DATI*.* A: /S	tar cvf /dev/fd0 -L 1440 -M /dati
FIND "saluti" PRIMO.DOC	grep "saluti" primo.doc
FOR %A IN (*.DOC) DO FIND "saluti" %A	grep "saluti" *.doc
MEM	free

Comparazione tra alcuni comandi Dos e gli equivalenti per GNU/Linux attraverso degli esempi.

5 - Princìpi del World Wide Web

Il più recente ed ambizioso dei servizi di Internet è il World Wide Web (ragnatela mondiale, abbreviato www o anche semplicemente web). Il web, come si è visto precedentemente, è nato al CERN di Ginevra come progetto accademico tendente a semplificare l'accesso alle informazioni e ai documenti.

Poiché il web è stato concepito quando Internet già esisteva, utilizza molto di ciò che è stato realizzato in informatica negli ultimi decenni, ma utilizza anche importanti nuove tecnologie, realizzate appositamente per il web:

Il web funziona con il meccanismo **client-server**. La parte client del web è il browser, un programma che l'utente esegue sul proprio computer e mediante il quale richiede le informazioni desiderate ad un server web. Alla risposta del server, il browser visualizze le informazioni in pagine indipendenti.

Le pagine web sono descritte mediante il linguaggio di formattazione **HTML** (*HyperText Markup Language*). HTML consente agli autori di pagine web di creare pagine ipertestuali che qualunque server web può distribuire e qualunque browser può visualizzare. I codici HTML sono detti tag e consentono di indicare la formattazione del testo (dimensione, stile, posizionamento, ecc.) e la posizione delle immagini, dei filmati, delle animazioni. Queste indicazioni sono poi interpretate dal browser per la visualizzazione.

Una pagina web può contenere collegamenti (*link*) che puntano ad altre pagine web, anche su macchine diverse. I collegamenti possono essere incorporati nel testo o associati ad una immagine. È il concetto di **ipertesto**, un termine coniato da Ted Nelson negli anni 60. Un ipertesto è un testo che consente una lettura non lineare di un documento. Se in un testo che parla di vino c'è una informazione sugli alberi che producono il sughero, e a questa informazione è associato un link, basta fare clic sul link per andare ad una nuova pagina che parla di tali alberi, interrompendo la lettura della parte dedicata al vino, alla quale si potrà tornare successivamente.

Il sistema degli **URL** (*Uniform Resource Locator*) consente a quasi tutti i tipi di informazione di essere recuperati da quasi tutti i punti di Internet, ed è stato messo a punto appositamente per il web.

Il protocollo **HTTP** (*HyperText Transfer Protocol*) è il protocollo di comunicazione tra browser (client web) e server web.

5.1. Il protocollo HTTP

Il modo più comune per pubblicare informazioni attraverso la rete è quello di utilizzare un servente HTTP (HyperText Transfer Protocol).

Le informazioni pubblicate in questo modo sono rivolte a tutti gli utenti che possono raggiungere il servizio, nel senso che normalmente non viene richiesta alcuna identificazione.

Per offrire un servizio HTTP occorre un programma in grado di gestirlo. Di solito si tratta di un demone, il servente HTTP consente l'accesso a una directory particolare e alle

sue discendenti. Questa directory viene identificata spesso con il nome *document root* e si tratta in pratica di una sorta di directory personale degli utenti (anonimi) che accedono attraverso questo protocollo.

Per poter usufruire di un servizio HTTP occorre un programma cliente adatto. In generale, tale programma cliente è in grado di accedere anche ad altri servizi, pertanto, in questo senso viene definito semplicemente «navigatore». Il programma di navigazione tipico dovrebbe consentire anche la visualizzazione di immagini, ma un buon programma che utilizza soltanto un terminale a caratteri può essere utilizzato in qualunque condizione, quindi, tale possibilità non deve essere scartata a priori.

L'integrazione di diversi protocolli impone l'utilizzo di un sistema uniforme per indicare gli indirizzi, per poter conoscere subito in che modo si deve effettuare il collegamento. Per questo, quando si utilizza un navigatore, si devono usare indirizzi espressi in modo standard, precisamente secondo il formato URI, o *Uniform resource identifier*. Attualmente, è ancora in uso la vecchia definizione, URL, *Uniform resource locator*, che in pratica rappresenta un sottoinsieme di URI. Attraverso questa modalità, è possibile definire tutto quello che serve per raggiungere una risorsa: protocollo, nodo (*host*), porta, percorso.

Il formato di un URI potrebbe essere definito secondo lo schema seguente:

```
protocollo indirizzo_della_risorsa[dati_aggiuntivi]
```

Alcuni tipi di protocolli sono in grado di gestire dei dati aggiuntivi in coda all'indirizzo della risorsa. Nel caso del protocollo HTTP combinato con CGI, può trattarsi di ***richieste*** o di ***percorsi aggiuntivi***.

Quando un URI comprende anche una stringa di richiesta (*query*), questa viene distinta dall'indirizzo della risorsa attraverso un punto interrogativo.

```
protocollo indirizzo_della_risorsa?[richiesta]
```

L'utilizzo di una stringa di richiesta presume che la risorsa sia un programma in grado di utilizzare l'informazione contenuta in tale stringa. Segue un esempio banale di un URI contenente una richiesta:

```
http://www.brot.dg/cgi-bin/saluti.pl?buongiorno
```

Quando l'indirizzo della risorsa di un URI fa riferimento a un programma, questo può ricevere un'informazione aggiuntiva legata a un file o a una directory particolare. Si ottiene questo aggiungendo l'indicazione del percorso che identifica questo file o questa directory.

```
protocollo indirizzo_della_risorsa[percorso_aggiuntivo]
```

Segue un esempio banale di un URI, completo dell'indicazione di un percorso:

```
http://www.brot.dg/cgi-bin/elabora.pl/archivio.doc
```

Quando un simbolo di quelli non utilizzabili deve essere indicato ugualmente da qualche parte dell'URI, facendogli perdere il significato speciale che questo potrebbe avere altrimenti, si può convertire utilizzando la notazione `%hh`. La sigla *hh* rappresenta una coppia di cifre esadecimali. A questa regola fa eccezione lo spazio che viene codificato normalmente con il segno `+`, ma non in tutte le occasioni.

Generalmente, per gli indirizzi URI normali non c'è la necessità di preoccuparsi di questo problema, quindi, l'utilizzo di simboli particolari riguarda prettamente la costruzione delle richieste, come viene mostrato meglio in seguito.

La seguente tabella mostra l'elenco di alcune corrispondenze tra simboli particolari e la codifica alternativa utilizzabile negli URI:

Carattere	Codifica		Carattere	Codifica
%	%25		&	%26
+	%2B		/	%2F
=	%3D		~	%7E

Il funzionamento del protocollo HTTP è molto semplice. L'utilizzo di un servizio HTTP si compone di una serie di transazioni, ognuna delle quali si articola in queste fasi:

1. apertura della connessione;
2. invio da parte del cliente di una richiesta;
3. risposta da parte del servente;
4. chiusura della connessione.

In questo modo, il programma servente non deve tenere traccia delle transazioni che iniziano e finiscono ogni volta che un utente compie un'azione attraverso il suo programma cliente.

La richiesta inviata dal programma cliente deve contenere il metodo (i più comuni sono `GET` e `POST`), l'indicazione della risorsa cui si vuole accedere, la versione del protocollo ed eventualmente l'indicazione dei tipi di dati che possono essere gestiti dal programma cliente (si parla in questi casi di tipi MIME). Naturalmente sono possibili richieste più ricche di informazioni.

La risposta del servente HTTP è costituita da un'intestazione che, tra le altre cose, specifica il modo in cui l'informazione allegata deve essere interpretata. È importante comprendere subito che l'intestazione viene staccata dall'inizio dell'informazione allegata attraverso un riga vuota, composta dalla sequenza `<CR><LF>`.

5.1.1. Analisi di una connessione HTTP

Per comprendere in pratica il funzionamento di una connessione HTTP, si può utilizzare il programma `telnet` al posto di un navigatore normale. Si suppone di poter accedere al nodo `www.brot.dg` nel quale è stato installato Apache (un comune web server) con successo. Dal servente viene prelevato il file `index.html` che si trova all'interno della directory *document root*.

```
$ telnet www.brot.dg http[Invio]
```

`telnet` risponde e si mette in attesa di ricevere il messaggio da inviare al servente.

```
Trying 192.168.1.1...
Connected to www.brot.dg.
Escape character is '^]'.
```

Si deve iniziare a scrivere, cominciando con una riga contenente il metodo, la risorsa e la versione del protocollo, continuando con una riga contenente le possibilità di visualizzazione del cliente (i tipi MIME).

```
GET /index.html HTTP/1.0[Invio]
Accept: text/html[Invio]
[Invio]
```

Appena si invia una riga vuota, il servente intende che la richiesta è terminata e risponde.

```
HTTP/1.1 200 OK
Date: Tue, 27 Jan 1998 17:44:46 GMT
Server: Apache/1.2.4
Last-Modified: Tue, 30 Dec 1997 21:07:24 GMT
ETag: "6b003-792-34a9628c"
Content-Length: 1938
Accept-Ranges: bytes
Connection: close
Content-Type: text/html

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<HTML>
 <HEAD>
  <TITLE>Test Page for Linux's Apache Installation</TITLE>
 </HEAD>
<!-- Background white, links blue (unvisited), navy (visited), red
(active) -->
 <BODY
  BGCOLOR="#FFFFFF"
  TEXT="#000000"
  LINK="#0000FF"
  VLINK="#000080"
  ALINK="#FF0000"
 >
  <H1 ALIGN="CENTER">It Worked!</H1>
  <P>
  If you can see this, it means that the installation of the
  <A
   HREF="http://www.apache.org/"
  >Apache</A>
  software on this Linux system was successful. You may now add
content to
  this directory and replace this page.
  </P>
```

```
...
...
 </BODY>
</HTML>
Connection closed by foreign host.
```

Come già accennato, il messaggio restituito dal servente è composto da un'intestazione (*header*) in cui l'informazione più importante è il tipo di messaggio allegato, cioè in questo caso `Content-Type: text/html`, seguita da una riga vuota e quindi dall'oggetto richiesto, cioè il file `index.html`.

Al termine della ricezione dell'oggetto richiesto, la connessione ha termine. Lo si può osservare dal messaggio dato da `telnet`:

```
Connection closed by foreign host.
```

Il lavoro di un programma cliente è tutto qui: inviare richieste al servente HTTP, ricevere le risposte e gestire i dati, possibilmente visualizzandoli o mettendo comunque l'utente in grado di fruirne.

5.1.2. Tipi MIME

MIME è una codifica standard per definire il trasferimento di documenti multimediali attraverso la rete. L'acronimo sta per *Multipurpose Internet Mail Extentions* e la sua origine è appunto legata ai trasferimenti di dati allegati ai messaggi di posta, come il nome lascia intendere.

Il protocollo HTTP utilizza lo stesso standard e con questo il programma servente informa il programma cliente del tipo di oggetto che gli viene inviato. Nello stesso modo, il programma cliente, all'atto della richiesta di una risorsa, informa il servente dei tipi MIME che è in grado di gestire.

Il servente HTTP, per poter comunicare il tipo MIME al cliente, deve avere un modo per riconoscere la natura degli oggetti che costituiscono le risorse accessibili. Questo modo è dato dall'estensione, per cui, la stessa scelta dell'estensione per i file accessibili attraverso il protocollo HTTP è praticamente obbligatoria, ovvero, dipende dalla configurazione dei tipi MIME. Di seguito alcuni tipi MIME con le possibili estensioni:

Tipo MIME	Estensioni	Descrizione
application/postscript	ps eps	PostScript.
application/rtf	rtf	Rich Text Format.
application/x-tex	tex	Documento TeX/LaTeX.
audio/basic	au snd	File audio.
audio/x-wav	wav	File audio.
image/gif	gif	Immagine GIF.
image/jpeg	jpeg jpg	Immagine JPEG.
image/tiff	tiff tif	Immagine TIFF.
image/x-xwindowdump	xwd	Immagine X Window Dump.
text/html	html htm	Testo formattato in HTML.
text/plain	txt	Testo puro.
video/mpeg	mpeg mpg mpe	Animazione MPEG.
video/quicktime	qt mov	Animazione Quicktime.

5.1.3. Campi di Richiesta

Come si è visto dagli esempi mostrati precedentemente, la richiesta fatta dal programma cliente è composta da una prima riga in cui si dichiara il tipo, la risorsa desiderata e la versione del protocollo.

```
GET /index.html HTTP/1.0
```

Di seguito vengono indicati una serie di campi, più o meno facoltativi. Questi campi sono costituiti da un nome seguito da due punti (`:`), da uno spazio e dall'informazione che gli si vuole abbinare.

Campo «Accept»

Una o più righe contenenti un campo `Accept` possono essere incluse per indicare i tipi MIME che il cliente è in grado di gestire (cioè di ricevere). Se non viene indicato alcun campo `Accept`, si intende che siano accettati almeno i tipi `text/plain` e `text/html`.

I tipi MIME sono organizzati attraverso due parole chiave separate da una barra obliqua. In pratica si distingue un tipo e un sottotipo MIME. È possibile indicare un gruppo di tipi MIME mettendo un asterisco al posto di una o di entrambe le parole chiave, in modo da selezionare tutto il gruppo relativo. Per esempio,

```
Accept: */*
```

rappresenta tutti i tipi MIME;

```
Accept: text/*
```

rappresenta tutti i sottotipi MIME che appartengono al tipo `text`;

mentre

```
Accept: audio/basic
```

rappresenta un tipo e un sottotipo MIME particolare.

Campo «User-Agent»

Il campo `User-Agent` permette di informare il servente sul nome e sulla versione dell'applicativo particolare che svolge la funzione di cliente. Per convenzione, il nome di questo è seguito da una barra obliqua e dal numero della versione. Tutto quello che dovesse seguire sono solo informazioni addizionali per le quali non è stabilita una forma precisa. Per esempio, nel caso di Netscape, si potrebbe avere un'indicazione del tipo seguente:

```
User-Agent: Mozilla/4.04 [en] (X11; I; Linux 2.0.32 i586)
```

5.1.4. Campi di Risposta

La risposta del servente HTTP a una richiesta del programma cliente si compone di un'intestazione seguita eventualmente da un allegato, che costituisce la risorsa a cui il cliente voleva accedere. L'intestazione è separata dall'allegato da una riga vuota.

La prima riga è costituita dal codice di stato della risposta. Nella migliore delle ipotesi dovrebbe presentarsi come nell'esempio seguente:

```
HTTP/1.0 200 OK
```

Nella seguente tabella alcuni codici di ritorno più frequenti:

Codice	Descrizione
200	OK
201	Creato.
202	Accettato.
204	Nessun contenuto.
300	Scelte multiple.
301	Spostato in modo permanente.
302	Spostato temporaneamente.
304	Non modificato.
400	Richiesta errata.
401	Non autorizzato.
403	Proibito.
404	Non trovato.
500	Errore interno del servente HTTP.
501	Servizio non realizzato (non disponibile).
502	Gateway errato.
503	Servizio non disponibile.

Il resto dell'intestazione è composto da campi, simili a quelli utilizzati per le richieste dei programmi clienti:

Campo «Allow»

Il campo `Allow` viene utilizzato dal programma servente per informare il programma cliente dei metodi che possono essere utilizzati. Viene restituita tale informazione quando il cliente tenta di utilizzare un metodo di richiesta che il servente non è in grado di gestire. Segue un esempio.

```
Allow: GET, HEAD, POST
```

Campo «Content-Length»

Il campo `Content-Length` indica al programma cliente la dimensione (in byte) dell'allegato. Se viene utilizzato il metodo `HEAD`, con cui non viene restituito alcun allegato, permette di conoscere in anticipo la dimensione della risorsa.

```
Content-Length: 1938
```

Campo «Content-Type»

Il campo `Content-Type` indica al programma cliente il tipo MIME a cui appartiene la risorsa (allegata o meno). Segue l'esempio più comune.

```
Content-Type: text/html
```

5.1.5. Variabili GET e POST

Tramite le richieste GET e POST è possibile inviare delle variabili al server insieme alla richiesta di una pagina.

La differenza tra i due metodi è che nel modo GET le variabili vengono definite all'interno della URL mentre in POST queste vengono inviate nel BODY della richiesta

Ad esempio se volessimo richiedere la pagina `index.html` e contemporaneamente inviare una variabile di nome `pippo` con associato il valore `pluto` dovremmo effettuare la richiesta come segue:

Metodo GET:

```
GET /index.html?pippo=pluto HTTP/1.0[Invio]
Accept: text/html[Invio]
[Invio]
```

Metodo POST:

```
POST /index.html HTTP/1.0[Invio]
Accept: text/html[Invio]
[Invio]
pippo=pluto[Invio]
[Invio]
```

Questa funzionalità del protocollo HTTP è quella che permette di richiedere delle pagine *dinamiche*, infatti a ricevere le suddette richieste potrebbe essere un'applicazione in grado di elaborare le variabili e di conseguenza inviare come risposta delle informazioni calcolate sul momento.

Vedremo in seguito come creare delle pagine html in modo da far inviare dal browser al server queste variabili e come poi elaborarle attraverso delle applicazioni in PHP o PERL.

5.1.6. Cookie

Assieme alle variabili GET e POST il server riceve anche le informazioni relative ai cookie presenti nel browser.

Un cookie è un piccola informazione scritto dal browser sul disco locale in modo che possa essere successivamente riletta dal browser stesso e spedita al server. Questo è il modo in cui il browser può memorizzare alcune informazioni per poi renderle disponbili al server ogni qual volta si faccia una richiesta.

L'impostazione del cookie avviene ad opera del server inviando al richiesta di memorizzazione al browser, questa deve avvenire nell'*header* della risposta HTTP del server in questo modo:

Richiesta del browser

```
GET /index.html HTTP/1.0[Invio]
Accept: text/html[Invio]
[Invio]
```

Risposta del server

```
HTTP/1.1 200 OK
Date: Tue, 27 Jan 1998 17:44:46 GMT
Server: Apache/1.2.4
Last-Modified: Tue, 30 Dec 1997 21:07:24 GMT
ETag: "6b003-792-34a9628c"
Content-Length: 1938
Accept-Ranges: bytes
Connection: close
Content-Type: text/html
Set-Cookie:utente=tizio;expires=Sun, 27-Feb-2005 21:43:03
GMT;path=/;domain=lnf.infn.it;

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<HTML>
 <HEAD>
.....
```

In questo modo il server chiede al browser di memorizzare un cookie con nome `utente`, con valore `tizio`, che venga considerato scaduto e dunque eliminato alle 21:43:03 di Domenica 27 febbraio 2005, ed infine che venga reso disponibile solo quando si facciano richieste ad un web server nel dominio `lnf.infn.it`.

Quando il browser andrà dunque a fare una qualsiasi richiesta ad esempio al server `www.lnf.inf.it` invierà sempre le informazioni relative ai cookie che questo gli ha chiesto di trattenere:

Richiesta del browser

```
GET /index.html HTTP/1.0[Invio]
Accept: text/html[Invio]
Cookie: utente=tizio[Invio]
[Invio]
```

Quindi il server trasmetterà l'informazione del cookie all'eventuale applicazione che risponderà al path `/index.html` oppure nel caso in cui questa sia una semplice pagina statica l'informazione verrà ignorata.

5.1.7. Approfondimenti

Per una guida completa e approfondimenti sul protocollo HTTP è necessario fare riferimento alla relativa sezione del W3C: http://www.w3.org/Protocols/

5.2. Il linguaggio HTML

Il linguaggio HTML (HyperText Markup Language) è il linguaggio di formattazione del testo (da non confondere con un linguaggio di programmazione) utilizzato per descrivere pagine web. Oltre alle istruzioni di formattazione del testo, con HTML si possono creare collegamenti ipertestuali, inserire immagini, suoni e animazioni. L'evoluzione di HTML è molto rapida, e spesso le implementazioni hanno preceduto gli standard ufficiali. Ecco una minicronologia delle varie versioni di HTML.

- 1989: Prima proposta di un sistema ipertestuale (Tim Berners-Lee)
- 1991: HTML (1)
- 1993: HTML + (mai standardizzato)
- 1994: HTML 2
- 1994: HTML 3 (mai standardizzato)
- 1996: HTML 3.2
- 1997: HTML 4
- 2000: HTML 4.01
- 2000: XHTML 1
- 2001: XHTML 1.1

Oggi lo standard HTML viene deciso dal **W3C, World Wide Web Consortium** (http://www.w3.org), entità che si occupa di definire e pubblicare tutte le specifiche del codice HTML e come verdemo in seguito, dei CSS.

In questo manuale verrà descritto nello specifico lo standard HTML/4.01 che rimane ad oggi quello più utilizzato e del quale è sicuramente più definita la sintassi.

La versione 4.01 di html, rispetto alle precedenti ha la particolarità di separare il contenuto dagli stili di visualizzazione, infatti in linguaggio html non dovranno essere definiti font, colori o simili, solo il testo e la formattazione dei paragrafi e delle tabelle.

Vedremo in seguito che tutti gli altri attributi relativi all'aspetto della pagina dovranno essere definiti tramite i CSS (*Cascading Style Sheets*).

Per indicare la formattazione del testo, html utilizza i cosiddetti tag, cioè metaindicazioni per il rendering da parte del browser del contenuto della pagina.

Ecco un documento HTML molto breve ma completo:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<HTML>
<HEAD>
<TITLE>Esempio 1</TITLE>
</HEAD>
<BODY>
<H1>Esempio 1</H1>
Un documento molto poco interessante.
<P>Secondo paragrafo di questo documento poco interessante che contiene un <A
HREF="un_altro_documento.html">link ad un altro documento</A> poco interessante.
</BODY>
</HTML>
```

L'HTML è un linguaggio di composizione basato sull'SGML. Come tale, un documento HTML inizia sempre con la dichiarazione del DTD; poi tutto il documento viene racchiuso nell'elemento principale di questa struttura: `<html></html>`.

Queste istruzioni vengono registrate in un file di testo. Aprendo questo file di testo con un browser, per esempio Netscape Navigator, si ottiene un risultato del tipo:

Esempio 1

Un documento molto poco interessante.

Secondo paragrafo di questo documento poco interessante che contiene un link ad un altro documento poco interessante.

5.2.1. Struttura di un documento HTML

All'interno dei tag <html> si devono specificare altre due sezioni principali:

- Head
- Body

Nella prima verranno definite delle istruzioni che verranno interpretate dal browser per sapere come trattare alcuni aspetti del documento, come ad esempio:

- Il titolo della pagina;
- Stili di formattazione (CSS)
- Tempo di scadenza della validità del documento
- ecc...

Nella seconda, BODY, viene scritto il vero contenuto della pagina che verrà visualizzata.

Per definire ad esempio un titolo "Pagina di prova" per la pagina dovremmo inserire all'interno della sezione delimitata da <head></head> il tag

```
<title>Pagina di prova</title>
```

Se volessimo inoltre che la pagina venga ricaricata ogni 30 secondi automaticamente dal browser potremmo inserire anche:

```
<meta HTTP-EQUIV='Refresh' CONTENT='180'>
```

che per altro equivale ad inviare un header HTTP con le stesse informazioni prima del body HTTP.

5.2.2. Tag di formattazione principali

Come dicevamo il BODY è l'effettivo contenuto che si vedrà nel browser, ma vediamo come organizzare la formattazione tramite i tag più importanti:

- tag <P>: Viene utilizzato per indicare un paragrafo, il tag dovrà essere aperto all'inizio e chiuso alla fine:

```
<p>questo paragrafo potrebbe andare anche su piu' righe ma verrebbe sempre considerato nella formattazione come un paragrafo</p>
```

Risultato:

questo paragrafo potrebbe andare anche su piu' righe ma verrebbe sempre considerato nella formattazione come un paragrafo

- tag <DIV>: Viene utilizzato per indicare una sezione all'interno di un paragrafo, vedremo che sarà utilizzato soprattutto per definire stili diversi per righe diverse con i CSS. Il tag dovrà essere aperto all'inizio e chiuso alla fine:

```
<p>Per cominciare dobbiamo:
  <div>essere sicuri di conoscere il linguaggio html</div>
</p>
```

Risultato:

Per cominciare dobbiamo:
essere sicuri di conoscere il linguaggio html

- tag <H1>: Indica al browser che si tratta di un titolo di Primo livello, esistono anche i tag <H2>, <H3>, e così via per indicare i livelli inferiori:

```
  <H1>I livello</H1>
  <H2>II livello</H2>
  <H3>III livello</H3>
  <H4>IV livello</H4>
```

Risultato:

I livello

II livello

III livello

IV livello

Alcuni tag invece non hanno necessità di essere `chiusi` poiché vengono utilizzate per generare delle singole azioni e non per definire il tipo di testo in una selezione.

tag
: Viene utilizzato per obbligare il testo ad andare a capo, pur rimanendo nello stesso paragrafo.

```
<p>questo paragrafo potrebbe andare anche su piu'<br>
righe ma verrebbe<br>
 sempre considerato<br>
 nella formattazione come un paragrafo</p>
```

Risultato:

questo paragrafo potrebbe andare anche su piu'
righe ma verrebbe
sempre considerato
nella formattazione come un paragrafo

tag <HR>: Horizzontal Line, genera appunto una linea orizzontale utile per separare le parti tel testo:

```
<p>questo paragrafo potrebbe andare anche su piu'<br>
righe ma verrebbe<br>
 sempre considerato<br>
 nella formattazione come un paragrafo</p>
<hr>
<p>questa e' una frase che non c'entra niente con quella sopra</p>
```

Risultato:

questo paragrafo potrebbe andare anche su piu'
righe ma verrebbe
sempre considerato
nella formattazione come un paragrafo

---

questa e' una frase che non c'entra niente con quella sopra

5.2.3. Link – collegamenti ipertestuali

La peculiarità del linguaggio HTML è sicuramente la possibilità di creare dei collegamenti *ipertestuali* ad altri documenti html, per utilizzare questa funzione si usa il tag `<a>` in particolare nella forma:

```
 <a href="destinazione">testo del link</a>
```

Esempio:

```
quì c'e' il link per la home page dei
<a href="http://lnf.infn.it/">Laboratori Nazionali di Frascati</a>
dell'<a href="http://www.infn.it/">INFN</a>
```

Risultato:

quì c'e' il link per la home page dei Laboratori Nazionali di Frascati dell'INFN

I link possono *puntare* ad indirizzi **assoluti**, come nell'esempio precedente, oppure **relativi**, ovvero definendo solo il percorso a partire dalla posizione corrente. Ad esempio, se in una pagina html sita in:

`http://www.pippo.com/test/provevarie/uno.html`

scrivessimo:

```
Vai alla <a href=”due.html”>prossima pagina</a>
```

Cliccando sul link “prossima pagina” il browser andrebbe a caricare:
`http://www.pippo.com/test/provevarie/due.html`

È possibile anche specificare una sottodirectory di quella attuale ad esempio scrivendo sempre nel file:
`http://www.pippo.com/test/provevarie/uno.html`

```
Vai alla <a href=”archivio/due.html”>prossima pagina</a>
```

la destinazione sarebbe:
`http://www.pippo.com/test/provevarie/archivio/due.html`

Normalmente le destinazioni dei link vengono aperte nella finestra che mostra la pagina sorgente, è tuttavia possibile aprire un collegamento in una nuova pagina del browser inserendo nel tag `<a>` l’attributo `target` ed definendone il valore come “_blank”:

```
Vai alla <a href=”due.html” target=”_blank”>prossima</a>
```

5.2.4. Immagini

All’interno della pagina html è possibile inserire delle immagini, queste possono essere in vari formati comprensibili dal browser, generalmente questi sono:

- JPEG;
- GIF
- PNG.

Per inserire un’immagine è necessario utilizzare il tag `<img>`:

```
<img src=”imgs/pippo.jpg” alt=”Foto di Pippo” border=5>
```

Questo mostrerà l’immagine `pippo.jpg` dalla directory `imgs/`, alt definisce un testo alternativo per chi non volesse visualizzare le immagini nel browser, nota che **l’attributo `alt` è obbligatorio**; inoltre è stato definito un bordo intorno all’immagine di 5 pixel, per eliminarlo è sufficiente impostarlo a 0 (zero).

5.2.5. Sintassi: annidameno dei tag

Ora che abbiamo visto i vari tag, vediamo come annidarli tra loro, infatti un’altra caratteristica dell’html è che generalmente i tag hanno una gerarchia dalla quale verrà definito l’aspetto definitivo della pagina.

Per questo è molto utile nella scrittura del codice, indentare il testo a seconda del livello, vedremo nel corso del manuale che questo risulterà molto utile poi per la lettura a posteriori e per evitare i classici errori di sintassi, è infatti proibito in html aprire i tag in un ordine e chiuderli in un altro.

Esempio sbagliato:

```
<p>questo è un <a href="pippo.html">link</p></a>
```

Esempio corretto:

```
<p>questo è un <a href="pippo.html">link</a></p>
```

Ad esempio potremmo utilizzare al posto del testo del link un'immagine in questo modo:

```
<a href="http://www.pippo.it">
  <img src="imgs/pippo.jpg"
       alt="Foto di Pippo"
       border=5>
</a>
```

Come vedete è possibile organizzare l'aspetto del codice anche andando a capo all'interno dei tag, in questo modo questo sarà molto più leggibile.

5.2.6. Caratteri speciali

Come già accennato in precedenza per le url anche all'interno del codice html ci sono dei caratteri che non possono essere scritti "tranquillamente", in particolare la maggior parte sono caratteri che identificano le metainformazioni della pagine, come ad esempio i tag e che quindi se vogliono essere mostrati nel risultato finale della pagine o anche come valori di attributi dei tag, devono essere trattati in modo particolare.

Segue una tabella con i caratteri più comunemente utilizzati e proibiti nel codice html:

Carattere	Codice html
<	<
>	>
&	&

Un altro carattere speciale che troviamo è ** ** che vuol dire testualmente "Non Breaking Space" ovvero uno spazio non rompibile, questo carattere viene usato in due occasioni, la prima, nella sua funzione nativa, è quella di inserire uno spazio che non possa essere spezzato da un invio a capo, quindi le due parole alla sua destra e alla sinistra rimarranno sempre sulla stessa riga; la seconda funzione, quella più comune è di inserire più spazi all'interno di una riga, infatti all'interno del codice uno o più spazi semplici ( ) verranno sempre considerati come uno.

Esempi:

Sbagliato	Corretto
`<p>5 è < di 10</p>`	`<p>5 è < di 10</p>`
`<p>Dael & Mario</p>`	`<p>Dael & Mario</p>`
`<p> <img alt="Dael & Mario" src="dm.jpg"> </p>`	`<p> <img alt="Dael & Mario" src="dm.jpg"> </p>`
`<p>spazi fino a quì</p>`	`<p>spazi      fino a quì</p>`

5.2.7. Tabelle

Le tabelle sono l'elemento più utilizzato per formattare le parti visibili dell'html, infatti vedremo che saranno molto utili per posizionare testo e/o immagini dove vogliamo e allo stesso tempo fare in modo che indipendentemente dalla grandezza della finestra del browser questi rimangano ordinati secondo un criterio da noi definito.

Le tabelle a differenza dei tag che abbiamo visto fin ora è completa se e solo se vengono definite al suo interno almeno una riga ed almeno una cella per ogni riga.

Per indicare l'inizio di una tabella si usa il tag `<table>`, per definirne la fine invece si usa, come ormai dovrebbe essere pacifico, il tag `</table>`; questo però non basta appunto perchè la tabella sia definita, al suo interno dobbiamo infatti inserire il tag `<tr>` per iniziare una riga ed il tag `<td>` per le celle di una riga.

Il contenuto può essere inserito solo all'interno di una cella e **mai prima della definizione del** `<td>`.

Esempio:

```
<table border=1>
  <tr>
    <td>
       cella 1 della riga 1
    <td>
       cella 2 della riga 1
  <tr>
    <td>
       cella 1 della riga 2
    <td>
       cella 2 della riga 2
  <tr>
    <td>
       cella 1 della riga 3
    <td>
       cella 2 della riga 3
</table>
```

Risultato:

cella 1 della riga 1	cella 2 della riga 1
cella 1 della riga 2	cella 2 della riga 2
cella 1 della riga 3	cella 2 della riga 3

Ne risulta quindi una tabella con tre righe e due colonne, le colonne infatti sono rappresentate dal numero di celle per ogni riga.

Il numero di celle deve essere uguale per ogni riga per evitare errori nella visualizzazione, a meno che non vogliamo che una cella occupi lo spazio di due o più celle definite nelle altre righe, per questo si può utilizzare l'attributo `colspan` nel tag `<td>`:

Esempio:

```
<table border=1>
  <tr>
    <td>
       cella 1 della riga 1
    <td>
       cella 2 della riga 1
    <td>
       cella 3 della riga 1
  <tr>
    <td>
       cella 1 della riga 2
    <td colspan=2>
       cella 2 della riga 2 su due colonne
  <tr>
    <td>
       cella 1 della riga 3
    <td>
       cella 2 della riga 3
    <td>
       cella 3 della riga 3
</table>
```

Risultato:

cella 1 della riga 1	cella 2 della riga 1	cella 3 della riga 1
cella 1 della riga 2	cella 2 della riga 2 su due colonne	
cella 1 della riga 3	cella 2 della riga 3	cella 3 della riga 3

È possibile anche fare in modo che una cella venga a cavallo tra due righe tramite l'attributo `rowspan` sempre nel tag `<td>`:

Esempio:

```
<table border=1>
  <tr>
    <td>
       cella 1 della riga 1
    <td>
       cella 2 della riga 1
    <td>
       cella 3 della riga 1
  <tr>
    <td>
       cella 1 della riga 2
    <td rowspan=2>
       cella 2 della riga 2 su due righe
    <td>
       cella 3 della riga 2
  <tr>
    <td>
       cella 1 della riga 3
    <td>
       cella 3 della riga 3
</table>
```

Risultato:

cella 1 della riga 1	cella 2 della riga 1	cella 3 della riga 1
cella 1 della riga 2	cella 2 della riga 2 su due righe	cella 3 della riga 2
cella 1 della riga 3		cella 3 della riga 3

È da notare che nella terza riga abbiamo dovuto definire solo due celle perché la cella 2, quella centrale viene occupata dalla cella 2 della riga 2, sono state quindi definite solo le celle che andranno a posizionarsi nella prima e terza colonna.

Nel tag `<table>` come visto negli esempi è stato definito l'attributo `border`, questo serve appunto per mostrare un bordo intorno alle celle ed alla tabella stessa, questo bordo può essere tuttavia omesso assegnando il valore 0 (zero) permettendoci di organizzare il testo senza mostrare la presenza di una tabella:

Esempio:

```
<table border=0>
  <tr>
    <td>
       cella 1 della riga 1
    <td>
       cella 2 della riga 1
    <td>
       cella 3 della riga 1
  <tr>
```

```
    <td>
       cella 1 della riga 2
    <td rowspan=2>
       cella 2 della riga 2 su due righe
    <td>
       cella 3 della riga 2
  <tr>
    <td>
       cella 1 della riga 3
    <td>
       cella 3 della riga 3
</table>
```

Risultato:

cella 1 della riga 1	cella 2 della riga 1	cella 3 della riga 1
cella 1 della riga 2	cella 2 della riga 2 su due righe	cella 3 della riga 2
cella 1 della riga 3		cella 3 della riga 3

È possibile definire anche la dimensione degli elementi della tabella, in modo assoluto in pixel oppure in modo relativo in percentuale rispetto all'oggetto in cui l'elemento è contenuto, questa può essere definita tramite gli attributi `width` e `height`, il primo per la larghezza ed il secondo per l'altezza.

Questi attributi possono essere definiti entrambi per le celle, solo la larghezza per la tabella e nessuno per la riga; per quanto riguarda le relazioni delle misure in percentuale seguire la presente tabella:

Elemento	**Width %**	**Height %**
`<table>`	Relativo alla pagina	**NON definibile**
`<tr>`	**NON definibile**	**NON definibile**
`<td>`	Relativo alla riga	Relativo alla tabella

Mentre per le misure in pixel è possibile utilizzare la forma:

```
<td width=130 height=50>
```

Per quelle in percentuale è necessario inserire il valore tra doppi apici in quanto il carattere % potrebbe creare problemi di sintassi:

```
<td width="20%" height="5%">
```

Un altro attributo utile nella definizione delle celle è `valign` che specifica l'allineamento verticale del contenuto della cella, quindi nel tag `<td>`, questo può essere: **`top`** (alto), **`middle`** (centro), **`bottom`** (basso).

```
<td valign=top width="20%" height="5%">
```

È da ricordare che come tutti i tag html anche le tabelle possono essere annidate, è possibile infatti inserirne di nuove all'interno di celle di altre tabelle, sempre ovviamente dopo un tag `<td>`:

Esempio:

```
<table border=1>
  <tr>
    <td>
       cella 1 della riga 1
    <td>
       cella 2 della riga 1
  <tr>
    <td>
       <table border=1>
         <tr>
           <td>
             c1r1 sottotabella
           <td>
             c2r1 sottotabella
         <tr>
           <td>
             c1r2 sottotabella
           <td>
             c2r2 sottotabella
       </table>
    <td>
       cella 2 della riga 2
</table>
```

Risultato:

cella 1 della riga 1	cella 2 della riga 1
<table><tr><td>c1r1 sottotabella</td><td>c2r1 sottotabella</td></tr><tr><td>c1r2 sottotabella</td><td>c2r2 sottotabella</td></tr></table>	cella 2 della riga 2

5.2.8. Approfondimenti

Per una guida completa e approfondimenti sul linguaggio HTML/4.01 è necessario fare riferimento alla relativa sezione del W3C in inglese: http://www.w3.org/Markup/

Esiste una versione tradotta in italiano da Michele Diodati reperibile da:
http://www.lnf.infn.it/~dmaselli/stage/html401/

6 - CSS (Cascading Style Sheets)

6.1. Introduzione

Css (Cascading Style Sheets) è un linguaggio di formattazione dello stile di un documento Html, o di una serie di documenti in cascata, da qui il loro nome.

In cascata vuol dire che è possibile aggiornare lo stile di parte di un documento o di un documento intero, o di una serie di documenti, lavorando su di una sola fonte.

I limiti di cui soffre l'Html si evincono dalla scarsa possibilità che questo linguaggio offre sul controllo dello stile di un documento, lasciando allo sviluppatore un maggior dispendio di tempo (per formattare ogni singolo elemento) e di risorse (occupando più spazio in termini di byte).

Riassumendo, i Css nei confronti dell'Html sono:

- più pratici
- più potenti

6.1.1. Praticità dei Css

Esaminiamo il seguente codice Html/4.0 atto a formattare un semplice testo, impostando il colore rosso ed il grassetto:

```
<p><font color="#800000"><b>Testo formattato con semplice
Html</b></font><p>
```

Nulla di particolarmente complicato, soluzione ideale per una singola pagina informativa creata al volo, o per un sito di poche, massimo una decina di pagine, senza alcuna pretesa di tipo grafico.

Consideriamo un sito composto da 500 pagine, di portata quindi ancora bassa per un portale o un sito di medio-grandi dimensioni, ed immaginiamo di volerlo formattare completamente con i Tag Html, li dove ci si dovesse rendere conto che il colore rosso ad un testo non va più bene, ci si troverebbe di fronte ad un grosso problema: aprire ogni singolo file e modificare a mano il colore di tutti i testi rossi presi singolarmente.

I Css, vedremo più avanti come, consentono di agire con un solo cambiamento, o comunque con pochi passaggi, su tutte le parti interessate.

6.1.2. Potenza dei Css

I Css, oltre a consentirci di limitare un lavoro altrimenti lungo e faticoso a pochi e semplici passaggi, consentono di effettuare stilizzazioni impossibili da ottenere con semplice codice Html.

Tanto per dire la prima che mi viene in mente, non è possibile con l'Html decidere di posizionare un elemento in un determinato punto della pagina se non creando una serie di tabelle ad hoc, dove comunque non è detto che l'effetto ottenuto sia quello desiderato.

6.1.3. Caratteristiche standard dei Css

Il linguaggio Css è nato grazie alla collaborazione di alcune tra le più famose softwarehouse produttrici di browser con il W3C, World Wide Web Consortium (http://www.w3.org), organizzazione mondiale atta a definire gli standard per lo sviluppo Web.

Questa collaborazione, come tutte queste forme di standardizzazione (ove possibile) dei linguaggi per il Web, è atta a definire la compatibilità del linguaggio nei confronti di tutti i browser e software inerenti lo sviluppo e/o la consultazione del Web.

6.1.4. Limiti dei Css

Non sono certo perfetti i risultati di tali collaborazioni, ovviamente ogni softwarehouse mira a portare l'acqua al suo mulino, o comunque determinate esigenze limitano il supporto dei Css a parte degli elementi.

Dunque NON ci troviamo di fronte ad un linguaggio del tutto crossbrowser, ovvero standard per tutte le piattaforme, quindi è necessario prestare attenzione agli elementi che si desidera utilizzare con la formattazione Css, consiglio quindi di installare sul proprio computer, al fine di effettuare dei test, almeno browser tipo Netscape Navigator ed Opera, oltre ad Internet Explorer, normalmente installato sui sistemi Windows.

Nel corso della guida esamineremo in generale gli elementi compatibili con questi tre browser, con i quali si può lavorare con una certa tranquillità.

6.2. Il codice CSS

Il codice Css può essere implementato in tre diversi modi a seconda delle esigenze:

- direttamente su di un elemento Html
- nell'header della pagina
- in un file esterno con estensione `.css`

Esaminiamo nel dettaglio queste tre categorie, dette rispettivamente fogli di stile incorporati, fogli di stile interni ed in fine fogli di stile esterni.

6.2.1. Fogli di stile incorporati

Questa pratica è in genere la meno usata, dato che finisce per equivalere al normale metodo di formattazione Html, anche se conserva la potenza e la versatilità dei css rispetto all'Html, ma può tornare utile per formattare un singolo elemento che, per necessità, deve essere diverso dai suoi simili, già definiti in un foglio di stile globale (interni, esterni).

Si utilizza l'attributo Html `style` all'interno del tag che si intende formattare con quel criterio, ecco un esempio:

```
<div style="color: #FF0000;">Questo testo apparirà in rosso</div>
```

6.2.2. Fogli di stile interni

Questo metodo è più utilizzato, consiste nell'inserire un blocco di stile all'interno dell'header della pagina, definendo un unico stile per i singoli elementi contenuti, i Tag di testo, i link, le tabelle, ecc.

Si utilizza il Tag `<style>` e `</style>` con l'attributo `type="text/css"` all'interno dei tag `<head>` del documento html:

```
<html>
   <head>
      <title>Il mio primo foglio di stile</title>
      <style type="text/css">
         div { color: #FF0000; font-family: Verdana; }
         a { color: #3366CC; text-decoration: None; }
      </style>
   </head>
<body>

<div>Testo formattato con i Css</div>
<a href="#">Link formattato con i Css</a>

</body>
</html>
```

6.2.3. Fogli di stile esterni

La terza modalità di utilizzo dei Css è rappresentata dall'inclusione di un file di testo semplice esterno con estensione `.css`.

La stringa di codice che si utilizza per richiamare il file esterno nella pagina, sempre all'interno di `<head>`, è la seguente:

```
<link rel="stylesheet" type="text/css" href="style.css">
```

nel caso il file si trovi nella stessa cartella del file Html che lo richiama, se invece si trovasse nella cartella stili la stringa sarebbe la seguente:

```
<link rel="stylesheet" type="text/css" href="stili/style.css">
```

Questo metodo è senza dubbio il migliore perchè rende possibile richiamare il file di stile in ogni pagina ottenendo ovunque l'effetto stilistico progettato nel file esterno, e li dove dovesse essere apportato un cambiamento globale ad una tipologia di elementi, ad esempio il colore dei link, è possibile cambiare lo stile ovunque con una sola operazione.

Ultima nota, nel file esterno non bisogna utilizzare i Tag `<style>` e `</style>`, ma solo codice Css.

6.3. Ereditarietà

Una cosa molto utile dei CSS è che le definizioni dei livelli alti sono ereditate dagli stili di livello più basso.

In pratica se viene definito un certo font ed un colore per il tag `P`, definendo una classe `P.intestazione`, è possibile ridefinire solo gli aspetti che vogliamo cambiare rispetto a `P`, quindi ad esempio ridefinendo solo il colore, il font riamarrà quello definito in `P`.

Inoltre l'ereditarietà funziona anche tra i vari modi di definire i css, infatti possono essere sovrascritti attributi definiti in un file `.css` esterno ridefinendo solo alcuni aspetti nell'attributo `style` dei tag html. Si dice che gli stili vanno in "cascata" in un nuovo stile "virtuale" con le seguenti regole di priorità (dal meno al più prioritario):

1. Browser default;
2. Fogli di stile esterni;
3. Fogli di stile interni (dentro il tag `<STYLE>`);
4. Stili inline (dentro gli elementi HTML con `style=`).

6.4. Sintassi

Esaminiamo questo codice Html/4.0 ormai obsoleto nonché *deprecated* e mettiamolo a confronto con l'equivalente codice Css:

```
<div><font color="#FF0000">Questo testo apparirà in
rosso</font></div>
```

vediamo ora come ottenere lo stesso risultato con i Css:

```
<div style="color: #FF0000;">Questo testo apparirà in rosso</div>
```

La sintassi Css differisce in verità non tantissimo da quella Html, possiamo notare due cose, non si utilizza come operatore di assegnazione il carattere = (uguale) ma il carattere : (due punti); ogni comando viene sempre chiuso col carattere ; (punto e virgola), questo per permettere di separare un'istruzione dall'altra, per questo nell'ultima istruzione di un blocco può essere omesso il punto e virgola, ma le buone norme di programmazione consigliano di utilizzarlo comunque.

Abbiamo tre parti:

- un Selettore
- una Proprietà
- un Valore

```
selettore { proprietà: valore; proprietà2: valore2; }
```

Un selettore, in genere è il tag HTML che si vuole definire, la proprietà è l'attributo da cambiare, e ad ogni proprietà viene associato un valore.

Vediamo come implementare un blocco di stile esterno al Tag, sia che si trovi all'interno dell'header della pagina, sia che faccia parte di un file esterno:

```
div {
   text-align: Center;
   color: #FF0000;
   font-size: 12px;
   font-family: Verdana;
   font-weight: Bold;
}
```

Questo metodo di indentazione del codice favorirà sicuramente la chiarezza e la leggibilità, permettendo un eventuale intervento successivo in maniera più agevole.

Nota: precedentemente ho fatto riferimento agli operatori di assegnazione, per chi non sapesse di cose si tratta, nella terminologia tecnica sia matematica che di programmazione, si intende per operatore di assegnazione un carattere utilizzato per assegnare il valore sulla destra all'attributo o proprietà sulla sinistra.

6.4.1. Commentare il codice

La sintassi Css utilizza la stessa simbologia del Javascript per definire un commento, sia che sia su di una sola riga o su più righe, l'apertura è rappresentata dai caratteri `/*` e la chiusura dai caratteri `*/`.

Vediamo lo stesso esempio riportato in questa lezione implementato con opportuni commenti, il che servirà a farsi un'idea preliminare della funzionalità dei codici Css:

```
div {
   /* Centra il testo rispetto alla pagina */
      text-align: Center;
   /* Imposta il colore del testo */
      color: #FF0000;
   /* Imposta le dimensioni del testo */
      font-size: 12px;
   /* Imposta il carattere del testo */
      font-family: Verdana;
   /* Evidenzia il testo col grassetto */
      font-weight: Bold;
}
```

Questo testo è il risultato della stilizzazione utilizzata nell'esempio.

6.5. Formattazione del Testo

Creiamo ora un primo esempio concreto di foglio di stile sulla scorta delle nozioni di base fin ora acquisite, preoccupiamoci di formattare una serie di testi utilizzando tutti gli attributi di stile dediti allo svolgimento di tali funzioni e tutti i trucchi per snellire il codice ed il lavoro.

La prima cosa che ci preoccupiamo di stabilire per la formattazione di un file testuale è la dimensione del testo, il carattere, ovvero il font da utilizzare ed eventualmente il colore, immaginiamo di voler stilizzare il testo contenuto in un Tag :

```
<style type="text/css">
   div {
      color: 003366;
      font-size: 12px;
      font-family: Verdana;
   }
</style>
```

Lì dove tutti gli elementi di testo rispettino un determinato tipo di stilizzazione, conviene stabilire all'interno del body le proprietà, ad esempio:

```
<style type="text/css">
   body {
      color: 003366;
      font-size: 12px;
      font-family: Verdana;
   }
</style>
```

Prego di prestare attenzione ad un particolare, non a caso ho evidenziato il codice inerente alle dimensioni del testo, lì dove un testo si trova all'interno di una cella di una tabella Html, anche se contenuto in un Tag `<div>`, le dimensioni del testo non saranno prese in considerazione, visualizzandole di default, è opportuno definire le dimensione dei `<td>` a parte, la soluzione migliore è questa:

```
<style type="text/css">
   body {
      color: 003366;
      font-family: Verdana;
   }
   td, div { font-size: 12px; }
</style>
```

Vediamo in questo modo anche come è possibile formattare due elementi contemporaneamente, ovvero col solo utilizzo di una virgola di separazione ed uno spazio opzionale, in questo caso "td, div".

Immaginiamo ora che una singola parola di un testo debba essere scritta in grassetto, con un carattere più grande ed un font diverso, utilizzeremo il Tag **`<span>`** che, a differenza del `<div>` non comporta un ritorno a capo, e gli assegneremo questo stile:

```
<span style="font-size: 15px; font-family: Tahoma; font-weight:
Bold;">www.lnf.infn.it</span>
```

Diventa in questo modo addirittura più conveniente utilizzare una formattazione Html semplice, proviamo a riscrivere il tutto:

```
<span style="font: Bold, 15px, Tahoma;">www.lnf.infn.it</span>
```

Nota: nel caso in cui il nome del font che si desidera utilizzare sia composto da più parole, sarà necessario inserirlo per intero tra singoli apici, ad esempio:

```
<div style="font-family: 'Times New Roman';"> www.lnf.infn.it </div>
```

Passiamo alla definizione di altri elementi di formattazione da assegnare ad elementi testuali:

- Testo centrato: `text-align: Center;`
- Colore di sfondo: `background-color: #FFFFBB;`
- Corsivo: `font-style: Italic;`
- Sottolineato: `text-decoration: Underline;` (da utilizzare con moderazione data la somiglianza con i link testuali)

6.6. Effetti sui Link

La potenza dei Css ci permette di gestire in maniera ottimale lo stile dei link di una pagina, ottenendo anche una serie di effetti particolarmente gradevoli.

Col normale Html possiamo agire sul colore dei link semplicemente implementando nel body il seguente codice:

```
<body link="#0000FF" alink="#FF0000" vlink="#CECECE">
```

impostando in questo modo il colore del link da visitare in blu (`#0000FF`), il colore del link attivo il rosso (`#FF0000`) ed il colore dei link visitati in grigio (`#CECECE`).

Con i Css possiamo agire analogamente sulle stesse tre proprietà e non solo, possiamo utilizzare una quarta proprietà che ci consente di scatenare un effetto al passaggio del mouse su di un link, vediamole:

```
a:link { color: #0000FF; } /* link da visitare */
a:active { color: #FF0000; } /* link attivo */
a:visited { color: #CECECE; } /* link visitato */
a:hover { color: #FF0000; } /* link al passaggio del mouse */
```

impostando, ad esempio, anche il colore del link al passaggio del mouse in rosso.

Specificate le quattro proprietà, vediamo come possiamo agire in maniera più snella e veloce, impostiamo il colore del link da visitare, attivo e visitato di una sola tonalità e con un solo passaggio, abbiamo due modi per fare questo, vediamoli entrambi:

```
a:link, a:active, a:visited { color: #0000FF; } /* primo metodo */
a { color: #0000FF; } /* secondo metodo */
```

Altro vantaggio offerto dai Css è quello di eliminare la sottolineatura dei link e di reimpostarla ad esempio al passaggio del mouse, o volendo possiamo scegliere anche di far apparire la sottolineatura al di sopra della scritta, vediamo un esempio:

```
a { color: #0000FF; text-decoration: None; } /* il link in partenza
non è sottolineato */
a:hover { color: #FF0000; text-decoration: Underline; } /* il link
diventa sottolineato  al passaggio del mouse */
```

in questo modo, invece, otteniamo l'effetto di "sopralineatura"

```
a { color: #0000FF; text-decoration: None; }
a:hover { color: #FF0000; text-decoration: Overline; }
```

Possiamo anche combinare i due effetti:

```
a { color: #0000FF; text-decoration: None; }
a:hover { color: #FF0000; text-decoration: Underline Overline; }
```

Gli effetti che si possono ottenere sono svariati, possiamo ad esempio scegliere di impostare un link che al passaggio del mouse aumenta le proprie dimensioni o viene evidenziato in grassetto o in corsivo, basta provare e riprovare finchè l'effetto desiderato no salta fuori!

6.7. Classi e Id

Un foglio di stile che consenta al designer di disporre di un'ampia gamma di elementi Html da stilizzare non è completamente fluido e versatile se non ci fosse l'opportunità di poter definire stili diversi per una stessa categoria di elementi Html, ad esempio, il Tag <p> che avrà la sua stilizzazione, non potrà essere utilizzato con un criterio grafico e stilistico diverso in un'altra occorrenza dello stesso in un dato punto della pagina o dl sito.

Mi spiego meglio, se imposto ad un Tag di testo il colore rosso, non potrò utilizzare lo stesso tipo di Tag per un testo che voglio che sia verde, se non scindendo lo stesso Tag in una serie di "sottotag": questo il principio dell'utilizzo delle classi.

Una classe potrebbe dunque definirsi come un sottoinsieme appartenente ad un'unica categoria che può avere caratteristiche diverse da un'altra classe creata come altro sottoinsieme della stessa categoria.

Chiariamo ancora di più le idee con un provvidenziale esempio: supponiamo di voler utilizzare il Tag <p> per definire i testi delle nostre pagine, ma sorge la necessità di utilizzare stili diversi a seconda della porzione di pagina in cui andiamo a scrivere, creiamo dunque due distinte classi come sottoinsiemi di <p>

```
<style type="text/css">
   p.testo_rosso { color: Red; }
   p.testo_verde { color: Green; }
</style>

<p class="testo_rosso">Questo testo è Rosso</p>
<p class="testo_verde">Questo testo è Verde</p>
```

La sintassi per la creazione di una classe è banale, la forma è "nome Tag - punto - nome classe", ad esempio p.testo_rosso.

Per richiamarla in un Tag Html si utilizza l'attributo `class`, ad esempio `<p class="testo_rosso">`.

In questo modo abbiamo creato una classe specifica per il Tag `<p>`, se dovessimo dunque provare a scrivere `<div class="testo_rosso">` non avremo alcuna risposta, essendo quella classe proprietaria di `<p>`, per ovviare a questo problema possiamo creare delle classi generiche, semplicemente utilizzando la forma "punto - nome classe", ad esempio `.testo_rosso`.

```
<style type="text/css">
.testo_rosso { color: Red; }
</style>

<p class="testo_rosso">Questo testo è Rosso</p>
<div class="testo_rosso">Anche questo testo è Rosso</div>
```

Ovviamente richiamare questo stile da un Tag che non utilizza l'attributo color, come nell'esempio, è perfettamente inutile.

Ma non finisce qui! possiamo creare delle classi di tipo id per richiamare lo stile mediante l'utilizzo di un foglio di stile Javascript.

La sintassi è pressoché analoga, utilizziamo il carattere "cancelletto" (#) al posto del punto, vediamo un esempio:

```
<style type="text/css">
   #nomeclasseid { color: Red; }
</style>

<div id="nomeclasseid">Questa è una classe di tipo id</div>
```

Attenzione: l'utilizzo di quest'altra tecnica è da utilizzarsi con cautela, assegnare lo stesso id ad un Tag vuol dire confondere un eventuale script Javascript che vi si riferisce, essendo l'id in programmazione un identificatore univoco.

6.8. Tabella degli attributi

Di seguito una tabella con le caratteristiche degli attributi CSS:

Nome	Valore	Valore default	Applicazione	Eredità	Percentuale
'background-attachment'	scroll | fixed | inherit	scroll		no	
'background-color'	<color> | transparent | inherit	transparent		no	
'background-image'	<uri> | none | inherit	none		no	
'background-position'	[[<percentage> | <length> | left | center | right] [<percentage> | <length> | top | center | bottom]?] | [[left | center | right] || [top | center | bottom]] | inherit	0% 0%		no	refer to the size of the box itself
'background-repeat'	repeat | repeat-x | repeat-y | no-repeat | inherit	Repeat		no	
'background'	['background-color' || 'background-image' || 'background-repeat' || 'background-attachment' || 'background-position'] | inherit	see individual properties		no	allowed on 'background-position'
'border-collapse'	collapse | separate | inherit	separate	'table' and 'inline-table' elements	yes	
'border-color'	[<color> | transparent]{1,4} | inherit	see individual properties		no	
'border-spacing'	<length> <length>? | inherit	0	'table' and 'inline-table' elements	yes	
'border-style'	<border-style>{1,4} | inherit	see individual properties		no	
'border-top' 'border-right' 'border-bottom' 'border-left'	[<border-width> || <border-style> || 'border-top-color'] | inherit	see individual properties		no	
'border-top-color' 'border-right-color' 'border-bottom-color' 'border-left-color'	<color> | transparent | inherit	the value of the 'color' property		no	
'border-top-style' 'border-right-style' 'border-bottom-style' 'border-left-style'	<border-style> | inherit	none		no	
'border-top-width' 'border-right-width' 'border-bottom-width' 'border-left-width'	<border-width> | inherit	medium		no	
'border-width'	<border-width>{1,4} | inherit	see individual properties		no	
'border'	[<border-width> || <border-style> || 'border-top-color'] | inherit	see individual properties		no	
'bottom'	<length> | <percentage> | auto | inherit	auto	positioned elements	no	refer to height of containing block
'caption-side'	top | bottom | inherit	top	'table-caption' elements	yes	
'clear'	none | left | right | both | inherit	none	block-level elements	no	
'clip'	<shape> | auto | inherit	auto	absolutely positioned elements	no	
'color'	<color> | inherit	depends on user agent		yes	
'content'	normal | [<string> | <uri> | <counter> | attr(<identifier>) | open-quote | close-quote | no-open-quote | no-close-quote]+ | inherit	normal	:before and :after pseudo-elements	no	
'counter-increment'	[<identifier> <integer>?]+ | none | inherit	none		no	
'counter-reset'	[<identifier> <integer>?]+ | none | inherit	none		no	
'cursor'	[[<uri> ,]* [auto | crosshair | default | pointer | move | e-resize | ne-resize | nw-resize | n-resize | se-resize | sw-resize | s-resize | w-resize | text | wait | help | progress]] | inherit	auto		yes	
'direction'	ltr | rtl | inherit	ltr	all elements, but see prose	yes	
'display'	inline | block | list-item | run-in | inline-block | table | inline-table | table-row-group | table-header-group | table-footer-group | table-row | table-column-group | table-column | table-cell | table-caption | none | inherit	inline		no	
'elevation'	<angle> | below | level | above | higher | lower | inherit	level		yes	
'empty-cells'	show | hide | inherit	show	'table-cell' elements	yes	
'float'	left | right | none | inherit	none	all, but see 9.7	no	
'font-family'	[[<family-name> | <generic-family>] [, <family-name>| <generic-family>]*] | inherit	depends on user agent		yes	
'font-size'	<absolute-size> | <relative-size> | <length> | <percentage> | inherit	medium		yes	refer to parent element's font size
'font-style'	normal | italic | oblique | inherit	normal		yes	
'font-variant'	normal | small-caps | inherit	normal		yes	
'font-weight'	normal | bold | bolder | lighter | 100 | 200 | 300 | 400 | 500 | 600 | 700 | 800 | 900 | inherit	normal		yes	
'font'	[['font-style' || 'font-variant' || 'font-weight']? 'font-size' [/ 'line-height']? 'font-family'] | caption | icon | menu | message-box | small-caption | status-bar | inherit	see individual properties		yes	see individual properties
'height'	<length> | <percentage> | auto | inherit	auto	all elements but non-replaced inline elements, table columns, and column groups	no	see prose
'left'	<length> | <percentage> | auto | inherit	auto	positioned elements	no	refer to width of containing block
'letter-spacing'	normal | <length> | inherit	normal		yes	
'line-height'	normal | <number> | <length> | <percentage> | inherit	normal		yes	refer to the font size of the element itself
'list-style-image'	<uri> | none | inherit	none	elements with 'display: list-item'	yes	
'list-style-position'	inside | outside | inherit	outside	elements with 'display: list-item'	yes	
'list-style-type'	disc | circle | square | decimal | decimal-leading-zero | lower-roman | upper-roman | lower-greek | lower-latin | upper-latin | armenian | georgian | none | inherit	disc	elements with 'display: list-item'	yes	
'list-style'	['list-style-type' || 'list-style-position' || 'list-style-image'] |	see individual	elements with 'display: list-item'	yes	

Nome	Valore	Valore default	Applicazione	Eredità	Percentuale
	inherit	properties			
'margin-right' 'margin-left'	<margin-width> | inherit	0	all elements except elements with table display types other than table and inline-table	no	refer to width of containing block
'margin-top' 'margin-bottom'	<margin-width> | inherit	0	all elements except elements with table display types other than table and inline-table	no	refer to width of containing block
'margin'	<margin-width>{1,4} | inherit	see individual properties	all elements except elements with table display types other than table and inline-table	no	refer to width of containing block
'max-height'	<length> | <percentage> | none | inherit	none	all elements except non-replaced inline elements and table elements	no	see prose
'max-width'	<length> | <percentage> | none | inherit	none	all elements except non-replaced inline elements and table elements	no	refer to width of containing block
'min-height'	<length> | <percentage> | inherit	0	all elements except non-replaced inline elements and table elements	no	see prose
'min-width'	<length> | <percentage> | inherit	0	all elements except non-replaced inline elements and table elements	no	refer to width of containing block
'orphans'	<integer> | inherit	2	block-level elements	yes	
'outline-color'	<color> | invert | inherit	invert		no	
'outline-style'	<border-style> | inherit	none		no	
'outline-width'	<border-width> | inherit	medium		no	
'outline'	['outline-color' || 'outline-style' || 'outline-width'] | inherit	see individual properties		no	
'overflow'	visible | hidden | scroll | auto | inherit	visible	block-level and replaced elements, table cells, inline blocks	no	
'padding-top' 'padding-right' 'padding-bottom' 'padding-left'	<padding-width> | inherit	0	all elements except elements with table display types other than table, inline-table, and table-cell	no	refer to width of containing block
'padding'	<padding-width>{1,4} | inherit	see individual properties	all elements except elements with table display types other than table, inline-table, and table-cell	no	refer to width of containing block
'page-break-after'	auto | always | avoid | left | right | inherit	auto	block-level elements	no	
'page-break-before'	auto | always | avoid | left | right | inherit	auto	block-level elements	no	
'page-break-inside'	avoid | auto | inherit	auto	block-level elements	yes	
'position'	static | relative | absolute | fixed | inherit	static		no	
'quotes'	[<string> <string>]+ | none | inherit	depends on user agent		yes	
'right'	<length> | <percentage> | auto | inherit	auto	positioned elements	no	refer to width of containing block
'table-layout'	auto | fixed | inherit	auto	'table' and 'inline-table' elements	no	
'text-align'	left | right | center | justify | inherit	'left' if 'direction' is 'ltr'; 'right' if 'direction' is 'rtl'	block-level elements, table cells and inline blocks	yes	
'text-decoration'	none | [underline || overline || line-through || blink] | inherit	none		no (see prose)	
'text-indent'	<length> | <percentage> | inherit	0	block-level elements, table cells and inline blocks	yes	refer to width of containing block
'text-transform'	capitalize | uppercase | lowercase | none | inherit	none		yes	
'top'	<length> | <percentage> | auto | inherit	auto	positioned elements	no	refer to height of containing block
'unicode-bidi'	normal | embed | bidi-override | inherit	normal	all elements, but see prose	no	
'vertical-align'	baseline | sub | super | top | text-top | middle | bottom | text-bottom | <percentage> | <length> | inherit	baseline	inline-level and 'table-cell' elements	no	refer to the 'line-height' of the element itself
'visibility'	visible | hidden | collapse | inherit	visible		yes	
'white-space'	normal | pre | nowrap | pre-wrap | pre-line | inherit	normal		yes	
'widows'	<integer> | inherit	2	block-level elements	yes	
'width'	<length> | <percentage> | auto | inherit	auto	all elements but non-replaced inline elements, table rows, and row groups	no	refer to width of containing block
'word-spacing'	normal | <length> | inherit	normal		yes	
'z-index'	auto | <integer> | inherit	auto	positioned elements	no	

6.9. Approfondimenti

Per una guida completa e approfondimenti sul linguaggio CSS è necessario fare riferimento alla relativa sezione del W3C in inglese: http://www.w3.org/Style/CSS/

In particolare per la versione 2.1: http://www.w3.org/TR/CSS21/

È possibile consultare in italiano una versione tradotta da Pasquale Popolizio della versione 1 dei CSS reperibile da:

http://www.lnf.infn.it/~dmaselli/stage/css1/

7 - Bibliografia

Appunti di informatica libera di Daniele Giacomini
url: http://a2.pluto.it/

La tecnologia di Internet di Mauro Boscarol
url: http://www.boscarol.com/pages/internet/

Internet & Applications di D. Mariano e C. Guadalupi
url: http://telemat.die.unifi.it/book/Internet/Applications/Welcome.html

W3C (World Wide Web Consortium):
HTTP - Hypertext Transfer Protocol
url: http://www.w3.org/Protocols/
HyperText Markup Language (HTML) Home Page
url: http://www.w3.org/Markup/
Cascading Style Sheets home page
url: http://www.w3.org/Style/CSS/

Linguaggi :: Css di Luca Ruggiero
url: http://www.lukeonweb.net/linguaggi.asp?lang=Css&cat=Manuale