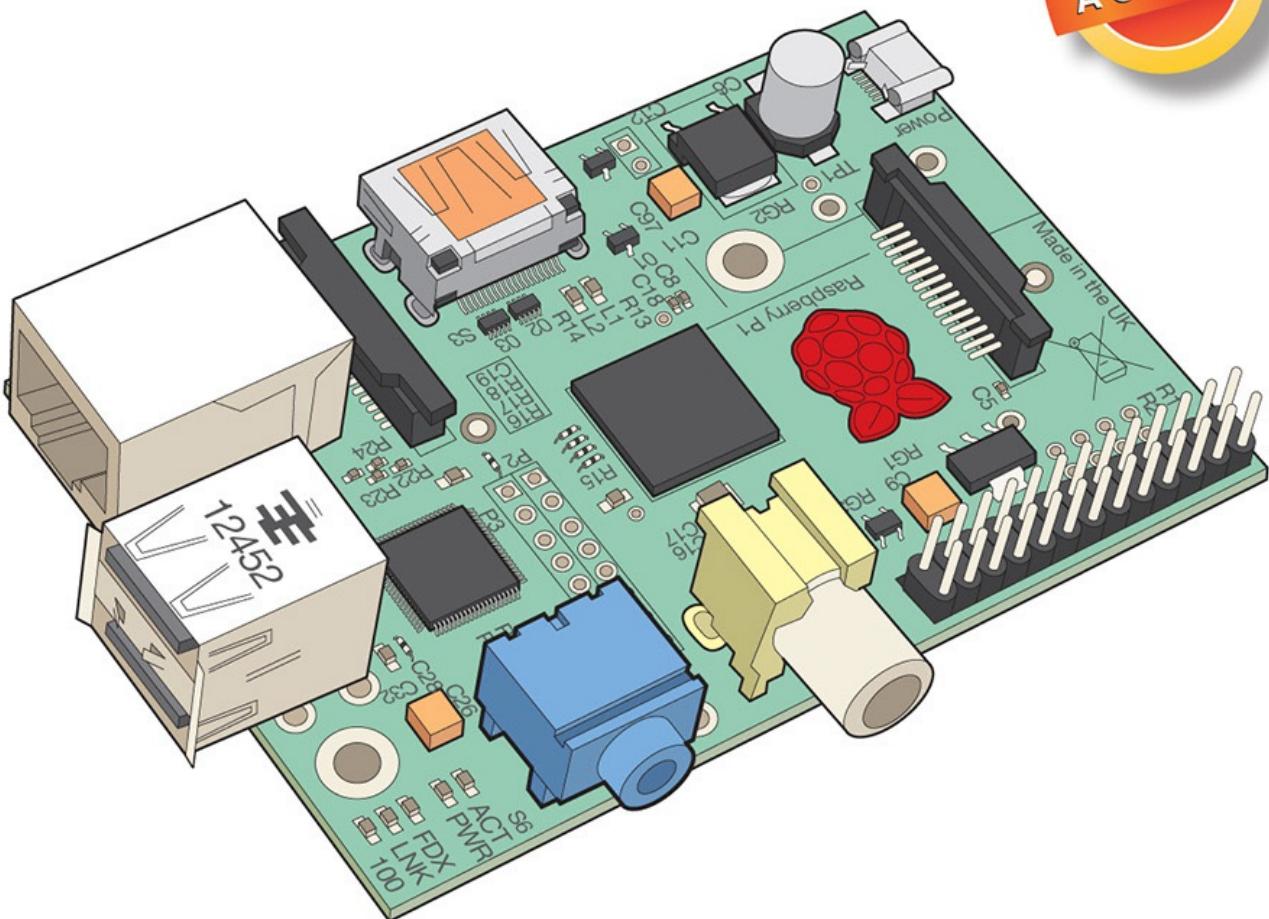


- Valter Minute -

Raspberry Pi

Guida all'uso



[Usare la command line di Linux e programmare con Python >>](#)

[Collegare sensori, attuatori e altri componenti hardware >>](#)

[Creare un media center e una stazione meteo >>](#)

[Cloud computing e Internet delle cose >>](#)

EDIZIONI
FAG
MILANO

*pro
DigitalLifeStyle

*pro
DigitalLifeStyle

Raspberry Pi

Guida all'uso

Valter Minute



Raspberry Pi | Guida all'uso

Autore: Valter Minute

Collana:

*^{pro}
DigitalLifeStyle

Publisher: Fabrizio Comolli

Editor: Marco Aleotti

Progetto grafico e impaginazione: Roberta Venturieri

Coordinamento editoriale, prestampa e stampa: escom - Milano

ISBN: 978-88-6604-415-4

Edizioni FAG Milano

Via G. Garibaldi 5 – 20090 Assago (MI) - www.fag.it

Introduzione

Chi, come l'autore, ha purtroppo passato la quarantina, ricorderà sicuramente il periodo degli home computer.

L'avvento di ZX81, VIC-20, ZX-Spectrum, Commodore 64, MSX, Commodore 128, Commodore 16 e tanti altri rese possibile la diffusione di massa dell'informatica, portando in tantissime case il computer che, fino a quel momento, era un oggetto misterioso visto di sfuggita in qualche ufficio, con tristi terminali a fosfori verdi, o magari al cinema, con tante lucette rosse che si accendevano e spegnevano secondo logiche comprensibili solo a tecnici in camice bianco.

I primi personal computer partivano con un interprete BASIC e anche solo per caricare (dal registratore a cassette!) qualche giochino bisognava digitare un po' di comandi.

E giocando con i comandi del VIC-20 (lussuoso home computer dotato di 3,5 Kbyte di RAM!) il sottoscritto ha appreso i primi rudimenti di programmazione.

L'avvento di computer sempre più potenti con sistemi operativi grafici e intuitivi, da una parte, e delle console, già "belle e pronte" per far girare i giochi dall'altra, ha reso l'informatica sempre più accessibile, ma anche sempre più astratta. Oggi per navigare in rete, scrivere documenti, disegnare, guardare e produrre video e foto non dobbiamo più scrivere alcun comando. Questo è sicuramente un enorme progresso (l'autore è nostalgico ma anche incredibilmente pigro!), ma ha anche limitato l'approccio alla programmazione.

Raspberry Pi è un piccolo ed economico personal computer che, a differenza dei suoi "fratelli maggiori", ci permette non solo di imparare a usare il software in dotazione, ma anche di modificarlo, estenderlo e, perché no, realizzarne di nuovo. Il suo costo ridotto e i suoi bassi consumi consentono di utilizzarlo come dispositivo dedicato (anche se solo per progetti amatoriali) e anche questo ci aiuta a esplorare e a scoprire.

Dopo aver familiarizzato con Linux e la programmazione in Python,

faremo interagire Raspberry Pi con il mondo reale, utilizzando componenti hardware facilmente reperibili e del costo di poche decine di euro (tutti elencati nell'Appendice alla fine del libro).

Sfrutteremo il nostro piccolo PC per imparare, nello spirito degli eroici home computer anni '80, immergendoci però da subito in progetti concreti che ci faranno comprendere le sue grandi potenzialità: dal server domestico al media center, dal sistema di videosorveglianza alla stazione meteorologica, in connessione con la rete e gli altri nostri device grazie ai servizi di cloud computing.

Partiremo, nel [Capitolo 1](#), esplorando l'hardware e il software, mentre nel [Capitolo 2](#) ci concentreremo su Linux e la sua interfaccia grafica.

Nel [Capitolo 3](#) inizieremo a "mettere le mani dentro al cofano" scoprendo la command line, che sfrutteremo nel [Capitolo 4](#) per trasformare Raspberry Pi in un server domestico. Nel [Capitolo 5](#) ci rilasseremo un attimo imparando a usare il Pi come media center, prima di immergerci nella programmazione con Python nel [Capitolo 6](#).

Il [Capitolo 7](#) è dedicato all'hardware e, in particolare, ai sensori, e ci consentirà di iniziare a "giocare" con i componenti elettronici e le basette sperimentali.

Il [Capitolo 8](#), invece, ci farà volare tra le nuvole del cloud-based computing, per poi precipitare a terra come le gocce di pioggia annunciate dalla nostra piccola stazione meteo, oggetto del [Capitolo 9](#).

A questo punto, come alla fine di ogni viaggio che si rispetti, il [Capitolo 10](#) vi darà spunti e idee per proseguire da soli e programmare la prossima tappa del vostro cammino, alla scoperta di quello che si può fare con un PC grande come una carta di credito.

Buon viaggio!

Valter Minute

Ringraziamenti

Questo è il mio primo libro e scriverlo è stata un'esperienza interessante, divertente, faticosa, esaltante e via aggettivando.

Il testo che leggete è stato reso possibile dalla fiducia di Andrea Maietta e Paolo Aliverti (ossia Frankenstein Garage), che ho avuto il piacere di conoscere personalmente e di sentire a diverse conferenze, che hanno ripescato dalla loro agenda i miei contatti quando si è trattato di scrivere un testo che presentasse Raspberry Pi. Spero di essere stato all'altezza della loro fiducia.

Ringrazio Marco Aleotti che mi ha seguito e aiutato nella mia esperienza da scrittore tecnico alle prime armi con pazienza infinita. Spero che i suoi sforzi siano ripagati dal risultato finale.

Ringrazio Paolo Patierno che, da appassionatissimo di tecnologia e di qualsiasi cosa in grado di eseguire una parvenza di software, ha contribuito a rivedere i contenuti segnalandomi idee interessanti e, soprattutto, un bel po' di correzioni. Gli errori che sono rimasti sono solo colpa mia.

Ringrazio mia moglie Francesca per la pazienza con cui sopporta da anni l'idea di dividermi con la passione per l'informatica, le nottate passate al PC o lo sguardo vitreo con cui la guardo mentre il mio cervello sta finendo di debuggare il codice che ho lasciato in ufficio mentre il mio corpo è apparentemente seduto a tavola. Chiedo scusa anche a Lorenzo e Leonardo per il tempo che gli ho rubato. Spero che la passione e l'interesse che ora mettono nei loro giochi li segua anche da grandi, qualunque sia la strada che decideranno di seguire. Continuando a divertirsi e ad aver voglia di imparare.

Infine voglio dedicare questo lavoro a mio padre. Mi ha insegnato a cercare sempre di capire come funzionano le cose, a non aver paura di sporcarsi le mani, a montare e smontare gli oggetti (più che altro rimontando e riaggiustando quelli che io maldestramente distruggevo). Mi ha insegnato il piacere di fare le cose provando a farle bene, anche se ci vuole più tempo e fatica. Sicuramente molta della mia passione

per la tecnologia deriva dai suoi insegnamenti, che più che da tante parole passavano dall'esempio di quelle mani grandi che trattavano con infinita delicatezza le cose che stava costruendo o riparando. Era un "maker", anche se nessuno aveva ancora inventato questa definizione. La cosa che più mi dispiace è che i miei figli non abbiano avuto il tempo di vedere quelle mani al lavoro. Proverò a rimpiazzarle, ma non sarà facile.

Alla scoperta di Raspberry Pi

L'hardware di Raspberry Pi e il software per sfruttarlo al meglio. Una guida passo passo per installare il sistema operativo.

Raspberry Pi è un piccolo personal computer, poco più grande di una carta di credito. Dal suo lancio commerciale nel febbraio 2012 ne sono stati venduti più un di un milione di esemplari.

Deve il suo successo a un prezzo molto interessante (25 dollari nella versione senza supporto di rete), unito a bassissimi consumi e a una discreta potenza di calcolo. Queste caratteristiche ne fanno una piattaforma molto interessante per chi vuole imparare e divertirsi sperimentando con l'elettronica e l'informatica.

Raspberry Pi viene fornito con una versione del sistema operativo Linux ed è quindi immediatamente pronto per cominciare a sperimentare. Il sistema è stato realizzato e commercializzato (tramite una serie di distributori) dalla Raspberry Foundation. Circa la metà dei dispositivi (quelli commercializzati in Europa e Nord America) vengono prodotti in Galles, in controtendenza rispetto alla spinta a portare in Estremo Oriente la produzione di tutti i prodotti elettronici di largo consumo.

La Raspberry Foundation ha avuto l'idea di realizzare un computer a basso costo per consentire ad appassionati e studenti di apprendere le basi dell'informatica, in modo analogo a quanto avveniva negli anni '80 con gli home computer come Commodore Vic20 e C64, Sinclair ZX81 e

Spectrum, e in particolare in Gran Bretagna con i BBC Micro.

Chi, come l'autore, inizia ad avere qualche capello bianco, ricorda con molta nostalgia quel periodo in cui i programmi si caricavano e salvavano su musicassette e diverse riviste specializzate si rivolgevano al vasto pubblico di chi voleva avvicinarsi al mondo dei computer apprendendo i rudimenti della programmazione.

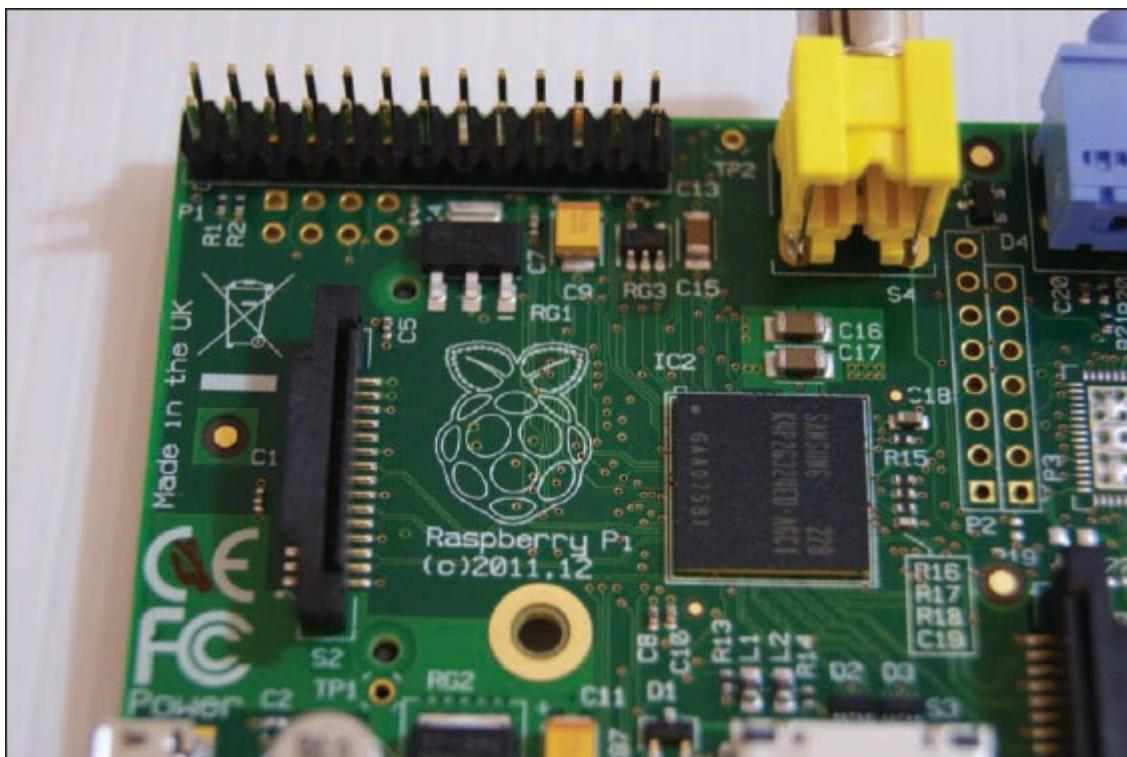


Figura 1.1 - Il logo della Raspberry Foundation stampato su Raspberry Pi.

Raspberry Pi è il risultato di diversi anni di lavoro e della collaborazione di moltissime persone sia in fase di design dell'hardware sia in fase di sviluppo del software.

Una buona parte dei progetti hardware e del codice sorgente del software è liberamente disponibile. Questo ha consentito la realizzazione di tantissimi progetti pubblicati in rete e la creazione di una community di appassionati in tutto il mondo. Oggi Raspberry Pi è, insieme ad Arduino (un sistema più semplice, basato su un microcontrollore senza sistema operativo), una delle piattaforme più utilizzate dai maker.

NOTA

La filosofia e gli strumenti operativi del movimento dei maker sono trattati in modo ampio e dettagliato ne *Il Manuale del Maker*, di Andrea Maietta e Paolo Aliverti, pubblicato da Edizioni FAG (www.ilmanualedelmaker.it).

In questo libro prenderemo confidenza con il sistema e affronteremo alcuni semplici progetti, sperando che questi non siano il fine ultimo, ma solo il gradino di partenza da cui ogni lettore potrà sperimentare e costruire qualcosa di nuovo e originale. Se avete voglia di sperimentare e imparare cose nuove spero che questo percorso vi dia gli strumenti per poter continuare e magari realizzare progetti ancora più interessanti e complessi di quelli, necessariamente semplici, proposti in queste pagine.

E adesso è ora di rimboccarsi le maniche, armarsi di passione e saldatore a stagno e scoprire cosa è possibile fare con un computer da 25 dollari!

L'hardware

Se state leggendo questo libro è molto probabile che abbiate già acquistato un Raspberry Pi e forse la definizione di "piccolo personal computer" vi ha lasciati un po' perplessi.

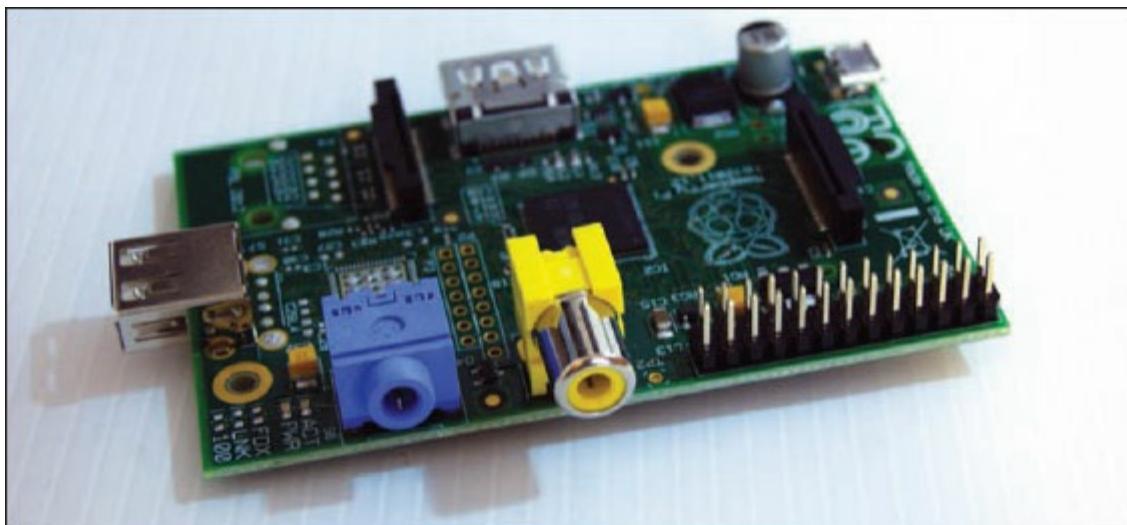


Figura 1.2 - Raspberry Pi modello A.

L'aspetto di Raspberry Pi è molto diverso da quello di un normale personal computer. È una piccola scheda elettronica che misura solo 8,5 x 5,5 centimetri, appena qualche millimetro in più di una carta di credito, e alta un paio di centimetri. Sulla scheda sono montati diversi connettori, alcuni dei quali sono simili a quelli che troviamo sui comuni PC desktop e portatili. Ci sono porte USB (una o due, a seconda del modello), un'uscita video HDMI, sempre più comune anche sui PC

portatili, un jack audio e, nel caso del modello B, una porta di rete Ethernet. C'è anche uno slot per una SD card, che sostituirà il disco fisso che siamo abituati a utilizzare con i nostri "normali" personal computer. Ci sono poi alcuni connettori che non capita di vedere spesso sui normali PC ma che, come vedremo nei prossimi capitoli, ci permetteranno di collegare al nostro Raspberry Pi sensori e altri dispositivi per fare cose impensabili con il nostro "tranquillo" PC da scrivania.

Esistono due modelli di Raspberry Pi, modello A e modello B. Il modello B è leggermente più potente del suo "fratellino" e costa 10 \$ in più. Le differenze tra i modelli riguardano la quantità di memoria RAM (256 MB per il modello A e i primi esemplari di modello B, 512 MB per il modello B prodotto da ottobre 2012 in poi), il numero di porte USB (il modello A ne ha una, il modello B due) e la porta Ethernet, presente solo sul modello B.

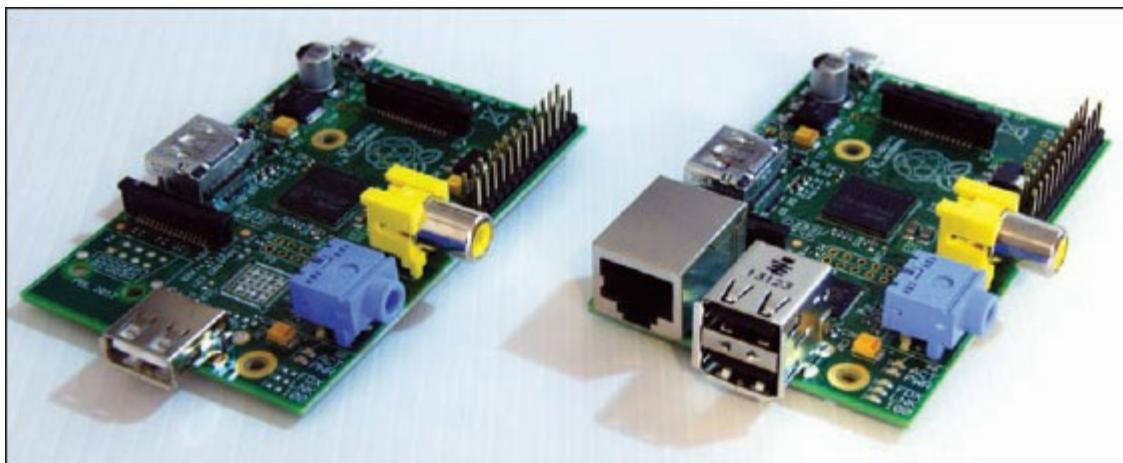


Figura 1.3 - Modello A e modello B a confronto.

La maggior parte degli esempi e dei progetti hardware proposti in questo libro funzionano su entrambi i modelli di Raspberry Pi. Nei Capitoli 4 e 7 si utilizzeranno le capacità di connessione in rete del sistema Linux. Il modello B potrà essere collegato direttamente a una rete via cavo, mentre entrambi i modelli potranno utilizzare una rete Wi-Fi o cablata tramite un adattatore USB, che si può acquistare per pochi euro nella maggior parte dei negozi di elettronica e informatica.

Se non avete ancora acquistato un Raspberry Pi spero che questo libro vi faccia venire voglia di sperimentare e di acquistarne uno. Sul sito www.raspberrypi.org potrete trovare una lista dei distributori ufficiali. Al momento quelli attivi in Europa sono RS components (it.rs.com)

comunque trovare molti altri rivenditori con una semplice ricerca sul web o sul sito di eBay.

I componenti di Raspberry Pi

Se avete avuto occasione di smontare un personal computer o, magari, vi siete assemblati personalmente il vostro PC, avrete sicuramente notato che la piccola scheda di Raspberry Pi è molto diversa dalle schede madri dei computer desktop.

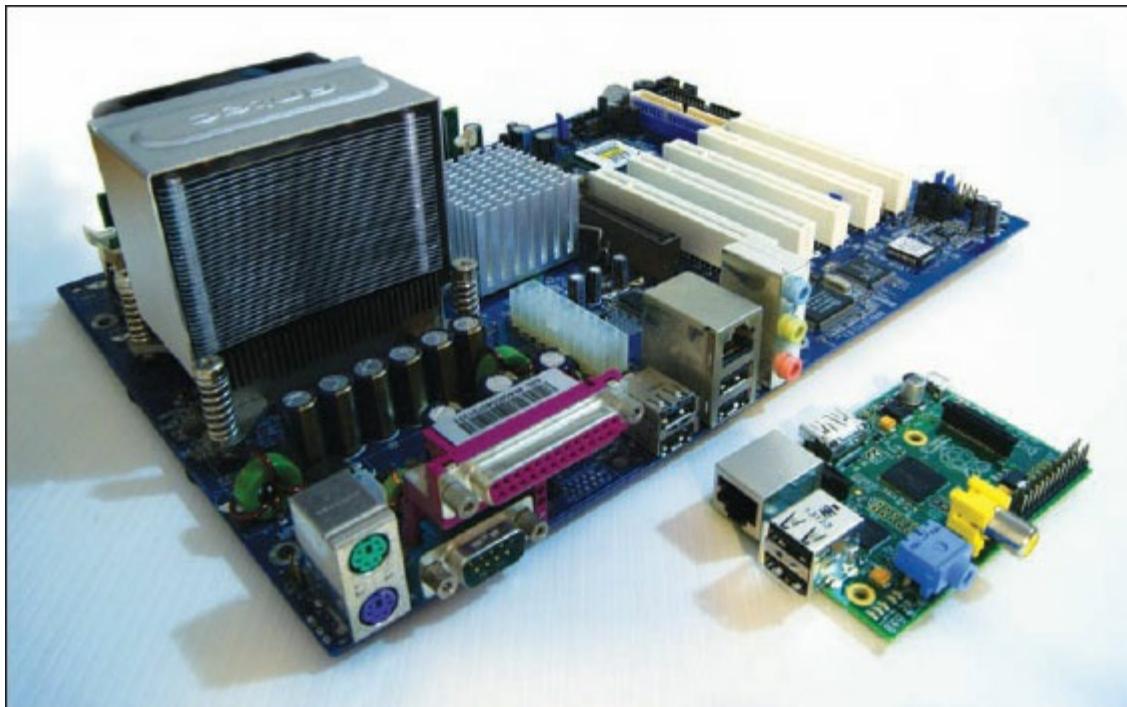


Figura 1.4 - Raspberry Pi e una motherboard per PC.

Mentre sulle motherboard è normale trovare una moltitudine di chip, slot di espansione, banchi di memoria, sul nostro Raspberry Pi campeggia un solo chip di circa un centimetro di lato. Questo chip racchiude la RAM e il processore. In realtà si tratta di due chip, montati uno sopra l'altro, con la RAM sopra e il processore sotto. Il "cuore" di Raspberry Pi è un processore Broadcom BCM2835. Anche questo è molto diverso dai processori comunemente utilizzati nei personal computer e molto più simile a quelli che si trovano negli smartphone, nei tablet, ma anche nei decoder per la TV via satellite o per il digitale terrestre. Il BCM2835 è quello che viene chiamato "System On Chip" (SOC). È un unico componente che integra al suo interno il processore

vero e proprio e tutta una serie di unità periferiche che nei normali personal computer sono esterne. Dentro quel centimetro quadrato di silicio troviamo un processore, un controller video dual core (equivalente alla scheda video dei normali PC), un controller audio, un controllore USB, una porta seriale, un controllore per SD card e controllori per altri bus e periferiche, che impareremo a conoscere nei prossimi capitoli, come I2C, SPI e GPIO.

I System On Chip presentano una serie di vantaggi rispetto ai processori tradizionali:

- costi più contenuti;
- dimensioni ridotte;
- minori consumi energetici.

Raspberry Pi sfrutta al meglio queste caratteristiche fornendoci a costi molto bassi un piccolo computer che consuma pochissima corrente.

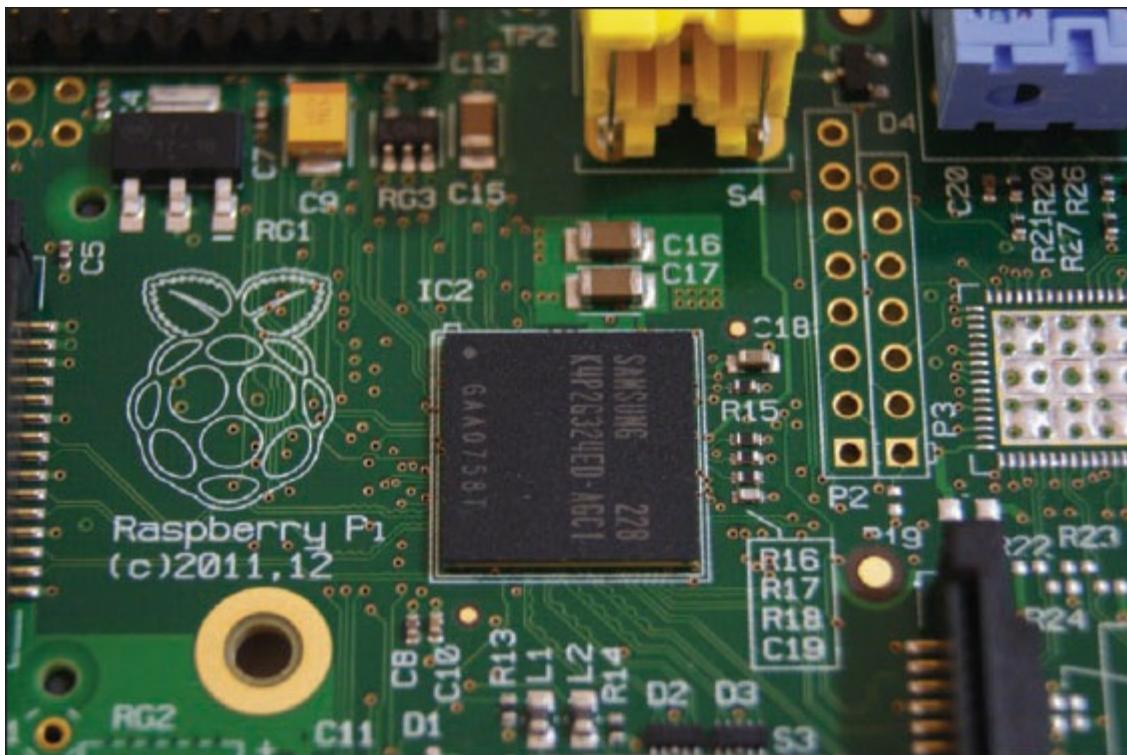


Figura 1.5 - Il processore di Raspberry Pi. Il SOC Broadcom è coperto dalla RAM, in questo caso prodotta da Samsung.

Il processore

Il processore di Raspberry Pi è basato sull'architettura ARM. ARM è una delle più diffuse architetture a microprocessore ed è diffusa su una miriade di dispositivi: dai tablet ai navigatori satellitari, dagli smartphone ai decoder per la TV digitale, dai dischi fissi ai router di

rete.

Nel solo 2012 sono stati venduti otto miliardi e settecento milioni (avete letto bene, miliardi!) di processori basati su core ARM.

NOTA

Fonte: "ARM Holdings plc - Annual Report & Accounts 2012",
[http://phx.corporate-ir.net/External.File?
item=UGFyZW50SUQ9MTczODc5fENoaWxksUQ9LTF8VHLwZT0z&t=1](http://phx.corporate-ir.net/External.File?item=UGFyZW50SUQ9MTczODc5fENoaWxksUQ9LTF8VHLwZT0z&t=1).

In realtà ARM Holdings, l'azienda che ha sviluppato questa tecnologia, non ha venduto nemmeno un pezzo di silicio. L'azienda inglese, infatti, progetta i processori e licenzia la tecnologia a diversi produttori tra cui Broadcom, Qualcomm, Samsung, Texas Instruments, Freescale, Marvell, NXP, Nvidia e altri, per un totale di più di ottocento licenze vendute.

Il primo processore ARM fu realizzato da Acorn, un'azienda inglese nata sull'onda del successo dei primi home computer (Commodore Vic 20 e C64, Sinclair ZX81 e Spectrum), che realizzò una serie di prodotti di discreto riscontro, soprattutto in Gran Bretagna dove furono utilizzati dalla BBC per i primi corsi di informatica trasmessi in televisione.

In realtà i processori ARM di primissima generazione non vennero mai utilizzati all'interno di prodotti di larga diffusione. Solo la seconda generazione raggiunse una discreta popolarità grazie al suo impiego in uno dei più potenti home computer prodotti negli anni '80: l'Acorn Archimedes.

Il processore includeva 30.000 transistor, meno della metà di quelli del processore Motorola 68000 utilizzato da Commodore, Amiga, Atari ST e Macintosh; tuttavia, proprio grazie al suo design più semplice, raggiungeva prestazioni decisamente migliori.

Il suo sistema operativo RiscOS proponeva un'interfaccia utente a finestre, ispirata a quella dei Macintosh lanciati qualche anno prima da Apple.

Il declino degli home computer, rimpiazzati dalle console per gli utilizzi ludici e dai PC compatibili per quelli "seri", portò Acorn ad attraversare un periodo difficile, segnato da problemi finanziari che portarono nel 1990 alla nascita di Advanced RISC Machines (ARM).

La tecnologia ARM si è sviluppata partendo da prodotti Apple, come il PDA Newton, e passando anche per diversi processori rilasciati da Intel,

fino ai giorni nostri.

I processori ARM sono oggi sinonimo di elevata integrazione, bassi consumi e ottime performance.

E nel resto di questo libro vedremo come sfruttare al meglio queste caratteristiche per divertirci e imparare con il nostro Raspberry Pi.

Memoria

A seconda del modello, Raspberry Pi può avere 256 MB (Model A) o 512 MB (Model B) di RAM. Abituati ai Gigabyte di memoria che ormai equipaggiano i personal computer, ma anche telefonini e tablet, questi numeri possono sembrare miseri. Se però pensiamo che i primi home computer erano equipaggiati con 64 Kilobyte di memoria (cioè 65.536 byte, contro i 268.435.456 del Raspberry Model A!) e che anche quelli più potenti come Amiga e Atari ST arrivavano ad appena 1 MB, possiamo capire che anche il nostro Raspberry ha grandi potenzialità e che starà a noi riuscire a sfruttarle al meglio per realizzare qualcosa di interessante e divertente.

E dobbiamo sempre ricordarci che l'uomo è arrivato sulla luna con computer che avevano 4 KB (4096 byte!) di RAM. I nostri 256 MB ci consentiranno di andare molto lontano!

Alimentazione

È possibile alimentare Raspberry Pi tramite la porta Micro-USB. Non è tuttavia possibile alimentarlo direttamente da un PC. È necessario utilizzare un alimentatore dedicato, anche un carica-batteria da telefonino, verificando che eroghi almeno 1 Ampère (troverete indicato "1A") di corrente; 2A potrebbero essere necessari se dovete collegare a Raspberry Pi diverse periferiche.

Se verificate malfunzionamenti o riavvi improvvisi del vostro Raspberry Pi, verificate che l'alimentatore sia adeguato. Il consumo di corrente del sistema varia a seconda dell'attività del processore, delle periferiche USB collegate e dell'attività in rete. Un alimentatore non abbastanza potente potrebbe consentirvi comunque di avviare il dispositivo e utilizzarlo limitatamente, ma potrebbe andare in crisi quando vengono lanciate applicazioni che usano pesantemente la CPU, la grafica oppure quando vengono collegati dispositivi USB esterni.



Figura 1.6 - Il connettore Micro-USB.

USB

Raspberry Pi dispone, a seconda del modello, di una o due porte USB Host. Queste porte possono essere utilizzate per collegare al nostro piccolo computer diverse periferiche che possono estendere le sue funzionalità. Per collegare più dispositivi alle nostre porte USB è sufficiente collegare un hub. Esistono hub USB a 4 o più porte reperibili per pochi euro nei negozi fisici e online di elettronica e informatica. Per collegare dispositivi che assorbono un quantitativo significativo di corrente è consigliabile collegare un hub alimentato. Questo tipo di hub è collegato a un alimentatore e non preleva quindi la corrente necessaria ad alimentare le periferiche direttamente dal nostro Raspberry Pi. Per utilizzare alcuni dispositivi USB sarà necessario installare i relativi driver e scopriremo meglio come farlo nei prossimi capitoli. Per installare il sistema operativo e utilizzare l'interfaccia grafica sarà necessario collegare a Raspberry Pi un mouse e una tastiera USB.

Se avete un PC desktop potrete prenderle in prestito. Impareremo in seguito a controllare il nostro Raspberry Pi direttamente da un PC o un Mac, che si vedranno così restituire la loro tastiera.



Figura 1.7 - Le porte USB di un Raspberry Pi Model B.

Storage

Raspberry Pi carica il sistema operativo da una SD card. È possibile utilizzare SD card fino a 32 GB e quindi memorizzare, oltre al sistema operativo e alle applicazioni, anche parecchi dati. Non è però possibile rimuovere la card mentre il sistema è in funzione, perlomeno quando si utilizza la versione di Linux (Raspbian) fornita con il dispositivo.

Per memorizzare grandi moli di dati o per trasferire informazioni dal vostro Raspberry Pi ad altri computer è possibile utilizzare dispositivi USB. Nel caso di dischi esterni la corrente erogata dalle porte USB di Raspberry Pi potrebbe essere insufficiente e sarà quindi necessario collegare alla porta USB di Raspberry Pi un hub USB alimentato.

Raspbian ci consentirà inoltre di collegarci in modo semplice a cartelle condivise da PC o altri sistemi in rete e, come vedremo nel [Capitolo 8](#), anche di memorizzare i nostri dati nella “nuvola” di internet. Quindi lo spazio per i nostri dati non mancherà. Nei prossimi capitoli vedremo come recuperare questi dati dal mondo che ci circonda tramite meravigliose diavolerie chiamate sensori.

Rete

Solo il Model B ha una porta Ethernet che può essere usata per collegarsi a reti locali cablate. Entrambi i modelli possono essere collegati a reti Wi-Fi utilizzando degli adattatori USB. Nel caso del Model A è possibile anche il collegamento a reti cablate, sempre tramite un adattatore USB. Il collegamento alle reti sarà oggetto dei Capitoli [3](#) e [4](#), in cui vedremo come utilizzare una console remota e configurare il

nostro Raspberry Pi per utilizzarlo come piccolo server domestico.

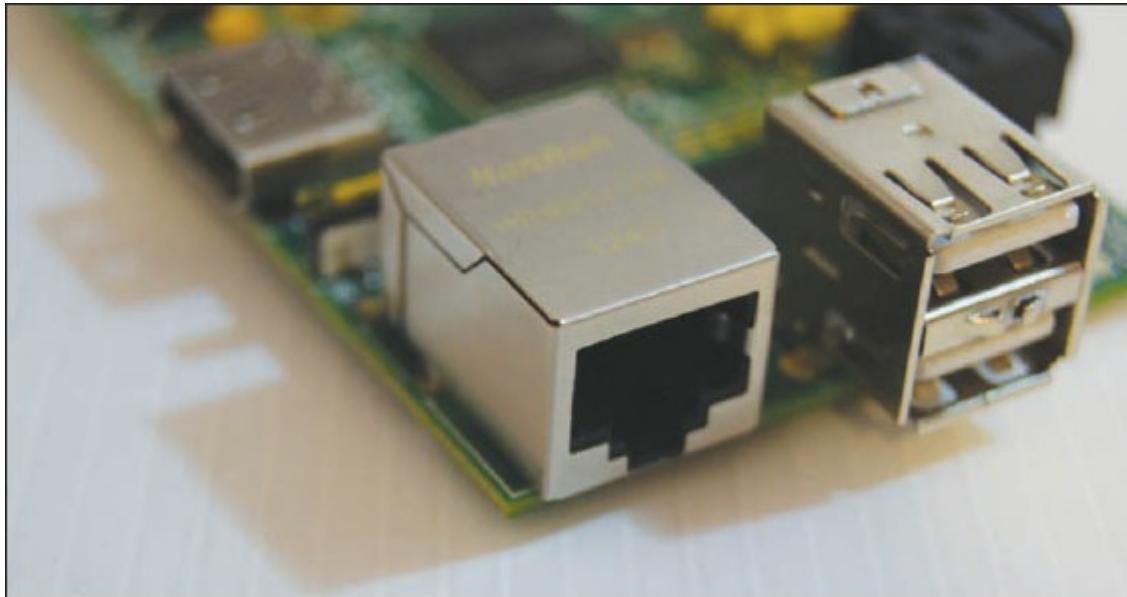


Figura 1.8 - Il connettore Ethernet di Raspberry Pi Model B.

Audio/Video

Raspberry Pi ha due uscite video: una HDMI e una RCA.

L'uscita HDMI (High-Definition Multimedia Interface) consente di trasferire i segnali audio e video in formato digitale e può essere utilizzata per collegare Raspberry Pi alla maggior parte dei televisori prodotti negli ultimi anni e a molti monitor per personal computer.

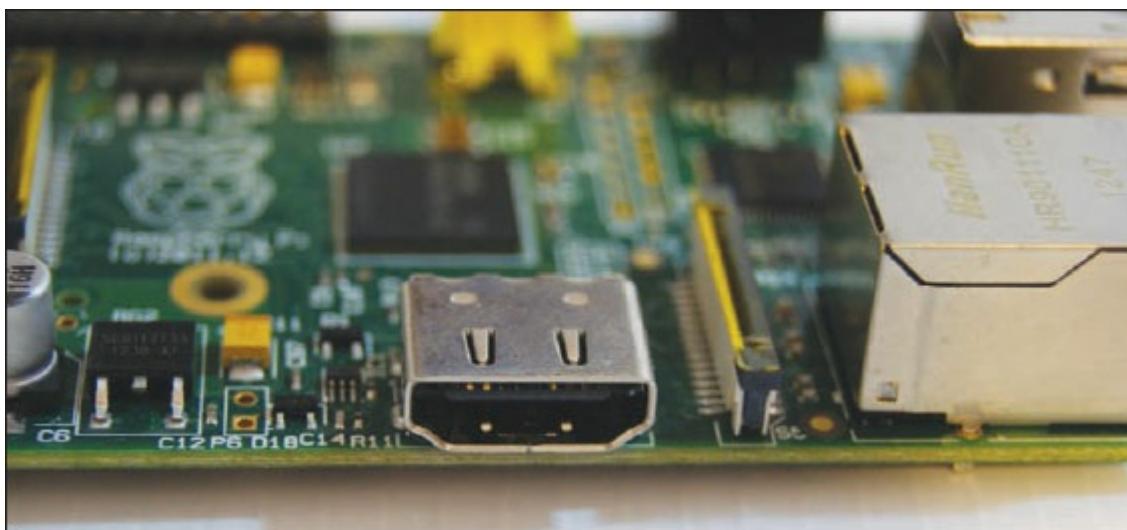


Figura 1.9 - Il connettore HDMI.

Se volete collegare Raspberry Pi a un monitor che non ha una porta HDMI e il monitor supporta un ingresso digitale DVI (Digital Visual Interface) è possibile utilizzare un cavo o un adattatore HDMI-DVI. Se invece il monitor supporta solamente l'ingresso analogico VGA (con il

connettore a 15 pin) sarà necessario utilizzare un convertitore HDMI-VGA.



Figura 1.10 - Convertitore HDMI-VGA.

Per collegare Raspberry Pi a televisori più vecchi è possibile utilizzare l'uscita RCA (Radio Corporation of America), che è in grado di generare segnali in formato PAL (standard europeo) o NTSC (standard statunitense) compatibili con la maggior parte dei TV color.

La risoluzione PAL/NTSC è più bassa di quella ottenibile via HDMI e la qualità del segnale RCA non è sempre ottimale. È preferibile, quando possibile, utilizzare l'uscita HDMI. Per avere un buon segnale senza troppi disturbi è utile cercare un cavo RCA ben schermato.

In ogni caso il collegamento al TV via cavo RCA farà rivivere a chi è già entrato negli "anta" i tempi degli home computer Commodore o Sinclair.

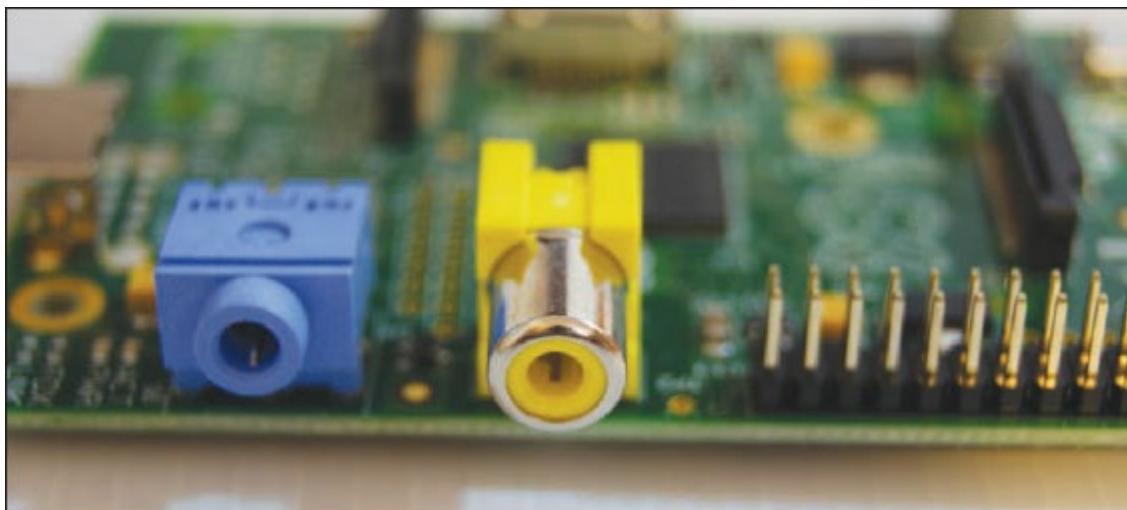


Figura 1.11 - Il connettore RCA.



Figura 1.12 - L'ingresso RCA di un vecchio TV color.

L'uscita RCA, a differenza di quella HDMI, non gestisce i segnali audio. Raspberry Pi ha un jack audio stereo da 3,5 mm che può essere utilizzato per collegare delle cuffie o delle casse esterne amplificate. Questo ci consentirà di poterlo utilizzare anche come media center, come descritto nel [Capitolo 5](#).

Non è invece disponibile un connettore audio in ingresso. Per registrare del suono con il nostro Raspberry Pi dovremo ricorrere a un microfono USB.

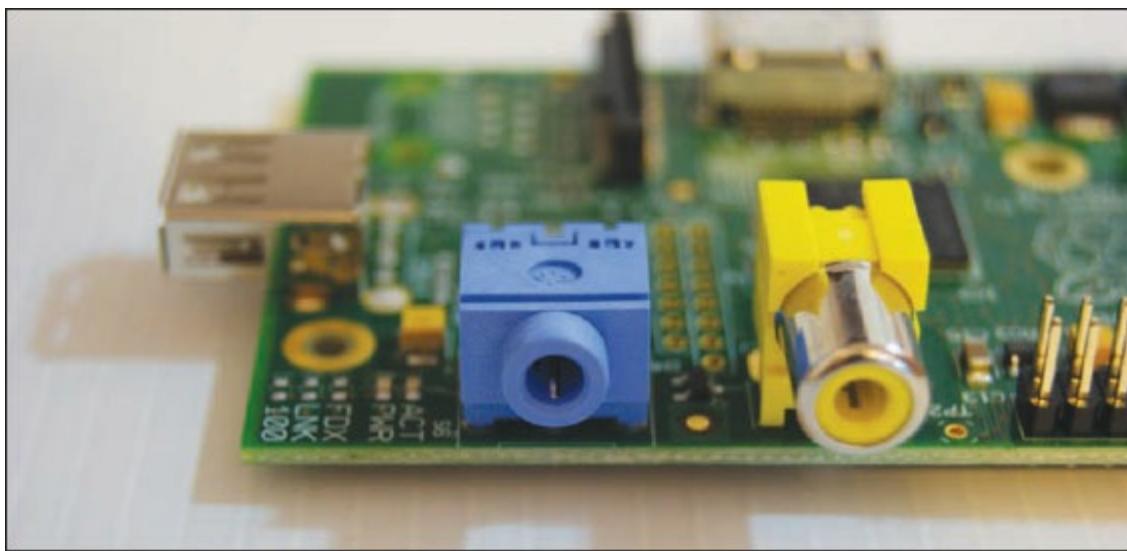


Figura 1.13 - Il connettore Audio out.

GPIO

Il connettore GPIO rappresenta la differenza principale tra il Raspberry Pi e un "normale" PC, tablet o smartphone. Se i connettori descritti nei paragrafi precedenti sono abbastanza comuni, il connettore GPIO è una caratteristica peculiare di Raspberry Pi. Questo connettore, composto da 26 pin a passo 2,45 mm (un decimo di pollice, il passo standard dei componenti elettronici agli albori dell'informatica), consente di collegare

al nostro piccolo computer tantissime "diavolerie" che potranno trasformarlo, di volta in volta, in un media center, in una stazione meteo e in tantissimi altri dispositivi più o meno specializzati. La possibilità di controllare direttamente i pin di input e di output non è comune nel mondo dei personal computer, mentre costituisce la peculiarità fondamentale dei sistemi a microcontrollore, come Arduino. Con Raspberry Pi potremo sperimentare con l'elettronica, estendere e trasformare il nostro sistema ben oltre quanto immaginato da chi lo ha progettato. Servirà solo un po' di pazienza, perché questi saranno gli argomenti trattati nella seconda metà di questo libro.

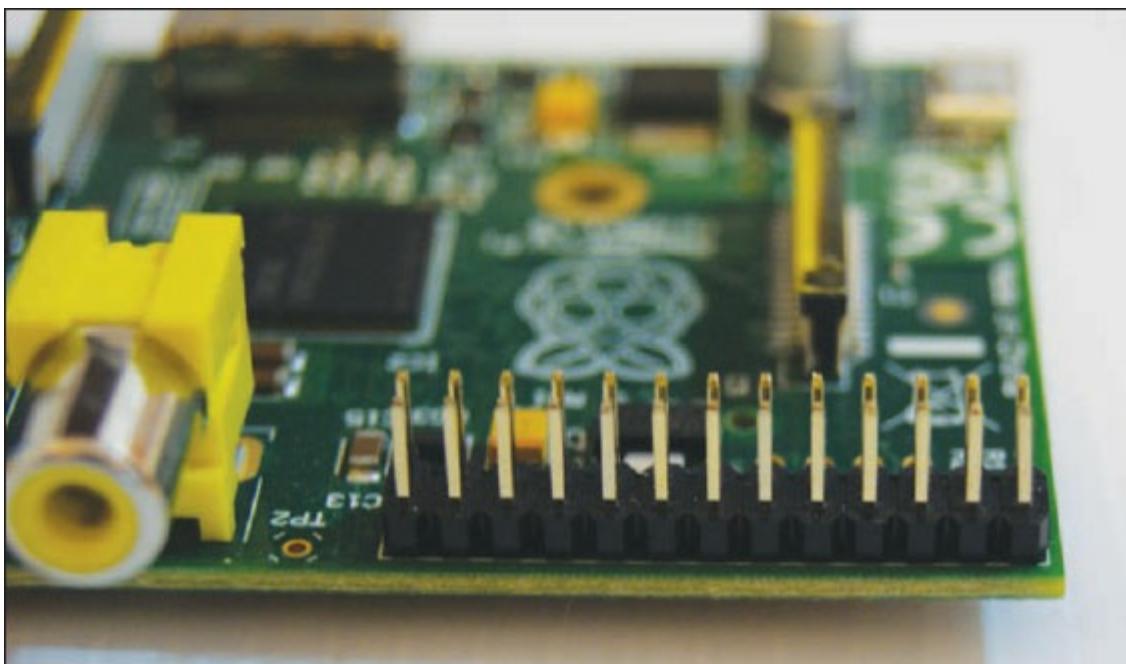


Figura 1.14 - Il connettore GPIO.

Il software

Lo scopo della Raspberry Foundation è fornire un sistema utile per la didattica e per questo motivo è stato scelto come sistema operativo Linux.

Il progetto Linux è stato iniziato nel 1991 da uno studente finlandese, Linus Torvalds, che voleva portare sui comuni PC un sistema operativo simile ai sistemi Unix utilizzati sui mainframe e sui sistemi di rete diffusi in ambito universitario.

Ovviamente, non iniziò il progetto sperando di guadagnare soldi a palate (il boom di internet, delle dotcom e delle startup non era nemmeno immaginabile all'epoca), ma per imparare e sperimentare.

Un'altra sua idea, sempre per imparare e sperimentare il più possibile

condividendo con altri le sue scoperte, era quella di realizzare il suo progetto seguendo un modello di sviluppo collaborativo, in cui chiunque potesse contribuire al sistema e in cui chi apportava modifiche avrebbe poi dovuto ridistribuirle in formato sorgente. Questo modello di sviluppo è stato definito “open source” e, in diverse accezioni e con diverse modalità di licenza, è oggi alla base di moltissimi progetti software. Si contrappone al modello del software proprietario in cui il codice sorgente è considerato un segreto da custodire in modo assoluto. Ma in questo libro non entreremo troppo nei dibattiti filosofici e ci limiteremo a sfruttare gli strumenti open source che abbiamo a disposizione sapendo che, se la cosa stuzzicherà il nostro interesse, potremo modificarli andando a prenderci il codice sorgente.

Il progetto Linux è cresciuto e si è sviluppato, probabilmente oltre ogni più rosea immaginazione, e oggi Linus Torvalds lavora per la Linux Foundation e si occupa di gestire lo sviluppo di questo sistema, coordinando gli sforzi di migliaia di sviluppatori in tutto il mondo.

Spesso questi sviluppatori sono stipendiati da grandi aziende come IBM, Google e tutti i maggiori produttori di silicio, da Intel a ARM, da Texas Instruments a Samsung. In realtà quello che si può scaricare dal sito www.linux.org è solamente il “kernel”, il nucleo del sistema, che fornisce i servizi base per poi eseguire delle applicazioni. Senza un insieme minimo di applicazioni il kernel non serve a molto.

Fortunatamente, mentre Linus si occupava del kernel, tantissimi altri sviluppatori hanno portato su Linux le applicazioni tipiche del mondo Unix, costruendo così un sistema operativo completo e totalmente open source. In tempi più recenti sono state sviluppate su Linux applicazioni che forse non erano neppure immaginabili ai tempi dei sistemi Unix: browser, programmi di fotoritocco, messaggistica testuale e audio/video, media player ecc.

Già dagli albori dello sviluppo di Linux, per semplificare l'utilizzo, si è sviluppato il concetto di “distribuzione”. La distribuzione aggrega al kernel tutta una serie di applicazioni che consentono all'utente finale di utilizzare il sistema senza dover recuperare e ricompilare i sorgenti di ogni singola applicazione necessaria.

Esistono distribuzioni per sistemi desktop (le più diffuse sono Ubuntu, Mint, Fedora, Debian), che forniscono un'interfaccia grafica e le funzionalità che siamo abituati a trovare su un PC Windows o su un Mac. Sono disponibili anche moltissime distribuzioni specializzate che

consentono di realizzare sistemi server, firewall, terminali o dispositivi dedicati.

Il kernel e la maggior parte delle applicazioni più diffuse su Linux sono open source e questo consente agli sviluppatori e a chi vuole imparare a programmare di scoprire come vengono implementate le diverse funzionalità del sistema e, se necessario, correggere errori o aggiungere nuove funzioni.

Raspberry Pi utilizza un processore diverso da quello dei normali PC e anche le sue risorse hardware sono più scarse. Per consentire agli utenti di sfruttare al meglio il loro piccolo computer è stata realizzata una distribuzione ad-hoc per Raspberry Pi chiamata Raspbian. Il nome è composto da RASPberry e debIAN. Debian è una delle distribuzioni storiche di Linux ed è la stessa su cui si basa anche Ubuntu, quella che al momento è la “versione” di Linux più diffusa nel mondo PC.

Raspbian è la distribuzione mantenuta e consigliata dalla stessa Raspberry Foundation.

Installazione

Oltre a Raspbian sono state sviluppate diverse altre distribuzioni per Raspberry Pi, alcune specializzate e dedicate alla realizzazione di un media player (argomento trattato in uno dei prossimi capitoli), altre più generiche.

Per non fare torto a nessuno (o perlomeno a nessuna delle distribuzioni più diffuse), da giugno del 2013 dal sito di Raspberry Pi è possibile scaricare NOOBS (New Out Of Box Software).

NOOBS è una raccolta di distribuzioni, tra cui Raspbian, che può essere utilizzata per configurare e installare Raspberry Pi al primo boot. Se non avete ricevuto una SD card con NOOBS insieme al vostro Raspberry Pi, questo è il momento di scaricare il software e preparare una SD per il vostro nuovo computer.

Aprite il vostro browser preferito sul sito www.raspberrypi.org, selezionate **Downloads** dalla barra in alto e salvate il file compresso che contiene tutto il necessario per eseguire NOOBS.

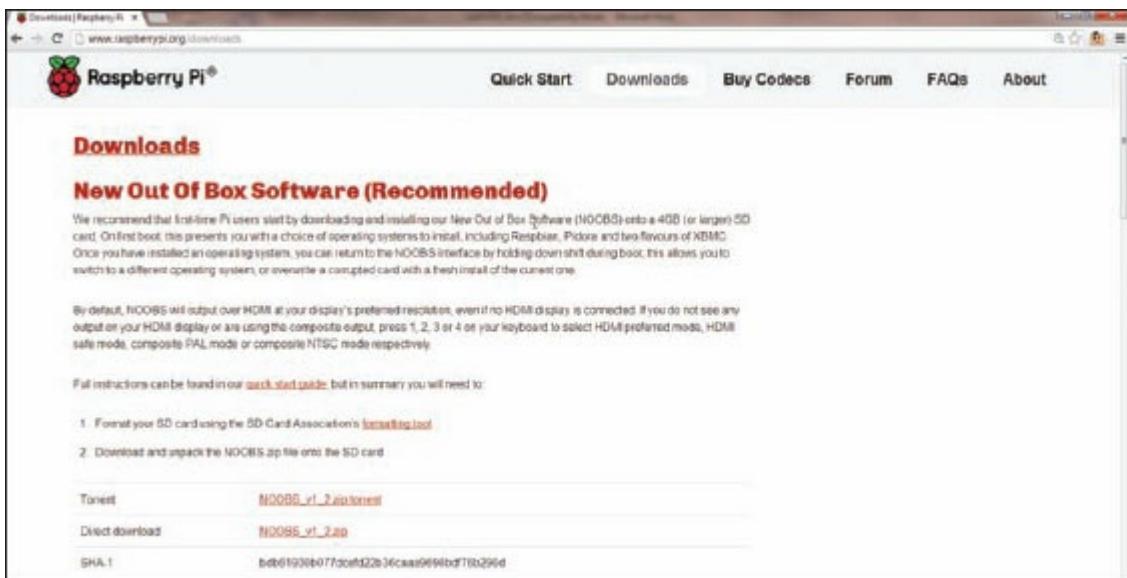


Figura 1.15 - La pagina per il download nel sito della Raspberry Foundation.

Tramite NOOBS installeremo un sistema Raspbian che utilizzeremo nei prossimi capitoli. Mentre completate il download di NOOBS potete iniziare a preparare la SD card da utilizzare con Raspberry Pi. Raspberry Pi utilizza schede SD fino a 32 GB di capacità; per usare Raspbian è consigliata una SD da almeno 4 GB.

Non è possibile utilizzare direttamente microSD card con Raspberry Pi; sono però disponibili adattatori che consentono di utilizzare queste card anche in dispositivi che accettano solo il formato più grande.

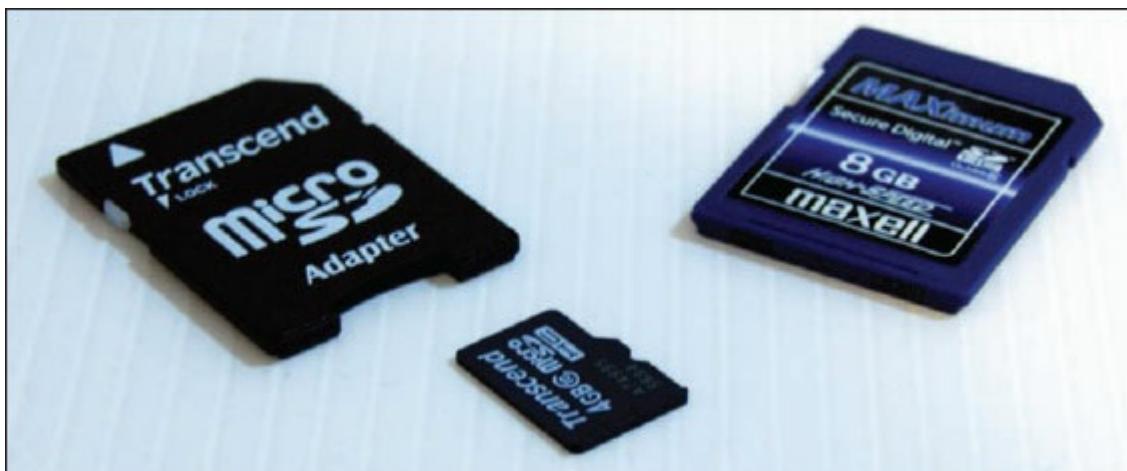


Figura 1.16 - SD Card (a destra) e microSD card con adattatore (a sinistra).

Le performance delle schede SD possono variare di molto e, da qualche anno, le schede sono etichettate con un identificativo di "classe" che corrisponde alla velocità di trasferimento dati in Megabyte al secondo. Più è alto il numero della classe, maggiore è la velocità di trasferimento dati garantita dalla SD card. Se il vostro Raspberry Pi vi

sembra lento, soprattutto mentre carica un'applicazione o accede ai file, sostituire la SD card con una di prestazioni più elevate potrebbe dare nuova vita al vostro sistema.



Figura 1.17 - Indicazione della classe (10) e della capacità (4 GB) su una microSD card.

Per utilizzare la SD card con NOOBS questa va formattata usando il tool fornito dalla SD Association (https://www.sdcard.org/downloads/formatter_4/), fornito in versione Windows e OS X. Una volta installato e lanciato, il tool consentirà di selezionare la SD card da formattare.

Selezionate la SD card dopo aver trasferito tutti i dati che non volete perdere. L'operazione di formattazione, infatti, cancellerà totalmente il contenuto della SD card.



Figura 1.18 - SDFormatter - schermata principale.

Prima di avviare la formattazione è necessario selezionare il tasto **Option** e impostare a **ON** l'opzione **FORMAT SIZE ADJUSTMENT**.

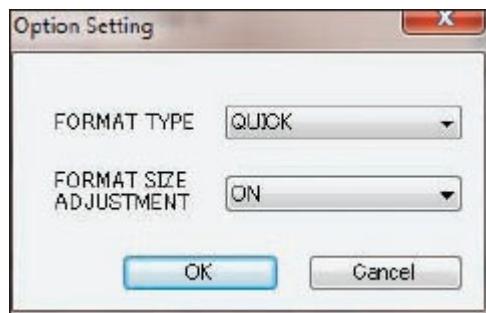


Figura 1.19 - SD Formatter - schermata delle opzioni.

Una volta premuto il tasto **Format** il software vi lascerà un’ultima opportunità di verificare che non ci siano dati importanti sulla SD card e procederà con la formattazione. Completata la formattazione potrete aprire il file zip di NOOBS scaricato dal sito della Raspberry Pi Foundation e copiarne i contenuti nella cartella principale della SD card. Collegate un cavo HDMI o RCA, una tastiera e un mouse (opzionale) al vostro Raspberry Pi, inserite la SD card appena preparata nello slot, alimentatelo tramite il connettore Micro-USB (basta il carica batterie di un telefonino) e sarete pronti a installare il vostro sistema operativo!

Accendete il vostro monitor (se avete collegato il cavo HDMI è importante accendere il monitor prima di Raspberry Pi) e, una volta alimentato il nostro Raspberry Pi, sullo schermo apparirà la schermata principale di NOOBS. Se avete collegato un cavo RCA a un TV color lo schermo resterà nero anche dopo aver impostato la corretta sorgente video sul televisore.

Dovrete premere il tasto **3** (PAL) o **4** (NTSC) per configurare la modalità video analogica (per i TV color europei è necessario scegliere PAL).

Se avete collegato un cavo HDMI l’immagine dovrebbe essere immediatamente visibile. Se non lo è premete il tasto **2** che imposta una modalità video HDMI differente e compatibile con gli adattatori HDMI/VGA.

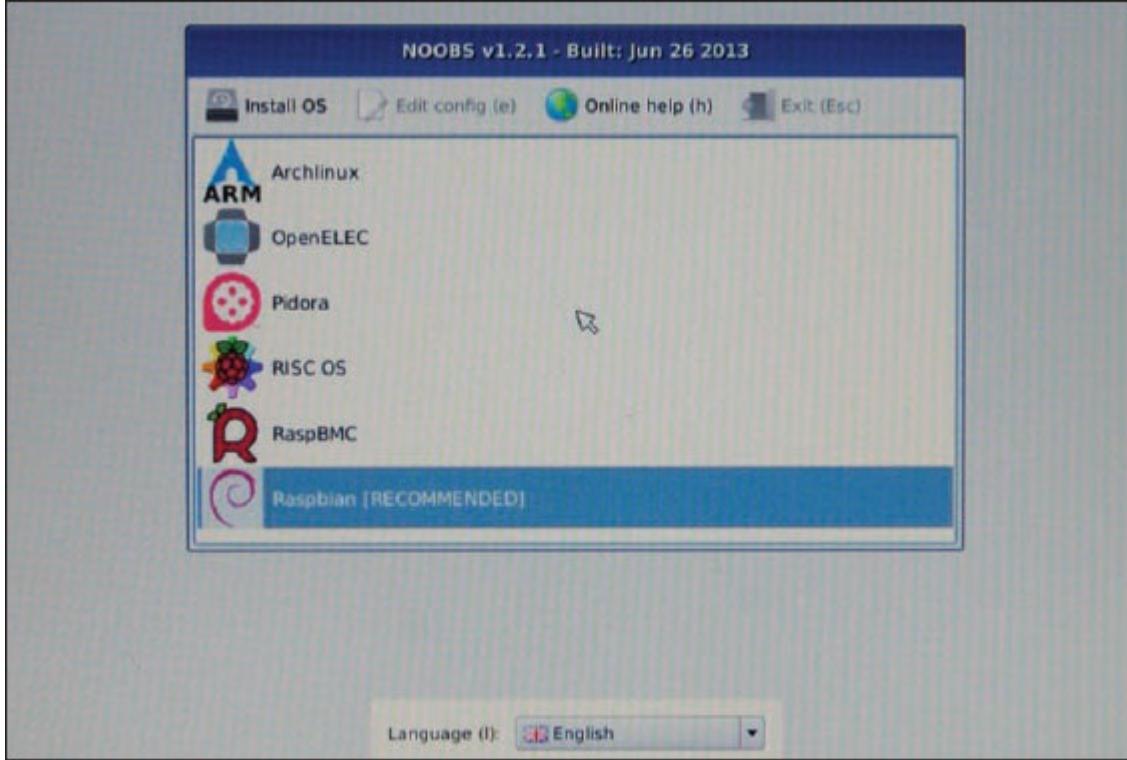


Figura 1.20 - NOOBS - schermata principale.

A questo punto è possibile selezionare la distribuzione desiderata (nel nostro caso Raspbian) e avviare l'installazione. NOOBS consente anche di selezionare la lingua per l'installazione ma, al momento, non è disponibile una versione in italiano. Prima di installare, il sistema ci aviserà del rischio di perdere tutti i dati attualmente sulla SD card; visto che si tratta di una SD appena formattata (a meno di uno scambio fortuito!) non ci saranno problemi a confermare e iniziare la copia dei file.

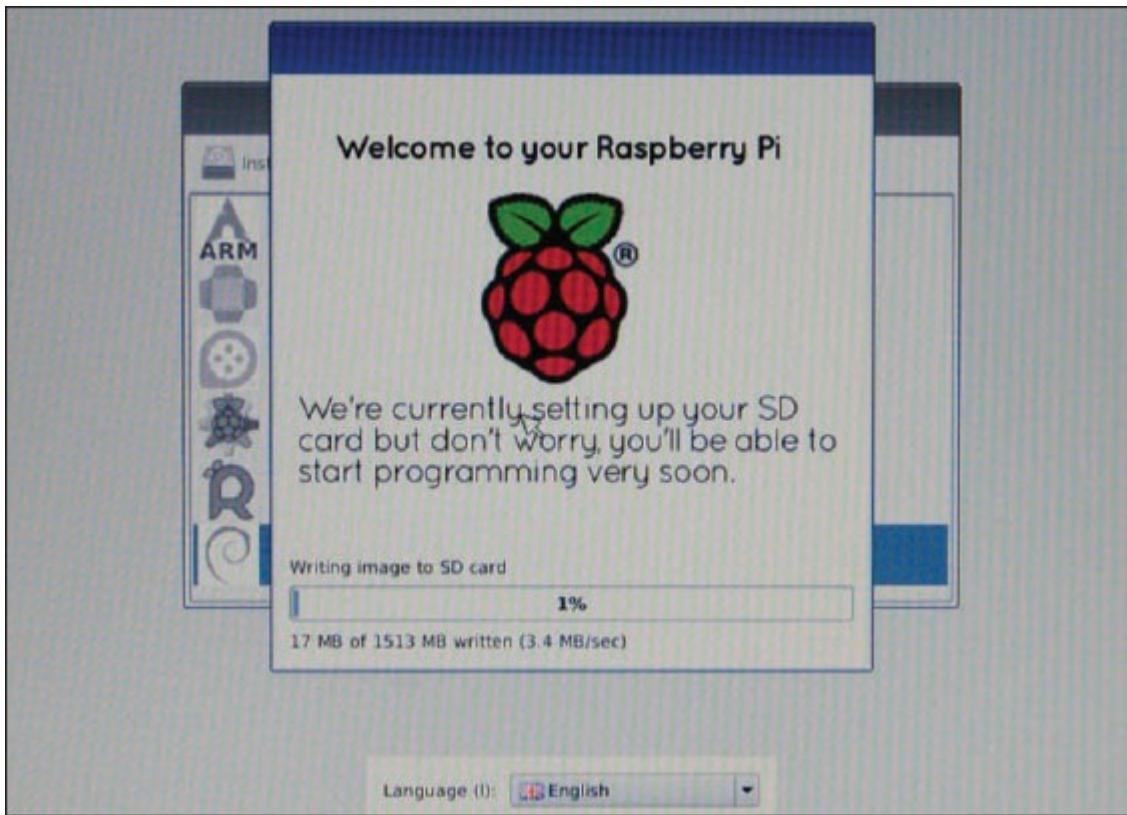


Figura 1.21 - NOOBS - copia dei file.

Una volta terminata la copia verrà mostrato un messaggio che ci aviserà dell'avvenuta installazione. Basterà confermare il messaggio e il sistema ripartirà automaticamente.

Dopo il riavvio verrà visualizzato un menu testuale che ci consentirà di effettuare alcune operazioni preliminari di configurazione del nostro sistema. È possibile navigare tra le opzioni usando i **tasti cursore**, per poi selezionarle premendo la **barra spaziatrice**. Per selezionare i pulsanti presenti nella parte bassa delle diverse schermate si può utilizzare il tasto **Tab**, premendolo ripetutamente fino a quando non viene selezionato il pulsante desiderato. Una volta fatta la giusta selezione i tasti **Spazio** o **Invio** ci consentiranno di “premere” il pulsante.

Selezionando la prima opzione (**Expand Filesystem**) potremo assicurarci che Raspbian vada a utilizzare l'intera capacità della SD card che stiamo utilizzando. Questa opzione non è necessaria se si è installato il sistema utilizzando NOOBS.

L'opzione numero due ci consentirà di cambiare la password dell'utente di sistema. Per default l'utente si chiama “pi” e la sua password è “raspberry”. Se non volete che altri lettori di questo libro possano prendere possesso del vostro amato piccolo PC, è il caso di trovare una nuova password!

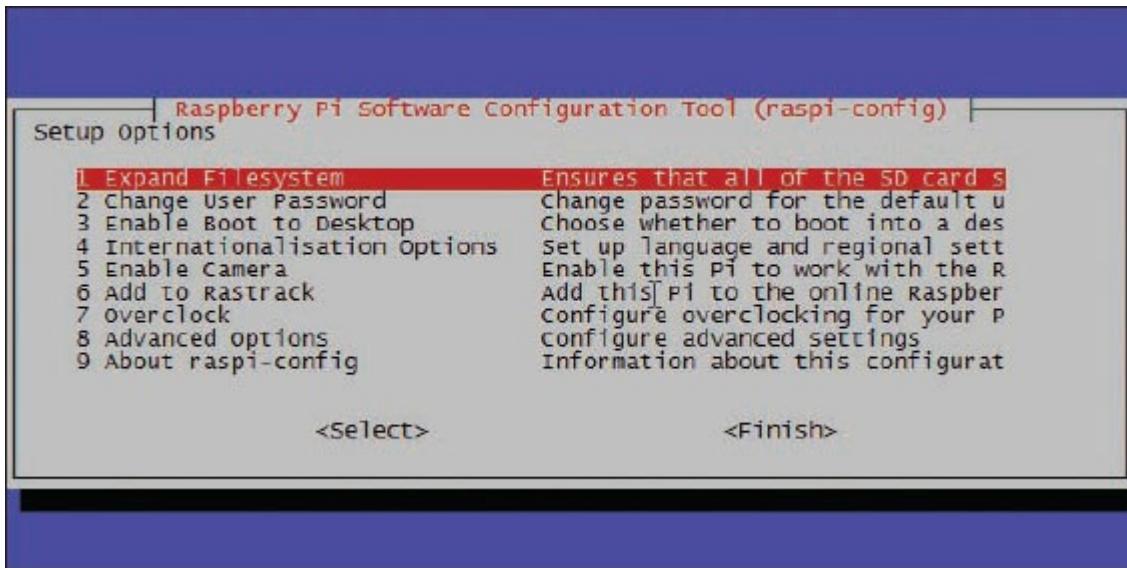


Figura 1.22 - Raspbian - configurazione del sistema.

La terza opzione consente di impostare l'avvio del sistema in modo che venga attivata l'interfaccia grafica. Questa è la modalità che utilizzeremo nel prossimo capitolo e quindi è una buona idea abilitare immediatamente questa opzione.

La quarta opzione ci consentirà di selezionare le impostazioni regionali. Questo ci permetterà di configurare Raspberry Pi per usare tastiera e impostazioni di formattazione di date e numeri italiane.

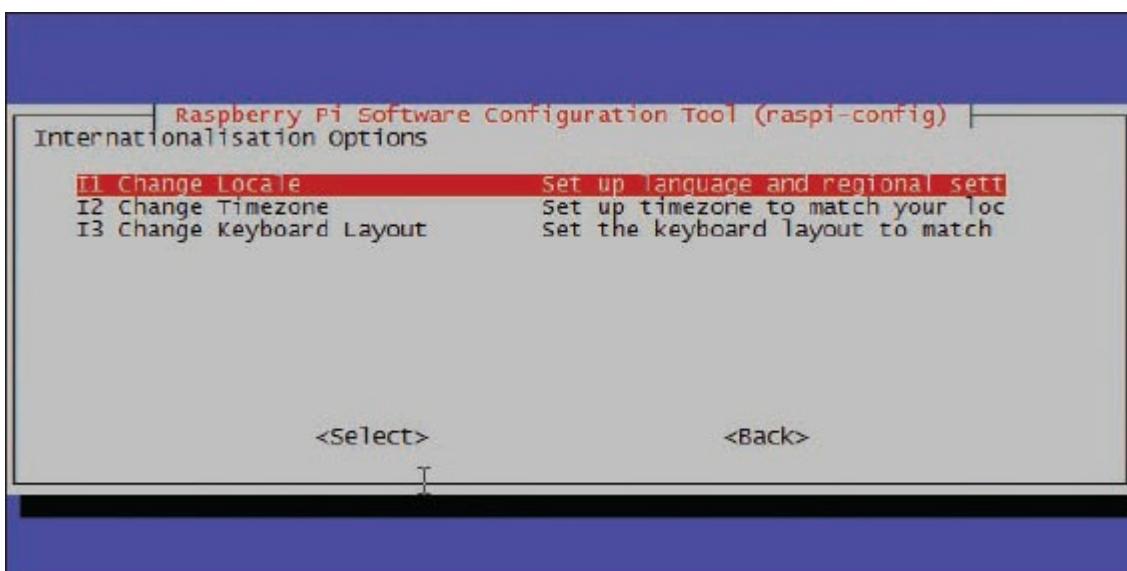


Figura 1.23 - Raspbian - il menu di localizzazione.

Saranno tradotti in italiano anche una parte dei messaggi del sistema operativo e delle applicazioni grafiche, anche se la maggior parte dei testi resterà in inglese.

La configurazione delle impostazioni internazionali prevede un menu con tre opzioni. Selezionando **Change Locale** sarà possibile impostare

i “locale” utilizzati dal sistema. Per locale si intende l’insieme di convenzioni relative alla formattazione di data e ora, numeri, valuta ecc.

Nella prima schermata è possibile selezionare **All Locale** oppure spuntare i singoli locale che ci interessano. Per supportare la lingua italiana è possibile selezionare **it_IT**. I nostri amici della Svizzera italiana potranno selezionare **it_CH**. È consigliabile selezionare anche i locale inglese/USA (prefisso **en_US**), visto che buona parte delle applicazioni in Raspbian non ha una traduzione nella nostra lingua.

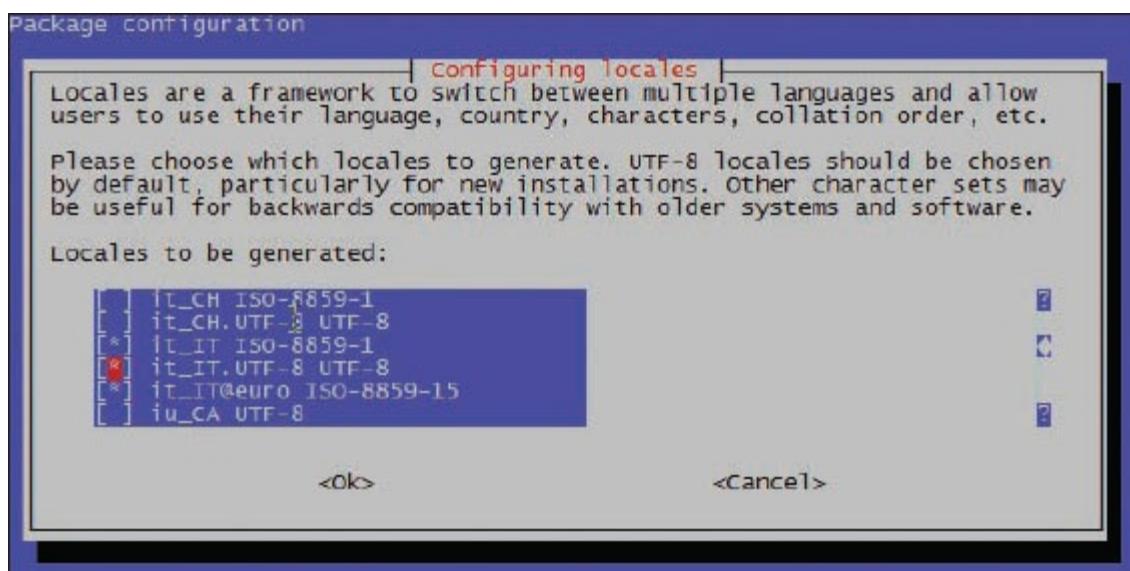


Figura 1.24 - Raspbian - selezione dei locale supportati.

Nel secondo passaggio di configurazione sarà possibile selezionare, tra i locale abilitati, quello di default. **it_IT-UTF8** dovrebbe garantire la massima compatibilità con le diverse applicazioni e i tool utilizzati nel resto dei capitoli di questo libro.

Il sistema passerà alla generazione dei locale selezionati. Se avete selezionato tutti i locale disponibili questo richiederà diversi minuti.

Tornati al menu delle opzioni di internazionalizzazione, è il momento di selezionare il fuso orario corretto. Il primo step prevede la selezione del continente, il secondo la selezione della “time zone”. La time zone definisce, oltre al fuso orario, anche le modalità di passaggio dall’ora legale all’ora solare e viceversa ed è normalmente identificata con una città. Roma è la città da selezionare per impostare la time-zone italiana.

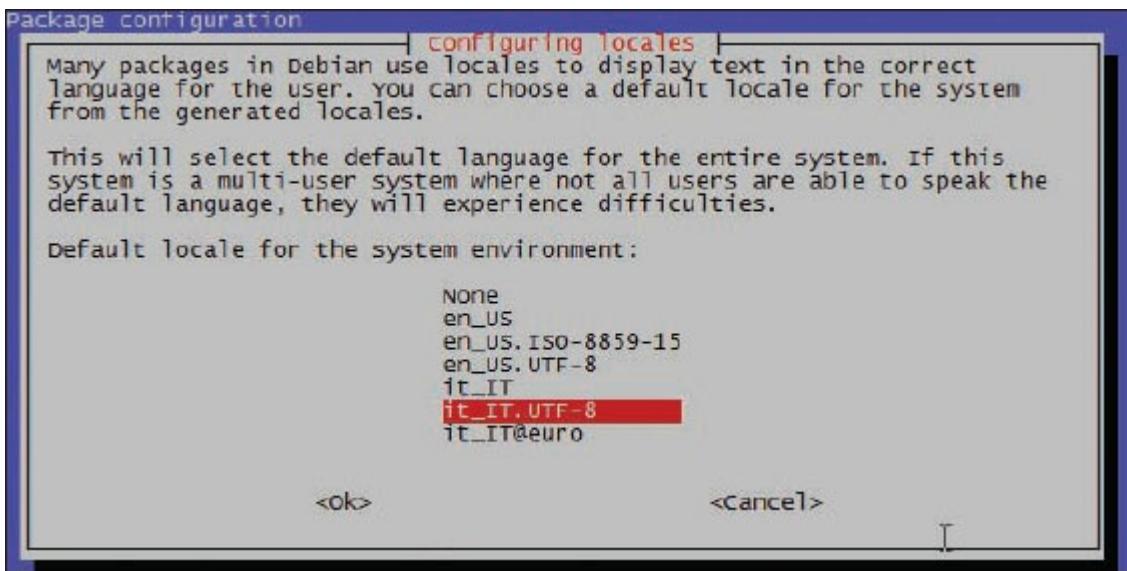


Figura 1.25 - Raspbian - impostazione del locale di default.

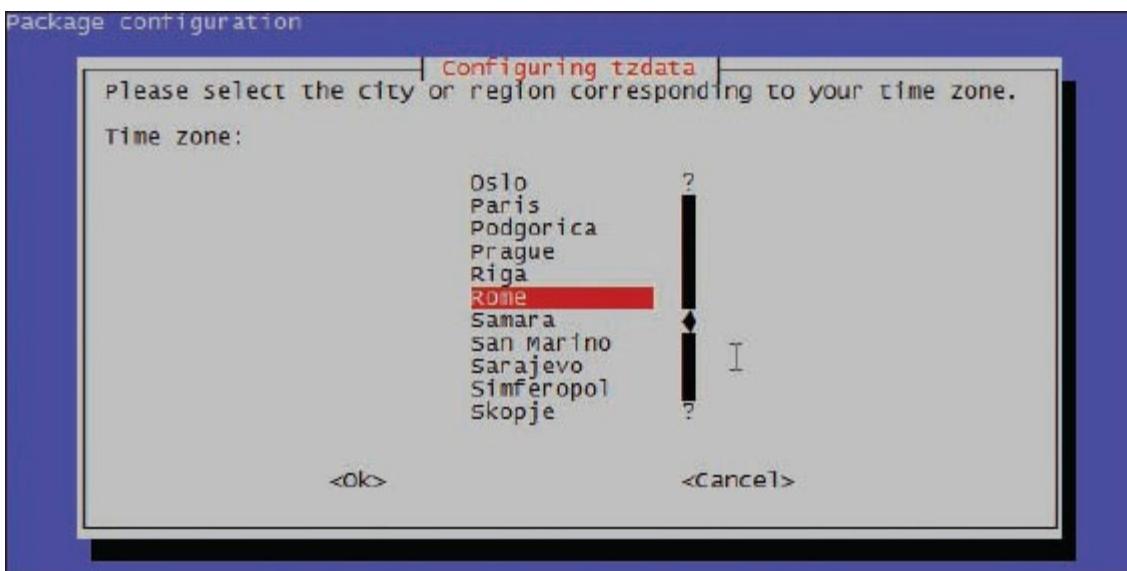


Figura 1.26 - Raspbian - selezione della time-zone.

Tornati di nuovo al menu di internazionalizzazione rimane da selezionare il layout di tastiera. Ogni nazione e lingua ha impostazioni diverse e tastiere che si distinguono, oltre che per la posizione dei caratteri alfabetici, anche per la presenza o meno di lettere accentate, simboli particolari ecc.

Il primo passaggio di configurazione ci chiederà di selezionare il tipo e il numero di tasti della nostra tastiera. Se non vi siete mai chiesti quanti tasti ha la vostra tastiera... è il momento di contare!

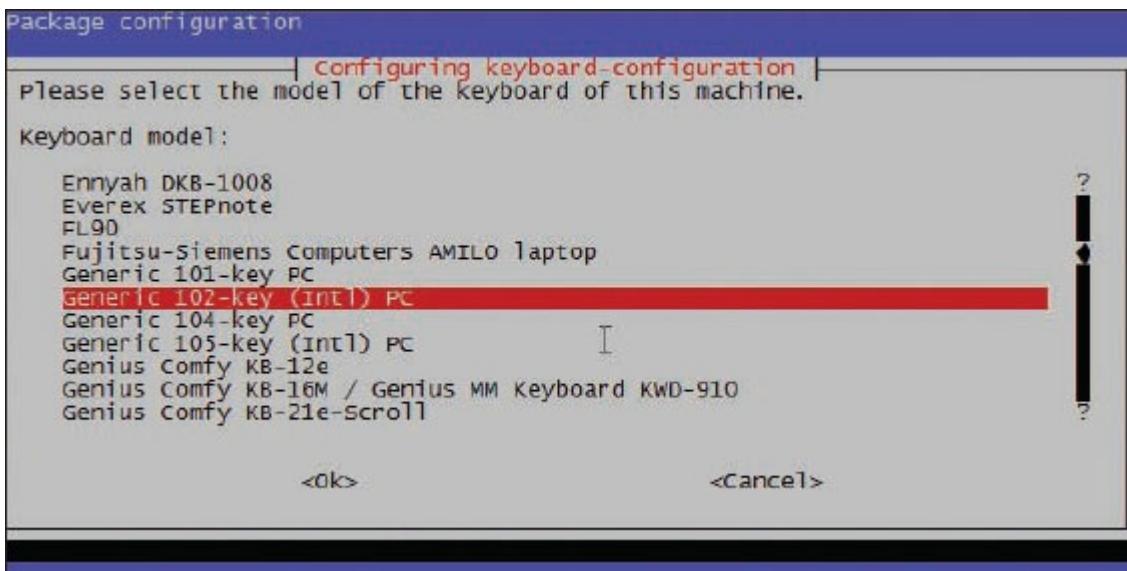


Figura 1.27 - Raspbian - selezione del tipo di tastiera

A questo punto apparirà la selezione del layout di tastiera vero e proprio. In realtà la prima lista presenta solo i layout per la lingua inglese e sarà quindi necessario selezionare **Other** per passare alla lista completa.

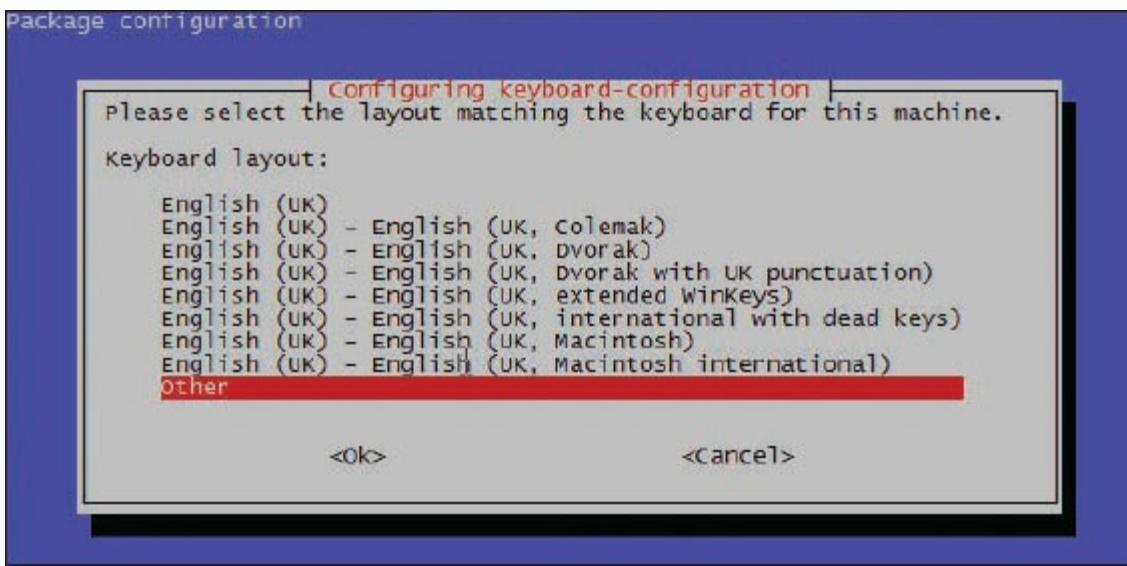


Figura 1.28 - Raspbian - lista dei layout per la lingua inglese.

La lista completa dei layout è molto lunga ma, con un po' di pazienza, potrete individuare il layout Italiano e, in un secondo passaggio, scegliere tra i diversi layout Italiani quello appropriato per la tastiera che state utilizzando. Il layout **Italian** dovrebbe funzionare per la maggior parte delle tastiere da PC. È comunque possibile rieseguire la configurazione senza dover reinstallare di nuovo il sistema. Scopriremo come farlo nel prossimo capitolo.

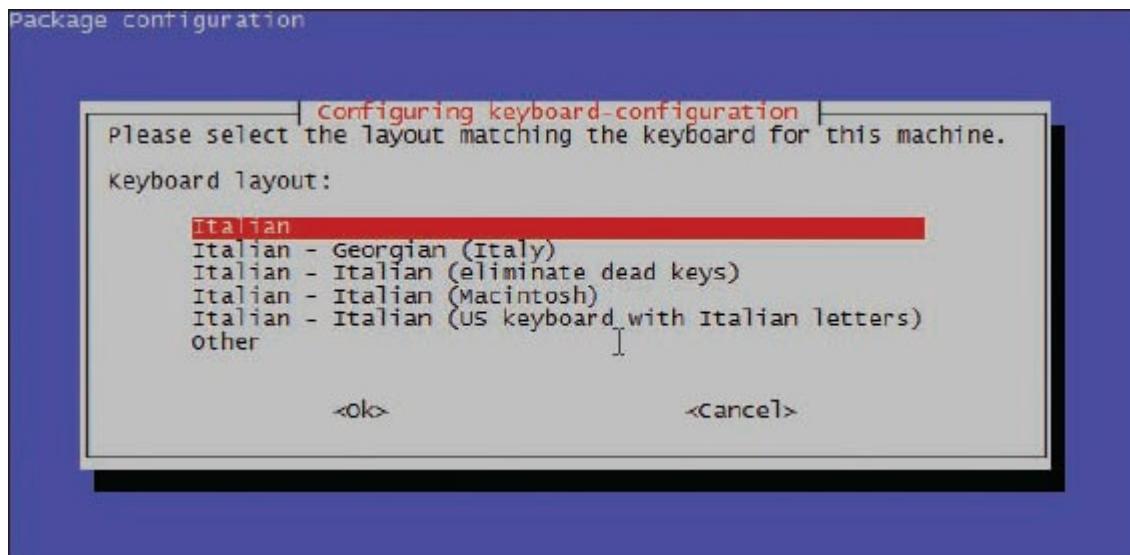


Figura 1.29 - Raspbian - elenco dei layout di tastiera per la lingua italiana.

Se siete collegati a Internet e volete registrare il vostro nuovo Raspberry Pi sul sito rastrack.co.uk potete selezionare l'opzione 6. La registrazione non è obbligatoria e non ha alcun crisma di ufficialità. Il sito è gestito da Ryan Walmsey, un appassionato inglese, e serve solo a farvi scoprire se vicino a voi c'è qualche altra persona "contagiata" dalla passione per i piccoli PC. Se non avete un collegamento internet, non siete collegati alla rete o avete un Raspberry Pi Model A che non ha una porta Ethernet, potrete eseguire la registrazione in un secondo tempo o farla direttamente sul sito www.rastrack.co.uk utilizzando il vostro PC.

Se visitate il sito, provate a trovare l'autore di questo libro tra le centinaia di Raspberry Pi registrati in Italia!

L'opzione numero 5 consente di abilitare il modulo camera. Questo modulo è opzionale e può essere collegato a Raspberry Pi per consentirgli di catturare immagini a una velocità molto più alta di quella consentita da una normale webcam USB. Se non possedete o non avete ancora collegato il modulo telecamera, potrete abilitarlo anche in seguito. Magari in tempo per "giocarci" un po' seguendo gli esempi del capitolo dedicato ai sensori.

L'opzione numero 7 permette di cambiare la frequenza di clock del processore. Sono disponibili frequenze da 700 MHz (il default) fino a 1 GHz. Alzare la frequenza porta a un miglioramento delle performance, ma anche a un maggiore consumo di energia e, di conseguenza, alla necessità di dissipare più calore. Se il calore non viene dissipato in modo efficace il processore potrebbe subire danni permanenti a seguito dell'overclock. In questo libro vi saranno offerte diverse altre occasioni per invalidare la garanzia del vostro Raspberry Pi e vi consiglio quindi di

lasciare per ora la frequenza al valore di default.

L'ultimo passaggio, prima di potersi godere finalmente il nostro piccolo PC, passa per il menu numero 8: **Advanced Options**. Nel sotto-menu la seconda opzione vi consentirà di assegnare un nome al vostro Raspberry Pi e potrebbe essere utile farlo se pensate di utilizzarlo all'interno di una rete domestica.

La quarta opzione, invece, consente di abilitare l'accesso da remoto tramite il protocollo SSH (Secure SHell) e ci permetterà di operare sul nostro piccolo computer senza bisogno di collegare fisicamente mouse e tastiera. Se il vostro Raspberry Pi è collegato in rete, abilitate questa opzione.

A questo punto siamo pronti per utilizzare il nostro Raspberry Pi. Selezionando il tasto **Finish** il sistema verrà riavviato e, dopo qualche decina di secondi di attesa, apparirà il desktop con al centro un gigantesco lampone. È arrivato il momento di iniziare a sperimentare davvero con Raspberry Pi.

Ma per farlo dovete passare al prossimo capitolo!

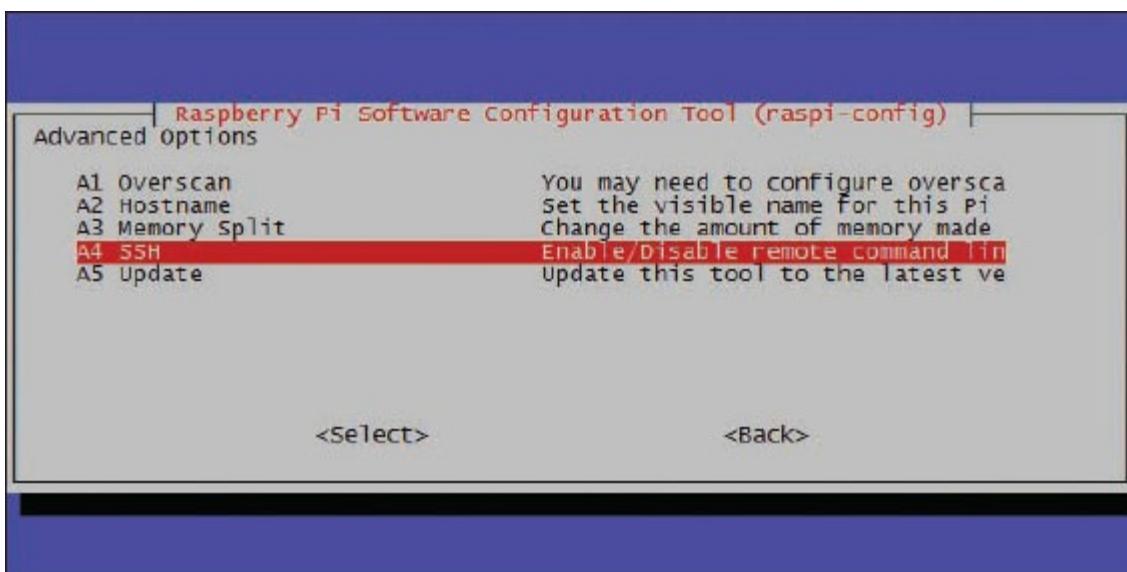


Figura 1.30 - Raspbian - opzioni avanzate.

Il mio piccolo personal computer

Prendiamo confidenza con Raspbian e la sua interfaccia grafica. Poi proviamo a rendere il nostro piccolo PC ancora più “personale”.

Il desktop di Raspbian, con il suo gigantesco lampone, non è troppo diverso dal desktop dei PC Windows (almeno prima di Windows 8) e dei Mac. Il sistema operativo è completamente diverso (Linux per Raspberry Pi, Windows per i PC, OS X per i Mac), ma molti dei concetti della “metafora della scrivania” sono comuni ai diversi ambienti. In questo capitolo vedremo quali sono i principali strumenti che ci mette a disposizione l’interfaccia grafica di Raspbian e scopriremo come è possibile installare nuove applicazioni o configurare quelle esistenti a seconda delle nostre esigenze.

Sui sistemi Unix l’interfaccia grafica non è parte del kernel di sistema, ma è demandata a un’applicazione chiamata X server. Questa applicazione riceve dalle altre applicazioni grafiche richieste per disegnare sullo schermo attraverso un protocollo chiamato X11. Le funzioni di X11 forniscono solo le primitive base per il disegno a video e per ricevere input dall’utente, non gestiscono finestre o tantomeno funzioni avanzate come menu o desktop.

Il desktop e le finestre sui sistemi Linux sono generalmente gestiti da un “window manager”. Esistono diversi window manager che si

caratterizzano per effetti grafici più o meno complessi, funzionalità avanzate e un utilizzo di risorse più o meno “dispendioso”. Nel caso di Raspbian il window manager utilizzato è Lightweight X11 Desktop Environment (LXDE). Come avranno intuito i lettori che masticano un po’ di inglese, si tratta di un desktop manager leggero (lightweight) e che quindi si adatta bene alle risorse limitate di Raspberry Pi, senza però farci perdere la maggior parte delle comodità a cui siamo abituati.

Gli elementi del desktop

La parte sinistra del desktop è occupata dalle applicazioni di uso più frequente. Sarà possibile aggiungere altre icone o rimuovere quelle che non ci servono, per personalizzare al massimo il nostro sistema e facilitarci l’accesso alle funzioni che utilizziamo più spesso. In questo capitolo impareremo come cambiare l’aspetto dell’interfaccia grafica di Raspbian. Vi consiglio però di trattenere la vostra creatività fino alla fine di questa breve introduzione, in modo che gli screenshot che troverete nelle prossime pagine corrispondano effettivamente a quanto vedete sul vostro monitor.

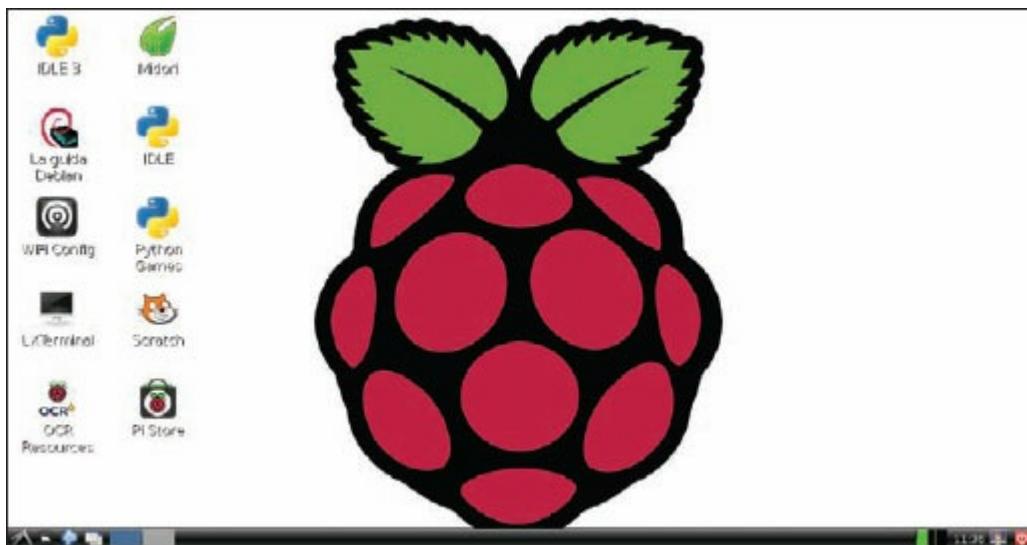


Figura 2.1 - Il desktop di LXDE.

Nella parte bassa dello schermo è presente una barra che consente di lanciare applicazioni non presenti sul desktop e presenta una serie di informazioni utili.



Figura 2.2 - La barra delle applicazioni.

Scorrendo la barra da sinistra a destra troveremo:

- il menu che ci consentirà di lanciare le diverse applicazioni installate;
- Il file manager che ci consentirà di navigare all'interno del file system;
- il web browser (Midori);
- un pulsante che consente di ridurre a icona (con un clic sinistro) o di "arrotolare" (con un clic del tasto centrale se avete un mouse a tre tasti o un clic sulla rotella di scorrimento) riducendole alla sola barra del titolo, tutte le finestre aperte;
- le miniature dei diversi desktop. Per default Raspbian crea due desktop e, con un clic sulle miniature nella barra delle applicazioni, è possibile passare da un desktop all'altro. Avere più desktop può essere molto utile se siete abituati a lavorare con più applicazioni attive contemporaneamente. In questo modo potrete, ad esempio, lasciare aperta le applicazioni di posta elettronica e messaging in un desktop e usare l'altro per lavorare. Un clic vi consentirà di vedere rapidamente se sono arrivate e-mail o se ci sono altre novità senza bisogno di spostare, iconizzare o chiudere le finestre delle applicazioni che state utilizzando;
- un'area in cui vengono mostrate le applicazioni aperte sul desktop attivo;
- un grafico che mostra l'occupazione della CPU. Le barre verdi che scorrono verso sinistra rappresentano l'occupazione di CPU negli ultimi secondi. Se il vostro Raspberry Pi non risponde ai comandi e la barra verde riempie tutta l'area del grafico, dategli un attimo di respiro per smaltire le operazioni in corso e tornerà veloce e reattivo come prima;
- l'orario corrente;
- un pulsante che consente di attivare lo screensaver. Prima di poterlo utilizzare sarà necessario installare uno screensaver, dovrete pazientare per qualche pagina prima di scoprire come farlo;
- un pulsante che consente di spegnere o riavviare il sistema. Una volta premuto questo pulsante verrà mostrato un menu che ci consentirà di spegnere o riavviare il nostro Raspberry Pi, oppure di terminare la sessione corrente, tornando alla schermata di login. La schermata di login non viene normalmente mostrata al boot.



Figura 2.3 - Il menu di logout.

Configurare la rete Wi-Fi

Se avete un adapter USB/Wi-Fi potrete sfruttarlo per collegare il vostro Raspberry Pi alla vostra rete domestica e, attraverso questa, a Internet. Non tutti gli adattatori sono compatibili, ma potete trovare una lunga lista di adattatori testati con Raspberry Pi sul sito elinux (elinux.org/RPi_USB_Wi-Fi_Adapters). Se non siete sicuri in merito alla compatibilità del vostro adattatore USB Wi-Fi, la cosa più semplice da fare è connetterlo a Raspberry Pi e provare! Alcuni adattatori richiedono una quantità abbastanza alta di corrente e potrebbero far riavviare Raspberry Pi quando li collegate direttamente alle sue porte USB. Potrebbe quindi essere necessario collegarli tramite un hub USB alimentato. Per configurare la rete Wi-Fi potete lanciare l'applicazione **Wifi Config** che trovate sul desktop. Una volta lanciata l'applicazione questa mostrerà una finestra di dialogo.

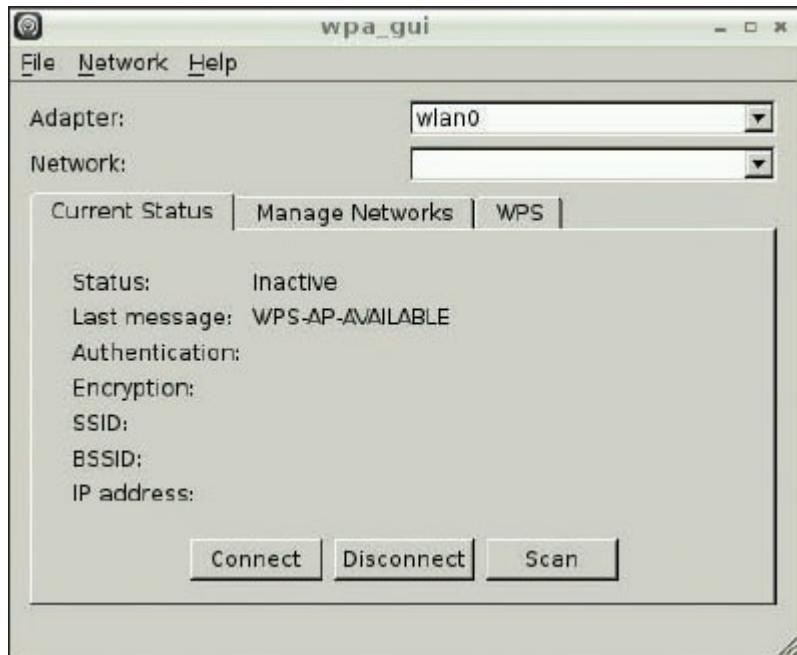


Figura 2.4 - Configurazione Wi-Fi - schermata principale.

Se nella prima casella di selezione compare l'adattatore Wi-Fi (sarà indicato con il nome wlan0, come nella schermata in [Figura 2.4](#)) il vostro adattatore Wi-Fi è stato riconosciuto correttamente dal sistema.

Se non dovesse comparire provate a scollarlo e ricollegarlo al vostro Raspberry Pi attraverso un hub alimentato. Se anche così non dovesse funzionare è possibile che non sia tra quelli supportati da Raspbian; una ricerca su internet potrà chiarire i vostri dubbi e magari aiutarvi a risolvere il problema o a trovare un adattatore compatibile. Potete premere il pulsante **Scan** e passare alla selezione della rete a cui collegarvi.

Scan results				
SSID	BSSID	frequency	signal	flag
TP-LINK_A90...	64:70:D2:a9:...	2452	-164 dBm	[WP]
MINUNET	00:26:f2:50:...	2412	-213 dBm	[WP]
Digicom_11n	00:a0:a2:44:...	2412	-207 dBm	[WP]

Figura 2.5 - Configurazione Wi-Fi - elenco delle reti rilevate.

Se la vostra rete non appare nella lista verificate che il router sia acceso e che il segnale raggiunga la posizione in cui avete installato Raspberry Pi con una qualità accettabile (potete provarlo con il vostro smartphone, se supporta il Wi-Fi, o con un PC). A questo punto è

necessario selezionare con un **doppio clic** la rete a cui collegarsi.

Il sistema dovrebbe riconoscere automaticamente la modalità di connessione e di sicurezza; in caso di dubbi consultate il manuale del vostro router e, se questo vi è stato fornito da un provider, le istruzioni per il collegamento di PC e altri dispositivi. Vedrete gli stessi parametri che trovate nella schermata di configurazione della rete di Raspberry Pi.

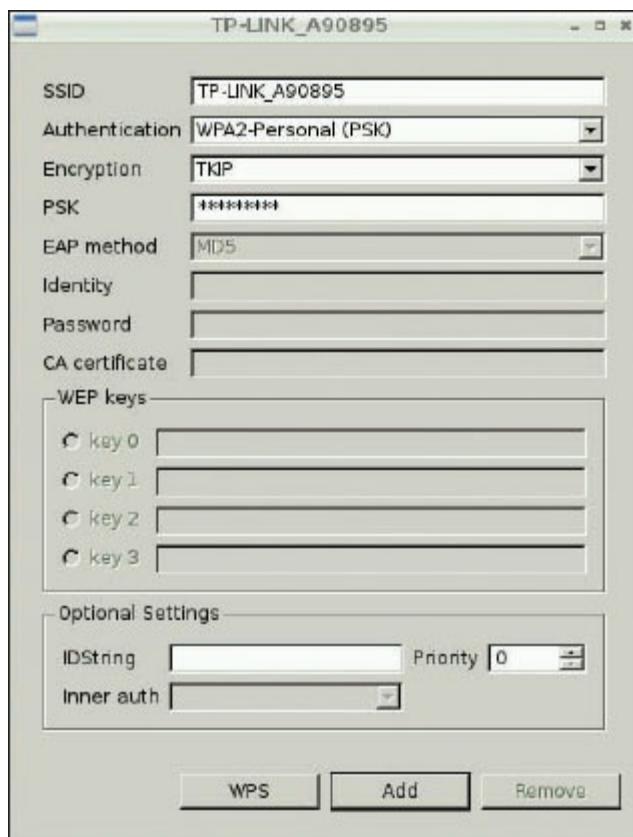


Figura 2.6 - Configurazione Wi-Fi - impostazione della rete.

L'unico parametro che, ovviamente, non può essere rilevato in automatico è la password di rete, da inserire nel campo **PSK**. Una volta inserite le informazioni necessarie potete salvare la configurazione premendo il pulsante **Add** e chiudere la lista delle reti Wi-Fi con il pulsante **Close**.

Premendo il pulsante **Connect** Raspberry Pi dovrebbe connettersi alla rete e mostrarvi l'indirizzo IP, come mostrato in [Figura 2.7](#).

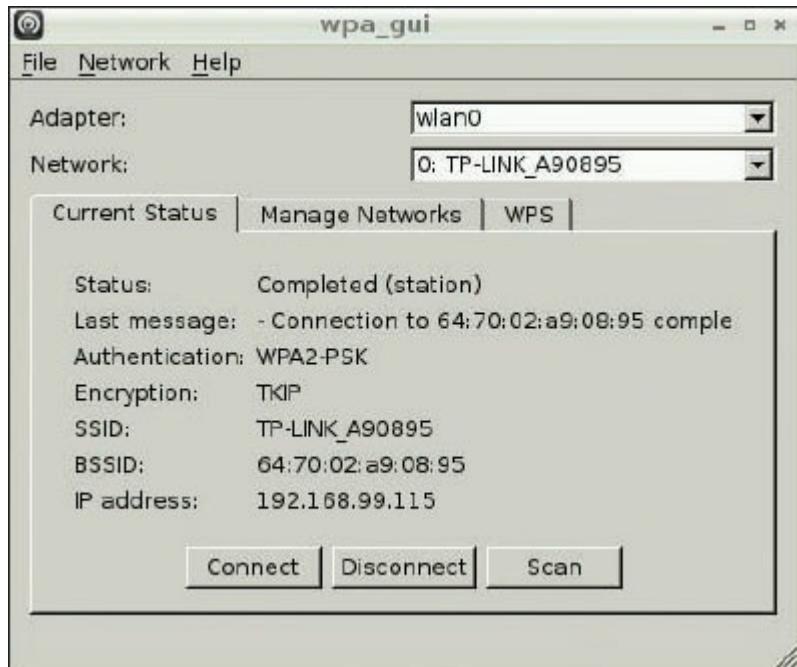


Figura 2.7 - Configurazione WiFi - connessione completata.

Se la connessione non viene attivata correttamente potrete verificare ed eventualmente modificare i parametri passando alla cartella **Manage Networks**, selezionando la vostra rete nella lista di quelle configurate e premendo il pulsante **Edit**.

Buona navigazione!

Le applicazioni

Le applicazioni installate sono accessibili tramite il menu di LXDE.

Il menu è accessibile selezionando l'icona con il logo LXDE in basso a sinistra dello schermo oppure premendo **Ctrl+Esc** sulla vostra tastiera.

In questo menu le applicazioni sono raggruppate per tipologia. Per mostrare l'icona di un'applicazione sul desktop è sufficiente fare un clic con il tasto destro sulla corrispondente voce di menu e selezionare **Aggiungere alla scrivania**. In questo modo potrete avviare direttamente dal desktop le applicazioni che utilizzate più di frequente. Per rimuovere una qualsiasi delle icone del desktop, è sufficiente selezionare l'icona da eliminare con un clic destro e scegliere l'opzione **Elimina** dal menu che verrà visualizzato. Dentro il menu di LXDE troverete molte applicazioni interessanti; nei prossimi paragrafi vedremo alcune di quelle di uso più comune.

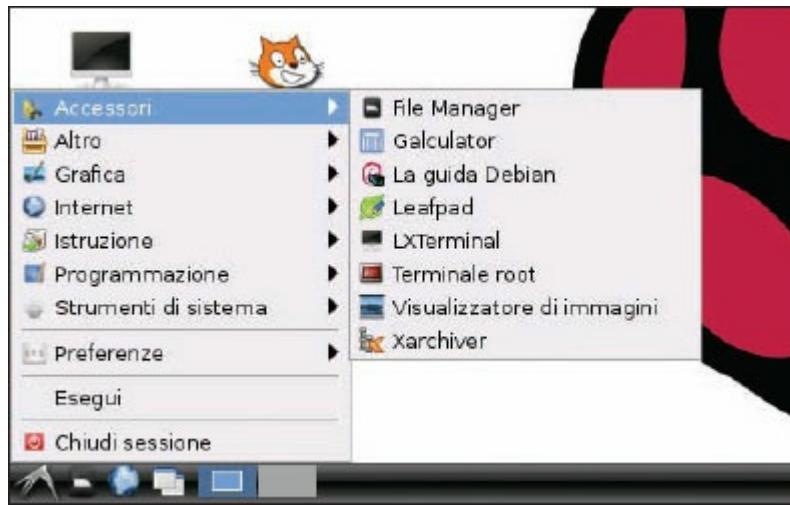


Figura 2.8 - Il menu di LXDE.

File Manager

L'applicazione file manager permette di navigare all'interno del filesystem del nostro Raspberry Pi. Affronteremo più in dettaglio la struttura del filesystem nei prossimi capitoli, ma il file manager può essere uno strumento utile per iniziare a esplorarlo.

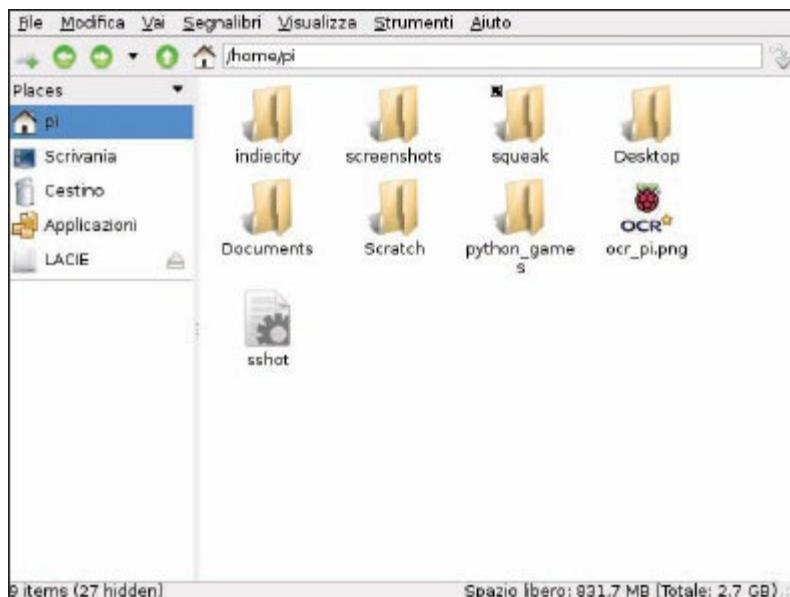


Figura 2.9 - La schermata principale del file manager.

La schermata mostra sulla sinistra un elenco delle cartelle di utilizzo più frequente:

- la cartella "home" è la cartella in cui vengono normalmente memorizzati i dati e i file di configurazione specifici per ogni singolo utente;
- la "scrivania" con i file presenti sul desktop; possiamo copiare qui i file o le cartelle che vogliamo vedere sul desktop;

- il cestino, perché anche i migliori sbagliano e recuperare un file cancellato per errore può sempre tornare utile;
- le applicazioni che possono essere lanciate da questa cartella o direttamente dal menu di LXDE;
- le memorie esterne (tipicamente USB) collegate al nostro Raspberry Pi, in modo da poter rapidamente scambiare contenuti con altri sistemi.

Una barra nella parte alta della finestra contiene una serie di pulsanti che consentono di aprire una nuova “pagina” dentro il file manager (in questo modo potremo tenere aperte più cartelle contemporaneamente), di tornare alle cartelle visitate in precedenza o di risalire di un livello nel filesystem. Visualizza inoltre il percorso della cartella correntemente visualizzata e ci consente di inserire un nuovo percorso per andare direttamente a visualizzare una qualsiasi directory. L’ultimo pulsante consente di effettuare un refresh della cartella corrente, aggiornando i suoi contenuti a seguito di creazioni o cancellazioni di file e sotto-cartelle.

Se volete scoprire tutti i segreti nascosti dentro il vostro Raspberry Pi potete selezionare l’opzione **mostra file nascosti** del menu **Visualizza**. In questo modo verranno mostrati anche i file e le cartelle il cui nome inizia con un punto. Questi file vengono normalmente nascosti all’utente sui sistemi Linux.

Prendetevi un po’ di tempo per navigare all’interno del filesystem. I file e le cartelle che contengono componenti fondamentali del sistema operativo non sono modificabili da parte dell’utente pi, quindi non correrete il rischio di danneggiare alcunché di critico. E se ci si perde all’interno dei meandri del file system si può sempre tornare “a casa” selezionando l’icona **home!**

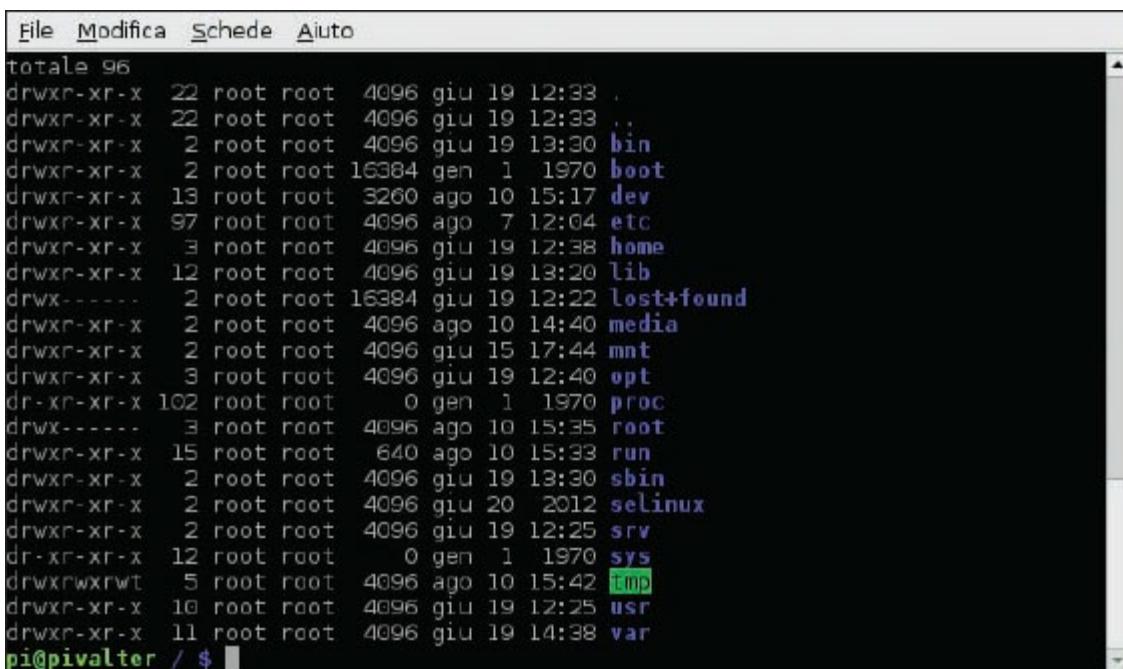
In realtà, come vedremo nel prossimo capitolo, è spesso più efficiente manipolare il filesystem utilizzando la linea di comando. Anche il file manager ci consente di accedere alla linea di comando selezionando l’opzione **Apri cartella corrente nel terminale** dal menu **Strumenti**. La seconda opzione del menu **Strumenti** consente di aprire un terminale con i diritti di amministrazione (l’utente root è l’utente amministratore dei sistemi Unix), quindi di essere esposti al rischio di combinare qualche disastro accidentale. Aspettate di aver letto il prossimo capitolo prima di cimentarvi con la command line da

utente root!

LXTerminal

Questa applicazione ci permetterà di accedere alla command line del sistema aprendo un “terminale”. Oltre che attraverso LXTerminal è possibile accedere ad altre console a linea di comando premendo i tasti **Ctrl+Alt+F1, F2, F3, F4, F5, F6**. Avere più console è utile per poter eseguire più applicazioni, senza dover aspettare il completamento di quella in corso. Scopriremo meglio come utilizzare la command line nel prossimo capitolo.

La prima console (quella accessibile con **Ctrl+Alt+F1**) mostrerà anche i messaggi di sistema. Per poter eseguire comandi e applicazioni in queste console è necessario eseguire il login con username **pi** e la password che avete definito in fase di installazione; non è necessario inserire le informazioni di login quando si lancia LXTerminal.



The screenshot shows a terminal window with a menu bar at the top containing "File", "Modifica", "Schede", and "Aiuto". The main area displays a list of files and directories in a terminal session. The output starts with "totale 96" followed by a list of entries. Some entries are color-coded: "bin" is blue, "boot" is red, "dev" is green, "etc" is blue, "home" is blue, "lib" is blue, "lost+found" is blue, "media" is green, "mnt" is blue, "opt" is blue, "proc" is red, "root" is blue, "run" is green, "sbin" is blue, "selinux" is blue, "srv" is blue, "sys" is red, "tmp" is green, "usr" is blue, and "var" is blue. The prompt at the bottom is "pi@pivalter ~ \$".

```
File Modifica Schede Aiuto
totale 96
drwxr-xr-x 22 root root 4096 giu 19 12:33 .
drwxr-xr-x 22 root root 4096 giu 19 12:33 ..
drwxr-xr-x 2 root root 4096 giu 19 13:30 bin
drwxr-xp-x 2 root root 16384 gen 1 1970 boot
drwxr-xr-x 13 root root 3260 ago 10 15:17 dev
drwxr-xr-x 97 root root 4096 ago 7 12:04 etc
drwxr-xr-x 3 root root 4096 giu 19 12:38 home
drwxr-xr-x 12 root root 4096 giu 19 13:20 lib
drwxr-xr-x 2 root root 16384 giu 19 12:22 lost+found
drwxr-xr-x 2 root root 4096 ago 10 14:40 media
drwxr-xr-x 2 root root 4096 giu 15 17:44 mnt
drwxr-xr-x 3 root root 4096 giu 19 12:40 opt
dr-xr-xr-x 102 root root 0 gen 1 1970 proc
drwxr-xr-x 3 root root 4096 ago 10 15:35 root
drwxr-xr-x 15 root root 640 ago 10 15:33 run
drwxr-xr-x 2 root root 4096 giu 19 13:30 sbin
drwxr-xr-x 2 root root 4096 giu 20 2012 selinux
drwxr-xr-x 2 root root 4096 giu 19 12:25 srv
dr-xr-xr-x 12 root root 0 gen 1 1970 sys
drwxrwxrwt 5 root root 4096 ago 10 15:42 tmp
drwxr-xr-x 10 root root 4096 giu 19 12:25 usr
drwxr-xr-x 11 root root 4096 giu 19 14:38 var
pi@pivalter ~ $
```

Figura 2.10 - LXTerminal.

Se non avete modificato la password in fase di installazione, se vi è venuta in mente una nuova password più sicura della prima o se volete cambiare qualcun altro dei parametri definiti in fase di setup di Raspberry Pi, potete rilanciare l'applicazione di setup da una qualsiasi command line digitando il comando:

```
sudo raspi-config
```

Potrete così ripetere le operazioni descritte nel capitolo precedente per riconfigurare tastiera, time zone e altri parametri del sistema.

La command line è accessibile anche da remoto tramite il protocollo SSH, che tratteremo nel [Capitolo 4](#). Questo ne fa un ottimo strumento per la gestione e l'amministrazione remota.

Dimenticavo... per tornare al desktop dalle console testuali è sufficiente premere **Ctrl+Alt+F7**.

Nel menu di LXDE, sotto la voce **Accessori** è presente un secondo terminale, identificato come **Terminale root**. Questo terminale consente di eseguire operazioni potenzialmente rischiose per il sistema e non concesse all'utente normale. Nel corso di questo libro non sarà mai necessario utilizzare questo terminale e le operazioni che richiedono privilegi elevati verranno eseguite tramite il comando **sudo** (Super User Do).

Come già anticipato, nel prossimo capitolo utilizzeremo parecchio la linea di comando, quindi per ora possiamo riprendere il nostro fedele mouse e continuare nell'esplorazione delle applicazioni grafiche.

Midori

Se avete a portata di mano un PC con Windows o un Mac e provate a vedere qual è l'occupazione di memoria di uno qualsiasi dei browser più diffusi per queste piattaforme (Internet Explorer, Chrome, Safari, Firefox); vi renderete immediatamente conto del motivo per cui nessuno di questi software è disponibile per Raspberry Pi. Tutti utilizzano una quantità impressionante di memoria e, già aprendo poche pagine, la memoria utilizzata eccede i 256 MB del Model A. Per questo motivo Raspbian fornisce un browser più leggero. Questo browser si chiama Midori e la sua icona fa bella mostra di sé sul desktop di default. Midori è basato sul "motore" WebKit, lo stesso utilizzato da Chrome e Safari, due dei browser più diffusi su PC, Mac e piattaforme mobili. Questo gli garantisce una buona compatibilità con la maggior parte di siti web. Midori ha la capacità di mostrare più schede, per tenere aperte contemporaneamente più pagine nella stessa finestra, gestisce i segnalibri, la ricerca direttamente dalla barra di navigazione, la cronologia e buona parte delle funzioni presenti nei browser più comuni. Nonostante la gamma abbastanza estesa di funzionalità, Midori riesce a contenere la quantità di memoria necessaria al suo funzionamento entro limiti compatibili con quelli di Raspberry Pi.

Se avete collegato il vostro Raspberry Pi a internet potrete usare Midori per navigare, cercare informazioni e scaricare file. Parleremo più in dettaglio del collegamento alla rete nei prossimi capitoli.

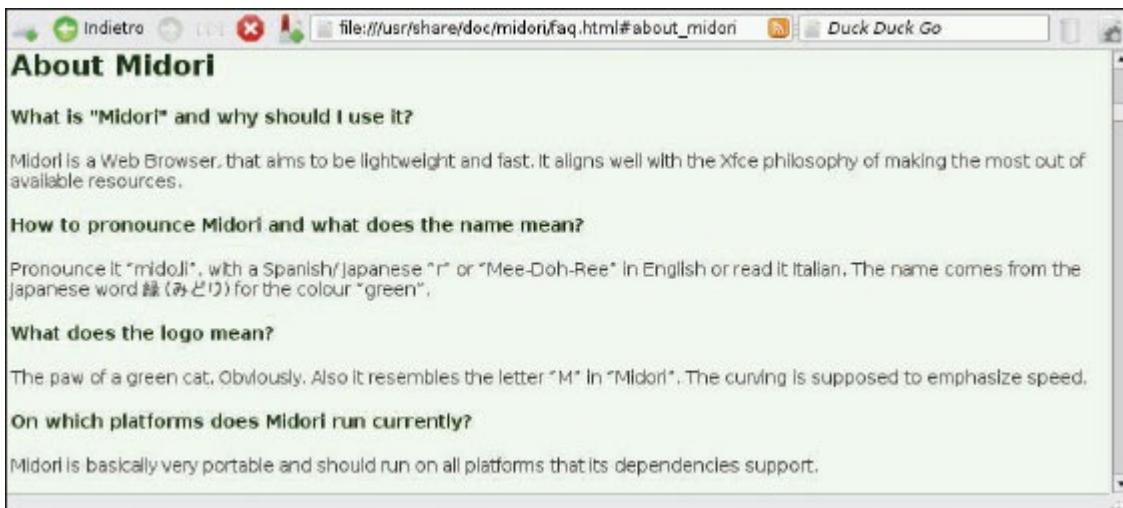


Figura 2.11 - Midori.

Pi Store

Molte delle piattaforme mobili e non come iOS, OS X, Android e Windows 8 consentono agli utenti di scaricare nuove applicazioni da un “marketplace” centralizzato, normalmente gestito da chi sviluppa la piattaforma. Raspberry Pi non è da meno e il Pi Store vi consente di scaricare diverse applicazioni gratuite e a pagamento. Sono disponibili giochi, applicazioni “produttive”, strumenti di sviluppo e tutorial. Per scaricare le applicazioni, anche quelle gratuite, è necessario registrarsi fornendo il proprio indirizzo e-mail.

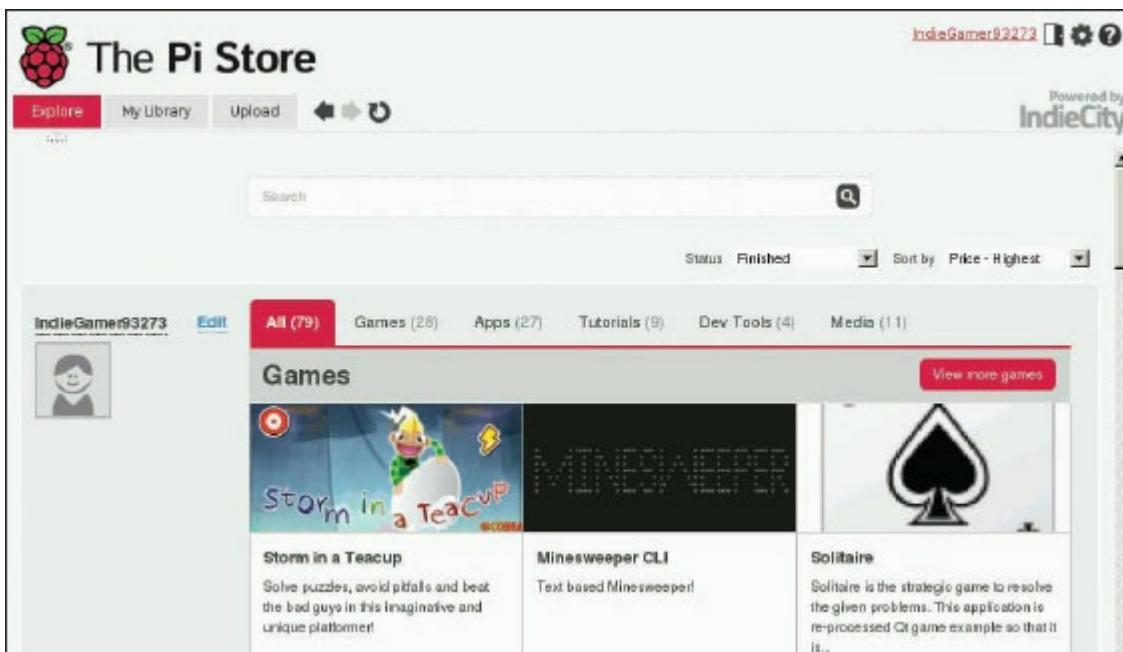


Figura 2.12 - Pi Store.

Divertitevi a scoprire quante applicazioni interessanti esistono per il vostro piccolo computer e ricordate che quelle nel Pi Store sono solo una piccola parte di quanto è disponibile in rete.

Una volta selezionata l'applicazione da installare questa verrà scaricata e installata automaticamente. Spesso l'installazione richiede qualche minuto e mostra una schermata terminale.

Le applicazioni installate e quelle in corso di installazione sono mostrate nella cartella **My Library**. Questa cartella consente di lanciare le applicazioni installate oppure di rimuoverle.

Pi Store non è l'unico sistema per installare nuove applicazioni sul nostro Raspberry Pi. Il sistema, a differenza di quanto avviene con la maggior parte dei dispositivi commerciali legati a un marketplace, consente di installare software semplicemente scaricandolo dalla rete. E non dimenticate che tantissimo del software che utilizzeremo durante questo "viaggio" alla scoperta di Raspberry Pi è rilasciato come open source, cosa non molto frequente sulle piattaforme commerciali.

Scratch

Scratch è un linguaggio di programmazione creato dal Media Lab del MIT (Massachusetts Institute of Technology, una delle università più prestigiose del mondo) come strumento educativo per insegnare i principi della programmazione a bambini in età scolare e adolescenti.

L'idea di creare un linguaggio semplice e adatto a semplificare

l'apprendimento non è nuova. Diversi linguaggi, come Logo, sono stati creati con questo obiettivo già alla fine degli anni '60. Lo stesso Basic, diffuso sui primi home computer, ma ancora oggi molto utilizzato anche per lo sviluppo di applicazioni professionali, nasceva come linguaggio per principianti. Il suo acronimo, infatti, sta per "Beginner's All-purpose Symbolic Instruction Code". Linguaggio per principianti, appunto.

Questi ambienti di sviluppo si dovevano però scontrare con le risorse limitate e la scarsa interattività di quei sistemi. Sui primi home computer il sistema partiva presentando l'interprete Basic, ma l'interfaccia a caratteri e l'assenza di dispositivi evoluti di interazione oltre alla tastiera (il mouse stava facendo i primi passi e solo i Macintosh ne avevano uno di serie) rappresentavano un ostacolo che rendeva la programmazione un'arte per pochi iniziati armati di passione e pazienza.

Scratch nasce nel 2003, quando ormai l'interfaccia grafica si è ampiamente diffusa, quindi sfrutta molto gli elementi grafici per semplificare e rendere più intuitivo l'approccio alla definizione di un "programma". Essendo uno strumento principalmente rivolto ai ragazzi privilegia lo sviluppo di giochi e avventure interattive, per unire allo studio dei principi di programmazione un aspetto ludico e motivante.



Figura 2.13 - Scratch.

Ovviamente la filosofia di Scratch si combina benissimo con gli scopi per cui è stato realizzato e prodotto Raspberry Pi, ed è per questo che il gatto sorridente, icona di Scratch, fa bella mostra di sé sul desktop del nostro piccolo personal computer.

Per gli esempi di programmazione in questo libro utilizzeremo il linguaggio Python, ma potete trovare moltissimo materiale su Scratch partendo dal sito ufficiale scratch.mit.edu.

Dal sito potrete anche scaricare tantissimi esempi e giochi completi che possono servire a sollecitare la curiosità di qualche programmatore in erba.

Python

Python è un linguaggio sviluppato da Guido Van Rossum, un programmatore olandese, alla fine degli anni '80.

Deve il suo nome al famoso gruppo comico inglese Monty Python e, come i film dei succitati comici, gode di un ampio apprezzamento tra gli sviluppatori, soprattutto sui sistemi Linux.

Anche Python, come Linux, segue un modello di sviluppo open source ed evolve sotto la guida del suo sviluppatore originale, che si auto-definisce "Benevolent Dictator For Life". Il successo di Python è stato travolgente e ha aggregato a mano a mano al progetto moltissimi sviluppatori che hanno contribuito a migliorare il linguaggio e a estenderne le funzionalità, implementando moltissime librerie aggiuntive.

Nella seconda metà di questo libro utilizzeremo Python per interagire con i sensori e sviluppare semplici applicazioni.

Python è un linguaggio interpretato. Questo vuol dire che per utilizzare i nostri programmi Python dovremo avere un interprete in grado di eseguirli sul nostro Raspberry Pi.

Uno degli interpreti Python più diffusi è IDLE.

Sul desktop di Raspbian ne troviamo due versioni. Quella identificata con IDLE supporta la versione 2.7 di Python, mentre quella denominata IDLE 3 supporta la versione 3.2. Gli esempi di questo libro verranno realizzati con quest'ultima versione.

```
File Edit Shell Debug Options Windows Help
Python 3.2.3 |default, Mar 1 2013, 11:53:50|
[GCC 4.6.3] on linux2
Type "copyright", "credits" or "license()" for more information.
>>> copyright
Copyright (c) 2001-2012 Python Software Foundation.
All Rights Reserved.

Copyright (c) 2000 BeOpen.com.
All Rights Reserved.

Copyright (c) 1995-2001 Corporation for National Research Initiatives.
All Rights Reserved.

Copyright (c) 1991-1995 Stichting Mathematisch Centrum, Amsterdam.
All Rights Reserved.
>>> credits
    Thanks to CWI, CNRI, BeOpen.com, Zope Corporation and a cast of thousands
    for supporting Python development. See www.python.org for more information.
>>> print("Hello World!")
Hello World!
>>>
```

Ln: 21 Col: 4

Figura 2.14 - IDLE 3.

Un linguaggio interpretato ha solitamente performance inferiori a quelle dei linguaggi compilati, come C e C++, il cui codice viene trasformato in codice macchina da un compilatore direttamente in fase di sviluppo e non richiede quindi l'interpretazione in fase di esecuzione. L'interprete di Python è tuttavia molto efficiente e una buona parte delle funzioni della libreria di Python sono implementate in codice compilato, garantendo le performance dei componenti più critici.

La flessibilità di Python e la possibilità di eseguire direttamente gli script senza dover passare per una fase di compilazione lo rendono un ottimo strumento per chi vuole imparare a programmare e questo si adatta perfettamente allo spirito del nostro viaggio alla scoperta di Raspberry Pi.

Spesso si considerano i linguaggi interpretati come linguaggi adatti più alla realizzazione di “script” (semplici sequenze di comandi utilizzate per automatizzare operazioni ripetitive) che allo sviluppo di programmi complessi. Python, però, unisce alla semplicità dei più diffusi linguaggi di scripting una sintassi molto chiara e leggibile, il supporto di caratteristiche avanzate come la programmazione object oriented e una

ricchissima libreria di funzioni. Queste caratteristiche consentono di utilizzarlo per sviluppare anche applicazioni complesse e strutturate.

In noto servizio di sincronizzazione file e memorizzazione in cloud Dropbox utilizza molto pesantemente Python sia per i suoi server, sia per il software client disponibile per moltissime piattaforme. Python è così importante per l'azienda che, dopo aver lavorato per diversi anni alle dipendenze di Google, il buon Guido Van Rossum è ora un dipendente di Dropbox!

Approfondiremo la conoscenza di Python nei prossimi capitoli, ma se volete cimentarvi nella realizzazione del vostro primo semplicissimo programma Python potete lanciare **IDLE 3**, scrivere:

```
print("Hello World!")
```

nella shell IDLE e premere **Invio**.

Benvenuti nel novero dei programmatori!

Se poi volete scoprire qualche applicazione ludico-pratica di Python su Raspberry Pi, potete lanciare i Python Games presenti sul desktop di Raspbian.

Leafpad

Leafpad è un semplice editor di testo che può essere utile per modificare o anche semplicemente consultare file testuali. Come scopriremo nel prossimo capitolo (scusate se vi ho rovinato la sorpresa, ma questo libro ne ha in serbo ancora moltissime!), anche tutti i file di configurazione del sistema sono file di testo e Leafpad è lo strumento ideale per modificarli.

The screenshot shows a window of the Leafpad text editor. The menu bar at the top includes 'File', 'Modifica', 'Cerca', 'Opzioni', and 'Aiuto'. The main text area contains several blocks of text, likely build instructions for a Raspberry Pi application. The text is as follows:

```
File Modifica Cerca Opzioni Aiuto
Building on Pi
+++++
To build the test apps on the pi, first build the libs:
make -C libs/ilclient
make -C libs/vgfont

then by entering each test app directory and run make. E.g
cd hello_world
make
./hello_world.bin

Running ./rebuild.sh will rebuild the all libs and and app

Building on a different PC
+++++
If you want to build the samples on a different machine (c
```

Figura 2.15 - Leafpad.

Dal file manager è possibile aprire qualsiasi file utilizzando Leafpad, semplicemente selezionandolo con un clic destro e scegliendo l'opzione **Leafpad** del menu popup.

Altre utility

Prendetevi un po' di tempo per esplorare il menu di LXDE e cercare altre applicazioni interessanti.

Ci sono le classiche utility da desktop come la calcolatrice, un visualizzatore di immagini, un gestore di file compressi, un task manager e molto altro.

C'è anche la **Guida di Debian** che è un ottimo testo introduttivo (purtroppo in lingua inglese) per approcciare Linux e la command line. Se non ve la cavate troppo bene con l'inglese, non disperate, la command line sarà l'oggetto del prossimo capitolo.

Personalizzare il desktop

Una delle caratteristiche più interessanti di LXDE è la possibilità di personalizzarlo ampiamente.

Il suo aspetto grafico può essere modificato per adattarlo al meglio al nostro modo di utilizzare Raspberry Pi e, perché no, ai nostri gusti in campo estetico.

Sfondo e screensaver

Se siete allergici ai lamponi, o se avete in mente altre immagini che potrebbero fare bella mostra di sé al centro del desktop di Raspbian, con un clic destro su un'area del desktop sgombra da icone e pannelli potete accedere all'opzione **Preferenze del Desktop**.



Figura 2.16 - Il menu Desktop.

Da questa schermata sarà possibile cambiare sia l'immagine di sfondo (e il relativo posizionamento) sia il colore e il tipo di carattere utilizzati per il testo delle icone.



Figura 2.17 - Preferenze del desktop.

Ora che avete personalizzato lo sfondo è il momento di cambiare un altro aspetto grafico del sistema, lo screen saver.

Lo screen saver è una piccola applicazione grafica che viene lanciata dopo un certo periodo di inattività dell'utente. Ai tempi dei monitor a fosfori verdi lo screen saver serviva a evitare che un'immagine statica visualizzata a lungo rimanesse "impressa" in modo permanente sullo schermo. Oggi questo rischio non esiste più e lo screen saver ha

soprattutto la funzione di nascondere il nostro lavoro da occhi indiscreti se, per qualche motivo, ci siamo allontanati dal computer. Spesso gli screen saver si utilizzano solamente per mostrare qualche animazione o effetto grafico e dilettrare chi si trova a passare di fronte alla nostra postazione. Per default in Raspbian non sono installati screensaver. Per installarli è necessario ricorrere alla command line.

Se siete collegati a internet, dopo aver aperto **LxTerminal**, digitate:

```
sudo apt-get install sxcreensaver xscreensaver-data-extra
```

In questo modo verranno installati gli screensaver che potrete poi selezionare.

Prima di continuare con l'installazione il sistema ci informerà in merito allo spazio su disco richiesto e sarà necessario premere il tasto **S** per eseguire l'installazione vera e propria.

Il comando `apt-get` consente di aggiungere componenti al sistema operativo scaricandoli dalla rete. Utilizzeremo questo comando nei prossimi capitoli per aggiungere nuove funzionalità al nostro sistema.

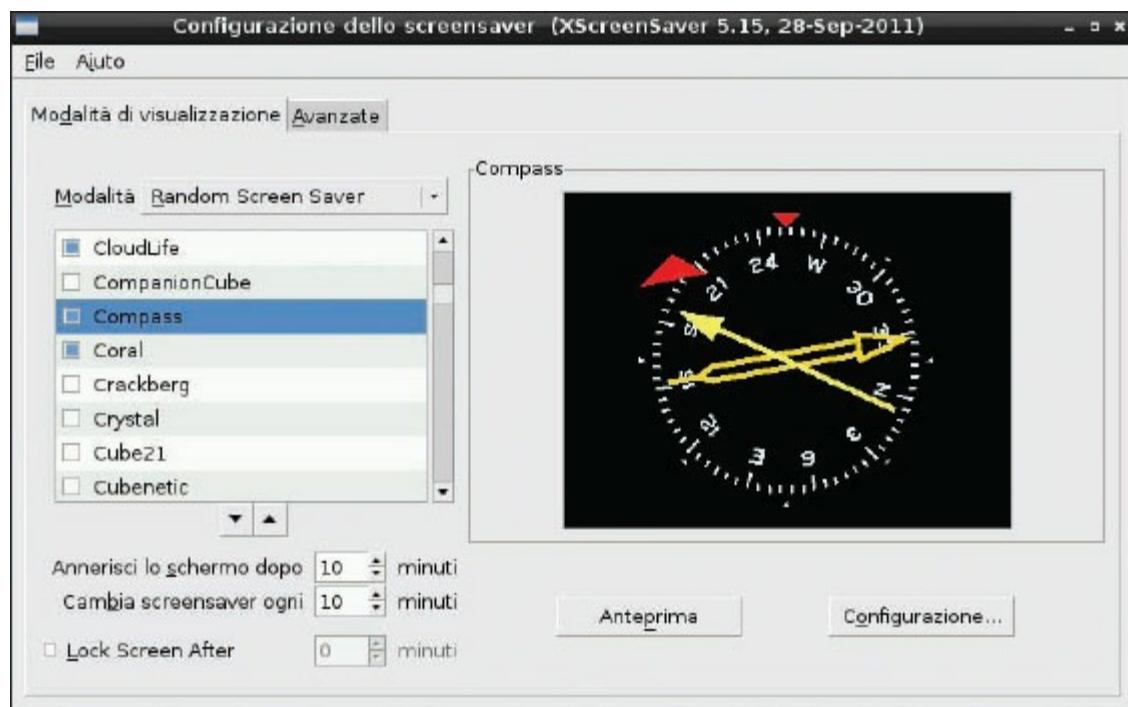


Figura 2.18 - Selezione dello screen saver.

Avere una gestione centralizzata per questo tipo di pacchetti riduce il rischio di installare versioni incompatibili dei diversi componenti di sistema e semplifica moltissimo la gestione del nostro sistema operativo.

Una volta installati gli screen saver dovrete riavviare il sistema con il comando:

`sudo reboot`

Al riavvio troverete l'opzione **Screensaver** nel sotto-menu **Preferenze** del menu di LXDE. Potrete selezionare lo screen saver che preferite o lasciare che il sistema lo scelga in modo casuale ogni volta che resterà inattivo. Gli screen saver sono anche una buona dimostrazione delle capacità grafiche di Raspberry Pi.

Aspetto delle finestre e dei controlli

È possibile modificare l'aspetto grafico delle finestre lanciando l'applicazione **Openbox Configuration Manager** dalla cartella **Preferenze** del menu di LXDE. È possibile selezionare uno dei diversi temi e cambiare l'aspetto della barra del titolo e dei menu collegati alle diverse finestre. Nuovi temi possono essere scaricati dal sito di Openbox nella sezione dedicata ai temi (www.openbox.org/download-themes.php).

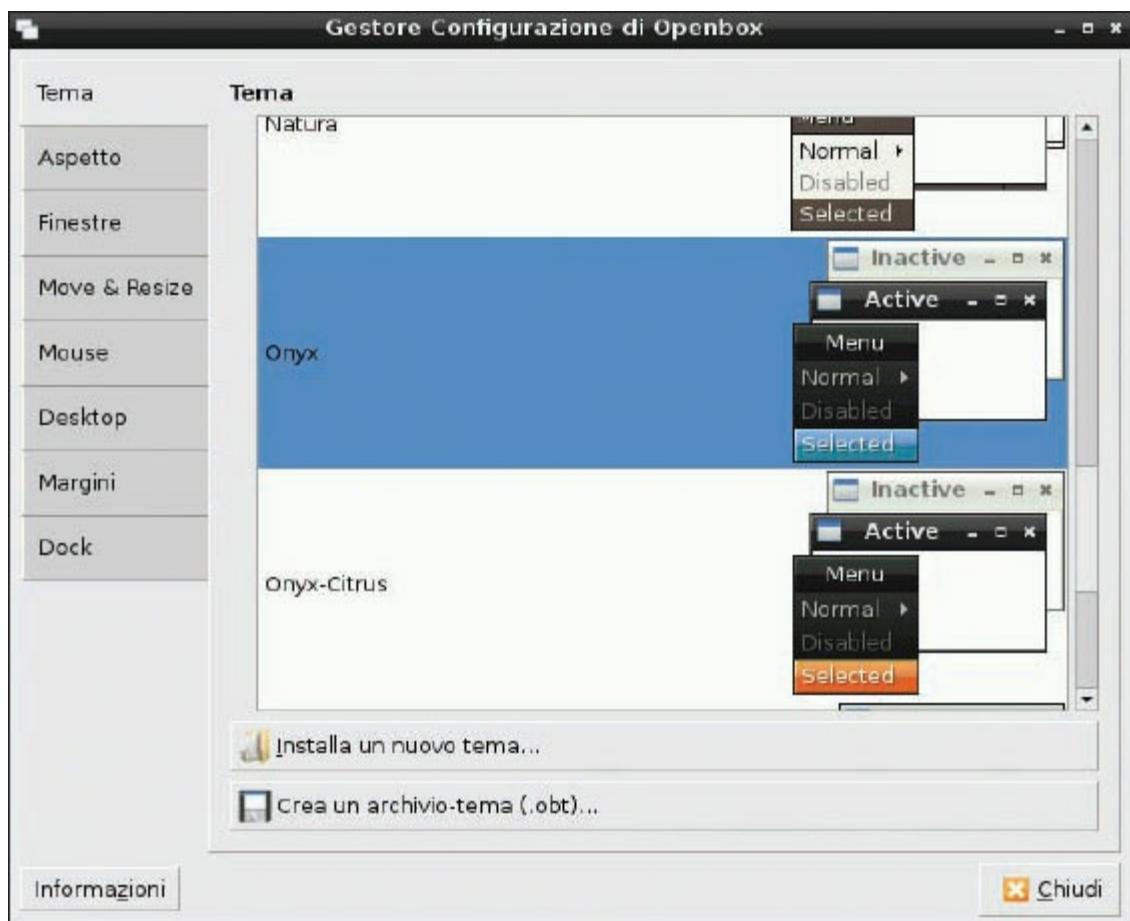


Figura 2.19 - Selezione del tema per le finestre.

Le altre opzioni di Configuration Manager consentono di modificare le modalità di ridimensionamento delle finestre, di lasciare un margine su

desktop non coperti da altre finestre o anche di aumentare il numero dei desktop virtuali gestiti da LXDE.

Openbox gestisce solo l'aspetto esterno delle finestre. Ogni applicazione disegna al suo interno widget e controlli (i pulsanti, i campi per l'inserimento del testo ecc.) utilizzando diverse librerie grafiche. Questo spiega perché non sempre le interfacce grafiche delle diverse applicazioni mantengono un aspetto estetico uniforme. Le librerie grafiche più diffuse sono GTK e QT. Una buona parte delle applicazioni incluse in Raspbian utilizza le librerie GTK, realizzate dal progetto Gnome. Per cambiare l'aspetto dei widget di GTK è possibile usare l'applicazione **Personalizza Aspetto e Stile** del menu **Preferenze** di LXDE. Anche in questo caso potremo configurare l'aspetto dei nostri widget selezionandolo tra diversi temi predefiniti. Potremo personalizzare anche le icone utilizzate dalle diverse applicazioni e lo stile dei puntatori del mouse.

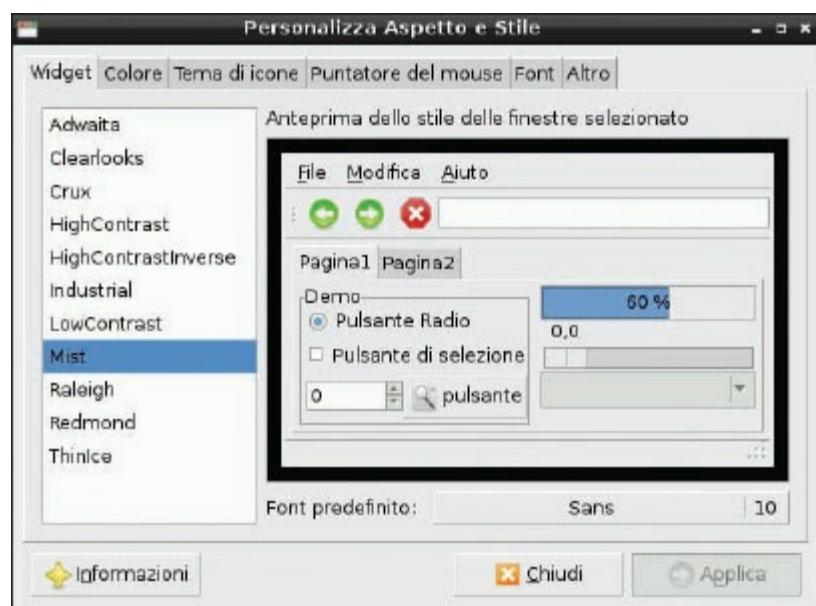


Figura 2.20 - Selezione del tema per i widget.

Date pure sfogo al vostro senso estetico personalizzando a piacimento l'aspetto grafico del sistema. Abbiamo definito Raspberry Pi "piccolo personal computer" e personalizzarlo è sicuramente una delle attività più divertenti.

Pannelli

La barra delle applicazioni di LXDE può essere personalizzata aggiungendo o rimuovendo i diversi pannelli che la compongono. È

anche possibile aggiungere nuove barre per mostrare più informazioni rispetto a quelle disponibili nella configurazione di default.

Per aggiungere un nuovo pannello basta un clic destro sulla barra delle applicazioni. Questo farà apparire un menu popup da quale sarà possibile scegliere l'opzione **Crea nuovo pannello**.



Figura 2.21 - Impostazioni pannello.

La finestra di impostazioni del pannello ci consentirà di determinarne la posizione, le dimensioni, i colori ecc. e, soprattutto, di aggiungere nuove applet al nostro pannello. Le applet consentono di tenere sotto controllo alcuni aspetti del sistema oppure di attivare direttamente le diverse funzionalità di Raspbian, senza dover passare per il menu o le icone del desktop.



Figura 2.22 - Le applet disponibili.

Con l'applet **Barra avvio di applicazioni** è possibile aggiungere dei pulsanti per lanciare direttamente le applicazioni presenti nel menu (come già avviene con Midori e il file manager), mentre l'applet **Menu cartella** consente di aggiungere un riferimento diretto per aprire una cartella attraverso il file manager (potrebbe essere utile per creare un link ai sorgenti dei diversi esempi di questo testo).

Nuovi plugin possono essere aggiunti utilizzando il Pi Store facendo diventare i vostro piccolo computer sempre più “personal”.

L'autore si è trattenuto dal dare sfogo alla sua creatività e al suo (a detta di molti discutibile) senso estetico, lasciando la configurazione ai default, per evitare che gli screenshot riportati in queste pagine non trovassero corrispondenza con quanto mostrato sui Raspberry Pi dei lettori.

Dopo aver scoperto le meraviglie dell'interfaccia grafica e averla adattata ai nostri gusti possiamo passare al prossimo capitolo e scoprire l'altra interfaccia utente del nostro sistema: la command line.

Guarda mamma, senza mouse!

La **command line** permette di semplificare e automatizzare tantissime operazioni. Conoscere i **comandi base** del sistema **Linux** e imparare a realizzare semplici **script** ci permetterà di sfruttare ancora più a fondo il nostro piccolo computer.

La command line è stata la prima forma di interfaccia utente supportata da Unix e da tutti i sistemi mainframe diffusi dagli anni '70 e '80. Un terminale in modo testo consentiva agli utenti di lanciare applicazioni, inserire dati e, in generale, di interagire con il sistema. Nella prima generazione di home computer l'interfaccia era costituita da un interprete Basic, sempre in modalità testuale.

Il messaggio:

```
****CBM BASIC V2 ****  
3583 BYTE FREE
```

e il prompt "READY" del Vic-20 sono stati per chi scrive il primo impatto con un vero computer.

Solo con l'avvento della seconda generazione di home computer (Amiga, Atari ST, Archimedes, Apple IIGS) e dei Macintosh l'interfaccia grafica andò lentamente a soppiantare quella in modo testo.

Il mondo dei PC arrivò buon ultimo con Windows e OS/2.

La command line resta comunque un'interfaccia molto utilizzata su diversi sistemi operativi, anche quelli che ormai utilizzano l'interfaccia grafica in modo molto estensivo, per compiti di gestione, automazione e amministrazione del sistema.

In Linux la command line riveste un ruolo importantissimo e conoscerne i meccanismi di funzionamento ci permetterà di sfruttare ancora meglio tutte le potenzialità di Raspberry Pi.

Questo capitolo presenterà alcuni dei comandi principali che si possono utilizzare attraverso la command line, poi spiegherà come combinarli tra loro per realizzare funzionalità più complesse arrivando a realizzare semplici programmi, chiamati script.

La shell

Sui sistemi Linux la command line è gestita da un'applicazione detta **shell** che si occupa di interpretare quanto scritto sulla console e tradurlo nell'esecuzione di comandi. Esistono diverse shell. Raspbian utilizza una delle più diffuse: bash.

Per accedere alla command line è sufficiente lanciare **LXTerminal** oppure premere **Alt+Ctrl+F1** sulla tastiera. Se utilizziamo questa seconda modalità la nostra command line sarà a tutto schermo e prima di poterla utilizzare dovremo inserire username e password.

Nel prossimo capitolo scopriremo anche come accedere al nostro sistema tramite una command line remota.

La command line ci accoglie con un messaggio di benvenuto che può sembrare piuttosto criptico:

```
Linux pivalter 3.6.11+ #474 PREEMPT Thu Jun 13 17:14:42 BST 2013 armv6l
```

```
The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/*copyright.
```

```
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
```

```
Last login: Mon Aug 12 15:22:03 2013
```

La prima linea ci fornisce diverse informazioni sul nostro sistema. Ci dice che stiamo eseguendo Linux, ci fornisce l'host name della nostra macchina (sarà un'informazione importante quando la utilizzeremo in rete) e la versione del kernel che stiamo utilizzando. Nel nostro caso si

tratta di Linux versione 3.6.11 compilato il 13 luglio del 2013 nella versione per processore ARM.

La seconda parte del messaggio fornisce informazioni sulla licenza del software che stiamo utilizzando, dicendoci che si tratta di software libero, distribuito in modalità open source.

Raspberry Pi ci fornisce anche la data dell'ultimo login. Purtroppo il sistema non ha un orologio con batteria tampone, quindi l'orario si resetta a ogni restart. All'avvio (e poi periodicamente) Raspberry Pi provvede a sincronizzare il suo orario usando il protocollo NTP (Network Time Protocol), che consente di recuperare data e ora corrente da diversi server internet.

Dopo il messaggio di benvenuto viene mostrato il prompt di sistema. Il prompt è un'indicazione del fatto che la nostra console è pronta a ricevere nuovi comandi.

```
pi@pivalter ~ $
```

Anche il prompt ci fornisce parecchie informazioni utili. Ci indica l'utente corrente (`pi`) e, dopo il simbolo "at" (quella che in italiano viene chiamata comunemente "chiocciolina") il nome della macchina su cui stiamo lavorando. Viene poi mostrata la directory corrente. Per default la command line è posizionata nella cartella "home" del nostro utente e, invece di ripeterne il path completo (`/home/pi`) viene riportato il simbolo tilde (`~`). Potremo usare questo simbolo per indicare la cartella home dell'utente corrente anche all'interno dei nostri script, ma non corriamo troppo... prima di scrivere uno script dovremo imparare i comandi principali della command line!

Il carattere `$` (dollaro) funge da terminatore del prompt e ci indica che il sistema è pronto a ricevere un nuovo comando.

Digitando il comando `ls` (abbreviazione di LiSt) verrà mostrato l'elenco dei file e delle sotto-cartelle della cartella corrente:

```
pi@pivalter ~ $ ls
Desktop indiecity python_games Documents ocr_pi.png
pi@pivalter ~ $
```

Da questo elenco non è possibile capire quali nomi si riferiscono a file e quali a sottocartelle.

Se vogliamo avere qualche informazione in più sui file possiamo aggiungere al comando `ls` il parametro `-l`. I parametri devono essere scritti dopo il nome del comando e prefissati con il carattere `-` (meno). È

importante fare attenzione all'utilizzo di maiuscole e minuscole. Nei sistemi Unix i comandi, i nomi di file e i parametri sono case-sensitive. Questo vuol dire che `-l` e `-L` non sono equivalenti, così come `ls` e `LS`!

```
pi@pivalter ~ $ ls -l
totale 12
drwxr-xr-x 2 pi pi 4096 giu 19 14:18 Desktop
drwxr-xr-x 3 pi pi 4096 ago  7 13:12 Documents
drwxr-xr-x 3 pi pi 4096 ago  7 12:04 indiecity
drwxrwxr-x 2 pi pi 4096 mar 10 11:20 python_games
-rw-r--r-- 1 pi pi 5781 feb  3  2013 ocr_pi.png
pi@pivalter ~ $
```

In questo caso, oltre ai nomi dei file e delle sotto-cartelle, vengono mostrate anche altre informazioni utili.

La prima riga indica il numero di blocchi (normalmente da 1024 byte) occupati dal contenuto della cartella.

Il primo carattere indica il tipo di oggetto è può essere `d` per indicare una sotto cartella, `l` per indicare un link (ne parleremo tra poco!) e `-` per indicare un normale file.

I nove caratteri seguenti indicano i diritti di accesso all'oggetto corrispondente. Ogni oggetto può essere letto (`r` - Read), scritto (`w` - Write) o eseguito (`x` - eXecuted). Bastano quindi tre lettere per rappresentare i possibili diritti di accesso al file.

Ogni file è poi associato a un utente e a un gruppo (chiamati entrambi `pi` nel nostro caso) elencati di fianco ai diritti di accesso. I diritti di accesso si riferiscono al proprietario (prima tripletta), al gruppo e a tutti gli altri utenti. Vedremo più avanti come cambiare questi diritti di accesso, come associare un diverso utente o gruppo al file o come aggiungere un utente a uno o più gruppi.

Tra la maschera dei diritti di accesso e il nome dell'utente proprietario viene indicato il numero di sotto-oggetti contenuti. Nel caso dei file questo valore non è significativo e sarà sempre pari a uno; per le sotto-cartelle rappresenta il numero di file, link e cartelle contenute.

La dimensione dei file è riportata dopo il nome dell'utente ed è espressa in byte.

La penultima colonna indica la data di ultima modifica del file. Nel caso di date recenti (meno di sei mesi dalla data corrente) vengono riportati giorno e ora di modifica; nel caso di modifiche meno recenti l'anno sostituisce le informazioni relative all'orario. L'ultima colonna, finalmente, propone il nome del file.

Se vogliamo vedere anche i file e le cartelle nascoste possiamo utilizzare il parametro `a`. I parametri possono essere passati separatamente o raggruppati dopo un unico `-`, quindi `ls -l -a` e `ls -la` sono equivalenti. Nei sistemi Unix i file il cui nome inizia con un `.` non vengono mostrati nel normale elenco dei file.

```
pi@pivalter ~ $ ls -la
totale 172
drwxr-xr-x 25 pi pi 4096 ago 12 15:17 .
drwxr-xr-x 3 root root 4096 giu 19 12:38 ..
-rw----- 1 pi pi 3738 ago 12 13:51 .bash_history
-rw-r--r-- 1 pi pi 220 giu 19 12:38 .bash_logout
-rw-r--r-- 1 pi pi 3263 ago 7 08:50 .bashrc
drwxr-xr-x 6 pi pi 4096 ago 7 10:39 .cache
...
-rw-r--r-- 1 pi pi 5781 feb 3 2013 ocr_pi.png
-rw-r--r-- 1 pi pi 675 giu 19 12:38 .profile
drwx----- 2 pi pi 4096 ago 12 11:43 .pulse
-rw----- 1 pi pi 256 ago 7 13:23 .pulse-cookie
drwxrwxr-x 2 pi pi 4096 mar 10 11:20 python_games
drwx----- 4 pi pi 4096 ago 7 08:58 .thumbnails
-rw----- 1 pi pi 155 ago 12 15:17 .Xauthority
-rw----- 1 pi pi 268 ago 12 15:17 .xsession-errors
-rw----- 1 pi pi 0 ago 12 15:17 .xsession-errors.old
```

È possibile lanciare `ls` in directory diverse da quella corrente passando come argomento sulla linea di comando il percorso dei file da mostrare.

Digitando:

```
pi@pivalter ~ $ ls -l /
```

verranno mostrati tutti i file della cartella root, la cartella base di tutto il filesystem:

```
drwxr-xr-x 2 root root 4096 giu 19 13:30 bin
drwxr-xr-x 2 root root 16384 gen 1 1970 boot
drwxr-xr-x 12 root root 3180 ago 12 15:17 dev
drwxr-xr-x 97 root root 4096 ago 7 12:04 etc
drwxr-xr-x 3 root root 4096 giu 19 12:38 home
drwxr-xr-x 12 root root 4096 giu 19 13:20 lib
drwx----- 2 root root 16384 giu 19 12:22 lost+found
drwxr-xr-x 2 root root 4096 ago 10 14:40 media
drwxr-xr-x 2 root root 4096 giu 15 17:44 mnt
drwxr-xr-x 3 root root 4096 giu 19 12:40 opt
dr-xr-xr-x 94 root root 0 gen 1 1970 proc
drwx----- 3 root root 4096 ago 10 15:35 root
drwxr-xr-x 14 root root 560 ago 12 16:38 run
drwxr-xr-x 2 root root 4096 giu 19 13:30 sbin
drwxr-xr-x 2 root root 4096 giu 20 2012 selinux
drwxr-xr-x 2 root root 4096 giu 19 12:25 srv
dr-xr-xr-x 12 root root 0 gen 1 1970 sys
drwxrwxrwt 5 root root 4096 ago 12 17:17 tmp
```

```
drwxr-xr-x 10 root root 4096 giu 19 12:25 usr  
drwxr-xr-x 11 root root 4096 giu 19 14:38 var
```

Quello di Linux, come tutti i filesystem dei sistemi Unix e derivati, è un filesystem ad albero dove sotto un'unica cartella radice (root, indicata con "/") vengono raccolti tutti i file e le sotto-cartelle. Se siete abituati a utilizzare un PC vi starete chiedendo come è possibile accedere a file e cartelle che non siano sulla SD card. Nel filesystem dei sistemi Windows vengono mostrati i singoli drive e ognuno ha un suo filesystem indipendente. In questo caso si parla di filesystem "a foresta".

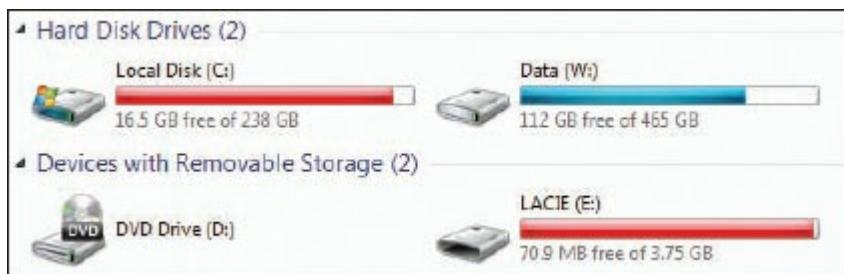


Figura 3.1 - Diversi drive collegati a un PC Windows 7.

Sui sistemi Linux le diverse memorie di massa (dischi esterni USB, dischi di rete ecc.) vengono "montate" all'interno del filesystem come sotto-cartelle della cartella root.

Se provate a collegare una chiavetta USB al vostro Raspberry Pi questa verrà montata come sotto-cartella della cartella /media.

Digitando `ls -l /media/` potrete vedere la nuova cartella:

```
pi@pivalter ~ $ ls -l /media/  
totale 4  
drwx----- 5 pi pi 4096 gen 1 1970 LACIE
```

Il nome della sotto-cartella dipende dalla label che avete assegnato al vostro disco USB. Digitando `ls -l /media/<nome cartella>` vedrete i file che sono memorizzati sulla vostra fedele chiavetta USB!

Un'altra differenza importante da sottolineare è l'utilizzo del carattere slash ("/") al posto del carattere backslash ("\") utilizzato dai sistemi Windows per dividere i diversi elementi di un percorso dentro il filesystem.

È possibile filtrare la ricerca indicando il nome di un singolo file o, più comunemente, una "maschera". Questa maschera può essere composta da una parte del nome del file e dai caratteri "wildcard" asterisco (*) indica zero o più caratteri) o punto di domanda (?) indica un carattere).

Per esempio il comando `ls *.png` mostrerà tutti i file con estensione

png.

Il comando `ls` prevede molte altre opzioni interessanti.
Lanciando `ls --help` verrà mostrato un messaggio l'aiuto in linea in italiano sul comando `ls`:

```
pi@pivalter ~ $ ls --help
Uso: ls [OPZIONE]... [FILE]...
Elenca informazioni sui FILE (predefinito: la directory corrente).
Ordina alfabeticamente le voci se non è usato uno di -cftuvSUX oppure --sort.

Gli argomenti obbligatori per le opzioni lunghe lo sono anche per quelle brevi.

-a, --all
-A, --almost-all
--author
-b, --escape
-B, --block-size=DIM
-B, --ignore-backups
-c
-C
--color[=QUANDO]
-d, --directory
-D, --dired
-f
-F, --classify
--file-type
--format=TIPO
--full-time
-g
--group-directories-first
-G, --no-group
-h, --human-readable
--si
-H, --dereference-command-line
--hide=MODELLO
--indicator-style=TIPO

non nasconde le voci che iniziano con .
non elenca le voci . e ..
con -l stampa l'autore di ogni file
stampa escape in stile C per i caratteri non grafici
scala le dimensioni di DIM prima di stamparle. Ad es.
"--block-size=M" stampa le dimensioni in unità di
1.048.576 byte. Consultare il formato di DIM in bas
non elenca le voci che terminano con ~
con -lt: mostra e ordina secondo il ctime (ora
di modifica delle informazioni di stato del file)
con -l: mostra il ctime e ordina secondo il nome
altrimenti: ordina secondo il ctime, prima il più r
elenca le voci per colonne
colora l'output. QUANDO è predefinito ad "always"
o può essere "never" o "auto".
Maggiori informazioni in basso
elenca le voci di directory invece del contenuto,
e non segue i collegamenti simbolici
genera un output adatto al modo dired di Emacs
non ordina, abilita -aU, disabilita -ls --color
accoda un indicatore alle voci (uno tra */=>@|)
similmente ma non accoda "**"
across -x, commas -m, horizontal -x, long -l,
single-column -1, verbose -l, vertical -C
come -l --time-style=full-iso
come -l, ma non elenca il proprietario
raggruppa le directory prima dei file.
Più efficace con un'opzione --sort, ma l'uso
di --sort=none (-U) disabilita il raggruppamento
con l'elenco lungo non stampa i nomi dei gruppi
con -l, stampa le dimensioni in formato leggibile
(es.: 1K 234M 2G)
similmente, ma usa multipli di 1000, non 1024
segue i collegamenti simbolici elencati sulla riga di
--dereference-command-line-symlink-to-dir
segue ciascun collegamento simbolico nella riga di co
che punta a una directory
non elenca le voci corrispondenti al MODELLO della sh
(annullato da -a o -A)
accoda ai nomi delle voci l'indicatore con lo stile
none (predefinito), slash (-p),
file-type (--file-type), classify (-F)
```

```

-i, --inode           stampa il numero d'indice di ogni file
-I, --ignore=MODELLO non elenca le voci corrispondenti al MODELLO della sh
-k                   come --block-size=1K
-l                   usa un formato di elenco lungo
-L, --dereference   quando mostra le informazioni su un collegamento,
                     simbolico, mostra le informazioni sul file a cui si
                     riferisce invece che sul collegamento stesso
-m                   elenca le voci separandole con virgole
-n, --numeric-uid-gid come -l, ma elenca gli id utente e di gruppo
-N, --literal         stampa i nomi grezzi (es.: non tratta in modo
                     speciale i caratteri di controllo)
-o                   come -l, ma non elenca le informazioni sul gruppo
-p, --indicator-style=slash appende / come indicatore alle directory
-q, --hide-control-chars stampa ? al posto dei caratteri non grafici
--show-control-chars  mostra i caratteri non grafici come sono (predefiniti
                     a meno che il programma sia «ls» e l'output un terminale
                     racchiude tra doppi apici i nomi delle voci
-Q, --quote-name    usa lo stile TIPO con i nomi delle voci:
--quoting-style=TIPO           literal, locale, shell, shell-always, c, escape
-r, --reverse        inverte il senso dell'ordinamento
-R, --recursive     elenca ricorsivamente le sottodirectory
-s, --size          stampa la dimensione allocata in blocchi di ogni file
-S                   ordina secondo le dimensioni dei file
--sort=TIPO         ordina per TIPO invece che per nome: none -U,
                     extension -X, size -S, time -t, version -v
--time=TIPO         con -l, usa il TIPO di orario invece che quello di
                     modifica: atime -u, access -u, use -u, ctime -c,
                     o status -c; se --sort=time usa l'orario specificato
                     chiave di ordinamento
--time-style=STILE  con -l mostra gli orari usando lo STILE specificato:
                     full-iso, long-iso, iso, locale, +FORMATO.
                     FORMATO è interpretato come da "date"; se è
                     FORMATO1<newline>FORMATO2, FORMATO1 è applicato
                     ai file non recenti e FORMATO2 a quelli recenti;
                     se STILE ha il prefisso "posix-" avrà effetto
                     solo fuori dal locale POSIX
-t                   ordina secondo l'orario di modifica, prima il più recente
-T, --tabsize=COL   assume che le tabulazioni siano ad ogni COL invece di
-u                   con -lt; ordina secondo l'orario di accesso e lo mostre
                     con -l mostra l'orario di accesso e ordina per nome
                     altrimenti ordina secondo l'orario di accesso
-U                   non ordina; elenca le voci nell'ordine della directory
-v                   ordina naturalmente secondo i numeri (di versione) ne
-w, --width=COL    considera lo schermo largo COL invece del valore attuale
-x                   elenca le voci per righe invece che per colonne
-X                   ordina alfabeticamente secondo le estensioni
-Z, --context      stampa il contesto di sicurezza SELinux di ogni file
-1                   elenca un file per riga
--help              mostra questo aiuto ed esce
--version           stampa le informazioni sulla versione ed esce

```

DIM può essere uno dei seguenti (o optionalmente un intero seguito da):
KB 1000, K 1024, MB 1000*1000, M 1024*1024, e così via per G, T, P, E, Z e Y.

L'uso dei colori per distinguere i tipi di file è disabilitato sia in modo predefinito con --color=never. Con --color=auto, ls emette i codici di colore solo se lo

standard output è connesso a un terminale. La variabile di ambiente LS_COLORS può cambiare queste impostazioni. Usare il comando dircolors per impostarla.

Stato di uscita:

- 0 se OK,
- 1 per problemi minori (es.: impossibile accedere alla sottodirectory),
- 2 per problemi seri (es.: impossibile accedere all'argomento della riga di coma)

Segnalare i bug di ls a <bug-coreutils@gnu.org>

Sito web di GNU coreutils: <<http://www.gnu.org/software/coreutils/>>

Aiuto sull'uso del software GNU in generale: <<http://www.gnu.org/gethelp/>>

Segnalare i bug di traduzione di ls a <tp@lists.linux.it>

Per la documentazione completa, eseguire: info coreutils "ls invocation"

Una documentazione ancora più esaustiva (ma in inglese) può essere consultata utilizzando il comando `info`.

Potrete scoprire tutti i segreti di ls digitando al prompt:

```
pi@pivalter ~ $ info coreutils "ls invocation"
```

Per uscire dalla visualizzazione dell'help basta premere Il tasto **Q**. I collegamenti tra le voci dell'help sono prefissati con un asterisco; per seguire il collegamento basta posizionarsi con il cursore sulla riga corrispondente e premere **Invio**. Il tasto **Backspace** ci riporterà alla pagina precedente, ripercorrendo il link a ritroso.

Un'altra possibilità per scoprire qualcosa in più sui comandi del sistema è utilizzare il comando `man`. Anche `man` mostrerà della documentazione, normalmente in lingua inglese, sul comando passatogli come argomento.

Per esempio:

```
pi@pivalter ~ $ man ls
```

mostrera la descrizione dei vari parametri del comando `ls`.

Provate a utilizzare le funzioni di help per scoprire come ordinare in ordine alfabetico la nostra lista di file.

Il filesystem

Il comando `ls` ci permette di scoprire vita, morte e miracoli di file e cartelle, ma non ci consente di spostarci all'interno del filesystem.

Per spostarsi all'interno del filesystem è possibile utilizzare il comando `cd` (change directory).

Il comando `cd` cambia la directory corrente. Il comportamento di diversi comandi cambia a seconda della directory corrente.

Se digitiamo:

```
pi@pivalter ~ $ cd /
pi@pivalter / $ ls
bin dev home lost+found mnt proc run selinux sys usr
boot etc lib media opt root sbin srv tmp var
pi@pivalter / $
```

I’elenco dei file che ci viene mostrato dal comando `ls` è diverso da quello visto all’inizio del paragrafo precedente. Questo perché, non specificando alcun percorso, il comando `ls` agisce sulla cartella corrente e il comando `cd /` ha cambiato la cartella corrente dalla cartella `home` del nostro utente alla `root`.

Questo cambiamento è visibile anche nel prompt di sistema.

Per tornare alla `home` possiamo digitare `cd` senza alcun parametro.

```
pi@pivalter / $ cd
pi@pivalter ~ $
```

Esistono due modi diversi di referenziare un percorso all’interno del filesystem. La prima modalità consiste nell’indicare il percorso completo a partire dalla `root`. Per esempio, possiamo referenziare il file `dummy.png` nella cartella `home` del nostro utente come `/home/pi/dummy.png`. Questo indirizzamento viene chiamato “path assoluto”. I path assoluti iniziano sempre con il carattere slash, per indicare la `root`.

È anche possibile indicare un percorso parziale. Per esempio, se già ci troviamo nella cartella `home`, lo stesso file può essere referenziato semplicemente come `dummy.png`. Questa modalità di indirizzamento viene definita “path relativo”. Si tratta infatti di un percorso relativo alla directory corrente.

Utilizzare i path è analogo a fornire indicazioni stradali.

In qualche caso può essere comodo riportarsi a un punto noto, per esempio “Piazza Garibaldi” (nel nostro caso la `root` è la “Piazza Garibaldi” del filesystem!) e in altri è meglio partire dalla nostra posizione corrente e fornire il percorso fino alla destinazione.

Se eseguiamo il comando `ls -a` l’elenco delle cartelle inizia invariabilmente con due cartelle nascoste chiamate `.` e `..`; non si tratta di codice Morse, ma di due path relativi:

```
pi@pivalter / $ ls -a
. bin dev home lost+found mnt proc run selinux sys usr
.. boot etc lib media opt root sbin srv tmp var
```

La cartella `.` indica la directory corrente e ci consente di referenziare i file e le sottocartelle con path relativi che iniziano con `./`; vedremo come questo sia importante per lanciare programmi dalla cartella corrente tra qualche paragrafo. La cartella `..` ci consente di indicare la cartella che contiene la cartella corrente. Questo apparente non-sense permette di referenziare percorsi relativi anche al di sopra della cartella di lavoro. Per esempio, quando siamo posizionati nella nostra cartella `home` (`/home/pi`) e digitiamo `ls ..`:

```
pi@pivalter ~ $ ls ..  
pi
```

verrà mostrato il contenuto della cartella `home`.

È anche possibile concatenare più riferimenti all'indietro, fino ad arrivare alla cartella root:

```
pi@pivalter ~ $ ls ../../  
bin dev home lost+found mnt proc run selinux sys usr  
boot etc lib media opt root sbin srv tmp var
```

Dopo tutto questo navigare nel filesystem può essere utile scoprire dove siamo finiti! Per conoscere la directory corrente possiamo utilizzare il comando `pwd` (Print Working Directory):

```
pi@pivalter ~ $ pwd  
/home/pi
```

Il comando `pwd` restituisce sempre un path assoluto (le famose indicazioni a partire da Piazza Garibaldi!).

Per sperimentare con i file e il filesystem utilizzeremo una sottocartella nella nostra `home`, in modo da non rischiare di danneggiare accidentalmente qualche file importante.

Per creare una nuova cartella è possibile utilizzare il comando `mkdir` (MaKe DIRectory). Prima di lanciare `mkdir` dobbiamo accertarci di essere effettivamente nella nostra cartella `home` dando un'occhiata al prompt. Se la cartella `home` non è la cartella attiva possiamo tornarci digitando `cd`, senza alcun argomento.

Digitando:

```
pi@pivalter / $ mkdir dummy  
pi@pivalter ~ $ cd dummy  
pi@pivalter ~/dummy $
```

avrete creato una nuova cartella `dummy` (è il classico nome per file e

cartelle di esempio) e l'avrete resa directory corrente.

Per sperimentare con i file dovremo creare un file “cavia”.

Con il comando `touch` possiamo creare un file vuoto nella cartella corrente.

```
pi@pivalter ~/dummy $ touch file1
```

Il comando `ls` ci mostrerà il nostro nuovo file:

```
pi@pivalter ~/dummy $ ls  
file1
```

I file possono essere copiati utilizzando il comando `cp` (CoPy). Questo comando accetta sia percorsi assoluti sia percorsi relativi.

Il breve esempio qui di seguito mostra il contenuto di una cartella prima e dopo aver eseguito il comando `cp`.

```
pi@pivalter ~/dummy $ ls  
file1  
pi@pivalter ~/dummy $ cp file1 file2  
pi@pivalter ~/dummy $ ls  
file1 file2
```

Una volta copiato, il file è indipendente dal file originale e potrà essere modificato indipendentemente da esso. A volte può essere utile creare un collegamento allo stesso file in una posizione diversa del filesystem. Questo consentirà di poter modificare i dati del file originale utilizzando indifferentemente il file vero e proprio o il link appena creato.

Per creare un link è possibile utilizzare il comando `ln` (LiNk) specificando il parametro `-s` per indicare un link simbolico. I link simbolici sono semplici riferimenti e possono funzionare anche tra rami del filesystem che fisicamente sono memorizzati su memorie differenti.

```
pi@pivalter ~/dummy $ ln -s file2 file3  
pi@pivalter ~/dummy $ ls  
file1 file2 file3
```

Anche in questo caso l'esecuzione del comando ha fatto apparire un nuovo file nella cartella ma, se state utilizzando una console a colori, avrete notato che questo file è diverso dai precedenti.

Utilizzando il comando `ls -l` il mistero viene svelato:

```
pi@pivalter ~/dummy $ ls -l  
totale 0  
-rw-r--r-- 1 pi pi 0 ago 12 18:07 file1  
-rw-r--r-- 1 pi pi 0 ago 12 18:07 file2
```

```
lrwxrwxrwx 1 pi pi 5 ago 12 18:13 file3 -> file2
```

Il primo carattere `l` indica che `file3` non è un normale file ma un link. E il percorso del file originale viene mostrato dopo il nome del link.

In altri casi può essere utile spostare un file da una cartella a un'altra o semplicemente rinominarlo. Entrambe le operazioni possono essere effettuate utilizzando il comando `mv` (MoVe):

```
pi@pivalter ~/dummy $ mv file2 file4  
pi@pivalter ~/dummy $ ls  
file1 file3 file4
```

Notate che, anche dopo aver rinominato `file2`, il link `file3` rimane presente nel file-system:

```
pi@pivalter ~/dummy $ ls -la  
totale 8  
drwxr-xr-x 2 pi pi 4096 ago 12 18:19 .  
drwxr-xr-x 26 pi pi 4096 ago 12 16:48 ..  
-rw-r--r-- 1 pi pi 0 ago 12 18:07 file1  
lrwxrwxrwx 1 pi pi 5 ago 12 18:13 file3 -> file2  
-rw-r--r-- 1 pi pi 0 ago 12 18:07 file4
```

Rinominare o rimuovere un file non causa l'aggiornamento dei link che lo referenziano. Il comando `mv` consente anche di spostare o rinominare le sotto-cartelle.

Possiamo anche cancellare dei file usando il comando `rm` (ReMove):

```
pi@pivalter ~/dummy $ touch file5  
pi@pivalter ~/dummy $ ls  
file1 file3 file4 file5  
pi@pivalter ~/dummy $ rm file5  
pi@pivalter ~/dummy $ ls  
file1 file3 file4
```

Il comando `rm` supporta anche la possibilità di cancellare ricorsivamente file e sottocartelle utilizzando il parametro `-r`. Quindi `rm -r *` cancellerà tutti i file della cartella corrente.

Per ora non è il caso di fare questo esperimento!

Per cancellare una cartella è possibile usare il comando `rmdir` (ReMove DIRectory), ma per poterlo fare è necessario che la cartella sia completamente vuota.

Se vi fate prendere la mano creando e cancellando file potrebbe essere utile tenere sotto controllo lo spazio libero sul nostro sistema. Il comando `df` (Disk Free) mostrerà queste informazioni:

```
pi@pivalter ~/dummy $ df
```

File system	1K-blocchi	Usati	Disponib.	Uso%	Montato su
rootfs	2595624	1597136	866552	65%	/
/dev/root	2595624	1597136	866552	65%	/
devtmpfs	216132	0	216132	0%	/dev
tmpfs	44880	248	44632	1%	/run
tmpfs	5120	0	5120	0%	/run/lock
tmpfs	89740	0	89740	0%	/run/shm
/dev/mmcblk0p5	57288	18888	38400	33%	/boot

Il comando `df` mostra lo spazio libero e occupato sulle diverse memorie di massa del sistema. Come abbiamo visto qualche paragrafo addietro, il filesystem ha una root unica e le diverse memorie (o partizioni) sono montate come sotto-cartelle nel filesystem.

Il comando `du` (Disk Usage) mostra invece lo spazio occupato da file e cartelle nella directory corrente, fornendo anche il dettaglio per ogni sotto-cartella e i relativi contenuti. Il parametro `-h` propone la dimensione dei file in byte e multipli (Kilobyte, Megabyte, Gigabyte).

```
pi@pivalter ~ $ du -h
8,0K ./indiecity/pistore/users
92K ./indiecity/pistore
104K ./indiecity
180K ./screenshots
4,0K ./gconf
76K ./fontconfig
12K ./dillo
4,0K ./gvfs
```

Magie da tastiera

Arrivati a questo punto vi sarete probabilmente stancati di scrivere i nomi dei comandi e i path dei file, senza contare il tempo perso per qualche errore di battitura. La command line viene in vostro soccorso con l'auto completamento. Il tasto **Tab** può essere utilizzato per completare il comando che state scrivendo. Se il completamento è univoco (non esistono altri comandi che iniziano con i caratteri che avete già scritto) il comando verrà completato immediatamente. Se esistono più alternative verrà completata solo una parte del testo e una seconda pressione del tasto **Tab** mostrerà le possibili alternative.

Per esempio scrivendo sulla command line:

```
pi@pivalter ~/dummy $ ca
```

e premendo due volte **Tab** verranno mostrati tutti i comandi che iniziano con "ca":

```
cal      caller  captoinfo cat      catman
calendar cancel  case    catchsegv
```

Se il numero di possibili alternative è molto alto il sistema chiederà una conferma prima di mostrarle tutte. L'auto completamento funziona sia per i comandi sia per i loro argomenti ed è un ottimo sistema per aumentare la vostra produttività e ridurre gli errori. Potete inoltre scorrere la lista degli ultimi comandi digitati usando i tasti cursore verso l'alto e verso il basso.

Nano, nano!

Per modificare i nostri file di testo utilizzando l'interfaccia grafica, come abbiamo visto nel capitolo precedente, possiamo utilizzare Leafpad.

Ci sono però occasioni in cui modificare i file direttamente dalla command line può tornare molto comodo: per esempio, quando siamo collegati da remoto e l'unica interfaccia disponibile è quella testuale. Per modificare i file dalla command line possiamo utilizzare `nano`. Lanciadolo senza parametri sarà possibile creare un nuovo file:

```
pi@pivalter ~ $ nano
```

Per modificare un file esistente basta passarne il nome come argomento sulla command line:

```
GNU nano 2.2.6          Nuovo buffer
^G Guida   ^O Salva     ^R Inserisci ^Y Pag Prec. ^K Taglia   ^C Posizione
^X Esci    ^J Giustifica^W Cerca     ^V Pag Succ. ^U Incolla  ^T Ortografia
```

I comandi principali di `nano` sono elencati nella parte bassa dello schermo e sono accessibili premendo il tasto **Ctrl** e la lettera evidenziata alla sinistra del comando.

Per esempio **Ctrl+G** aprirà la guida.

Una volta scritto il vostro capolavoro potrete salvarlo premendo **Ctrl+O**. Nano vi chiederà un nome per il vostro file (sia che si tratti di un nuovo file sia che abbiate aperto un file esistente per modificarlo):

```
GNU nano 2.2.6          Nuovo buffer          Modificato
Era una notte buia e tempestosa...
Nome del file in cui salvare: snoopy.txt
^G Guida   M-D Formato DOS   M-A Accoda   M-B File di backup
^C Annulla M-M Formato Mac   M-P Scrivi in testa
```

La sequenza di tasti per uscire dall'editor è **Ctrl+X**.

Ora che abbiamo creato un nuovo file non vuoto possiamo provare a stamparne il contenuto sulla console. Per farlo possiamo utilizzare il comando `cat` (CATenate):

```
pi@pivalter ~ $ cat snoopy.txt  
Era una notte buia e tempestosa...
```

Se volete visualizzare un file con la libertà di scorrerne il contenuto e di ricercare al suo interno, ma senza rischiare di modificarlo accidentalmente, potete utilizzare il comando `less`.

Nella schermata di `less` è possibile scorrere il testo con i **tasti cursore** o **pagina su** e **pagina giù**, e cercare stringhe di testo digitando uno **slash** (/) prima della stringa da ricercare. L'elenco completo dei comandi di `less` si può visualizzare digitando `h`. Per terminare il programma o uscire dalla schermata di help è sufficiente premere **Q**.

In alcuni casi può essere utile stampare soltanto le prime o le ultime righe di un file. I comandi `head` (testa) e `tail` (coda) per default stampano solo le prime o le ultime 10 righe di un file di testo. Il parametro `-n` consente di cambiare questo valore stampando più o meno linee.

Utenti e diritti

Unix e tutti i suoi derivati sono sistemi multi-utente. Più utenti possono accedere al sistema, con diversi diritti di accesso. Abbiamo definito il nostro Raspberry Pi un sistema “personale” e questo potrebbe farci pensare che si tratti di un sistema monoutente. In realtà la maggior parte delle applicazioni che lanceremo dall’interfaccia grafica o dalla command line verrà effettivamente eseguita per conto dell’utente “pi”, che è quello di default creato da Raspbian. Molti altri servizi di sistema, invece, vengono eseguiti utilizzando account utente diversi.

Questo accorgimento serve a evitare che problemi o bug di sicurezza in uno dei servizi (il server FTP, per esempio) possano consentire l’accesso non autorizzato ad altre funzionalità del sistema.

Per vedere l’elenco degli utenti correntemente attivi sul vostro sistema potete utilizzare il comando `cat` e visualizzare il file `/etc/passwd`:

```
pi@pivalter ~ $ cat /etc/passwd  
root:x:0:0:root:/root:/bin/bash  
daemon:x:1:1:daemon:/usr/sbin:/bin/sh  
bin:x:2:2:bin:/bin:/bin/sh
```

```
sys:x:3:3:sys:/dev/bin/sh
sync:x:4:65534:sync:/bin/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
uucp:x:10:10:uucp:/var/spool/uucp:/bin/sh
proxy:x:13:13:proxy:/bin:/bin/sh
www-data:x:33:33:www-data:/var/www:/bin/sh
backup:x:34:34:backup:/var/backups:/bin/sh
list:x:38:38:Mailing List Manager:/var/list:/bin/sh
irc:x:39:39:ircd:/var/run/ircd:/bin/sh
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/bin/sh
nobody:x:65534:65534:nobody:/nonexistent:/bin/sh
libuuid:x:100:101:/var/lib/libuuid:/bin/sh
pi:x:1000:1000:,:/home/pi:/bin/bash
sshd:x:101:65534::/var/run/sshd:/usr/sbin/nologin
ntp:x:102:104::/home/ntp:/bin/false
statd:x:103:65534::/var/lib/nfs:/bin/false
messagebus:x:104:106::/var/run/dbus:/bin/false
usbmux:x:105:46:usbmux daemon,,,:/home/usbmux:/bin/false
lightdm:x:106:109:Light Display Manager:/var/lib/lightdm:/bin/false
```

Come potete vedere da questo elenco, il nostro sistema è parecchio affollato!

In realtà nel corso dei vari capitoli utilizzeremo principalmente due utenti: l'utente "pi" e l'utente "root". L'utente root, chiamato anche "super-user", è l'utente che ha i diritti per modificare la configurazione del sistema o consentire ad altri utenti di farlo.

Il file `/etc/passwd` riporta, separati dal simbolo due punti (:), il nome utente, uno user-id numerico, l'id (sempre numerico) del gruppo a cui l'utente appartiene, un nome descrittivo e il path della cartella home dell'utente in oggetto. Il carattere `x` tra il nome utente e il suo id numerico era in passato adoperato per memorizzare una chiave utilizzata per validare la password. Ora le password sono memorizzate in modo più sicuro. Gestire i diritti per ogni singolo utente è complesso. Per semplificare la gestione gli utenti possono essere raggruppati in gruppi e i diritti di accesso essere assegnati a uno specifico gruppo anziché a ogni singolo utente. A un gruppo corrisponde spesso il diritto di utilizzare uno specifico servizio o una specifica funzionalità di sistema. Ed è per questo motivo che anche la lista dei gruppi presenti sul nostro piccolo sistema è abbastanza lunga.

Anche in questo caso possiamo consultarla stampando sulla console il file di testo `/etc/group`:

```
pi@pivalter ~ $ cat /etc/group
```

```
root:x:0:  
daemon:x:1:  
bin:x:2:  
sys:x:3:  
adm:x:4:pi  
tty:x:5:  
disk:x:6:  
lp:x:7:  
mail:x:8:  
news:x:9:  
uucp:x:10:  
man:x:12:  
proxy:x:13:  
kmem:x:15:  
dialout:x:20:pi  
fax:x:21:  
voice:x:22:  
cdrom:x:24:pi  
floppy:x:25:  
tape:x:26:  
sudo:x:27:pi  
audio:x:29:pi  
dip:x:30:  
www-data:x:33:  
backup:x:34:  
operator:x:37:  
list:x:38:  
irc:x:39:  
src:x:40:  
gnats:x:41:  
shadow:x:42:  
utmp:x:43:  
video:x:44:pi  
sasl:x:45:  
plugdev:x:46:pi  
staff:x:50:  
games:x:60:pi  
users:x:100:pi  
nogroup:x:65534:  
libuuid:x:101:  
crontab:x:102:  
pi:x:1000:  
ssh:x:103:  
ntp:x:104:  
netdev:x:105:pi  
input:x:999:pi  
messagebus:x:106:  
lpadmin:x:107:  
fuse:x:108:  
lightdm:x:109:  
indiecity:x:1001:root,pi
```

Ogni gruppo è identificato con un nome e un id numerico, in modo analogo a quanto avviene per gli utenti. La lista degli utenti che appartengono a un determinato gruppo è riportata separando tra loro i

nomi utente con una virgola.

Come potete vedere l'utente "pi" è presente in diversi gruppi. Questo ci consentirà di utilizzare molte delle funzioni di sistema senza doverci preoccupare di effettuare un login aggiuntivo.

Alcune delle operazioni più critiche e potenzialmente pericolose per la stabilità del sistema sono accessibili solo per l'utente "root". Sulla maggior parte dei sistemi Linux l'utente "root" non può effettuare il login direttamente in una console interattiva. Questo ci impedisce di loggarci come super-user e modificare a nostro piacimento il sistema, ma anche di effettuare operazioni potenzialmente utili, come riconfigurare il layout di tastiera. Per permetterci di eseguire specifici comandi con i permessi da super-user viene in nostro aiuto il comando `sudo` (Super User DO). Gli utenti che fanno parte del gruppo sudo possono invocare questo comando e attribuirsi temporaneamente i diritti di root. Questo approccio limita l'utilizzo interattivo del super-user.

Per esempio, se proviamo a eseguire l'applicazione di configurazione di Raspbian **raspi-config** come utente normale avremo questo risultato:

```
pi@pivalter ~ $ raspi-config  
Script must be run as root. Try 'sudo raspi-config'
```

Eseguendo, come suggerito dal messaggio di errore, `sudo raspi-config` avremo invece accesso alla configurazione.

Nei vari esempi che vedremo in questo libro utilizzeremo il comando `sudo` in diverse occasioni.

In alcune circostanze può essere preferibile utilizzare la modalità super-user per dare anche a utenti "normali", singoli o in un gruppo, i diritti di accedere a uno specifico file o a una specifica risorsa.

Il comando `chown` (CHange OWNer) consente di cambiare l'utente "proprietario" di un determinato file. Tornando nella nostra cartella di prova (`/home/pi/dummy`) e usando il comando:

```
pi@pivalter ~/dummy $ ls -la  
totale 12  
drwxr-xr-x 2 pi pi 4096 ago 12 20:13 .  
drwxr-xr-x 26 pi pi 4096 ago 12 21:17 ..  
-rw-r--r-- 1 pi pi 0 ago 12 18:07 file1  
-rw-r--r-- 1 pi pi 0 ago 12 19:48 file2  
lrwxrwxrwx 1 pi pi 5 ago 12 18:13 file3 -> file2  
-rw-r--r-- 1 pi pi 0 ago 12 18:07 file4  
-rw-r--r-- 1 pi pi 35 ago 12 20:13 snoopy.txt
```

possiamo notare che tutti i file sono "proprietà" dell'utente "pi".

Questo è normale poiché per default un file è assegnato a chi lo ha creato. Per cambiare la proprietà di uno dei file è possibile utilizzare, da super-user, il comando `chown`:

```
pi@pivalter ~/dummy $ sudo chown nobody file1
pi@pivalter ~/dummy $ ls -la
totale 12
drwxr-xr-x  2 pi      pi  4096 ago 12 20:13 .
drwxr-xr-x 26 pi      pi  4096 ago 12 21:17 ..
-rw-r--r--  1 nobody pi      0 ago 12 18:07 file1
-rw-r--r--  1 pi      pi      0 ago 12 19:48 file2
lrwxrwxrwx  1 pi      pi      5 ago 12 18:13 file3 -> file2
-rw-r--r--  1 pi      pi      0 ago 12 18:07 file4
-rw-r--r--  1 pi      pi     35 ago 12 20:13 snoopy.txt
```

Ora il `file1` è proprietà dell'utente `nobody`.

Con il comando `chown` è possibile anche assegnare un gruppo diverso. L'alternativa più semplice (e più facile da memorizzare) è il comando `chgrp` (CHange GRouP). Anche questo comando deve essere lanciato come super-user.

```
pi@pivalter ~/dummy $ sudo chgrp staff file1
pi@pivalter ~/dummy $ ls -la
totale 12
drwxr-xr-x  2 pi      pi  4096 ago 12 20:13 .
drwxr-xr-x 26 pi      pi  4096 ago 12 21:17 ..
-rw-r--r--  1 nobody staff  0 ago 12 18:07 file1
-rw-r--r--  1 pi      pi      0 ago 12 19:48 file2
lrwxrwxrwx  1 pi      pi      5 ago 12 18:13 file3 -> file2
-rw-r--r--  1 pi      pi      0 ago 12 18:07 file4
-rw-r--r--  1 pi      pi     35 ago 12 20:13 snoopy.txt
```

Perché è importante assegnare la proprietà (in inglese ownership) dei diversi file a utenti e gruppi? Quando all'inizio di questo capitolo abbiamo presentato le varie opzioni del comando `ls` avrete notato come l'autore, dopo una rapida spiegazione, abbia glissato elegantemente sul significato dei diritti di accesso. Ogni file ha diritti di accesso (Read, Write ed eXecute) specifici per l'utente proprietario, per il gruppo e per tutto il resto degli utenti di sistema. Assegnare quindi un file a uno specifico utente o gruppo può consentirci di leggerlo, modificarlo o eseguirlo, cosa che potrebbe essere impossibile se il file non fosse di nostra proprietà direttamente o tramite uno dei gruppi a cui appartiene anche il nostro utente.

Per cambiare i diritti di accesso a un file è possibile utilizzare il comando `chmod` (CHange MODe).

La sintassi di `chmod` è leggermente più complessa di quella dei

comandi visti fino a ora, ma con un po' di pazienza è possibile impararne almeno i fondamentali.

I diritti più comuni sono Read, Write ed eXecute, abbreviati con le lettere `r`, `w` e `x`. Per abilitare un diritto si utilizza il carattere più (+), per toglierlo il meno (-). Quindi, per esempio, `+w` abiliterà l'accesso in scrittura e `-r` negherà quello in lettura.

I diritti sono specifici per utente (`u` - User), gruppo (`g` - Group) o altri utenti (`o` - Other), oppure generici per tutti (`a` - All) e questo dovrà essere prefissato al comando. `u+x`, per esempio, abiliterà l'utente all'esecuzione del file. Se non si specifica il prefisso il comando verrà considerato valido per tutti gli utenti.

Provando sui nostri file di esempio:

```
pi@pivalter ~/dummy $ ls -la
totale 12
drwxr-xr-x  2 pi      pi      4096 ago 12 20:13 .
drwxr-xr-x 26 pi      pi      4096 ago 12 21:17 ..
-rw-r--r--  1 nobody staff    0 ago 12 18:07 file1
-rw-r--r--  1 pi      pi      0 ago 12 19:48 file2
lrwxrwxrwx  1 pi      pi      5 ago 12 18:13 file3 -> file2
-rw-r--r--  1 pi      pi      0 ago 12 18:07 file4
-rw-r--r--  1 pi      pi     35 ago 12 20:13 snoopy.txt
pi@pivalter ~/dummy $ chmod u-w file4
pi@pivalter ~/dummy $ chmod a+w file2
pi@pivalter ~/dummy $ ls -la
totale 12
drwxr-xr-x  2 pi      pi      4096 ago 12 20:13 .
drwxr-xr-x 26 pi      pi      4096 ago 12 21:17 ..
-rw-r--r--  1 nobody staff    0 ago 12 18:07 file1
-rw-rw-rw-  1 pi      pi      0 ago 12 19:48 file2
lrwxrwxrwx  1 pi      pi      5 ago 12 18:13 file3 -> file2
-rw-r--r--  1 pi      pi      0 ago 12 18:07 file4
-rw-r--r--  1 pi      pi     35 ago 12 20:13 snoopy.txt
```

In questo caso non abbiamo dovuto eseguire `chmod` come super-user perché i file erano di proprietà dell'utente "pi". In altri casi sarà necessario ricorrere a `sudo` per cambiare i diritti.

Come al solito lanciando `chmod --help` o `man chmod` potrete scoprire tutti i parametri e le opzioni di questo comando.

Se tutto questo parlare di utenti, diritti di accesso, sicurezza vi ha fatto venire il dubbio che la password che avete scelto per il vostro utente sia poco sicura, potete cambiarla utilizzando il comando `passwd` (PASSWorD).

```
pi@pivalter ~/dummy $ passwd
Cambio password per pi.
Password UNIX (corrente):
```

```
Immettere nuova password UNIX:  
Reimmettere la nuova password UNIX:  
passwd: password aggiornata correttamente
```

Come vedete il comando `passwd` per prima cosa richiede la password corrente, in modo da evitare che qualche burlone, trovando il vostro terminale incustodito, possa decidere di cambiarvi la password!

A volte aggiungere il nostro utente a un determinato gruppo è il sistema più rapido per garantirgli i diritti di accesso a una determinata funzionalità. Modificare direttamente il file `/etc/group` non è però una buona idea. Un errore di sintassi potrebbe renderlo non valido e bloccare il funzionamento di qualche componente del sistema. Per aggiungere un utente a uno specifico gruppo possiamo utilizzare il comando `adduser` passando il nome dell'utente e il nome del gruppo a cui aggiungerlo.

Se, per esempio, vogliamo consentire all'utente `pi` di accedere al floppy disk, possiamo aggiungerlo al gruppo `floppy`:

```
pi@pivalter ~/dummy $ sudo adduser pi floppy  
Aggiunta dell'utente «pi» al gruppo «floppy» ...  
Aggiunta dell'utente pi al gruppo floppy  
Fatto.
```

Ora dobbiamo solo andare in soffitta a cercare un floppy disk!

Il comando `adduser` può essere utilizzato anche per creare nuovi utenti e nuovi gruppi.

Se volete scoprire come farlo l'opzione `--help` vi darà tutte le informazioni necessarie.

```
pi@pivalter ~/dummy $ adduser --help  
adduser [--home DIR] [--shell SHELL] [--no-create-home] [--uid ID]  
[--firstuid ID] [--lastuid ID] [--gecos GECOS] [--ingroup GRUPPO | --gid ID]  
[--disabled-password] [--disabled-login] UTENTE  
    Aggiunge un utente normale  
  
adduser --system [--home DIR] [--shell SHELL] [--no-create-home] [--uid ID]  
[--gecos GECOS] [--group | --ingroup GRUPPO | --gid ID] [--disabled-password]  
[--disabled-login] UTENTE  
    Aggiunge un utente di sistema  
  
adduser --group [--gid ID] GRUPPO  
addgroup [--gid ID] GRUPPO  
    Aggiunge un gruppo  
  
addgroup --system [--gid ID] GRUPPO  
    Aggiunge un gruppo di sistema  
  
adduser UTENTE GRUPPO
```

Aggiunge un utente esistente a un gruppo esistente

opzioni generali:

--quiet -q	non mostra le informazioni sul processo sullo stdout
--force-badname	permette l'uso di nomi utente che non verificano la variabile di configurazione NAME_REGEX
--help -h	informazioni sull'uso del programma
--version -v	versione e copyright
--conf -c FILE	usa FILE come file di configurazione

Processi

In Linux con il termine “processo” si indica un programma in esecuzione. Tutti comandi che abbiamo utilizzato in queste pagine, e anche le applicazioni che abbiamo lanciato attraverso l’interfaccia grafica, lanciano nuovi processi. Per vedere l’elenco dei processi in esecuzione possiamo utilizzare il comando `ps` (Process Status).

```
pi@pivalter ~/dummy $ ps
 PID TTY          TIME CMD
2269 pts/0    00:00:03 bash
2602 pts/0    00:00:00 ps
```

Ogni processo è identificato da un codice numerico ed è associato a una console (TTY). Il comando `ps` lanciato senza parametri mostra i processi in esecuzione per la console corrente. In questo caso abbiamo la shell bash (l’applicazione che interpreta ed esegue i comandi) e lo stesso comando `ps` che è appena stato eseguito (ed è terminato prima che il sistema tornasse al prompt). Se proviamo a rilanciare di nuovo il comando `ps`:

```
pi@pivalter ~/dummy $ ps
 PID TTY          TIME CMD
2269 pts/0    00:00:03 bash
2603 pts/0    00:00:00 ps
```

I’output sarà simile all’esecuzione precedente a eccezione dell’identificativo numerico del processo (`PID` - Process ID) che sarà cambiato. Il sistema assegna numeri progressivi agli id di processo in modo che esecuzioni consecutive dello stesso comando possano comunque essere identificate come operazioni separate.

Vi aspettavate qualche processo in più in esecuzione sul nostro Raspberry Pi?

Lanciando il comando `ps` con l’opzione `ax` potrete avere un elenco completo.

```

pi@pivalter ~/dummy $ ps ax
  PID TTY      STAT   TIME  COMMAND
    1 ?        Ss     0:01  init [2]
    2 ?        S      0:00  [kthreadd]
    3 ?        S      0:00  [ksoftirqd/0]
    5 ?        S<    0:00  [kworker/0:0H]
    6 ?        S      0:00  [kworker/u:0]
    7 ?        S<    0:00  [kworker/u:0H]
    8 ?        S<    0:00  [khelper]
    9 ?        S      0:00  [kdevtmpfs]
   10 ?       S<    0:00  [netns]
   12 ?       S      0:00  [bdi-default]
   13 ?       S<    0:00  [kblockd]
   14 ?       S      0:00  [khubd]
   15 ?       S<    0:00  [rpciod]
   16 ?       S      0:00  [khungtaskd]
   17 ?       S      0:00  [kswapd0]
   18 ?       S      0:00  [fsnotify_mark]
   19 ?       S<    0:00  [nfsiod]
   20 ?       S<    0:00  [crypto]
   27 ?       S<    0:00  [kthrotld]
   28 ?       S<    0:00  [VCHIQ-0]
   29 ?       S<    0:00  [VCHIQr-0]
   30 ?       S<    0:00  [VCHIQs-0]
   31 ?       S<    0:00  [iscsi_eh]
   32 ?       S<    0:00  [dwc_otg]
   33 ?       S<    0:00  [DWC Notificatio]
   35 ?       S<    0:00  [deferwq]
   36 ?       S      0:01  [mmcqd/0]
   37 ?       S      0:00  [kworker/u:2]
   39 ?       S      0:00  [jbd2/mmcblk0p6-]
   40 ?       S<    0:00  [ext4-dio-unwrit]
  155 ?      Ss    0:00  udevd --daemon
  647 ?      S      0:00  udevd --daemon
  651 ?      S      0:00  udevd --daemon
 1550 ?      S      0:03  /usr/sbin/ifplugd -i eth0 -q -f -u0 -d10 -w -I
 1585 ?      S      0:00  /usr/sbin/ifplugd -i lo -q -f -u0 -d10 -w -I
 1708 ?      S      0:00  [kworker/0:2]
 1876 ?      S1    0:00  /usr/sbin/rsyslogd -c5
 1904 ?      Ss    0:00  /usr/sbin/thd --daemon --triggers /etc/triggerhappy/t
 1945 ?      Ss    0:00  /usr/sbin/cron
 1951 ?      Ss    0:00  /usr/bin/dbus-daemon --system
 2009 ?      S1    0:00  /usr/sbin/lightdm
 2039 ?      Ss    0:00  /usr/sbin/ntp -p /var/run/ntp.pid -g -u 102:104
 2045 tty7   Ss+   0:05  /usr/bin/X :0 -auth /var/run/lightdm/root/:0 -nolisten
 2077 ?      Ss    0:00  /usr/sbin/sshd
 2108 tty1   Ss+   0:00  /sbin/getty --noclear 38400 tty1
 2109 tty2   Ss+   0:00  /sbin/getty 38400 tty2
 2110 tty3   Ss+   0:00  /sbin/getty 38400 tty3
 2111 tty4   Ss+   0:00  /sbin/getty 38400 tty4
 2112 tty5   Ss+   0:00  /sbin/getty 38400 tty5
 2113 tty6   Ss+   0:00  /sbin/getty 38400 tty6
 2114 ?      Ss+   0:00  /sbin/getty -L ttyAMA0 115200 vt100
 2117 ?      S1    0:00  lightdm --session-child 12 15
 2120 ?      S1    0:00  /usr/sbin/console-kit-daemon --no-daemon
 2187 ?      S1    0:00  /usr/lib/policykit-1/polkitd --no-debug
 2199 ?      Ssl   0:00  /usr/bin/lxsession -s LXDE -e LXDE

```

```

2220 ? Ss 0:00 /usr/bin/ssh-agent /usr/bin/dbus-launch --exit-with-s
2223 ? S 0:00 /usr/bin/dbus-launch --exit-with-session x-session-ma
2224 ? Ss 0:00 /usr/bin/dbus-daemon --fork --print-pid 5 --print-add
2230 ? S 0:00 openbox --config-file /home/pi/.config/openbox/lxde-r
2232 ? S 0:12 lxpanel --profile LXDE
2234 ? S 0:01 pcmanfm --desktop --profile LXDE
2239 ? S1 0:00 /usr/lib/arm-linux-gnueabihf/lxpolkit
2242 ? S 0:00 /usr/lib/gvfs/gvfssd
2247 ? S 0:00 /usr/lib/arm-linux-gnueabihf/libmenu-cache1/libexec/m
2250 ? S 0:00 /usr/lib/gvfs/gvfs-gdu-volume-monitor
2252 ? S1 0:00 /usr/lib/udisks/udisks-daemon
2253 ? S 0:00 udisks-daemon: not polling any devices
2256 ? S 0:00 /usr/lib/gvfs/gvfs-gphoto2-volume-monitor
2258 ? S1 0:01 /usr/lib/gvfs/gvfs-afc-volume-monitor
2260 ? Ss 0:00 sshd: pi [priv]
2264 ? S 0:00 /usr/sbin/ntp -p /var/run/ntp.pid -g -u 102:104
2268 ? S 0:01 sshd: pi@pts/0
2269 pts/0 Ss 0:03 -bash
2599 ? S 0:00 [kworker/0:1]
2604 pts/0 R+ 0:00 ps ax

```

Come potete vedere il nostro povero processore ha parecchio lavoro da fare!

I processi mostrati tra parentesi quadre sono in esecuzione all'interno del kernel. Quelli senza parentesi sono stati eseguiti a livello utente e vengono chiamati processi "user mode".

L'id di un processo può essere recuperato con il comando `pidof` (Process ID OF):

```

pi@pivalter ~ $ pidof bash
2628
pi@pivalter ~ $ pidof getty
2114 2113 2112 2111 2110 2109 2108

```

L'id di un processo è utile per poterlo terminare.

Il comando utilizzato in questo caso ha un nome un po' truce: `kill`.

Il comando `kill` invia una richiesta di chiusura al programma incriminato. Se questa richiesta non basta a farlo uscire la terminazione è possibile utilizzare l'opzione `-9` che forza la terminazione incondizionata del processo.

```
pi@pivalter ~ $ kill -9 $(pidof bash)
```

Sorpresa dal risultato?

In questo caso abbiamo sfruttato la possibilità di passare il risultato di un comando (`pidof`) come argomento di un altro includendolo tra parentesi e prefissandolo con il carattere dollaro (`$`). Questo è solo uno dei tanti meccanismi che Linux ci mette a disposizione per concatenare

l'esecuzione di più comandi al fine di costruire qualcosa di più complesso e funzionale.

Costruiamo con i mattoncini

Uno dei principi base che ha dettato il design dei sistemi Unix fin dalle origini è stato quello di preferire a programmi complessi e ricchi di funzioni applicazioni più piccole e semplici, che potessero però essere facilmente utilizzate in combinazione tra loro. Questo consente di partire da pochi semplici strumenti per costruire soluzioni decisamente più complesse ed è stato uno dei fattori che hanno decretato il successo dei sistemi basati su questa filosofia.

Il sistema operativo e la shell mettono a disposizione una serie di strumenti per consentire lo scambio di informazioni tra i diversi processi lanciati dalla command line.

Normalmente le applicazioni command line inviano il loro output sulla console. Questo può essere mostrato all'interno del terminale grafico, sullo schermo in modalità testuale o anche in remoto. A volte, però, può far comodo salvare l'output di un comando in un file, per poterlo poi elaborare, modificare o riutilizzare. Questa operazione è possibile tramite la redirezione dell'input/output. Utilizzando l'operatore > (maggiore) sarà possibile inviare quanto verrebbe normalmente stampato a video in un file di testo.

Proviamo a eseguire il comando:

```
pi@pivalter ~/dummy $ ls -la > lista.txt
```

In questo caso il sistema non stamperà sulla console il solito elenco dei file della cartella corrente.

L'elenco è stato salvato nel file `lista.txt` che possiamo stampare sulla console tramite il comando `cat`:

```
pi@pivalter ~/dummy $ cat lista.txt
totale 12
drwxr-xr-x  2 pi      pi      4096 ago 13 01:48 .
drwxr-xr-x 26 pi      pi      4096 ago 12 21:17 ..
-rw-r--r--  1 nobody staff    0 ago 12 18:07 file1
-rw-rw-rw-  1 pi      pi      0 ago 12 19:48 file2
lrwxrwxrwx  1 pi      pi      5 ago 12 18:13 file3 -> file2
-r--r--r--  1 pi      pi      0 ago 12 18:07 file4
-rw-r--r--  1 pi      pi      0 ago 13 01:48 lista.txt
-rw-r--r--  1 pi      pi     35 ago 12 20:13 snoopy.txt
```

In questo modo abbiamo salvato l'elenco dei file in modo permanente.

Possiamo anche utilizzare il comando `find` per generare un elenco completo dei file presenti sul nostro sistema:

```
pi@pivalter ~/dummy $ find / > tuttiifile.txt
find: "/proc/tty/driver": Permesso negato
find: "/proc/1/task/1/fd": Permesso negato
find: "/proc/1/task/1/fdinfo": Permesso negato
find: "/proc/1/task/1/ns": Permesso negato
find: "/proc/1/fd": Permesso negato
find: "/proc/1/fdinfo": Permesso negato
find: "/proc/1/ns": Permesso negato
find: "/proc/2/task/2/fd": Permesso negato
find: "/proc/2/task/2/fdinfo": Permesso negato
find: "/proc/2/task/2/ns": Permesso negato
find: "/proc/2/fd": Permesso negato
find: "/proc/2/fdinfo": Permesso negato
find: "/proc/2/ns": Permesso negato
...
find: "/var/cache/ldconfig": Permesso negato
find: "/var/lib/polkit-1": Permesso negato
find: "/var/lib/lightdm": Permesso negato
find: "/var/spool/cron/crontabs": Permesso negato
find: "/run/udisks": Permesso negato
find: "/run/lightdm": Permesso negato
find: "/etc/polkit-1/localauthority": Permesso negato
find: "/etc/ssl/private": Permesso negato
```

In questo caso, a differenza di quanto successo con il comando `ls`, avremo un output sia nel file `tuttiifile.txt` sia sulla console. I messaggi sulla console sono messaggi di errore e nei comandi Linux costituiscono un output diverso rispetto a quello vero e proprio del comando.

Se vogliamo salvare i messaggi di errore possiamo usare l'operatore `2>`.

Per esempio, digitando:

```
pi@pivalter ~/dummy $ find / > tuttiifile.txt 2> errori.txt
```

Avremo la lista dei file in `tuttiifile.txt`, i messaggi di errore in `errore.txt` e nessun output sulla console. Redirigere anche gli errori può essere molto utile, per esempio, per inviare a un amico che ci vuole aiutare l'elenco degli errori che vengono generati sul nostro sistema. Un'altra possibilità è quella di redirigere l'output... nel nulla. Per un sistema Linux il nulla è rappresentato dal file virtuale `/dev/null`; redirigere l'output verso `/dev/null` lo toglierà dalla nostra console senza doverlo necessariamente salvare in modo permanente.

Per esempio:

```
pi@pivalter ~/dummy $ find / > tuttiifile.txt 2> /dev/null
```

genererà di nuovo il nostro elenco di file facendo sparire i messaggi di errore.

Il file `tuttiifile.txt` è stato rigenerato a ogni esecuzione del comando `find`. Questo è il comportamento che normalmente si desidera. Se però fosse necessario “appendere” l’output generato a un file esistente basterà utilizzare l’operatore `>>` (due volte maggiore) oppure `>>>` se volete farlo per i messaggi di errore.

Il sistema consente anche di redirigere l’input, inviando a un comando il contenuto di un file di testo invece di ciò che viene digitato sulla tastiera.

Per provare questa funzionalità possiamo usare il comando `grep` (Global Regular Expression Print).

Questo comando filtra il suo input stampando sull’output solo le righe che contengono una determinata stringa passata come argomento.

Se lanciamo `grep` dalla command line il risultato non è dei più interessanti:

```
pi@pivalter ~/dummy $ grep ciao
hello world
ciao mondo
ciao mondo
^C
```

Si limiterà a ristampare le righe che contengono la parola `ciao` e dovremo premere **Ctrl+C** per terminarlo.

Il suo potenziale diventa più chiaro se invece utilizziamo l’operatore `<` (minore) per redirigere l’output. Se, per esempio, vogliamo utilizzare la lista completa dei file generata in precedenza per trovare tutti i file che contengono nel nome la stringa `raspberry` possiamo lanciare:

```
pi@pivalter ~/dummy $ grep raspberry < tuttiifile.txt
/var/lib/dpkg/info/libraspberrypi-bin.md5sums
/var/lib/dpkg/info/libraspberrypi0.md5sums
/var/lib/dpkg/info/libraspberrypi0.postinst
/var/lib/dpkg/info/libraspberrypi-doc.list
/var/lib/dpkg/info/libraspberrypi-bin.list
...
```

Decisamente più utile, no?

In realtà sarebbe ancora più utile passare direttamente l’output del comando `find` al comando `grep`, in modo da non dover creare dei file permanenti per completare l’operazione. L’operatore `|` (pipe, lo trovate

sopra il tasto \ se avete una tastiera italiana) serve proprio a questo.

Lanciando:

```
pi@pivalter ~ /dummy $ find / 2> /dev/null | grep raspberry
/var/lib/dpkg/info/libraspberrypi-bin.md5sums
/var/lib/dpkg/info/libraspberrypi0.md5sums
/var/lib/dpkg/info/libraspberrypi0.postinst
/var/lib/dpkg/info/libraspberrypi-doc.list
/var/lib/dpkg/info/libraspberrypi-bin.list
/var/lib/dpkg/info/raspberrypi-bootloader.postrm
...
```

avremo la nostra lista di file senza bisogno di generare alcun file permanente.

E se vogliamo trovare i file .png all'interno della lista? Possiamo concatenare un'istanza di grep con una successiva! Et voilà:

```
pi@pivalter ~ /dummy $ find / 2> /dev/null | grep raspberry | grep -F .png
/usr/share/raspberrypi-artwork/raspberry-pi-logo.png
/usr/share/icons/gnome/32x32/emotes/face-raspberry.png
/usr/share/icons/gnome/16x16/emotes/face-raspberry.png
/usr/share/icons/gnome/22x22/emotes/face-raspberry.png
/usr/share/icons/gnome/48x48/emotes/face-raspberry.png
/usr/share/icons/gnome/24x24/emotes/face-raspberry.png
```

Il parametro `-F` passato alla seconda istanza di grep serve a forzare la ricerca della stringa `.png` senza interpretare Il punto iniziale come carattere particolare. In realtà grep può eseguire ricerche molto più complesse della ricerca di una semplice stringa usando le espressioni regolari e, come già visto per molti altri comandi, `man grep` o `grep --help` sapranno soddisfare la vostra curiosità.

Altri comandi "filtro" simili a grep e utili quando utilizzati attraverso l'operatore pipe sono il comando `more`, che mostra il contenuto di un file in modo analogo a quanto visto con `cat`, ma fermando la visualizzazione a ogni pagina e aspettando la pressione di un tasto per continuare, e il comando `sort`, che ordina una lista con modalità definibili tramite parametri sulla command line.

Un'altra possibilità è quella di passare come argomento sulla command line di un comando il risultato di un altro comando.

Prima di esaminare questa possibilità vediamo un comando che ci farà da "cavia" per questi esperimenti.

Il comando `stat` mostra alcune informazioni significative sul file passato come argomento.

```
pi@pivalter ~ /dummy $ stat file1
```

```

File: "file1"
Dim.: 0 Blocchi: 0 Blocco di IO: 4096 file regolare vuoto
Device: b306h/45830d Inode: 71672 Coll.: 1
Accesso: (0644/-rw-r--r--)Uid: (65534/ nobody) Gid: ( 50/ staff)
Accesso : 2013-08-12 18:07:13.731021273 +0200
Modifica : 2013-08-12 18:07:13.731021273 +0200
Cambio : 2013-08-12 21:56:18.140186264 +0200
Creazione: -

```

Se volessimo eseguirlo su tutti i file della nostra cartella potremmo lanciare `stat *.*`, ma questo non ci consentirebbe di prendere in considerazione anche i file nascosti. Per eseguire `stat` anche su questi file possiamo sfruttare l'output del comando `ls -a`. Per passare sulla command line l'output di un comando è necessario usare l'operatore `$()`.

```

pi@pivalter ~/dummy $ stat $(ls -a)
File: "errori.txt"
Dim.: 27644 Blocchi: 56 Blocco di IO: 4096 file regolare
Device: b306h/45830d Inode: 71685 Coll.: 1
Accesso: (0644/-rw-r--r--)Uid: ( 1000/ pi) Gid: ( 1000/ pi)
Accesso : 2013-08-13 01:55:24.001409451 +0200
Modifica : 2013-08-13 01:55:25.041409555 +0200
Cambio : 2013-08-13 01:55:25.041409555 +0200
Creazione: -
File: "file1"
Dim.: 0 Blocchi: 0 Blocco di IO: 4096 file regolare vuoto
Device: b306h/45830d Inode: 71672 Coll.: 1
Accesso: (0644/-rw-r--r--)Uid: (65534/ nobody) Gid: ( 50/ staff)
Accesso : 2013-08-12 18:07:13.731021273 +0200
Modifica : 2013-08-12 18:07:13.731021273 +0200
Cambio : 2013-08-12 21:56:18.140186264 +0200
Creazione: -

```

Non sempre l'output di un comando è nel formato accettabile per la command line di un altro. Il comando `xargs` (eXpand ARGumentS) ci permette di convertire una lista di stringhe separate con un qualsiasi carattere (per default spazio, Tab o ritorno a capo) in una lista separata con spazi e quindi adatta a una command line. Per lanciare, per esempio, `stat` su tutti i file di tipo `.png` che contengono nel nome la stringa `raspberry`, possiamo digitare:

```

pi@pivalter ~/dummy $ find / 2> /dev/null | grep raspberry | grep -F .png | xargs
File: "/usr/share/raspberrypi-artwork/raspberry-pi-logo.png"
Dim.: 48162 Blocchi: 96 Blocco di IO: 4096 file regolare
Device: b306h/45830d Inode: 49626 Coll.: 1
Accesso: (0644/-rw-r--r--)Uid: ( 0/ root) Gid: ( 0/ root)
Accesso : 2013-06-19 13:34:28.000000000 +0200
Modifica : 2012-07-15 17:04:54.000000000 +0200
Cambio : 2013-06-19 13:34:29.080000000 +0200
Creazione: -

```

```

File: "/usr/share/icons/gnome/32x32/emotes/face-raspberry.png"
Dim.: 2594 Blocchi: 8 Blocco di IO: 4096 file regolare
Device: b306h/45830d Inode: 54339 Coll.: 1
Accesso: (0644/-rw-r--r--)Uid: ( 0/ root) Gid: ( 0/ root)
Accesso : 2013-06-19 13:34:42.000000000 +0200
Modifica : 2012-04-01 06:25:22.000000000 +0200
Cambio : 2013-06-19 13:35:09.050000002 +0200
Creazione: -
File: "/usr/share/icons/gnome/16x16/emotes/face-raspberry.png"
Dim.: 1054 Blocchi: 8 Blocco di IO: 4096 file regolare
Device: b306h/45830d Inode: 53967 Coll.: 1
Accesso: (0644/-rw-r--r--)Uid: ( 0/ root) Gid: ( 0/ root)
Accesso : 2013-06-19 13:34:42.000000000 +0200
Modifica : 2012-04-01 06:24:48.000000000 +0200
Cambio : 2013-06-19 13:35:08.840000002 +0200
Creazione: -
File: "/usr/share/icons/gnome/22x22/emotes/face-raspberry.png"
Dim.: 1385 Blocchi: 8 Blocco di IO: 4096 file regolare
Device: b306h/45830d Inode: 55076 Coll.: 1
Accesso: (0644/-rw-r--r--)Uid: ( 0/ root) Gid: ( 0/ root)
Accesso : 2013-06-19 13:34:42.000000000 +0200
Modifica : 2012-04-01 06:25:00.000000000 +0200
Cambio : 2013-06-19 13:35:09.450000002 +0200
Creazione: -
File: "/usr/share/icons/gnome/48x48/emotes/face-raspberry.png"
Dim.: 4165 Blocchi: 16 Blocco di IO: 4096 file regolare
Device: b306h/45830d Inode: 53368 Coll.: 1
Accesso: (0644/-rw-r--r--)Uid: ( 0/ root) Gid: ( 0/ root)
Accesso : 2013-06-19 13:34:42.000000000 +0200
Modifica : 2012-04-01 06:25:33.000000000 +0200
Cambio : 2013-06-19 13:35:08.520000002 +0200
Creazione: -
File: "/usr/share/icons/gnome/24x24/emotes/face-raspberry.png"
Dim.: 1395 Blocchi: 8 Blocco di IO: 4096 file regolare
Device: b306h/45830d Inode: 54707 Coll.: 1
Accesso: (0644/-rw-r--r--)Uid: ( 0/ root) Gid: ( 0/ root)
Accesso : 2013-06-19 13:34:42.000000000 +0200
Modifica : 2012-04-01 06:25:11.000000000 +0200
Cambio : 2013-06-19 13:35:09.250000002 +0200
Creazione: -

```

Variabili

A volte può essere necessario memorizzare informazioni o parametri senza doverli necessariamente salvare in un file. Questi dati possono essere memorizzati utilizzando le variabili.

Per creare una variabile non dobbiamo fare altro che assegnarle un valore usando l'operatore = (uguale):

```
pi@pivalter ~ $ dummy=ciao
```

Possiamo stampare la nostra variabile mediante il comando echo:

```
pi@pivalter ~ $ echo $dummy  
ciao
```

Utilizzando il nome della variabile prefissato con il carattere dollaro (\$) la shell sostituirà alla variabile il suo valore corrente. Possiamo adoperare variabili all'interno di qualsiasi comando o addirittura direttamente sul prompt della shell, come in questo esempio:

```
pi@pivalter ~ $ dummy="ls -la"  
pi@pivalter ~ $ $dummy  
totale 200  
drwxr-xr-x 27 pi pi 4096 ago 19 15:13 .  
drwxr-xr-x 3 root root 4096 giu 19 12:38 ..  
-rw----- 1 pi pi 5690 ago 19 11:00 .bash_history  
-rw-r--r-- 1 pi pi 220 giu 19 12:38 .bash_logout  
-rw-r--r-- 1 pi pi 3263 ago 7 08:50 .bashrc  
drwxr-xr-x 6 pi pi 4096 ago 7 10:39 .cache  
drwxr-xr-x 13 pi pi 4096 ago 12 14:02 .config  
...  
-rw----- 1 pi pi 999 ago 19 15:05 .xsession-errors  
-rw----- 1 pi pi 2359 ago 19 14:36 .xsession-errors.old  
pi@pivalter ~ $
```

Se vogliamo assegnare alla nostra variabile un valore che contiene degli spazi, come nel caso precedente, dovremo utilizzare i doppi apici attorno al testo che vogliamo assegnare. È importante non mettere spazi tra il nome della variabile e l'operatore uguale e tra questo e il valore da assegnare.

Le variabili non vengono salvate in modo permanente e verranno perse semplicemente chiudendo la console attiva o riavviando Raspberry Pi.

La variabile che abbiamo appena definito è una variabile locale: non può cioè essere vista al di fuori della shell corrente, nemmeno dai comandi o script che vengono lanciati dalla shell stessa.

Se vogliamo definire una variabile che sia visibile ai vari comandi lanciati dalla shell è necessario utilizzare il comando `export`.

```
pi@pivalter ~ $ export dummy
```

Possiamo anche combinare assegnamento ed `export` in un'unica istruzione:

```
pi@pivalter ~ $ export dummy=ciao
```

Digitando il comando `printenv` possiamo vedere un elenco delle variabili definite.

```

pi@pivalter ~ $ printenv
dummy=ls -la
TERM=xterm
SHELL=/bin/bash
XDG_SESSION_COOKIE=3b0e4b712f60d6b9547b25ae51c194dd-1376916688.40861-552521201
SSH_CLIENT=192.168.99.104 9382 22
SSH_TTY=/dev/pts/0
USER=pi
LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:bd=40;33;01:cd=
33;01:or=40;31;01:su=37;41:sg=30;43:ca=30;41:tw=30;42:ow=34;42:st=37;44:ex=01;32:
tar=01;31:*.tgz=01;31:*.arj=01;31:*.taz=01;31:*.lzh=01;31:*.lzma=01;31:*.tlz=01;31:*.txz=01;31:*.zip=01;31:*.z=01;31:*.Z=01;31:*.dz=01;31:*.gz=01;31:*.lz=01;31:*.xz=01;31:*.bz2=01;31:*.bz=01;31:*.tbz=01;31:*.tbz2=01;31:*.tz=01;31:*.deb=01;31:*.rpm=01;31:*.jar=01;31:*.war=01;31:*.ear=01;31:*.sar=01;31:*.rar=01;31:*.ace=01;31:*.zoo=01;31:*.cpio=01;31:*.7z=01;31:*.rz=01;31:*.jpg=01;35:*.jpeg=01;35:*.gif=01;35:*.bmp=01;35:*.pbm=01;35:*.pgm=01;35:*.ppm=01;35:*.tga=01;35:*.xbm=01;35:*.xpm=01;35:*.tif=01;35:*.tiff=01;35:*.png=01;35:*.svg=01;35:*.svgz=01;35:*.mng=01;35:*.pcx=01;35:*.mov=01;35:*.mpg=01;35:*.mpeg=01;35:*.m2v=01;35:*.mkv=01;35:*.webm=01;35:*.ogm=01;35:*.mp4=01;35:*.m4v=01;35:*.mp4v=01;35:*.vob=01;35:*.qt=01;35:*.nuv=01;35:*.wmv=01;35:*.ASF=01;35:*.rm=01;35:*.rmvb=01;35:*.flc=01;35:*.avi=01;35:*.fli=01;35:*.flv=01;35:*.gl=01;35:*.dl=01;35:*.xcf=01;35:*.xwd=01;35:*.yuv=01;35:*.cgm=01;35:*.emf=01;35:*.axv=01;35:*.anx=01;35:*.ogv=01;35:*.ogx=01;35:*.aac=00;36:*.au=00;36:*.flac=00;36:*.mid=00;36:*.midi=00;36:*.mka=00;36:*.mp3=00;36:*.mpc=00;36:*.ogg=00;36:*.ra=00;36:*.wav=00;36:*.axa=00;36:*.oga=00;36:*.spx=00;36:*.xspf=00;36:
MAIL=/var/mail/pi
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/local/games/usr/games
PWD=/home/pi
LANG=it_IT.UTF-8
SHLVL=1
HOME=/home/pi
LOGNAME=pi
SSH_CONNECTION=192.168.99.104 9382 192.168.99.20 22
DISPLAY=:0
_=~/usr/bin/printenv
pi@pivalter ~ $

```

Oltre alla variabile `dummy` appena definita il sistema mantiene diverse variabili che possono tornare molto utili nell'utilizzo della command line o nella realizzazione di script (argomento che tratteremo tra poche righe).

Generalmente queste variabili hanno nomi tutti in caratteri maiuscoli. Usando per le nostre variabili i caratteri minuscoli eviteremo qualsiasi sovrapposizione accidentale con quelle di sistema, poiché i nomi di variabile, come i nomi file, sono case-sensitive. La variabile `HOME`, per esempio, contiene il path della cartella home dell'utente corrente, quindi potrete usare `$HOME` per referenziarla, indipendentemente dalla vostra posizione corrente nel filesystem.

Altre variabili ci consentono di conoscere il nome dell'utente (`USER`), la directory corrente (`PWD`) e la lingua corrente (`LANG`).

La variabile `PATH` contiene, separate dal carattere due punti (:), le cartelle in cui il sistema andrà a cercare gli eseguibili dei comandi da lanciare.

In realtà tutti i comandi che abbiamo utilizzato fino a ora sono memorizzati in una delle cartelle referenziate dentro la variabile `PATH` e questo ci ha consentito di lanciarli senza doverne specificare il percorso completo. I programmi memorizzati in cartelle non referenziate nel `PATH` devono essere referenziati mediante un percorso assoluto o relativo. Anche gli eseguibili nella directory corrente devono essere lanciati prefissandone il nome con punto e slash (./).

Se volete scoprire dove è memorizzato un comando potete utilizzare il comando `which`:

```
pi@pivalter ~ $ which ls  
/bin/ls
```

E come possiamo scoprire dove è memorizzato `which`?

```
pi@pivalter ~ $ which which  
/usr/bin/which
```

Scripting

Nei paragrafi precedenti abbiamo imparato a conoscere alcuni dei comandi fondamentali del nostro sistema Linux e a utilizzarli in combinazione per poter eseguire attività più complesse. Dover riscrivere ogni volta le varie sequenze di comandi è però abbastanza scomodo e noioso.

Gli script ci consentono di scrivere sequenze di operazioni e di richiamarle in modo molto semplice dalla shell. Gli script sono semplici file di testo e per creare un nuovo script possiamo semplicemente utilizzare Nano dalla command line o Leafpad dall'interfaccia grafica.

La prima riga dello script serve a indicare con quale applicazione deve essere eseguito. Il path dell'applicazione da utilizzare va prefissato con i caratteri cancelletto e punto esclamativo (#!).

Per il momento utilizzeremo degli script con i comandi della shell, quindi indicheremo `/bin/bash` come applicazione. La prima riga del nostro script sarà dunque:

```
#!/bin/bash
```

Nelle righe successive potremo riportare i diversi comandi da eseguire.

Utilizzare il comando `echo` per fornire qualche informazione all'utente che esegue lo script è una buona idea.

```
echo "Buongiorno $USER, questo è il tuo primo script!"
```

Ora possiamo salvare il nostro script chiamandolo "hello".

Per salvare il file se state utilizzando Nano potete premere **Ctrl+O** e inserire il nome del file da salvare, quindi **Ctrl+X** per uscire. Se avete preferito l'interfaccia grafica di Leafpad potrete selezionare le opzioni **Salva** e poi **Esci** del menu **File** per salvare il vostro nuovo script e uscire dall'editor.

Il nostro file, però, non è ancora eseguibile; lo possiamo vedere utilizzando il comando `ls`:

```
pi@pivalter ~ $ ls -l hello
-rw-r--r-- 1 pi pi 69 ago 20 08:51 hello
```

Per renderlo eseguibile dobbiamo impostare il flag `x` e possiamo farlo con il comando `chmod`:

```
pi@pivalter ~ $ chmod a+x hello
pi@pivalter ~ $ ls -l hello
-rwxr-xr-x 1 pi pi 69 ago 20 08:51 hello
```

Ora possiamo finalmente lanciare il nostro primo script; per farlo dobbiamo usare il path relativo alla directory corrente:

```
pi@pivalter ~ $ ./hello
Buongiorno pi, questo è il tuo primo script!
```

Il comando `echo` ha scritto il messaggio sulla console e la variabile `USER` è stata sostituita dal suo valore (`pi` nel caso dell'utente di default).

Gli script sono utili per eseguire operazioni ripetitive, ma non è detto che le operazioni debbano essere ripetute sempre esattamente allo stesso modo.

Per esempio, possiamo costruire uno script che ci ritorni il numero di gruppi in cui è presente l'utente `pi`. Per farlo possiamo usare `grep` per filtrare le righe in cui è presente il nome dell'utente e poi il comando `wc` (Word Count) con il parametro `-l` per contare il numero di linee ritornate.

Possiamo creare uno script chiamato `mostragruppi` per eseguire la sequenza di comandi che ci serve:

```
#!/bin/bash
```

```
cat /etc/group | grep "pi" | wc -l
```

Poi possiamo lanciarlo (dopo averlo reso eseguibile):

```
pi@pivalter ~ $ chmod a+x mostragruppi  
pi@pivalter ~ $ ./mostragruppi  
14
```

Per vedere in quanti gruppi è citato l'utente root dovremmo modificare il sorgente dello script. E così via per ogni altro utente che ci interessa.

In questi casi è decisamente più comodo utilizzare dei parametri. I parametri possono essere adoperati all'interno dello script come variabili e conterranno il valore passato sulla command line. Fino a nove parametri possono essere referenziati utilizzando $\$1$, $\$2$ ecc. fino a $\$9$. Il nome dello script verrà passato come $\$0$.

Cambiando il nostro script per utilizzare i parametri basterà sostituire al nome dell'utente il parametro $\$1$:

```
#!/bin/bash  
cat /etc/group | grep "$1" | wc -l
```

In questo modo potremo invocarlo con il nome dell'utente che vogliamo controllare:

```
pi@pivalter ~ $ ./mostragruppi pi  
14  
pi@pivalter ~ $ ./mostragruppi root  
2  
pi@pivalter ~ $ ./mostragruppi  
53
```

Nell'ultima invocazione non è stato passato alcun parametro.

In questo caso il parametro $\$1$ verrà valorizzato come stringa vuota e questo genera un valore stampato pari al numero di righe del file `/etc/group`. Per evitare questo tipo di errori dobbiamo inserire delle istruzioni di test e controllo all'interno del nostro codice; vedremo come farlo tra pochissime pagine.

Cosa possiamo fare se al nostro script servono più di nove parametri? Ci sono due possibilità. La prima è quella di indirizzare i parametri dal decimo in poi con la notazione $\{\$\{xx\}\}$, dove xx indica il numero del parametro. La seconda possibilità è quella di usare l'istruzione `shift` per far scorrere i parametri verso sinistra. In questo modo $\$2$ rimpiazzerà $\$1$ e verrà rimpiazzato da $\$3$, e così via.

Ogni comando Linux ha un valore di ritorno numerico. Il valore zero,

per convenzione, indica un'esecuzione corretta, mentre valori diversi da zero indicano un errore. Il valore di ritorno viene memorizzato in una variabile di environment particolare identificata con il carattere punto di domanda (?).

```
pi@pivalter ~ $ cat hello
#!/bin/bash
echo "Buongiorno $USER, questo è il tuo primo script!"

pi@pivalter ~ $ echo $?
0
pi@pivalter ~ $ cat helooooo
cat: helooooo: File o directory non esistente
pi@pivalter ~ $ echo $?
1
```

In questo esempio la prima esecuzione del comando `cat` è andata a buon fine e ha ritornato il valore zero, la seconda no e infatti il codice di ritorno è pari a uno.

Nei nostri script possiamo controllare il valore di ritorno di un comando utilizzando l'istruzione `if`. Notate che parliamo di istruzione e non di comando perché, a differenza dei comandi visti finora, si tratta di un'operazione gestita direttamente dalla shell e non tramite l'invocazione di un comando esterno.

In questo esempio utilizziamo il comando `if` per controllare il valore di ritorno di un comando e stampiamo un messaggio in caso di successo:

```
#!/bin/bash
if cat $1
then
echo "ha funzionato!"
fi
```

Salviamo il nostro esempio come “verificafile” e rendiamolo eseguibile con `chmod` prima di lanciarlo:

```
pi@pivalter ~ $ chmod a+x verificafile
pi@pivalter ~ $ ./verificafile hello
#!/bin/bash
echo "Buongiorno $USER, questo è il tuo primo script!"

ha funzionato!
pi@pivalter ~ $ ./verificafile heloooooo
cat: heloooooo: File o directory non esistente
```

L'esecuzione con successo del comando `cat` causa l'esecuzione delle istruzioni contenute tra l'istruzione `then` e l'istruzione `fi`.

È anche possibile inserire una serie di comandi da eseguire in caso di fallimento, utilizzando la clausola `else`.

```
#!/bin/bash
if cat $1
then
echo "ha funzionato!"
else
echo "non ha funzionato"
fi
```

In questo caso lo script stampa un messaggio sia in caso di successo sia in caso di insuccesso:

```
pi@pivalter ~ $ ./verificafайл hello
#!/bin/bash
echo "Buongiorno $USER, questo è il tuo primo script!"
ha funzionato!
pi@pivalter ~ $ ./verificafайл heloooooooo
cat: heloooooooo: File o directory non esistente
non ha funzionato
```

L'istruzione `if` della shell è limitata a controllare il codice di ritorno di un comando. Nella logica di Unix, invece di complicare la gestione della shell è stato implementato un comando che è in grado di verificare una serie di condizioni ritornando zero se queste sono verificate e che può quindi essere utilizzato a seguito dell'istruzione `if`.

Questo comando è `test`. Il comando `test` ha diverse opzioni. L'opzione `e`, per esempio, serve a verificare se un file esiste. Possiamo cambiare il nostro script utilizzando il comando `test` al posto del di `cat`; questo eviterà la stampa del file se questo esiste o il messaggio di errore in caso contrario.

```
#!/bin/bash
if test -e $1
then
echo "il file esiste!"
else
echo "il file non esiste!"
fi
```

Lanciando il nostro nuovo script con nomi di file esistenti o meno vedremo i due diversi messaggi:

```
pi@pivalter ~ $ ./verificafайл hello
il file esiste!
pi@pivalter ~ $ ./verificafайл heloooooooo
il file non esiste!
```

Il comando `test` è utilizzato così spesso negli script che è stata prevista una sintassi ad-hoc per invocarlo. È possibile utilizzare l'operatore `[]` (parentesi quadre) indicando tra le parentesi i parametri del comando `test` da invocare. È importante ricordarsi di lasciare uno spazio dopo la parentesi aperta e uno prima di quella chiusa, altrimenti verrà generato un errore di sintassi in fase di esecuzione del nostro script.

Usando le parentesi il nostro script può essere reso più leggibile:

```
#!/bin/bash
if [ -e $1 ]
then
echo "il file esiste!"
else
echo "il file non esiste!"
fi
```

Anche in questo caso, però, invocando lo script senza parametri otteniamo un comportamento inaspettato:

```
pi@pivalter ~ $ ./verificafайл
il file esiste!
```

Possiamo utilizzare il parametro `-z` del comando `test` per controllare che il parametro non sia uguale a stringa vuota prima di eseguire la verifica:

```
#!/bin/bash
# controllo che sia stato passato almeno un parametro
if [ -z $1 ]
then
    echo "è necessario passare un nome file!"
    exit
fi

# verifico l'esistenza del file
if [ -e $1 ]
then
    echo "il file esiste!"
else
    echo "il file non esiste!"
fi
```

Ora il nostro script gestirà correttamente anche l'invocazione senza parametri:

```
pi@pivalter ~ $ ./verificafайл
è necessario passare un nome file!
```

Questo script ci fornisce anche qualche altro spunto interessante.

L'istruzione `exit` permette di terminare lo script. Può essere utile farlo se ci accorgiamo che i parametri non sono corretti o se si è verificato un errore che potrebbe precludere l'esecuzione dei comandi successivi.

Avrete notato che, rispetto alle versioni precedenti, i comandi tra le istruzioni `then`, `else` e `fi` sono spostati a destra rispetto al margine. Questa tecnica viene chiamata indentazione e permette di rendere più leggibile il codice identificando immediatamente i vari blocchi logici che lo costituiscono.

Il carattere `#` (diesis o cancelletto) può essere utilizzato per inserire dei commenti all'interno del nostro script. I commenti non rallentano l'esecuzione dello script, quindi non state parsimoniosi nel loro utilizzo. Quello che vi è chiaro mentre state scrivendo uno script potrebbe non esserlo quando vi troverete a modificarlo dopo qualche mese e un commento potrebbe risparmiarvi tempo e fatica.

Il comando `test` supporta moltissime operazioni su stringhe e file. Come al solito per avere l'elenco completo la cosa più semplice è usare il comando `man test`.

Qui sotto una tabella con le opzioni più comunemente utilizzate:

Opzione	Tipo dato	Funzione	Esempio
<code>=</code>	Stringa	Confronta due stringhe e ritorna zero se sono uguali	<pre>if [\$USER=="root"] then echo "sei il super-user" fi</pre>
<code>!=</code>	Stringa	Confronta due stringhe e ritorna zero se non sono uguali	<pre>if [\$USER!="root"] then echo "sei un normale utente" fi</pre>
<code>-n</code>	Stringa	Ritorna zero se la stringa ha una lunghezza maggiore di zero	<pre>if [-n \$1] then echo "parametro 1 definito" fi</pre>
<code>-z</code>	Stringa	Ritorna zero se la stringa è vuota	<pre>if [-z \$1] then echo "parametro non definito" fi</pre>
<code>-e</code>	File	Verifica se il file esiste e ritorna zero in caso positivo, sia che si tratti di un file, sia che si tratti di una cartella	<pre>if [-e \$1] then echo "path valido" fi</pre>

-f	File	Verifica se il file esiste e ritorna zero in caso positivo. Una cartella con lo stesso nome genererà invece un valore diverso da zero	if [-f \$1] then echo "file valido" fi
-d	File	Verifica se una cartella esiste e ritorna zero in caso positivo	if [-d \$1] then echo "cartella valida" fi

In uno script può essere utile anche ripetere una stessa operazione su una serie di valori o di file.

In questo caso è possibile utilizzare l'istruzione `for` per iterare su una serie di stringhe o file. Partiamo con un piccolo esempio:

```
#!/bin/bash

indice=0

for parametro in $@

do
    indice=$[ $indice+1 ]
    echo "parametro $indice = $parametro"
done
```

Questo piccolo esempio utilizza la variabile `$@` che contiene la lista completa dei parametri e la scorre utilizzando l'istruzione `for`. L'istruzione `for` ripete i comandi contenuti tra le istruzioni `do` e `done`.

Queste istruzioni vanno a incrementare un contatore e a stampare uno a uno i parametri passati allo script. Notate che per incrementare il contatore viene utilizzato l'operatore `$[]`. Questo operatore, in maniera analoga a quanto avviene con l'operatore parentesi quadre e il comando `test`, invoca il comando `expr` per calcolare l'espressione passata tra le parentesi quadre. La shell, sempre nel rispetto della filosofia di Unix, usa un "mattoncino" già pronto anche per eseguire i calcoli!

Proviamo a eseguire il nostro script dopo averlo salvato con il nome `parametri`:

```
pi@pivalter ~ $ chmod a+x parametri
pi@pivalter ~ $ ./parametri uno due tre
parametro 1 = uno
parametro 2 = due
parametro 3 = tre
pi@pivalter ~ $ ./parametri
```

```
pi@pivalter ~ $
```

In questo caso l'esecuzione dello script senza parametri non genera alcun errore; il ciclo `for` verrà semplicemente ignorato.

Il comando `for` può essere utilizzato anche per eseguire un'operazione su una serie di file o cartelle.

Possiamo, per esempio, scrivere uno script che conti le sotto-cartelle della directory corrente:

```
#!/bin/bash
contatore=0
for nome in *
do
    if [ -d $nome ]
    then
        contatore=$((contatore+1))
    fi
done

echo "la cartella corrente contiene $contatore sotto-cartelle"
```

In questo caso il ciclo `for` enumererà tutti i file e le cartelle. L'istruzione `if`, usando l'opzione `-d`, andrà a incrementare il contatore solo per le cartelle.

Proviamo a eseguire il nostro script dopo averlo reso eseguibile:

```
pi@pivalter ~ $ chmod a+x contacartelle
pi@pivalter ~ $ ./contacartelle
la cartella corrente contiene 8 sotto-cartelle
```

A volte il numero di iterazioni del ciclo potrebbe non essere noto a priori. In questi casi è possibile utilizzare le istruzioni `while` o `until`.

L'istruzione `while` verifica il codice di ritorno di un comando e ripete le istruzioni del loop fino a quando il comando non ritorna un valore diverso da zero. L'istruzione `until` lavora in modo simile, ma termina il loop quando il comando ritorna zero.

Possiamo riscrivere lo script parametri utilizzando l'istruzione `until` e lo `shift` dei parametri:

```
#!/bin/bash

indice=0

until [ -z $1 ]
do
    indice=$((indice+1))
    echo "parametro $indice = $1"
    shift
done
```

done

Il risultato non cambia:

```
pi@pivalter ~ $ ./parametri uno due tre
parametro 1 = uno
parametro 2 = due
parametro 3 = tre
```

In questo capitolo abbiamo soltanto sfiorato la quantità di funzioni disponibili attraverso la command line e gli script, ma quanto abbiamo visto ci tornerà molto utile nei prossimi capitoli e, soprattutto, nell'utilizzo quotidiano di Raspberry Pi.

Oltre al comando `man` e all'`help` dei singoli comandi esistono in rete e in libreria tantissime risorse che possono aiutarvi, se vi interessa, ad approfondire la vostra conoscenza della command line e degli script.

Lo stesso Raspberry Pi può essere un'ottima fonte di esempi. Molte operazioni sono automatizzate con script da shell. Quando ne trovate qualcuno, perdete un minuto a leggerne il sorgente.

A proposito, vi ricordate qual è il comando per stampare un file sulla console?

Vi aiuto: è il nome di un felino domestico in lingua inglese.

That's not all folks!

Prendendo a prestito la famosa chiusura dei cartoni animati della Warner Bros, si chiude questo capitolo. Però, a differenza di quanto succedeva con i famosi Looney Tunes, la storia non finisce qui o, perlomeno, siete liberi di continuare a scoprire nuovi comandi e di inventare nuovi modi di combinarli tra loro per semplificarvi la vita durante l'utilizzo di Raspberry Pi o di qualsiasi sistema Linux.

Ogni tanto può essere utile spegnere il nostro Raspberry Pi e dedicarci ad altre attività. Per farlo potete utilizzare il comando `shutdown`. Questo comando, però, può essere usato solo da un super-user, quindi dovete lanciarlo tramite `sudo`:

```
pi@piserver ~ $ sudo shutdown -h now
```

```
Broadcast message from root@piserver (pts/0) (Fri Sep 13 16:22:02 2013) :
```

```
The system is going down for system halt NOW!
```

Il comando `shutdown` può essere utilizzato anche per programmare uno

spegnimento ritardato. Come al solito l'opzione `help` o le `man` pages vi saranno di aiuto.

Il comando `shutdown` non spegnerà fisicamente il vostro Raspberry Pi. Per spegnerlo, una volta terminato lo shutdown, è necessario staccare l'alimentatore.

Se invece volette riavviare il sistema potete utilizzare l'opzione `-r` di `shutdown` oppure il comando `reboot`.

```
pi@piserver ~ $ sudo reboot
```

```
Broadcast message from root@piserver (pts/0) (Fri Sep 13 16:26:40 2013):
```

```
The system is going down for reboot NOW!
```

E mentre il vostro Raspberry Pi si riavvia, potete passare al prossimo capitolo per imparare a farne un piccolo server domestico!

Raspberry Pi come server domestico

Utilizziamo Raspberry Pi come server domestico. Impariamo a configurarlo per fornire servizi e contenuti ai diversi sistemi presenti in una rete locale.

La diffusione delle reti domestiche, spesso gestite direttamente dal router Wi-Fi usato per il collegamento a internet, e dei dispositivi adatti a fruire di contenuti multimediali in rete, dai tablet alle televisioni “smart”, rende interessante la possibilità di avere un dispositivo sempre attivo per condividere questi contenuti e utilizzare altri servizi di rete. Tenere un normale PC sempre acceso richiede però parecchia energia e uno spazio in cui piazzare un dispositivo con ventole rumorose. Raspberry Pi è in grado di supportare una connessione di rete cablata o wireless, consuma pochissimo e non produce alcun rumore molesto. Questo lo rende il candidato ideale a svolgere la funzione di server domestico.

Per usare Raspberry Pi come server domestico dobbiamo però modificare la configurazione di Raspbian e in particolare:

- disabilitare l’attivazione dell’interfaccia grafica all’avvio;
- configurare un indirizzo IP statico;
- aggiungere uno storage esterno via USB;
- aggiungere e configurare servizi di rete.

Anche i server domestici sono dispositivi “personalì”, quindi sentitevi liberi di sperimentare e di applicare solo i consigli che vi sembrano adatti al vostro modo di utilizzare la rete.

Esistono moltissimi esempi di utilizzo di Raspberry Pi come server domestico e sono tutti diversi tra loro, proprio perché ogni utente ha adattato il sistema alle sue specifiche necessità.

Bye bye GUI

L’interfaccia Grafica (Graphical User Interface, GUI) è molto utile per utilizzare svariate tipologie di applicazioni. La grafica consente di avere interfacce utente più intuitive e di mostrare contenuti che non sarebbero fruibili attraverso un terminale a caratteri. L’interfaccia grafica, però, richiede anche parecchie risorse e, nel caso di un sistema server dove non lavorano utenti in modalità interattiva, disabilitarla può permettere di dedicare tutte le risorse hardware del sistema ai compiti del server.

Possiamo disabilitare l’avvio automatico dell’interfaccia grafica lanciando l’utility di configurazione di Raspberry Pi.

```
pi@piserver ~ $ sudo raspi-config
```

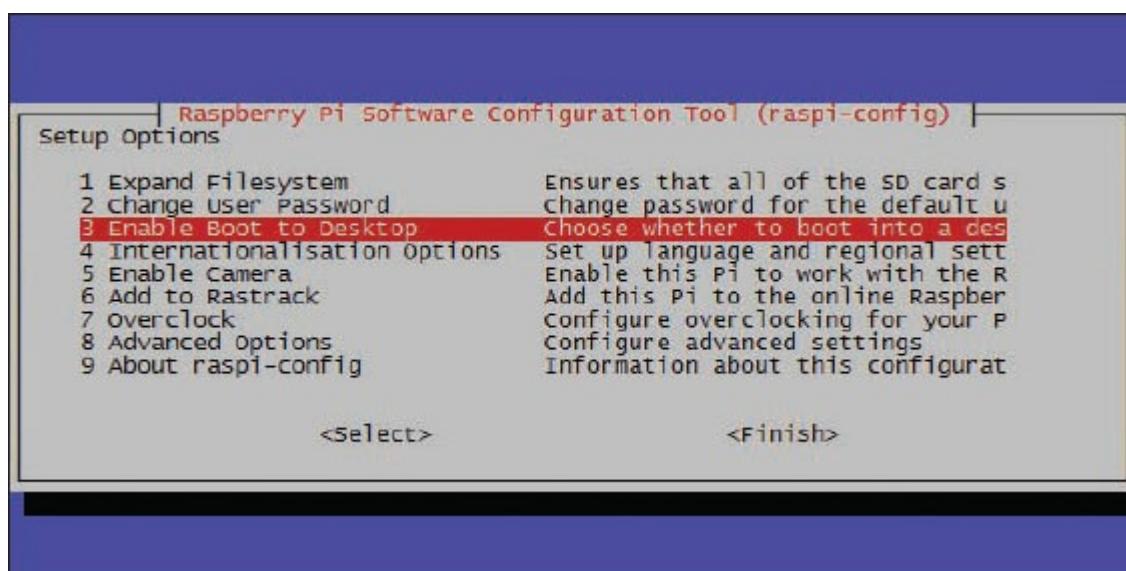


Figura 4.1 - Raspi-config - il menu principale.

Selezionando l’opzione **Enable boot to desktop** del menu principale potremo disabilitare l’avvio automatico dell’interfaccia grafica, semplicemente selezionando **No**.

Oltre a disabilitare l’interfaccia grafica possiamo effettuare altre

operazioni per migliorare le performance e le funzionalità del nostro server. Tutte queste operazioni passano dal menu **Advanced Options di raspi-config**.

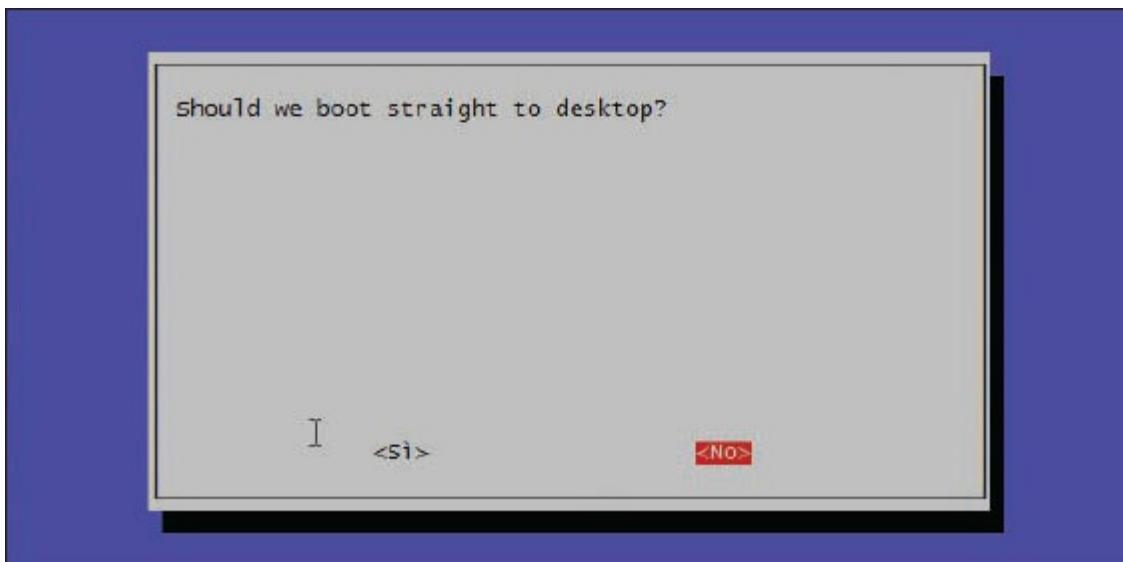


Figura 4.2 - Raspi-config - disabilitare l'avvio automatico dell'interfaccia grafica.

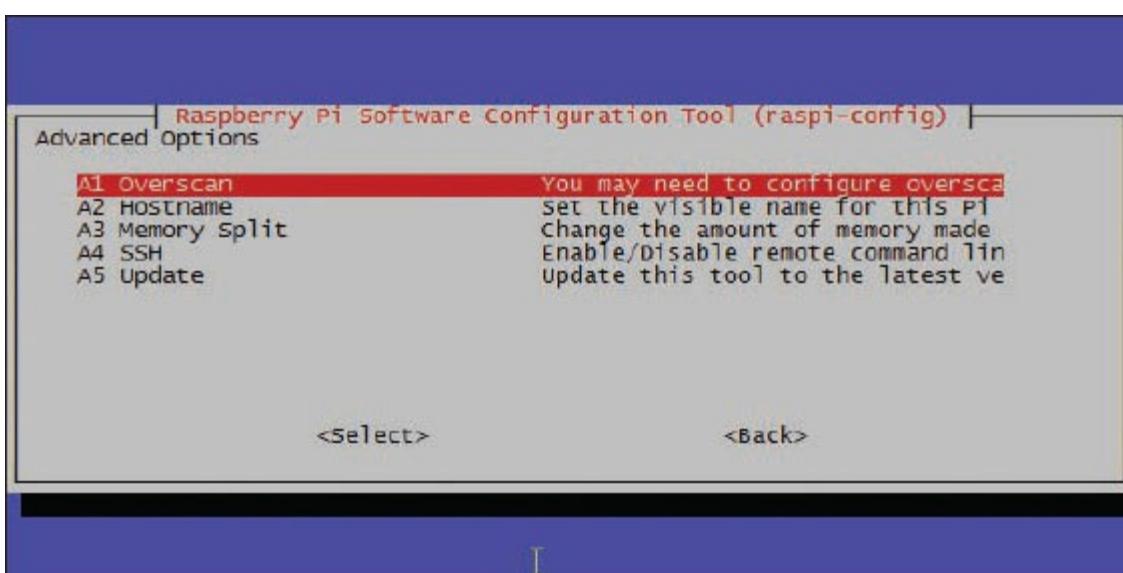


Figura 4.3 - Raspi-config - il menu Advanced Options.

Selezionando la seconda opzione del menu potremo assegnare un nome al nostro server. Questo nome ci sarà utile per identificarlo all'interno della rete. Prima di assegnare il nuovo nome al nostro server, raspi-config ci segnalerà quali sono i caratteri ammessi e quali le restrizioni per rendere il nostro nome di server compatibile con le diverse reti. Il nome scelto dall'autore, con notevole dispensio di fantasia, è `piserver`. Quando troverete questo nome nei diversi esempi dovrete sostituirlo con il nome che avrete scelto per il vostro server domestico.

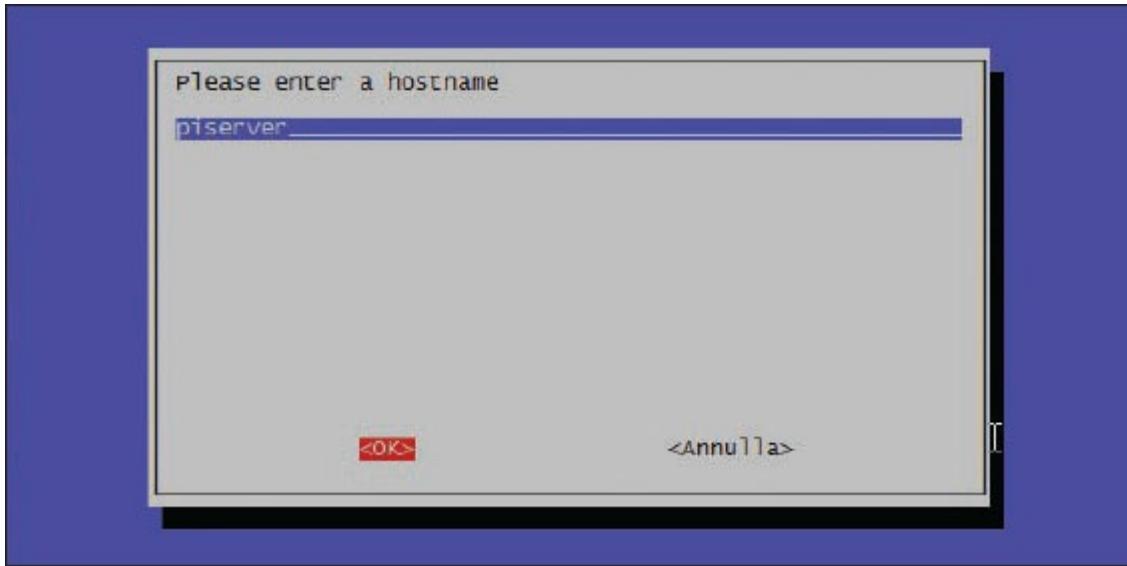


Figura 4.4 - Raspi-config - inserimento dell'hostname.

L'opzione **Memory Split** ci permetterà di ridurre la memoria allocata per l'interfaccia grafica e quindi di dedicare più memoria ai nostri servizi server. La quantità minima di memoria che è possibile riservare alla GPU (il co-processore grafico) è 16 Megabyte.

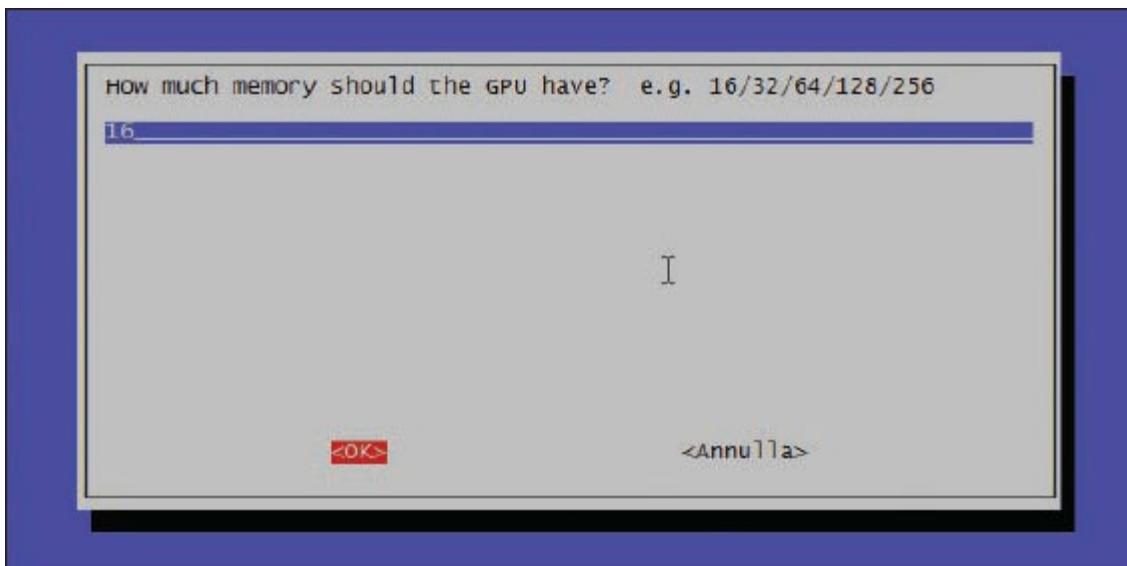


Figura 4.5 - Raspi-config - assegnazione della memoria alla GPU.

La quarta opzione del menu, **SSH**, ci permetterà di abilitare l'accesso al nostro sistema attraverso una shell remota. Questa opzione è fondamentale se avete deciso di rimuovere tastiera e mouse dal vostro Raspberry Pi una volta terminata la sua configurazione come server. Il servizio SSH consente infatti di avere una command line remota utilizzando una connessione crittografata e può essere quindi utilizzato per accedere al vostro server anche attraverso internet.

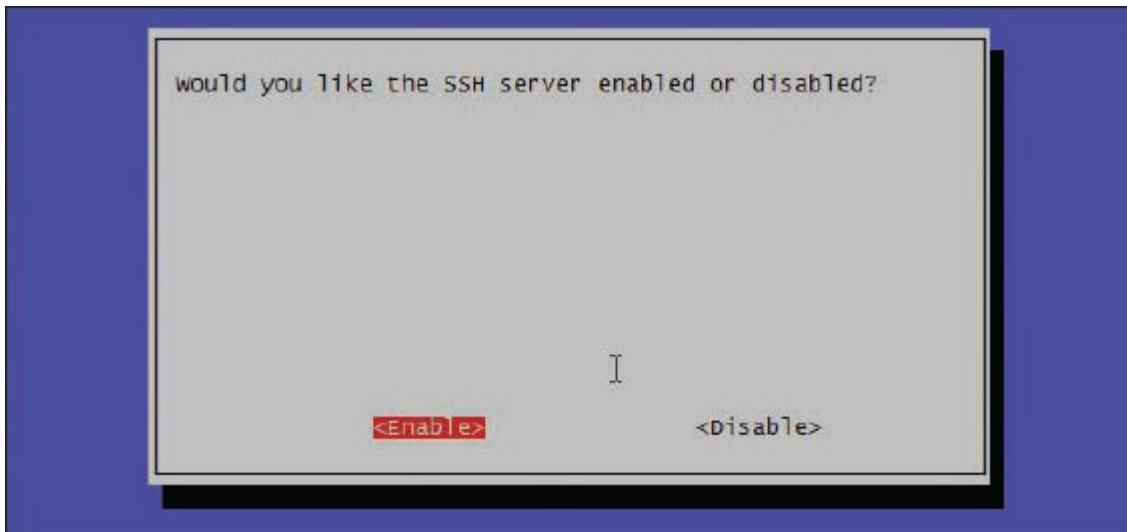


Figura 4.6 - Raspi-config - abilitazione di SSH per l'accesso remoto al server.

A questo punto possiamo riavviare Raspberry Pi e al boot successivo saremo accolti dal prompt di login al posto dell'interfaccia grafica. Basterà inserire username ("pi") e password per accedere alla command line. Se dovesse tornarvi nostalgia della GUI o fosse necessario utilizzare la GUI per qualche operazione particolare, per rilanciare l'interfaccia grafica basta digitare:

```
pi@piserver ~ $ startx
```

L'indirizzo IP

Normalmente gli indirizzi IP all'interno di una rete domestica sono assegnati dal router collegato alla linea internet.

Per scoprire l'indirizzo assegnato al nostro Raspberry Pi possiamo utilizzare il comando `ifconfig`:

```
pi@piserver ~ $ ifconfig
eth0    Link encap:Ethernet HWaddr b8:27:eb:d7:90:d1
        inet addr:192.168.99.20 Bcast:192.168.99.255 Mask:255.255.255.0
              UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
              RX packets:284 errors:0 dropped:1 overruns:0 frame:0
              TX packets:85 errors:0 dropped:0 overruns:0 carrier:0
              collisions:0 txqueuelen:1000
              RX bytes:22065 (21.5 KiB) TX bytes:12032 (11.7 KiB)

lo     Link encap:Local Loopback
        inet addr:127.0.0.1 Mask:255.0.0.0
              UP LOOPBACK RUNNING MTU:16436 Metric:1
              RX packets:0 errors:0 dropped:0 overruns:0 frame:0
              TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
              collisions:0 txqueuelen:0
              RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)
```

```
wlan0 Link encap:Ethernet HWaddr 80:1f:02:af:0b:cc  
inet addr:192.168.99.115 Bcast:192.168.99.255 Mask:255.255.255.0  
UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1  
RX packets:872 errors:0 dropped:883 overruns:0 frame:0  
TX packets:6 errors:0 dropped:0 overruns:0 carrier:0  
collisions:0 txqueuelen:1000  
RX bytes:123811 (120.9 KiB) TX bytes:1200 (1.1 KiB)
```

Il comando ci ritorna una serie di informazioni utili sulle connessioni di rete del nostro sistema. Sul nostro Raspberry Pi possiamo trovare tre tipologie di dispositivi di rete:

- `lo` è il dispositivo loopback. Non si tratta di una rete vera e propria, ma è utilizzato dalle applicazioni per comunicare tra loro;
- `eth0` è la rete LAN, è disponibile direttamente sul Model B e, utilizzando un adattatore USB/Ethernet, anche sul Model A;
- `wlan0` è l'adattatore USB/Wi-Fi che potrà essere utilizzato per le reti wireless.

Il dispositivo `lo` non serve per comunicare al di fuori del nostro sistema, quindi non lo considereremo nel corso di questo capitolo.

Il comando `ifconfig`, tra le altre informazioni, ci fornisce anche l'indirizzo IP del nostro dispositivo. Questa è un'informazione molto importante perché la maggior parte dei servizi richiederà l'IP del nostro server per poter scambiare informazioni e la vedete evidenziata nel blocco relativo alla connessione wireless visto nell'esempio precedente.

```
wlan0 Link encap:Ethernet HWaddr 80:1f:02:af:0b:cc  
inet addr:192.168.99.115 Bcast:192.168.99.255 Mask:255.255.255.0  
UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1  
RX packets:284 errors:0 dropped:1 overruns:0 frame:0  
TX packets:85 errors:0 dropped:0 overruns:0 carrier:0  
collisions:0 txqueuelen:1000  
RX bytes:22065 (21.5 KiB) TX bytes:12032 (11.7 KiB)
```

Il router può cambiare l'indirizzo assegnato a un dispositivo a seguito di uno spegnimento o di un reset.

Se questo non è normalmente un problema per i dispositivi client, potrebbe esserlo per un server, visto che per accedere ai suoi servizi i client devono conoscere il suo indirizzo. Per risolvere questo problema abbiamo due possibilità: fare in modo che il router assegni sempre lo stesso indirizzo al nostro Raspberry Pi oppure forzare l'utilizzo di un indirizzo statico.

La prima soluzione richiede una configurazione del router della nostra rete domestica. Se avete la password di configurazione del vostro router

(purtroppo alcuni provider non la forniscono o la nascondono all'interno della documentazione) e il suo manuale, normalmente scaricabile dal sito del produttore, potete cimentarvi nell'impresa.

Ovviamente non è possibile fornire indicazioni valide per qualsiasi tipologia di router ma, per darvi un'idea di cosa bisogna fare, anche se i comandi e le schermate potrebbero variare a seconda del modello utilizzato, vediamo adesso la sequenza delle operazioni da compiere sul router Netgear dell'autore.

Per prima cosa è necessario collegarsi attraverso il browser al router.

Una volta connessi è possibile accedere alle opzioni della rete locale (LAN).

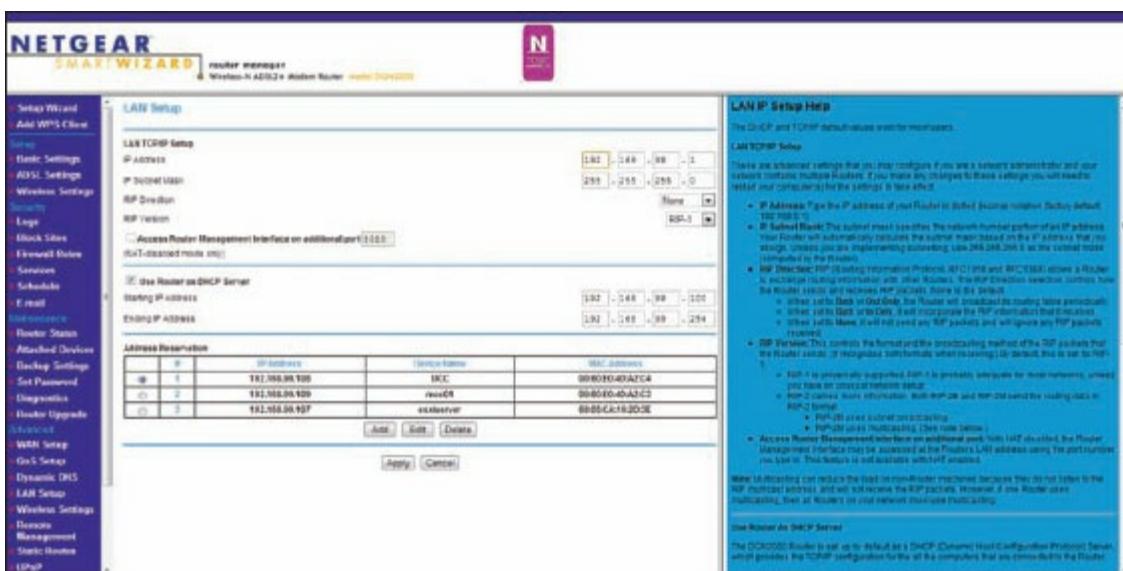


Figura 4.7 - Configurazione LAN di un router Netgear.

In questa pagina è possibile vedere l'indirizzo IP del router (ma per collegarsi al router bisogna conoscerne l'IP, e questo diventa il classico problema dell'uovo e della gallina!) e i parametri della rete (subnet mask ecc.).

Si vede anche che il servizio DHCP (Dynamic Host Configuration Protocol) è abilitato e che quindi il router assegnerà automaticamente gli indirizzi IP ai vari dispositivi nella LAN domestica.

In basso è già presente una lista di dispositivi con indirizzo fisso (la LAN domestica dell'autore combina una quantità preoccupante di gingilli elettronici, per la gioia del suo fornitore di corrente elettrica!). Per aggiungere il nostro Raspberry Pi è sufficiente premere il tasto **Add**. Verrà presentata una seconda lista dei dispositivi che al momento non hanno un indirizzo fisso.

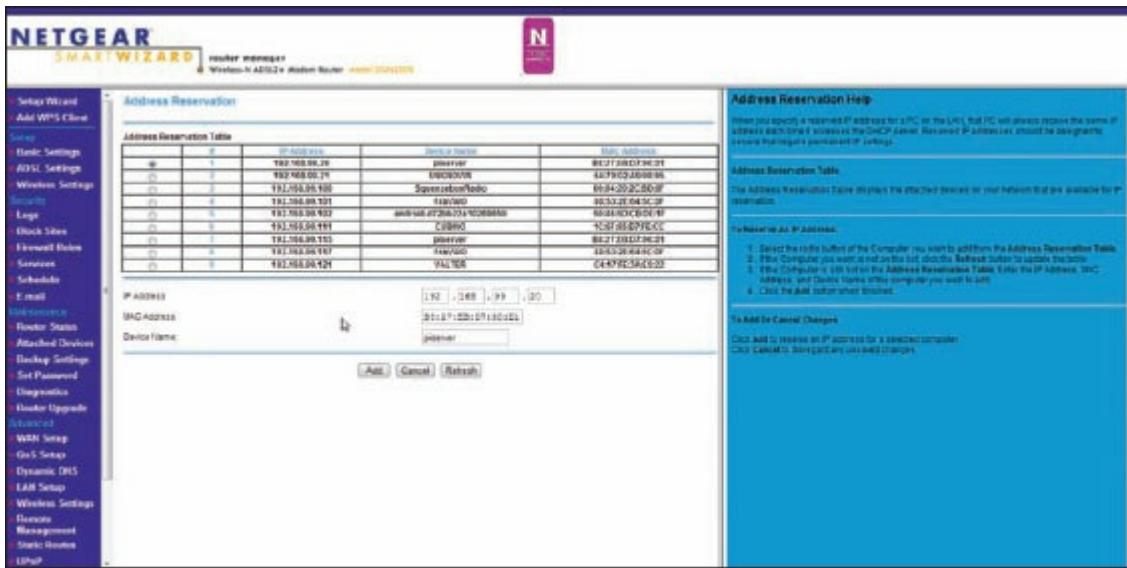


Figura 4.8 - Elenco dei dispositivi presenti in rete.

Questa lista (sempre per la gioia del fornitore di elettricità) è ancora più lunga e tra i diversi device c'è anche il nostro Raspberry Pi (il nome "piserver" è stato definito in fase di installazione). Selezionandolo il suo indirizzo esso viene mostrato nella parte bassa della schermata e, premendo il tasto **Add**, verrà aggiunto ai dispositivi fissi.

Ogni dispositivo è identificato, oltre che da un indirizzo IP, anche da un MAC address. Questo secondo indirizzo è fisso e legato a ogni specifico dispositivo, indipendentemente dalla rete a cui è connesso.

Se non avete accesso alla configurazione del vostro router domestico è possibile configurare per il vostro server un indirizzo IP statico.

Per farlo è necessario modificare il file `/etc/network/interfaces`.

Questo file contiene la configurazione dei dispositivi di rete.

```
pi@piserver ~ $ cat /etc/network/interfaces
auto lo

iface lo inet loopback

iface eth0 inet dhcp

allow-hotplug wlan0
iface wlan0 inet manual
wpa-roam /etc/wpa_supplicant/wpa_supplicant.conf
iface default inet dhcp
```

Il file può essere modificato solo dall'utente root, quindi per poterlo editare dovremo lanciare Nano usando il comando `sudo`:

```
pi@piserver ~ $ sudo nano /etc/network/interfaces
```

Ora dovremo modificare la sezione `eth0` (se stiamo utilizzando la

connessione Ethernet) o la sezione `wlan0` (se stiamo adoperando il Wi-Fi). Dovremo cambiare la riga:

```
iface default inet dhcp
```

in:

```
iface default inet manual
```

Questo, però, non basta per avere un IP statico. Dovremo anche inserire i parametri che vengono normalmente forniti dal server DHCP.

Dovremo scrivere questi parametri al di sotto della linea appena modificata:

```
address 192.168.99.19  
netmask 255.255.255.0  
network 192.168.99.0  
broadcast 192.168.99.255  
gateway 192.168.99.1
```

`Address` è l'indirizzo statico che vogliamo utilizzare per il nostro server. È necessario sceglierlo in modo che non vada a sovrapporsi con quelli assegnati dal router. In molti casi i router iniziano ad assegnare gli indirizzi partendo da numeri bassi, quindi usare un numero alto dovrebbe essere sufficiente per evitare problemi. In altri casi la logica è esattamente opposta e gli indirizzi assegnati partono da un valore alto. La cosa migliore è controllare gli indirizzi IP dei dispositivi che avete già in rete e regolarvi di conseguenza. Non usate gli indirizzi X.X.X.0, X.X.X.1, X.X.X.254 e X.X.X.255 perché sono solitamente utilizzati dal router o riservati come indirizzi di broadcast.

La **netmask** e l'indirizzo **broadcast** sono parte dell'output di `ifconfig`:

```
wlan0 Link encap:Ethernet HWaddr 80:1f:02:af:0b:cc  
      inet addr:192.168.99.115 Bcast:192.168.99.255 Mask:255.255.255.0  
        UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1  
        RX packets:284 errors:0 dropped:1 overruns:0 frame:0  
        TX packets:85 errors:0 dropped:0 overruns:0 carrier:0  
        collisions:0 txqueuelen:1000  
        RX bytes:22065 (21.5 KiB) TX bytes:12032 (11.7 KiB)
```

Il parametro `gateway` coincide con l'indirizzo IP del vostro router. Per recuperare l'indirizzo di gateway possiamo utilizzare il comando `netstat` con il parametro `-r`:

```
pi@piserver ~ $ netstat -r  
Kernel IP routing table  
Destination     Gateway         Genmask        Flags MSS Window irtt Iface
```

```
default      192.168.99.1  0.0.0.0          UG        0 0          0 eth0
192.168.99.0 *           255.255.255.0   U         0 0          0 eth0
```

Il parametro `network` rappresenta l'identificativo della vostra rete e, nel caso di reti LAN con netmask 255.255.255.0, corrisponde alle prime tre cifre dell'IP di un qualsiasi device e uno zero come ultima cifra.

Una volta impostati i parametri il vostro file dovrebbe avere questo aspetto (in questo caso sono stati modificati i parametri di `wlan0`):

```
auto lo

iface lo inet loopback
iface eth0 inet dhcp

allow-hotplug wlan0
iface wlan0 inet manual
wpa-roam /etc/wpa_supplicant/wpa_supplicant.conf
iface default inet static
address 192.168.99.19
netmask 255.255.255.0
network 192.168.99.0
broadcast 192.168.99.255
gateway 192.168.99.1
```

A questo punto è possibile utilizzare (sempre come utente root) i comandi `ifdown` e `ifup` per riavviare il nostro dispositivo di rete:

```
pi@piserver ~ $ sudo ifdown wlan0
RTNETLINK answers: No such process
pi@piserver ~ $ sudo ifup wlan0
ioctl[SIOCSIWAP]: Operation not permitted
ioctl[SIOCSIWENCODEEXT]: Invalid argument
ioctl[SIOCSIWENCODEEXT]: Invalid argument
pi@piserver ~ $
```

Rilanciando `ifconfig` possiamo verificare che i parametri siano stati applicati correttamente (il parametro `address` potrebbe comparire solo dopo qualche secondo; rilanciate `ifconfig` se non appare):

```
pi@piserver ~ $ ifconfig wlan0
wlan0  Link encap:Ethernet  HWaddr 80:1f:02:af:0b:cc
      inet addr:192.168.99.19  Bcast:192.168.99.255  Mask:255.255.255.0
            UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
            RX packets:69 errors:0 dropped:38959 overruns:0 frame:0
            TX packets:2 errors:0 dropped:0 overruns:0 carrier:0
            collisions:0 txqueuelen:1000
            RX bytes:6040975 (5.7 MiB)  TX bytes:4320 (4.2 KiB)
```

Alla conquista dello spazio

Un sistema server può essere utilizzato per condividere file. Se i file in

questione sono file multimediali (immagini, audio, video) è probabile che lo spazio sulla SD che Raspberry Pi utilizza per il sistema operativo inizi a scarseggiare. Raspberry Pi supporta SD Card fino a 32 Gigabyte, ma il costo di questo tipo di memorie è decisamente più elevato in confronto al costo degli hard disk USB esterni, che possono offrire svariate centinaia di Gigabyte o addirittura alcuni Terabyte di spazio per memorizzare i nostri dati.

Collegare un disco esterno USB al nostro Raspberry Pi può essere un buon metodo per ampliare lo spazio a disposizione. Esistono dischi esterni, tipicamente da 2,5 pollici, che possono essere alimentati direttamente da una porta USB. Le porte USB di Raspberry Pi, però, non forniscono la corrente necessaria a questo tipo di dispositivi; sarà quindi necessario procedere al collegamento tramite un hub USB alimentato.

Se non avete bisogno di condividere grosse quantità di dati potrete utilizzare la SD card. Per seguire i passaggi dei prossimi capitoli vi sarà sufficiente creare una cartella `data` nella root del filesystem con il comando:

```
pi@piserver ~ $ sudo mkdir /data  
pi@piserver ~
```

e potrete passare direttamente al prossimo paragrafo. Altrimenti seguite il testo qui di seguito per scoprire come configurare e utilizzare un disco esterno USB.

Per prima cosa è necessario collegare il disco USB a Raspberry Pi. Una volta collegato il disco possiamo vedere se è stato riconosciuto da Raspberry Pi usando il comando `dmesg`. Questo comando mostra gli ultimi messaggi generati dal kernel. Il kernel del nostro sistema Linux è responsabile del riconoscimento e del caricamento del driver corretto per gestire il nostro disco esterno.

```
pi@piserver ~ $ dmesg  
[ 374.887853] usb 1-1.2: new high-speed USB device number 5 using dwc_otg  
[ 374.989583] usb 1-1.2: New USB device found, idVendor=05e3, idProduct=0702  
[ 374.989616] usb 1-1.2: New USB device strings: Mfr=0, Product=1, SerialNumber=  
[ 374.989632] usb 1-1.2: Product: USB TO IDE  
[ 375.001206] usb-storage 1-1.2:1.0: Quirks match for vid 05e3 pid 0702: 520  
[ 375.008597] scsi0 : usb-storage 1-1.2:1.0  
[ 376.009651] scsi 0:0:0:0: Direct-Access Maxtor 6 Y160P0 0811 PQ: 0 ANSI: 0  
[ 376.011936] sd 0:0:0:0: [sda] 320173056 512-byte logical blocks: (163 GB/152 G  
[ 376.013570] sd 0:0:0:0: [sda] Test WP failed, assume Write Enabled  
[ 376.015305] sd 0:0:0:0: [sda] Cache data unavailable  
[ 376.015338] sd 0:0:0:0: [sda] Assuming drive cache: write through
```

```
[ 376.021320] sd 0:0:0:0: [sda] Test WP failed, assume Write Enabled
[ 376.023315] sd 0:0:0:0: [sda] Cache data unavailable
[ 376.023348] sd 0:0:0:0: [sda] Assuming drive cache: write through
[ 376.046662] sda: sda1
[ 376.052310] sd 0:0:0:0: [sda] Test WP failed, assume Write Enabled
[ 376.053930] sd 0:0:0:0: [sda] Cache data unavailable
[ 376.053962] sd 0:0:0:0: [sda] Assuming drive cache: write through
[ 376.053980] sd 0:0:0:0: [sda] Attached SCSI disk
```

Il sistema dovrebbe riconoscere il disco e mostrarne l'id; nel nostro caso (e nella maggior parte dei casi se al vostro Raspberry Pi non sono collegate altre memorie USB) il dispositivo è stato riconosciuto come `sda`.

Per utilizzare il nostro disco esterno con i vari servizi di rete è utile formattarlo con un filesystem Linux.

L'operazione di formattazione cancellerà tutti i dati sul disco. Quindi, se non state usando un disco nuovo appena acquistato, prima di procedere ricordatevi di copiare su un'altra memoria di massa i dati che vi interessano.

Per formattare il nostro disco utilizzeremo il comando `fdisk`.

Dovremo lanciarlo come utente root poiché le operazioni su dischi e partizioni possono compromettere le funzionalità del sistema e portare alla perdita di dati e sono quindi riservate al super user. Se avete copiato tutti i dati che vi interessavano dal vostro hard disk e siete pronti a cancellare tutte le informazioni che contiene, potete procedere con la preparazione delle partizioni.

```
pi@piserver ~ $ sudo fdisk /dev/sda
```

Command (m for help):

Il comando `fdisk` ha un suo prompt per i comandi e con il comando `m` potrete vedere l'elenco dei comandi principali:

```
Command (m for help): m
Command action
  a  toggle a bootable flag
  b  edit bsd disklabel
  c  toggle the dos compatibility flag
  d  delete a partition
  l  list known partition types
  m  print this menu
  n  add a new partition
  o  create a new empty DOS partition table
  p  print the partition table
  q  quit without saving changes
  s  create a new empty Sun disklabel
  t  change a partition's system id
  u  change display/entry units
  v  verify the partition table
```

```
w write table to disk and exit
x extra functionality (experts only)
```

Command (m for help) :

Possiamo usare il comando `o` per creare una nuova tabella delle partizioni; in questo modo rimuoveremo ogni partizione e ogni dato esistente.

```
Command (m for help) : o
```

```
Building a new DOS disklabel with disk identifier 0xe635de37.
Changes will remain in memory only, until you decide to write them.
After that, of course, the previous content won't be recoverable.
```

```
Warning: invalid flag 0x0000 of partition table 4 will be corrected by w(rite)
```

Command (m for help) :

`Fdisk` ci avvisa che le modifiche che stiamo apportando saranno effettive solo quando usciremo dall'applicazione mediante il comando `w`. Prima di uscire dobbiamo creare una nuova partizione per i dati del nostro server. Per farlo possiamo usare il comando `n`, confermando tutti i valori proposti come default premendo **Invio**.

```
Command (m for help) : n
Partition type:
  p primary (0 primary, 0 extended, 4 free)
  e extended
Select (default p):
Partition number (1-4, default 1):
First sector (2048-320173055, default 2048):
Using default value 2048
Last sector, +sectors or +size{K,M,G} (2048-320173055, default 320173055):
Using default value 320173055
```

Command (m for help) :

I numeri riportati come `First` e `Last sector` si riferiscono alla dimensione del disco e nel vostro caso saranno probabilmente diversi da quelli riportati in questo esempio.

In questo modo abbiamo creato una nuova partizione che occupa tutto lo spazio disponibile sul disco.

Ora dovremo configurare il tipo di partizione tramite il comando `t`. L'opzione `L` ci permetterà di visualizzare le tipologie di partizione supportate. Per creare una partizione dati Linux dovremo usare il codice 83:

```
Command (m for help) : t
```

```
Selected partition 1
```

```
Hex code (type L to list codes): L
```

0	Empty	24	NEC DOS	81	Minix / old Lin bf	Solaris
1	FAT12	27	Hidden NTFS Win	82	Linux swap / So c1	DRDOS/sec (FAT-
2	XENIX root	39	Plan 9	83	Linux	c4 DRDOS/sec (FAT-
3	XENIX usr	3c	PartitionMagic	84	OS/2 hidden C:	c6 DRDOS/sec (FAT-
4	FAT16 <32M	40	Venix 80286	85	Linux extended	c7 Syrinx
5	Extended	41	PPC PReP Boot	86	NTFS volume set da	Non-FS data
6	FAT16	42	SFS	87	NTFS volume set db	CP/M / CTOS / .
7	HPFS/NTFS/exFAT	4d	QNX4.x	88	Linux plaintext de	Dell Utility
8	AIX	4e	QNX4.x 2nd part	8e	Linux LVM	df BootIt
9	AIX bootable	4f	QNX4.x 3rd part	93	Amoeba	e1 DOS access
a	OS/2 Boot Manag	50	OnTrack DM	94	Amoeba BBT	e3 DOS R/O
b	W95 FAT32	51	OnTrack DM6 Aux	9f	BSD/OS	e4 SpeedStor
c	W95 FAT32 (LBA)	52	CP/M	a0	IBM Thinkpad hi	eb BeOS fs
e	W95 FAT16 (LBA)	53	OnTrack DM6 Aux	a5	FreeBSD	ee GPT
f	W95 Ext'd (LBA)	54	OnTrackDM6	a6	OpenBSD	ef EFI (FAT-12/16/
10	OPUS	55	EZ-Drive	a7	NeXTSTEP	f0 Linux/PA-RISC b
11	Hidden FAT12	56	Golden Bow	a8	Darwin UFS	f1 SpeedStor
12	Compaq diagnost	5c	Priam Edisk	a9	NetBSD	f4 SpeedStor
14	Hidden FAT16 <3	61	SpeedStor	ab	Darwin boot	f2 DOS secondary
16	Hidden FAT16	63	GNU HURD or Sys	af	HFS / HFS+	fb VMware VMFS
17	Hidden HPFS/NTF	64	Novell Netware	b7	BSDI fs	fc VMware VMKCORE
18	AST SmartSleep	65	Novell Netware	b8	BSDI swap	fd Linux raid auto
1b	Hidden W95 FAT3	70	DiskSecure Mult	bb	Boot Wizard hid	fe LANstep
1c	Hidden W95 FAT3	75	PC/IX	be	Solaris boot	ff BBT
1e	Hidden W95 FAT1	80	Old Minix			

```
Hex code (type L to list codes): 83
```

```
Command (m for help):
```

Infine con il comando **w** andremo a salvare la nostra nuova tabella delle partizioni, terminando anche **fdisk**.

```
Command (m for help): w
```

```
The partition table has been altered!
```

```
Calling ioctl() to re-read partition table.
```

```
Syncing disks.
```

```
pi@piserver ~ $
```

Una volta creata la partizione dovremo formattarla con il comando **mkfs.ext4**.

Utilizzeremo il parametro **-L** per assegnare una label alla nostra partizione.

Assegnare una label è importante perché consentirà di riconoscere il nostro disco dati anche se a Raspberry Pi verranno collegate altre memorie di massa USB.

```
pi@piserver ~ $ sudo mkfs.ext4 /dev/sda1 -L datadisk
mke2fs 1.42.5 (29-Jul-2012)
Etichetta del filesystem=datadisk
OS type: Linux
```

```

Dimensione blocco=4096 (log=2)
Dimensione frammento=4096 (log=2)
Stride=0 blocks, Stripe width=0 blocks
10010624 inodes, 40021376 blocks
2001068 blocks (5.00%) reserved for the super user
Primo blocco dati=0
Maximum filesystem blocks=0
1222 gruppi di blocchi
32768 blocchi per gruppo, 32768 frammenti per gruppo
8192 inode per gruppo
Backup del superblocco salvati nei blocchi:
    32768, 98304, 163840, 229376, 294912, 819200, 884736, 1605632, 2654208,
    4096000, 7962624, 11239424, 20480000, 23887872

Allocating group tables: fatto
Scrittura delle tavole degli inode: fatto
Creating journal (32768 blocks): fatto
Scrittura delle informazioni dei superblocchi e dell'accounting del filesystem: f
pi@piserver ~ $
```

Ora la nostra partizione è pronta, ma per poterla utilizzare dovremo montarla nel nostro filesystem.

Il primo passaggio per poter montare una partizione è creare una cartella, chiamata “mount point”. Il mount point può essere in qualsiasi punto del filesystem. Nel nostro caso andremo a memorizzarlo nella root con il nome `data`. I mount point sono normali cartelle e possono quindi essere creati con il comando `mkdir`.

```
pi@piserver ~ $ sudo mkdir /data
pi@piserver ~
```

Dovremo anche fare in modo che il nostro disco USB venga montato all'avvio; per farlo è necessario modificare il file `/etc/fstab`. Possiamo farlo lanciando Nano come utente root:

```
pi@piserver ~ $ sudo nano /etc/fstab
```

Alla fine del file possiamo aggiungere una riga per configurare il nostro nuovo disco USB.

proc	/proc	proc	defaults	0	0
/dev/mmcblk0p5	/boot	vfat	defaults	0	2
/dev/mmcblk0p6	/	ext4	defaults,noatime	0	1
LABEL=datadisk	/data	ext4	defaults	0	0

Se avete assegnato una label diversa al vostro disco, ricordatevi di riportarla correttamente nel file al posto di `datadisk`.

Ora è possibile montare tutti i filesystem configurati, compreso il

nostro nuovo disco USB.

```
pi@piserver ~ $ sudo mount -a  
pi@piserver ~ $
```

Accesso remoto

Ora che abbiamo configurato il nostro server aggiungendo, se necessario, spazio addizionale per i dati e riducendo le risorse necessarie per l'interfaccia utente, possiamo anche restituire la tastiera e il mouse USB che avevamo preso in prestito nel primo capitolo e controllare il nostro server tramite SSH.

Il protocollo SSH è molto utilizzato per la connessione ai server da remoto ed esistono implementazioni di SSH client (l'applicazione che servirà per controllare il nostro server) su tutti i sistemi operativi più diffusi.

Sui sistemi Linux e OS X un client SSH è già presente; sui sistemi Windows è necessario installare un'applicazione gratuita.

Esistono client SSH gratuiti e a pagamento anche per le diverse piattaforme mobili e potrete quindi controllare il vostro server anche tramite lo smartphone quando siete fuori casa, oppure comodamente seduti sul divano tramite il vostro tablet.

Per collegarsi da OS X è sufficiente lanciare l'applicazione Terminal. Per chi non ha mai usato la command line del suo Mac, il terminale si trova dentro la cartella Utilities della cartella Applicazioni.

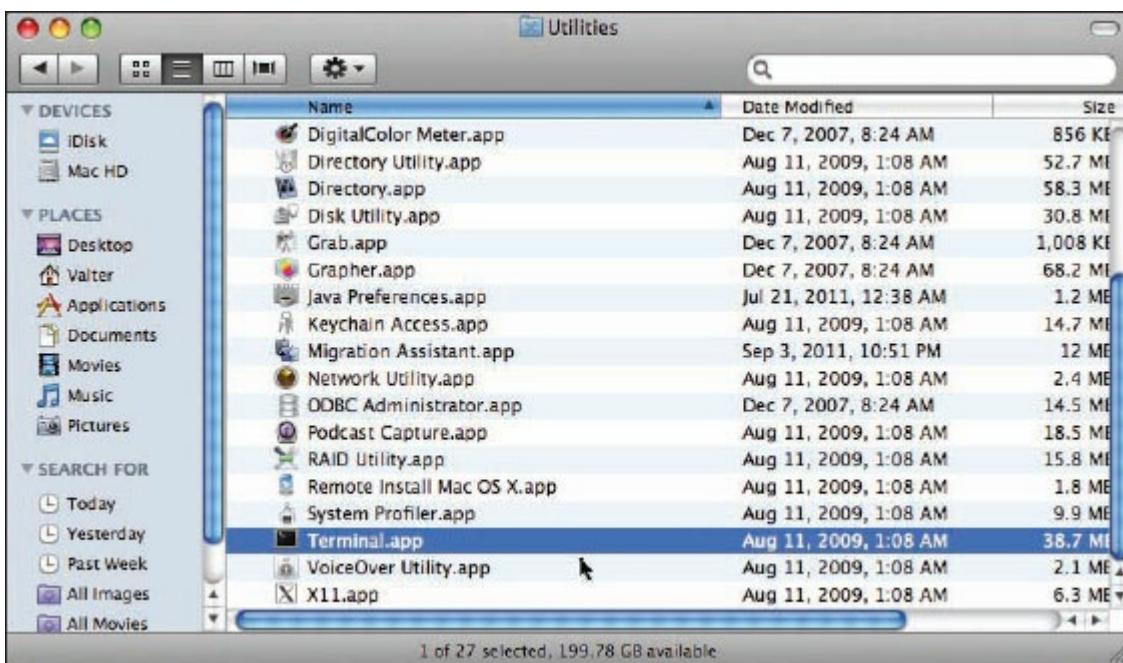


Figura 4.9 - La cartella Applicazioni/Utilities di OS X.

Una volta lanciato Terminale potremo usare la sua command line per lanciare SSH, passando l'indirizzo IP del nostro Raspberry Pi. Notate che la command line di OS X è molto simile a quella di Raspbian e supporta la maggior parte dei comandi discussi nel [Capitolo 3](#).

Anche il sistema operativo del vostro Mac è infatti basato su Unix.

```
VFMac:~ Valter$ ssh pi@192.168.99.20
```

Digiteremo lo stesso comando anche sul prompt di un sistema Linux (anche da un altro Raspberry Pi!).

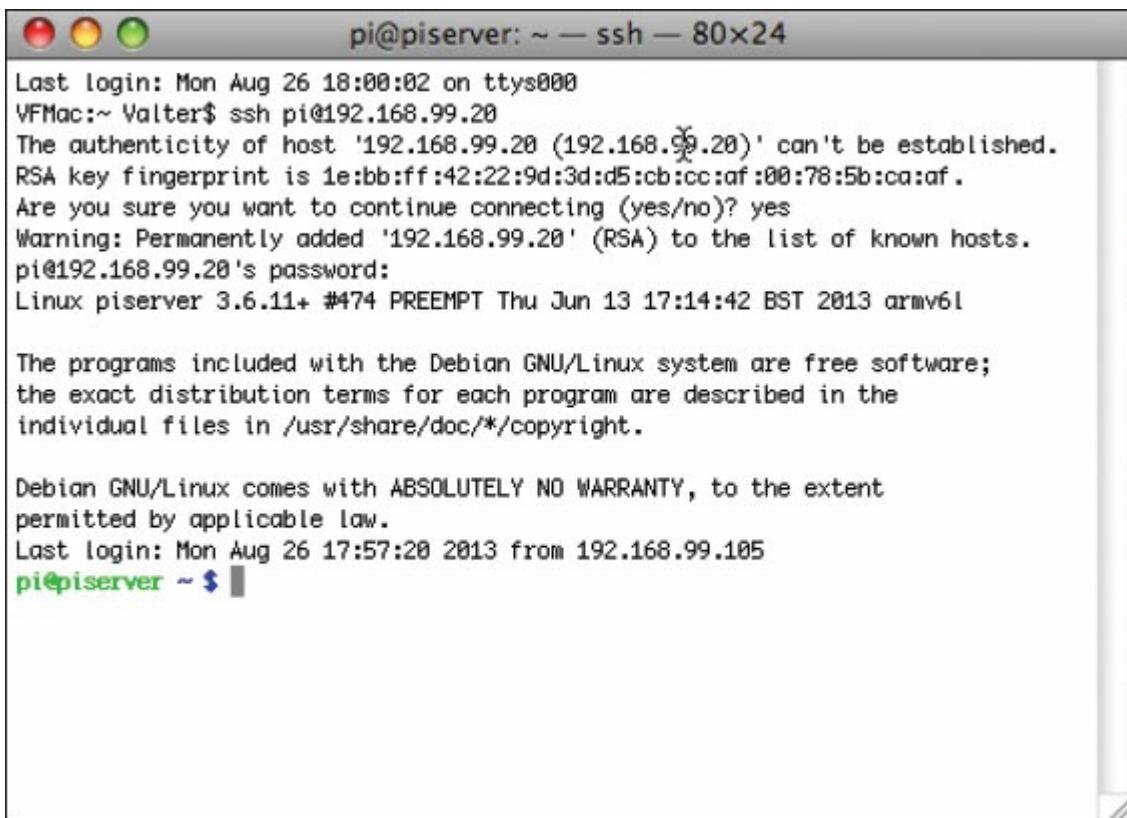
Al primo collegamento il client SSH ci avviserà che il nostro host non è tra quelli conosciuti e ci chiederà se proseguire il collegamento.

```
Last login: Mon Aug 26 18:00:02 on ttys000
The authenticity of host '192.168.99.20 (192.168.99.20)' can't be established.
RSA key fingerprint is 1e:bb:ff:42:22:9d:3d:d5:cb:cc:af:00:78:5b:ca:af.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.99.20' (RSA) to the list of known hosts.
pi@192.168.99.20's password:
Linux piserver 3.6.11+ #474 PREEMPT Thu Jun 13 17:14:42 BST 2013 armv6l
```

```
The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/*copyright.
```

```
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
```

```
Last login: Mon Aug 26 17:57:20 2013 from 192.168.99.105
pi@piserver ~ $
```



```
pi@piserver: ~ — ssh — 80x24
Last login: Mon Aug 26 18:00:02 on ttys000
VFMac:~ Valter$ ssh pi@192.168.99.20
The authenticity of host '192.168.99.20 (192.168.99.20)' can't be established.
RSA key fingerprint is 1e:bb:ff:42:22:9d:3d:d5:cb:cc:af:00:78:5b:ca:af.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.99.20' (RSA) to the list of known hosts.
pi@192.168.99.20's password:
Linux piserver 3.6.11+ #474 PREEMPT Thu Jun 13 17:14:42 BST 2013 armv6l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Mon Aug 26 17:57:20 2013 from 192.168.99.105
pi@piserver ~ $
```

Figura 4.10 - Primo collegamento a Raspberry Pi con Terminale sotto OS X.

Ora siamo collegati al nostro piccolo server e possiamo controllarlo via command line. Per accertarcene basta lanciare il comando `uname` usando il parametro `-n`:

```
pi@piserver ~ $ uname --n
piserver
```

Se avete un sistema Windows potete scaricare **PuTTY** dal sito www.putty.org. PuTTY non necessita di installazione e potrete lanciare direttamente l'eseguibile scaricato dal sito.

Una volta lanciato, PuTTY mostrerà una schermata di opzioni. Sarà necessario inserire l'indirizzo IP del nostro server nel campo **Host Name** e selezionare **SSH** come modalità di connessione. È possibile salvare la connessione, per poterla riattivare più rapidamente ogni volta che rilanciamo PuTTY, scrivendo un nome nel campo **Saved Sessions** e premendo **Save**.

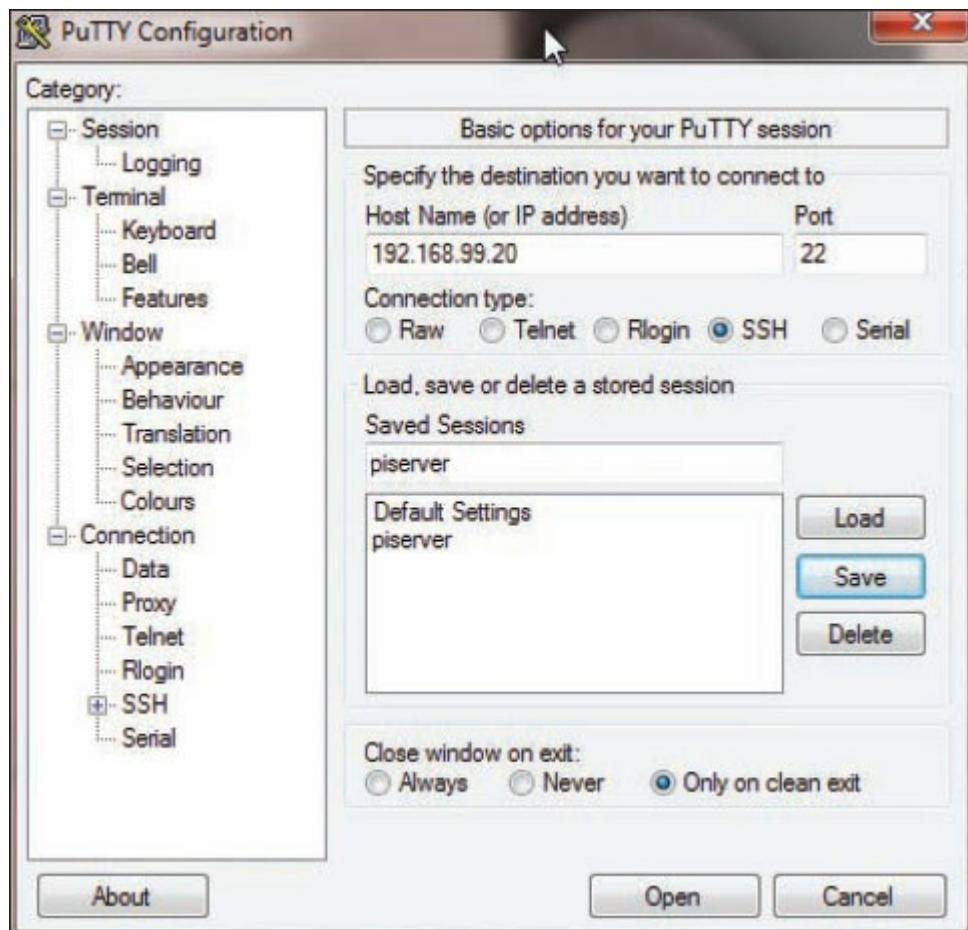


Figura 4.11 - Putty - la finestra principale.

A questo punto, premendo **Open**, PuTTY si collegherà al nostro Raspberry Pi aprendo una command line remota. Al primo collegamento verrà mostrato un messaggio analogo a quello visto per le versioni OS X e Linux, che segnalerà il mancato riconoscimento del nostro host. Premendo il pulsante **Yes** il nostro Raspberry Pi verrà memorizzato tra i dispositivi riconosciuti.



Figura 4.12 - Putty - il nostro host non è riconosciuto.

Anche con PuTTY dovremo inserire il nostro username e la nostra password e avremo accesso alla command line remota.

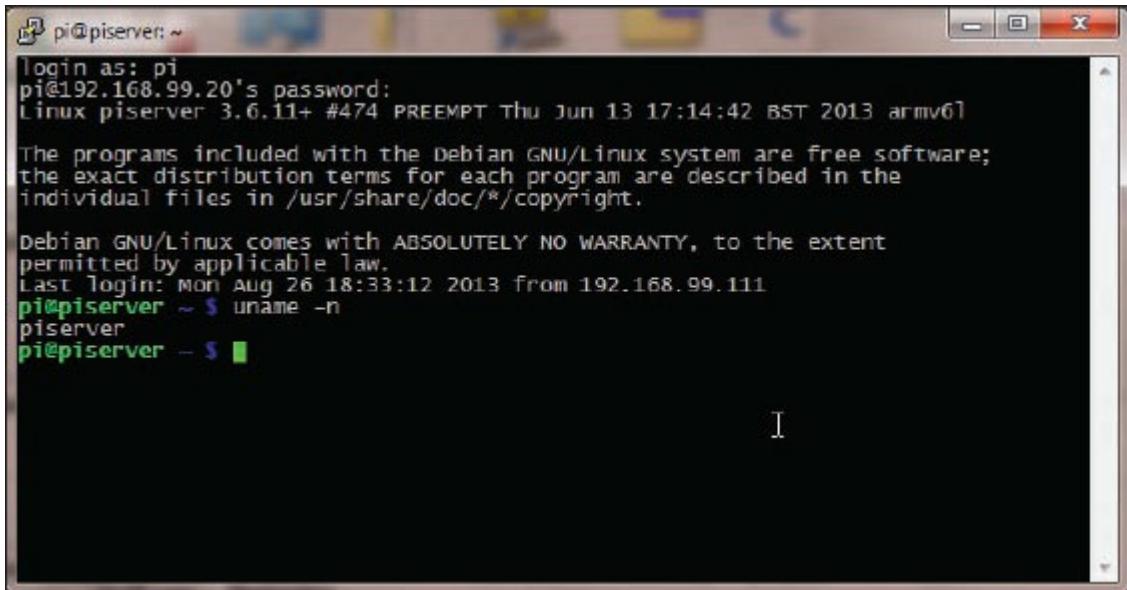


Figura 4.13 - PuTTY - la command line remota.

Ora che siamo in grado di collegarci da remoto possiamo anche staccare tastiera, mouse e cavo video dal nostro Raspberry Pi e lasciarlo tranquillo a lavorare come server domestico.

I servizi che vedremo nei prossimi capitoli potranno essere installati anche attraverso la command line remota.

Condivisione file

Uno degli scopi primari di un server è consentire ai diversi utenti della rete di condividere file. Esistono diversi protocolli adatti allo scopo e, a seconda delle modalità di utilizzo del vostro server, potrete scegliere quali si adattano meglio alle vostre esigenze. Un primo sistema di condivisione file è già supportato dal vostro server. Si chiama SSH File Transfer Protocol (SFTP) ed è basato sullo stesso protocollo SSH che utilizziamo per la command line remota. Esistono molte applicazioni client per il protocollo SFTP, grafiche e a command line, per tutti i sistemi operativi più diffusi. Sui sistemi Windows è possibile utilizzare WinSCP ([winscp.net](#)).

Dopo averlo installato, WinSCP ci chiederà di configurare una connessione.

Nel caso del nostro Raspberry Pi ci basterà selezionare SFTP come protocollo e inserire indirizzo IP, username e password.

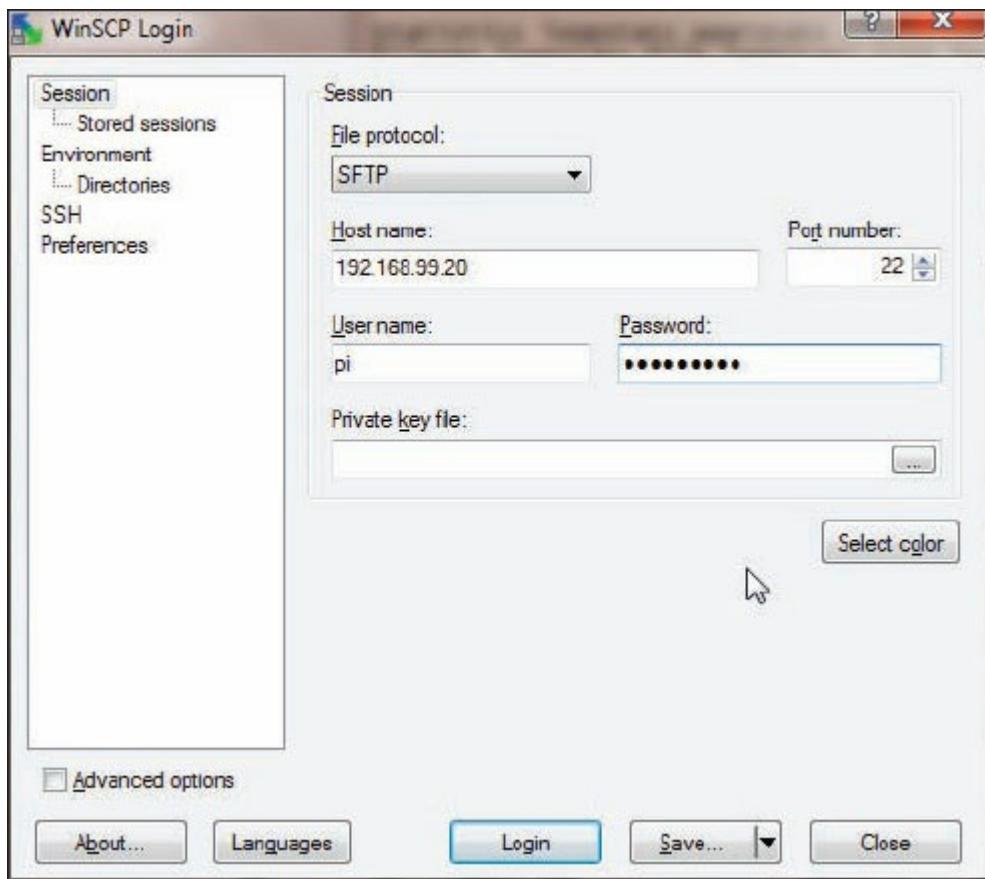


Figura 4.14 - WinSCP - la schermata principale.

Una volta configurati i parametri è possibile salvarli attraverso il pulsante **Save**. WinSCP ci chiederà di assegnare un nome alla nostra connessione.

Una volta tornati alla schermata principale, che a questo punto mostrerà una lista delle connessioni disponibili, potremo selezionare il nostro Raspberry Pi e premere **Login**.

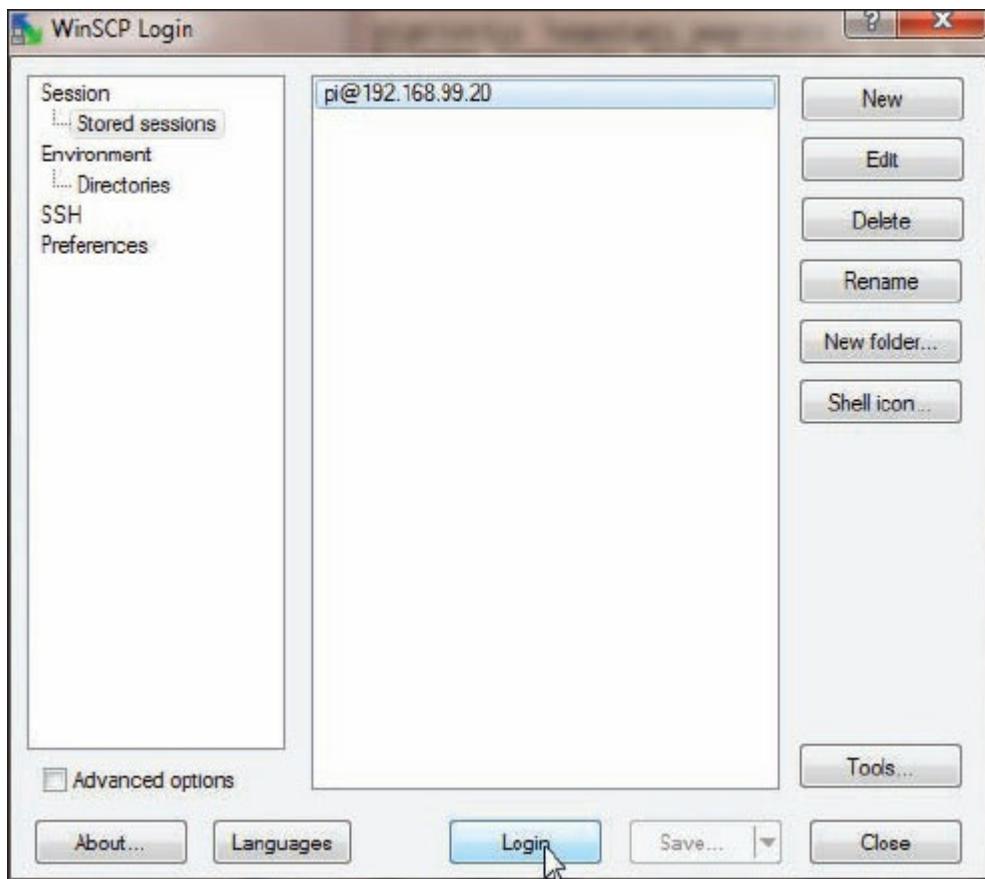


Figura 4.15 - WinSCP - la schermata principale con la lista delle connessioni disponibili.

Se abbiamo deciso di non salvare la password insieme ai parametri di connessione, questa ci verrà richiesta e, se l'abbiamo inserita correttamente, verrà mostrato il file manager.

Il file manager mostra nella parte sinistra della schermata il filesystem del nostro PC e a destra quello del nostro piccolo server. Possiamo trasferire file e anche copiare, rinominare e cancellare file e cartelle sul dispositivo remoto. Per default verrà mostrata la cartella home dell'utente che abbiamo utilizzato per il collegamento, ma potremo modificare tutti i file per cui l'utente ha diritti di accesso validi.

Il protocollo STFP può essere utilizzato per trasferire file, ma non per modificarli direttamente sul server. Per modificare un file dovremo trasferirlo sul nostro PC/Mac, modificarlo e poi ritrasferirlo. Le utility a command line possono essere utilizzate per automatizzare trasferimenti periodici ecc., e anche WinSCP supporta una funzionalità di scripting adatta a questi scopi.

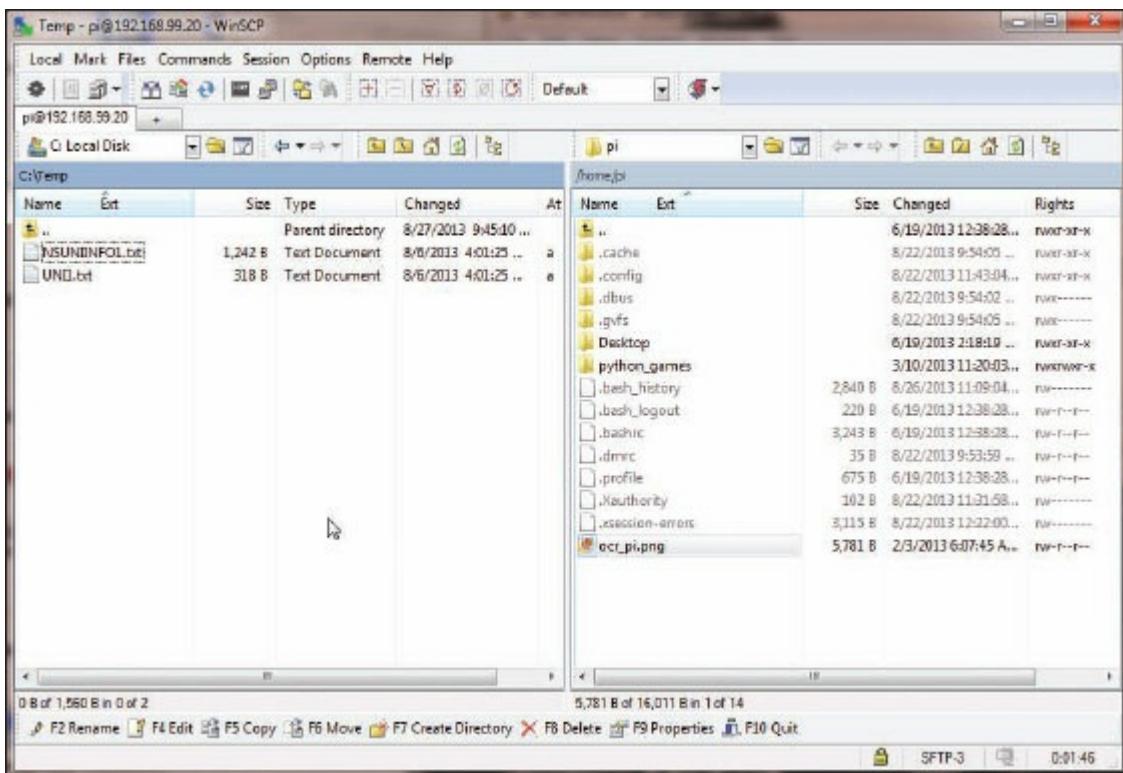


Figura 4.16 - WinSCP - il file manager.

A volte, però, può essere più comodo modificare o anche solo visualizzare i documenti condivisi direttamente sul server, senza doverli trasferire in locale.

Per farlo la cosa migliore è imparare un po' di... **Samba**.

Chi, come l'autore, è poco portato per il ballo non si deve preoccupare. Samba è un servizio di sistema che consente di condividere i file mediante Server Message Block (SMB, basta aggiungere le vocali per avere SaMBa!), il protocollo normalmente utilizzato dalle reti Windows e ormai supportato perfettamente anche da OS X e Linux. Samba non è parte dell'installazione di default di Raspbian, quindi per prima cosa dovremo installarlo sul nostro server.

Prima di installare nuovo software è utile aggiornare quello già installato.

Dovendo usare il nostro sistema come server, anche se solo in ambito domestico, è importante aggiornare i componenti di sistema per evitare che ci siano bug che possono generare problemi di sicurezza. Per farlo possiamo utilizzare il comando `apt-get update`, lanciato come super user. Alla fine dell'aggiornamento è utile fare un reboot per verificare immediatamente che tutto funzioni nel modo corretto.

```
pi@piserver ~ $ sudo apt-get update
Trovato http://raspberrypi.collabora.com wheezy Release.gpg
Scaricamento di:1 http://archive.raspberrypi.org wheezy Release.gpg [490 B]
```

```
Scaricamento di:2 http://mirrordirector.raspbian.org wheezy Release.gpg [490 B]
Trovato http://raspberrypi.collabora.com wheezy Release
Scaricamento di:3 http://mirrordirector.raspbian.org wheezy Release [14,4 kB]
Scaricamento di:4 http://archive.raspberrypi.org wheezy Release [7215 B]
Trovato http://raspberrypi.collabora.com wheezy/rpi armhf Packages
Scaricamento di:5 http://mirrordirector.raspbian.org wheezy/main armhf Packages
[7415 kB]
Scaricamento di:6 http://archive.raspberrypi.org wheezy/main armhf Packages [7214
Ign http://raspberrypi.collabora.com wheezy/rpi Translation-it_IT
Ign http://raspberrypi.collabora.com wheezy/rpi Translation-it
Ign http://archive.raspberrypi.org wheezy/main Translation-it_IT
Ign http://raspberrypi.collabora.com wheezy/rpi Translation-en
Ign http://archive.raspberrypi.org wheezy/main Translation-it
Ign http://archive.raspberrypi.org wheezy/main Translation-en
Trovato http://mirrordirector.raspbian.org wheezy/contrib armhf Packages
Scaricamento di:7 http://mirrordirector.raspbian.org wheezy/non-free armhf Packages
[48,0 kB]
Trovato http://mirrordirector.raspbian.org wheezy/rpi armhf Packages
Ign http://mirrordirector.raspbian.org wheezy/contrib Translation-it_IT
Ign http://mirrordirector.raspbian.org wheezy/contrib Translation-it
Ign http://mirrordirector.raspbian.org wheezy/contrib Translation-en
Ign http://mirrordirector.raspbian.org wheezy/main Translation-it_IT
Ign http://mirrordirector.raspbian.org wheezy/main Translation-it
Ign http://mirrordirector.raspbian.org wheezy/main Translation-en
Ign http://mirrordirector.raspbian.org wheezy/non-free Translation-it_IT
Ign http://mirrordirector.raspbian.org wheezy/non-free Translation-it
Ign http://mirrordirector.raspbian.org wheezy/non-free Translation-en
Ign http://mirrordirector.raspbian.org wheezy/rpi Translation-it_IT
Ign http://mirrordirector.raspbian.org wheezy/rpi Translation-it
Ign http://mirrordirector.raspbian.org wheezy/rpi Translation-en
Recuperati 7493 kB in 1min 5s (115 kB/s)
Lettura elenco dei pacchetti... Fatto
pi@piserver ~ $ sudo reboot
Broadcast message from root@piserver (pts/0) (Tue Aug 27 10:31:25 2013):
```

The system is going down for reboot NOW!

Una volta riavviato il nostro server possiamo ricollegarci via SSH e installare Samba, utilizzando il comando apt-get install.

```
pi@piserver ~ $ sudo apt-get install samba samba-common-bin
Lettura elenco dei pacchetti... Fatto
Generazione albero delle dipendenze
Lettura informazioni sullo stato... Fatto
I seguenti pacchetti saranno inoltre installati:
  tdb-tools
Pacchetti suggeriti:
  openbsd-inetd inet-superserver smbldap-tools ldb-tools ctdb
I seguenti pacchetti NUOVI saranno installati:
  samba samba-common-bin tdb-tools
0 aggiornati, 3 installati, 0 da rimuovere e 13 non aggiornati.
È necessario scaricare 6077 kB di archivi.
Dopo quest'operazione, verranno occupati 36,1 MB di spazio su disco.
Continuare [S/n]? s
Scaricamento di:1 http://mirrordirector.raspbian.org/raspbian/ wheezy/main samba
armhf 2:3.6.6-6 [3344 kB]
```

```
Scaricamento di:2 http://mirrordirector.raspbian.org/raspbian/ wheezy/main samba-common-bin armhf 2:3.6.6-6 [2707 kB]
Scaricamento di:3 http://mirrordirector.raspbian.org/raspbian/ wheezy/main tdb-tools armhf 1.2.10-2 [25,9 kB]
Recuperati 6077 kB in 14s (434 kB/s)
Preconfigurazione dei pacchetti in corso
Selezionato il pacchetto samba non precedentemente selezionato.
(Lettura del database... 62288 file e directory attualmente installati.)
Estrazione di samba (da .../samba_2%3a3.6.6-6_armhf.deb) ...
Selezionato il pacchetto samba-common-bin non precedentemente selezionato.
Estrazione di samba-common-bin (da .../samba-common-bin_2%3a3.6.6-6_armhf.deb) ...
Selezionato il pacchetto tdb-tools non precedentemente selezionato.
Estrazione di tdb-tools (da .../tdb-tools_1.2.10-2_armhf.deb) ...
Elaborazione dei trigger per man-db...
Configurazione di samba (2:3.6.6-6) ...
Generating /etc/default/samba...
Aggiunta del gruppo «sambashare» (GID 110) ...
Fatto.
update-alternatives: viene usato /usr/bin/smbstatus.samba3 per fornire /usr/bin/smbstatus (smbstatus) in modalità automatica
[ ok ] Starting Samba daemons: nmbd smbd.
Configurazione di samba-common-bin (2:3.6.6-6) ...
update-alternatives: viene usato /usr/bin/nmblookup.samba3 per fornire /usr/bin/nmblookup (nmblookup) in modalità automatica
update-alternatives: viene usato /usr/bin/net.samba3 per fornire /usr/bin/net (net) in modalità automatica
update-alternatives: viene usato /usr/bin/testparm.samba3 per fornire /usr/bin/testparm (testparm) in modalità automatica
Configurazione di tdb-tools (1.2.10-2) ...
update-alternatives: viene usato /usr/bin/tdbbackup.tdbtools per fornire /usr/bin/tdbbackup (tdbbackup) in modalità automatica
```

Prima di poter condividere file e cartelle con Samba dobbiamo organizzare il contenuto della nostra cartella `data`. Possiamo ipotizzare di avere una cartella `public` in cui mettere i dati condivisi per tutti gli utenti della nostra rete domestica e, se necessario, cartelle per i diversi utenti della stessa rete.

Per prima cosa dobbiamo creare gli utenti che ci servono tramite il comando `adduser`:

```
pi@piserver ~ $ sudo adduser --no-create-home valter
Aggiunta dell'utente «valter» ...
Aggiunta del nuovo gruppo «valter» (1002) ...
Aggiunta del nuovo utente «valter» (1001) con gruppo «valter» ...
La directory home «/home/valter» non è stata creata.
Immettere nuova password UNIX:
Reimmettere la nuova password UNIX:
passwd: password aggiornata correttamente
Modifica delle informazioni relative all'utente valter
Inserire il nuovo valore o premere INVIO per quello predefinito
Nome completo []: Valter Minute
Stanza n° []:
Numero telefonico di lavoro []:
Numero telefonico di casa []:
```

Altro []:

Le informazioni sono corrette? [S/n] S

L'opzione `--no-create-home` è utile per non creare alcuna cartella utente all'interno di / home. I nostri file condivisi saranno sul disco USB e non sulla SD card, che utilizzeremo solo per il sistema operativo e i file di configurazione.

Dopo aver creato gli utenti necessari dobbiamo assegnare loro una password tramite Samba. Per farlo useremo il comando `smbpasswd`:

```
pi@piserver ~ $ sudo smbpasswd -a valter
New SMB password:
Retype new SMB password:
Added user valter.
```

Se la password impostata è la stessa che utilizzate per il vostro utente Windows il sistema non vi chiederà una password nel momento in cui proverete ad accedere alle cartelle condivise. A questo punto possiamo usare il comando `addgroup` per creare un gruppo in cui metteremo tutti gli utenti che accederanno alla cartella `public`:

```
pi@piserver /data $ sudo addgroup public
```

e aggiungeremo al gruppo gli utenti appena creati usando di nuovo il comando **adduser**:

```
pi@piserver /data $ sudo adduser valter public
Aggiunta dell'utente «valter» al gruppo «public» ...
Aggiunta dell'utente valter al gruppo public
Fatto.
```

Ora è necessario creare le cartelle per i diversi utenti:

```
pi@piserver ~ cd /data
pi@piserver /data $ mkdir public
pi@piserver /data $ mkdir valter
```

Una volta create le cartelle potremo assegnare i corrispondenti diritti di accesso.

La cartella `public` dovrà essere assegnata al gruppo `public`, usando il comando `chgrp`. Dovremo inoltre modificarne i diritti di accesso dando agli utenti del gruppo `public` il diritto di scrivere, usando il comando `chmod` già visto nel capitolo precedente.

```
pi@piserver /data $ sudo chgrp public public/
pi@piserver /data $ sudo chmod g+w public
```

Per le cartelle private dei diversi utenti possiamo utilizzare il comando `chown` e assegnarle all'utente corrispondente:

```
pi@piserver /data $ sudo chown -R valter valter/
```

A questo punto dobbiamo configurare Samba per consentire l'accesso alle directory che abbiamo appena creato.

Per configurare Samba dovremo modificare il file `/etc/samba/smb.conf` usando Nano.

Prima di modificare il file è prudente farne una copia di backup:

```
pi@piserver /data $ sudo cp /etc/samba/smb.conf /etc/samba/smb.conf.bak
```

Samba prevede moltissime opzioni e descriverle in modo esaustivo in questo testo sarebbe impossibile; se volete consultare la documentazione ufficiale, potete utilizzare il comando `info`.

```
pi@piserver ~ $ info samba
```

Nel nostro caso ci limiteremo ad aggiungere due cartelle condivise: una pubblica in cui tutti gli utenti possano creare, modificare e leggere file e cartelle, e una riservata a uno specifico utente che sarà l'unico a potervi accedere.

È possibile, ovviamente, anche creare cartelle condivise solo tra alcuni utenti o cartelle in sola lettura dove mettere file di documentazione; alla fine di questo capitolo saprete come farlo e potrete sperimentare.

Per editare il file di configurazione possiamo utilizzare Nano.

```
pi@piserver /data $ sudo nano /etc/samba/smb.conf
```

Il file è diviso in sezioni identificate da un nome tra parentesi quadre. I parametri generali sono nella sezione `[global]`; non dovremo modificare questi parametri per attivare la condivisione dei file.

La prima sezione da modificare è `[homes]`. Per trovarla possiamo usare la funzione di ricerca di Nano premendo **Ctrl+W** e inserendo la stringa `[homes]`. Questa sezione consente di esportare le cartelle home dei vari utenti. Non abbiamo creato una cartella home per gli utenti di Samba, quindi questa sezione può essere commentata aggiungendo un punto e virgola (;) all'inizio di ogni riga.

Questo dovrebbe essere l'aspetto della sezione `[homes]` dopo le modifiche:

```
; [homes]
```

```

; comment = Home Directories
; browseable = no

# By default, the home directories are exported read-only. Change the
# next parameter to 'no' if you want to be able to write to them.
; read only = yes

# File creation mask is set to 0700 for security reasons. If you want to
# create files with group=rw permissions, set next parameter to 0775.
; create mask = 0700

# Directory creation mask is set to 0700 for security reasons. If you want to
# create dirs. with group=rw permissions, set next parameter to 0775.
; directory mask = 0700

# By default, \\server\username shares can be connected to by anyone
# with access to the samba server.
# The following parameter makes sure that only "username" can connect
# to \\server\username
# This might need tweaking when using external authentication schemes
; valid users = %S

```

A questo punto possiamo spostarci alla fine del file e aggiungere le sezioni relative alle condivisioni che vogliamo invece abilitare.

```

[public]
path=/data/public
comment=File condivisi
read only=No
valid users=@public

[valter]
path=/data/valter
comment=I file di Valter
read only=No
valid users=valter

```

Per ogni cartella dovremo fornire il path, un commento descrittivo, indicare che la cartella non sarà in modalità sola lettura (o indicarlo nel caso di cartelle che non vogliamo possano essere modificate). Il parametro `valid users`, infine, ci consentirà di indicare gli utenti o i gruppi che possono accedere alla cartella condivisa. Nel caso dei gruppi il nome va prefissato con il carattere at (@).

Una volta modificato il file di configurazione è necessario riavviare Samba per attivare le modifiche apportate al file di configurazione.

```

pi@piserver /data $ sudo /etc/init.d/samba restart
[ ok ] Stopping Samba daemons: nmbd smbd.
[ ok ] Starting Samba daemons: nmbd smbd.

```

A questo punto potremo accedere ai nostri file da un PC Windows, da

un Mac o da altri sistemi che supportino il protocollo SMB. Se avete un PC con Windows XP, Vista, 7 o 8 potete semplicemente aprire la lista delle risorse di rete. Raspberry Pi apparirà insieme alle altre macchine del vostro network.

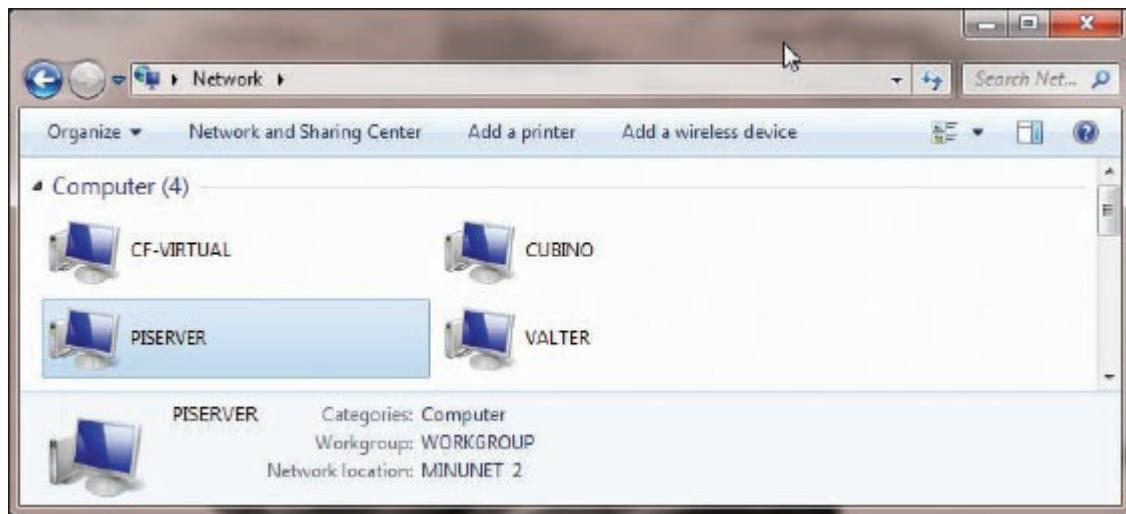


Figura 4.17 - Windows 7 - il nostro server appare tra le risorse di rete.

Con un doppio clic sarà possibile vedere le cartelle condivise che corrisponderanno a quelle inserite nel file di configurazione.



Figura 4.18 - Windows 7 - le cartelle accessibili.

Selezionando una cartella con un doppio clic il sistema ci chiederà una password (a meno che il nome utente e la password che abbiamo creato su Raspberry Pi non corrispondano a quelli con cui abbiamo effettuato il login sul PC), poi aprirà la cartella in cui potremo direttamente creare e modificare i nostri file.

Se utilizzate un Mac i passaggi da effettuare sono leggermente diversi.

Per accedere a una cartella condivisa è necessario selezionare l'opzione **Collega al server** del menu del Finder e inserire l'indirizzo della cartella a cui vogliamo accedere con il prefisso smb://.

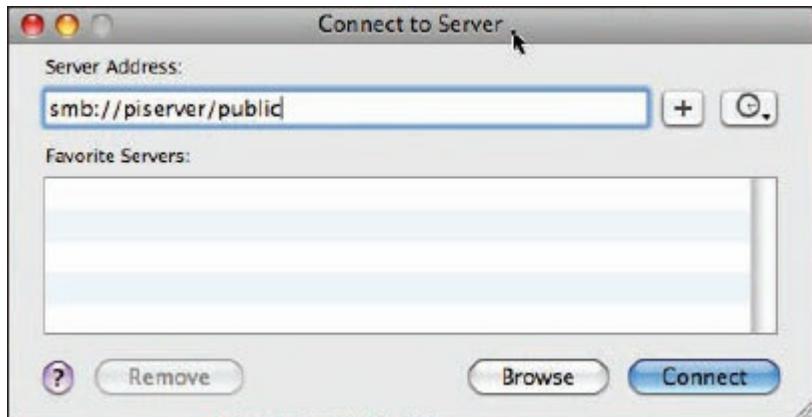


Figura 4.19 - OS X - la finestra Collega al server.

Una volta inserito l'indirizzo il sistema ci chiederà di inserire le credenziali per il login e, se avremo inserito la nostra password in modo corretto, presenterà la cartella condivisa.



Figura 4.20 - OS X - inserimento delle credenziali per l'accesso alla cartella.

Il test del DLNA

Spesso i server domestici sono utilizzati per condividere musica, immagini e video. Se vogliamo utilizzare questi file con PC e Mac possiamo tranquillamente utilizzare la condivisione via SMB. Se vogliamo invece utilizzarli con dispositivi più semplici come tablet, telefonini o smart TV è necessario aggiungere una nuova funzionalità al nostro Raspberry Pi: il server DLNA.

Digital Living Network Alliance è un'organizzazione controllata dai maggiori produttori di dispositivi multimediali e si occupa di definire degli standard che consentano a questi dispositivi di collaborare in una rete domestica nel modo più semplice e trasparente.

Per supportare questa funzionalità dovremo installare un altro pacchetto opzionale di Raspbian: **Minidlna**.

Possiamo farlo utilizzando di nuovo il comando `apt-get install`:

```
pi@piserver /data $ sudo apt-get install minidlna
Lettura elenco dei pacchetti... Fatto
Generazione albero delle dipendenze
Lettura informazioni sullo stato... Fatto
I seguenti pacchetti saranno inoltre installati:
  libavcodec53 libavformat53 libavutil51 libdirac-encoder0 libgsm1 libmp3lame0
  libschoroedinger-1.0-0 libspeex1 libtheora0 libval libvpx1 libx264-123
  libxvidcore4
Pacchetti suggeriti:
  speex
I seguenti pacchetti NUOVI saranno installati:
  libavcodec53 libavformat53 libavutil51 libdirac-encoder0 libgsm1 libmp3lame0
  libschoroedinger-1.0-0 libspeex1 libtheora0 libval libvpx1 libx264-123
  libxvidcore4 minidlna
0 aggiornati, 14 installati, 0 da rimuovere e 13 non aggiornati.
È necessario scaricare 5419 kB di archivi.
Dopo quest'operazione, verranno occupati 12,7 MB di spazio su disco.
Continuare [S/n]? s
Scaricamento di:1 http://mirrordirector.raspbian.org/raspbian/ wheezy/main
libavutil51 armhf 6:0.8.6-1+rpi1 [91,1 kB]
...
Configurazione di libavformat53:armhf (6:0.8.6-1+rpi1)...
Configurazione di minidlna (1.0.24+dfsg-1)...
```

Anche MiniDLNA deve essere configurato e, come nel caso di Samba, la configurazione è memorizzata in un file di testo. Il file da modificare è `/etc/minidlna.conf`. Possiamo utilizzare Nano per aprirlo e modificarlo.

Anche in questo caso potrebbe essere utile fare prima una copia di backup.

Il file è, fortunatamente, piuttosto semplice e abbondantemente commentato. Qui sotto è riportato con in grassetto i parametri modificati. Fate attenzione poiché alcuni parametri erano commentati (con il simbolo cancelletto) e sono stati scommentati nel file modificato.

```
# This is the configuration file for the Minidlna daemon, a DLNA/UPnP-AV media
# server.
#
# Unless otherwise noted, the commented out options show their default value.
#
# On Debian, you can also refer to the minidlna.conf(5) man page for
# documentation about this file.
# Path to the directory you want scanned for media files.
#
# This option can be specified more than once if you want multiple directories
# scanned.
#
# If you want to restrict a media_dir to a specific content type, you can
```

```

# prepend the directory name with a letter representing the type (A, P or V),
# followed by a comma, as so:
# * "A" for audio (eg. media_dir=A,/var/lib/minidlna/music)
# * "P" for pictures (eg. media_dir=P,/var/lib/minidlna/pictures)
# * "V" for video (eg. media_dir=V,/var/lib/minidlna/videos)
#
# WARNING: After changing this option, you need to rebuild the database. Either
#           run minidlna with the '-R' option, or delete the 'files.db' file
#           from the db_dir directory (see below).
#           On Debian, you can run, as root, 'service minidlna force-reload' instead
media_dir=/data/dlna

# Path to the directory that should hold the database and album art
# cache. db_dir=/data/minidlna

# Path to the directory that should hold the log file.
log_dir=/data/minidlna

# Minimum level of importance of messages to be logged.
# Must be one of "off", "fatal", "error", "warn", "info" or "debug".
# "off" turns off logging entirely, "fatal" is the highest level of importance
# and "debug" the lowest.
#log_level=warn

# Use a different container as the root of the directory tree presented to
# clients. The possible values are:
# * "." - standard container
# * "B" - "Browse Directory"
# * "M" - "Music"
# * "P" - "Pictures"
# * "V" - "Video"
# if you specify "B" and client device is audio-only then "Music/Folders" will be
used as root
#root_container=.

# Network interface(s) to bind to (e.g. eth0), comma delimited.
#network_interface=

# IPv4 address to listen on (e.g. 192.0.2.1).
#listening_ip=

# Port number for HTTP traffic (descriptions, SOAP, media transfer).
port=8200

# URL presented to clients.
# The default is the IP address of the server on port 80.
#presentation_url=http://example.com:80

# Name that the DLNA server presents to clients.
#friendly_name=

# Serial number the server reports to clients.
serial=12345678

# Model name the server reports to clients.
#model_name=Windows Media Connect compatible (MiniDLNA)

```

```

# Model number the server reports to clients.
model_number=1

# Automatic discovery of new files in the media_dir directory.
inotify=yes

# List of file names to look for when searching for album art. Names should be
# delimited with a forward slash ("/").
album_art_names=Cover.jpg/cover.jpg/AlbumArtSmall.jpg/albumartsmall.jpg/AlbumArt.
jpg/albumart.jpg/Album.jpg/album.jpg/Folder.jpg/folder.jpg/Thumb.jpg/thumb.jpg

# Strictly adhere to DLNA standards.
# This allows server-side downscaling of very large JPEG images, which may
# decrease JPEG serving performance on (at least) Sony DLNA products.
#strict_dlna=no

# Support for streaming .jpg and .mp3 files to a TiVo supporting HMO.
#enable_tivo=no

# Notify interval, in seconds.
#notify_interval=895

# Path to the MiniSSDPd socket, for MiniSSDPd support.
#minissdpdsocket=/run/minissdpd.sock

```

I parametri modificati, in grassetto nel listato, riguardano le directory in cui MiniDLNA memorizza i dati da condividere (`media_dir`), il database dei contenuti presenti sul server (`db_dir`) e i suoi file di log (`log_dir`). È stato inoltre modificato il parametro `inotify` che normalmente è commentato, questo per fare in modo che eventuali file aggiunti alle cartelle dei contenuti vengano indicizzati in automatico, senza doversi collegare al server.

Prima di riavviare il nostro server DLNA è necessario creare le cartelle per i contenuti.

```

pi@piserver ~ cd /data
pi@piserver /data $ mkdir dlna
pi@piserver /data $ mkdir minidlna
pi@piserver /data $ chmod a+w dlna
pi@piserver /data $ chmod a+w minidlna

```

Ora possiamo riavviare il server DLNA:

```

pi@piserver /data $ sudo service minidlna force-reload
[ ok ] Restarting DLNA/UPnP-AV media server: minidlna.

```

A questo punto, per verificare il funzionamento del nostro server, possiamo utilizzare un qualsiasi browser e puntarlo all'indirizzo IP del server sulla porta IP 8200, usando un indirizzo nel formato `http:<indirizzoIP>:8200`. Se tutto è stato configurato correttamente,

MiniDLNA ci mostrerà il numero di file indicizzati.



Figura 4.21 - MiniDLNA visto da un browser.

Ovviamente i contatori saranno a zero perché non sono stati copiati file nella cartella `/data/dlna`. Possiamo condividere questa cartella attraverso Samba oppure copiarci dei file attraverso STFP e vedremo che i contatori inizieranno ad aumentare.

Indicizzare parecchi file potrebbe richiedere un po' di tempo.

Come è possibile accedere ai nostri contenuti tramite DLNA? Le modalità cambiano da dispositivo a dispositivo e in alcuni casi è necessario installare applicazioni ad-hoc. Se avete un PC Windows, il vostro nuovo server DLNA apparirà tra le risorse di rete con un'icona diversa da quella utilizzata per i dispositivi che condividono file.

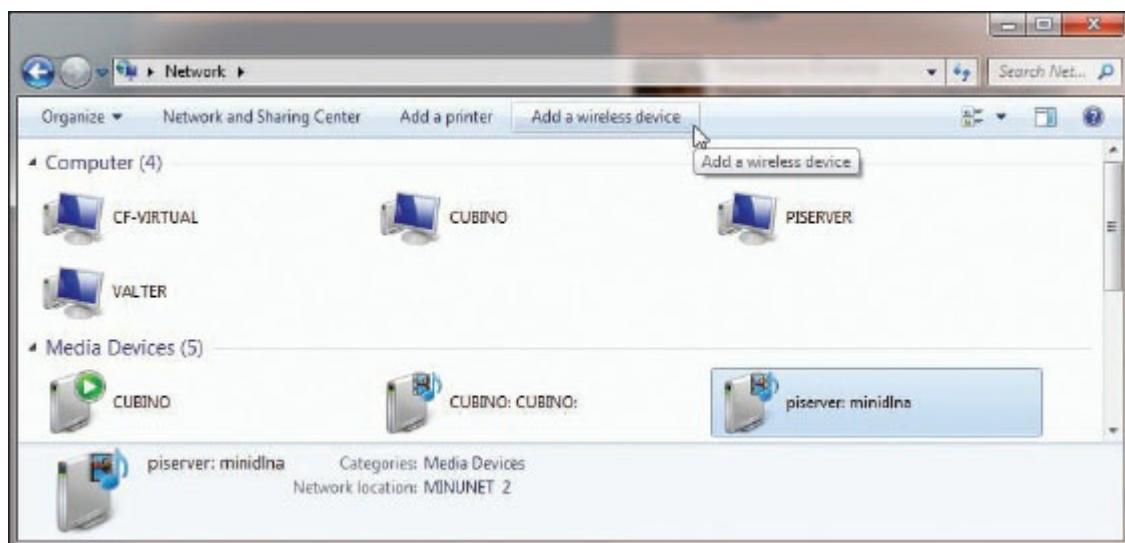


Figura 4.22 - Windows 7 - il server DLNA in Risorse di rete.

Con un doppio clic verrà aperto Windows Media Player e potrete

navigare tra i contenuti (musica, immagini e video) messi a disposizione dal vostro nuovo server DLNA.

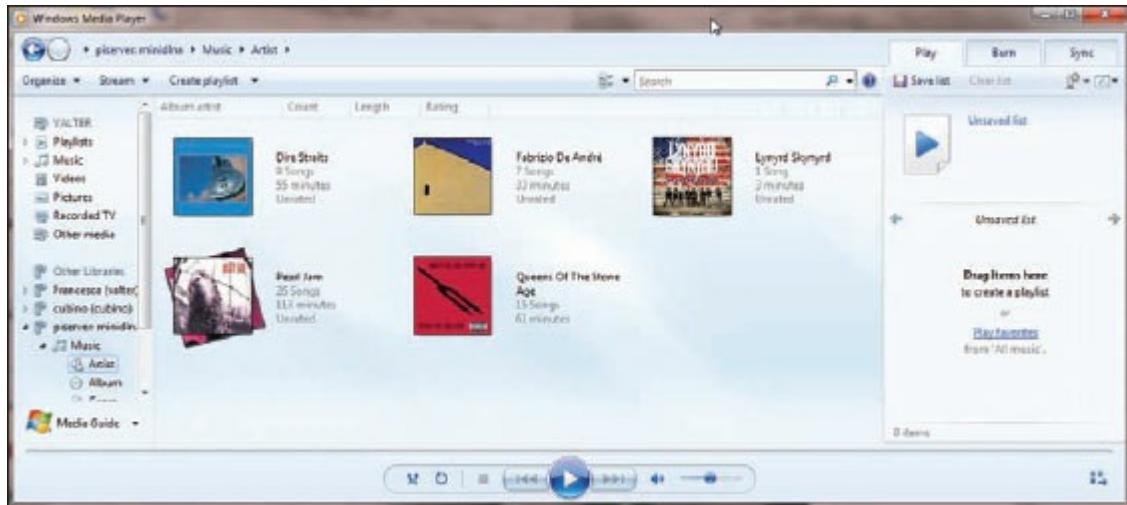


Figura 4.23 - Windows Media Player - i contenuti del nostro server DLNA.

OS X non prevede un client DLNA nativo, ma è possibile trovare diverse applicazioni, gratuite o a pagamento, senza dimenticare la possibilità di accedere comunque ai nostri file multimediali utilizzando le cartelle condivise da Samba.

È possibile utilizzare DLNA anche per fruire dei nostri contenuti multimediali su console di gioco (PS3 o XBOX), smartphone e tablet con i diversi sistemi operativi.

Questo vi consentirà di ascoltare la vostra musica o guardare i vostri film preferiti su diversi sistemi, senza preoccuparvi di trasferire file da un sistema all'altro.

Abbiamo visto come Raspberry Pi possa essere un ottimo server domestico e nei prossimi capitoli esploreremo altre opportunità per incrementarne le funzionalità, ma ora è venuto il momento di utilizzare il nostro piccolo PC come media player. Questo sarà infatti l'argomento del prossimo capitolo.

Raspberry Pi media center

Grazie alle **capacità multimediali**, ai bassi consumi e alla silenziosità Raspberry Pi può diventare un eccellente **media center**, che renderà il televisore un po' più intelligente.

Come abbiamo visto nel [Capitolo 1](#), il processore di Raspberry Pi integra un'unità grafica abbastanza potente, soprattutto per quanto riguarda la riproduzione di contenuti video, anche ad alte risoluzioni come il Full HD (1980 x 1080 pixel) ormai diffuso sui televisori di ultima generazione.

Nel capitolo precedente abbiamo imparato a condividere contenuti multimediali da un Raspberry Pi utilizzato come server. In questo impareremo a fruire degli stessi contenuti.

Raspbmc

Un media center collegato alla televisione richiede un'interfaccia grafica particolare (l'utente non è a pochi centimetri, come nel caso di un normale monitor) e un set di applicazioni dedicate. Impostare Raspbian per utilizzarlo come media center richiederebbe parecchie modifiche alla configurazione, modifiche che renderebbero, di fatto, complicato l'utilizzo di altre applicazioni grafiche.

Per questo sono state realizzate due distribuzioni dedicate, alternative a Raspbian: OpenElec e Raspbmc. Entrambe sono basate su XBMC. Questo software è nato per essere utilizzato per convertire le XBOX prima versione in media player e infatti l'acronimo significa XBox Media Center. L'idea di utilizzare un hardware molto diffuso e dal costo relativamente contenuto, collegato alla TV, per sfruttare contenuti in rete ha avuto un grande successo e XBMC è stato portato su diverse piattaforme, non ultimo Raspberry Pi.

In questo testo useremo Raspbmc perché è la distribuzione più simile a Raspbian e ci consentirà di applicare diversi dei concetti già visti nel libro. Tuttavia, se avete deciso che il posto del vostro Raspberry Pi (o di uno dei vostri Raspberry Pi!) è il salotto, vi consiglio di dare un'occhiata anche a OpenElec; troverete molte similitudini ma anche qualche differenza che, magari, ve la farà preferire.

XBMC è un sistema abbastanza giovane e molte delle funzionalità sono ancora a un livello sperimentale. Preparatevi a qualche riavvio inaspettato o a qualche comportamento strano del sistema durante la fase di configurazione. Durante l'utilizzo il sistema è stabile e questo consente una fruizione confortevole dal divano di casa.

Per usare Raspbmc dovremo ripetere la procedura di installazione tramite NOOBS. Se non volete perdere il lavoro fatto su Raspbian potete utilizzare una nuova SD card e ripetere i passaggi descritti nel [Capitolo 2](#). Allo stesso modo, se avete invece deciso di riutilizzare la SD che avete usato nei primi capitoli, potete ripartire dallo step di formattazione, ripetendo i passaggi descritti, selezionando Raspbmc dal menu principale di NOOBS.

Anche in questo caso verranno prima copiati i file e, terminata questa fase, il sistema verrà riavviato.

L'installazione di Raspbmc richiede diversi minuti durante i quali il sistema verrà riavviato più volte.



Figura 5.1 - Raspbmc - selezione della lingua.

Terminata la fase di installazione, verrà richiesta la lingua da utilizzare per l'interfaccia utente di XBMC. Possiamo selezionare la lingua italiana e a questo punto il nostro sistema si riavrà per l'ultima volta.

L'interfaccia utente di Raspbmc è pensata per poter essere utilizzata su uno schermo TV e presenta quindi scritte grandi, in caratteri leggibili con toni scuri (per evitare che il televisore illumini a giorno il vostro salotto durante l'utilizzo!).

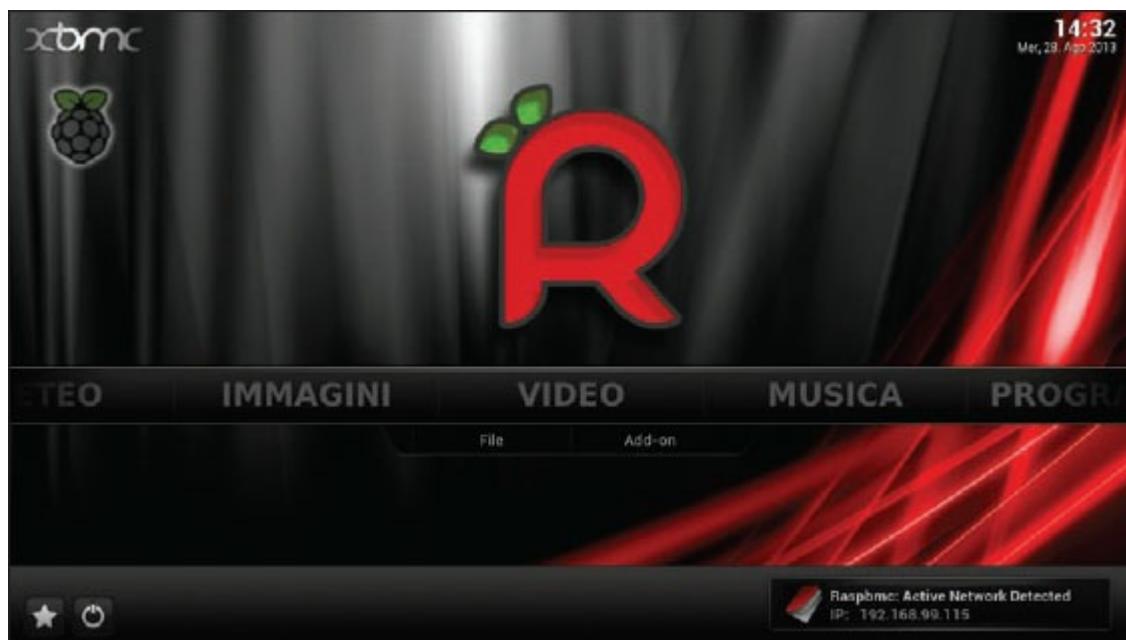


Figura 5.2 - Raspbmc - la schermata principale.

La schermata principale di Raspbmc ci permette di navigare tra le funzioni principali del sistema attraverso una barra centrale dove sono

rappresentate le diverse funzionalità: meteo, immagini, video, musica, programmi e impostazioni. È possibile navigare in questi menu con un mouse oppure con i tasti cursore della nostra tastiera.

Nella parte bassa dello schermo vengono mostrate, sulla sinistra, due icone: una per accedere ai favoriti (vedremo poi come aggiungere contenuti alla lista dei favoriti) e l'altra per spegnere il sistema. È importante spegnere sempre il sistema usando questa icona per evitare danni al filesystem o alla libreria multimediale.

L'area a destra nella barra inferiore viene usata per notifiche informative o per segnalare errori o malfunzionamenti dei diversi componenti di Raspbmc.

In alto a destra, infine, vengono mostrate data e ora correnti.

Configurazione iniziale e installazione

L'opzione **Sistema** della barra centrale è utile per controllare lo stato del nostro sistema e modificarne la configurazione.

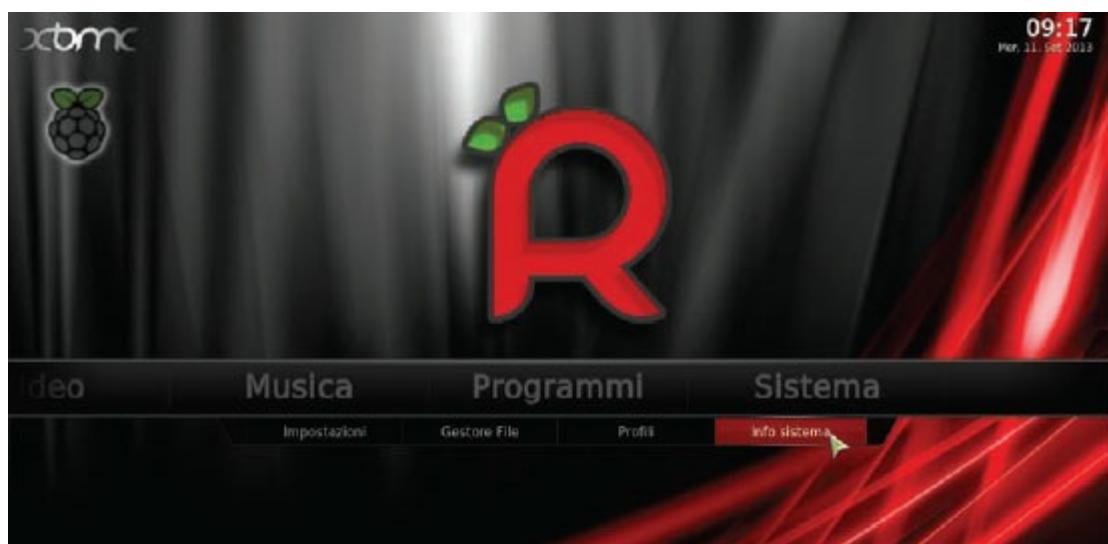


Figura 5.3 - Raspbmc - il menu Sistema.

Selezionando l'opzione **Info sistema** è possibile verificare le impostazioni generali e poi, in dettaglio, quelle di video, audio, rete ecc. È importante annotare l'indirizzo IP del nostro media center perché ci servirà per poterlo configurare e controllare da remoto.

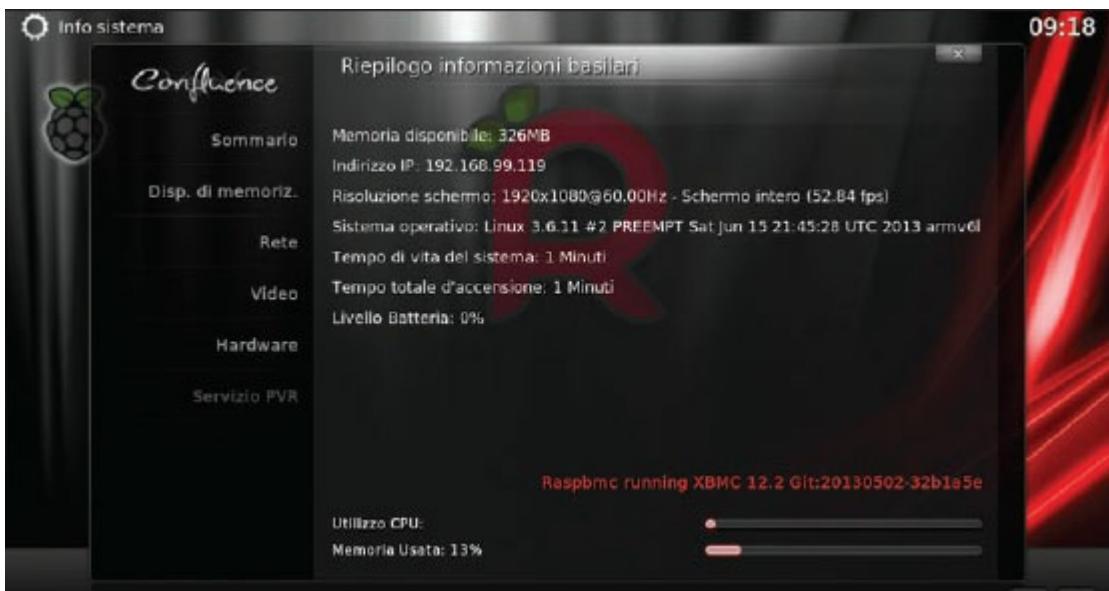


Figura 5.4 - Raspbmc - informazioni sul sistema.

Selezionando invece l'opzione **Impostazioni** è possibile modificare la configurazione del sistema. Utilizzeremo alcune di queste opzioni in seguito, ma può essere interessante navigare tra le schermate per farsi un'idea delle funzionalità e della configurabilità di XBMC.



Figura 5.5 - Raspbmc - le impostazioni.

L'opzione **Gestore file** ci permette di accedere al filesystem del nostro sistema (non dimentichiamoci che, ben nascosto dietro l'interfaccia grafica, c'è sempre un sistema Linux!) e di copiare file tra i due pannelli che costituiscono la schermata. Se dovete trasferire file sul vostro nuovo media player il file manager vi tornerà molto utile.



Figura 5.6 - Raspbmc - gestore file.

L'opzione **Profili** consente di configurare diversi profili utente, filtrando i contenuti e le funzionalità per i vari utenti. Può essere utile in alcune condizioni o per limitare l'accesso ad alcuni utenti tramite password, ma in generale dover effettuare un login ogni volta che si vuole utilizzare la TV è piuttosto noioso, quindi lasciare il profilo di default senza login può essere la soluzione migliore.

Sorgenti e contenuti

Il primo passo per utilizzare il nostro Raspberry Pi come media center è quello di aggiungere alle diverse tipologie di media (immagini, musica e video) delle sorgenti di contenuti. Raspbmc supporta diverse tipologie di contenuti:

- contenuti locali sulla SD o su un disco USB esterno;
- contenuti condivisi in rete con protocolli di condivisione come SMB (visto nel capitolo precedente e utilizzato dai sistemi Windows), NFS (tipicamente utilizzato sui sistemi Unix) e AFS (utilizzato dai sistemi Apple);
- contenuti condivisi in rete da server DLNA.

I contenuti possono essere aggiunti partendo dalla categoria di appartenenza (musica, immagini, video). XBMC, inoltre, consente di estendere le funzionalità del sistema attraverso plug-in. Questo ci permetterà, per esempio, di ascoltare stazioni radio via internet oppure di guardare i trailer dei film più recenti attraverso YouTube.

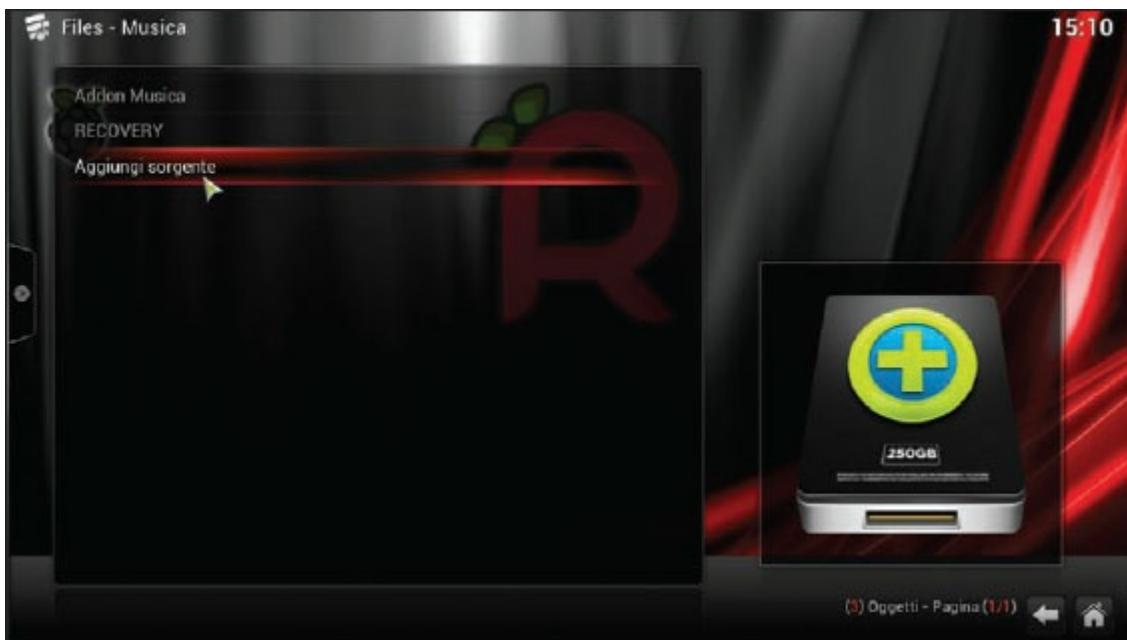


Figura 5.7 - Raspbmc - aggiungere sorgenti musicali.

Selezionando, per esempio, l'opzione **Musica** e facendo un clic su **Files** potremo aggiungere nuove sorgenti alla nostra libreria musicale.

Nella selezione file verranno mostrati i dischi locali (la SD card è chiamata **RECOVERY**) e l'opzione **Aggiungi sorgente**, che ci consentirà di arricchire la selezione di contenuti utilizzabili tramite Raspbmc.

Una volta selezionata l'opzione potremo creare una nuova sorgente. Una sorgente può contenere più risorse al suo interno. Per aggiungere la prima sorgente possiamo utilizzare il pulsante **Esplora**.

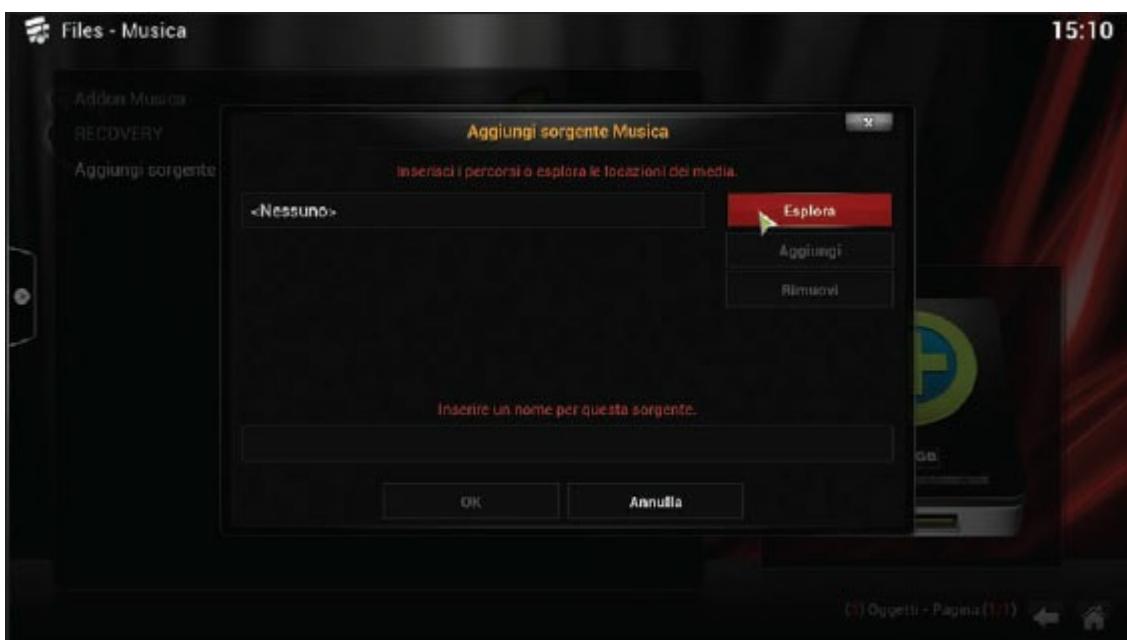


Figura 5.8 - Raspbmc - definizione di una nuova sorgente.

A questo punto potremo selezionare la tipologia di risorsa da aggiungere alla nostra sorgente. Potremo scegliere una cartella locale (ricordiamoci che anche le memorie USB vengono viste come cartelle locali), una condivisione di rete che utilizzi SMB o NFS, un server DLNA o uno stream di rete che utilizza il protocollo SAP, solitamente tramite il software VLC su un sistema PC o Linux.

Selezionando l'opzione **UPnP Devices** potremo scegliere un server DLNA (UPnP, Universal Plug and Play è il protocollo su cui è basata l'architettura DLNA), magari proprio il Raspberry Pi configurato come home-server seguendo le istruzioni del capitolo precedente!

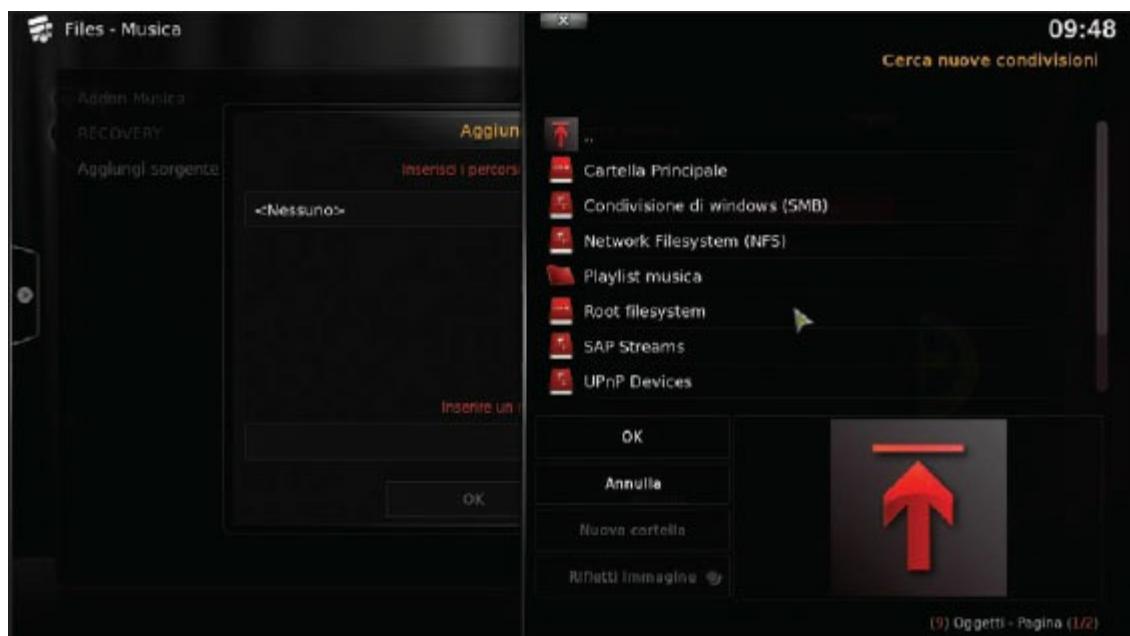


Figura 5.9 - Raspbmc - tipologie di sorgenti disponibili.

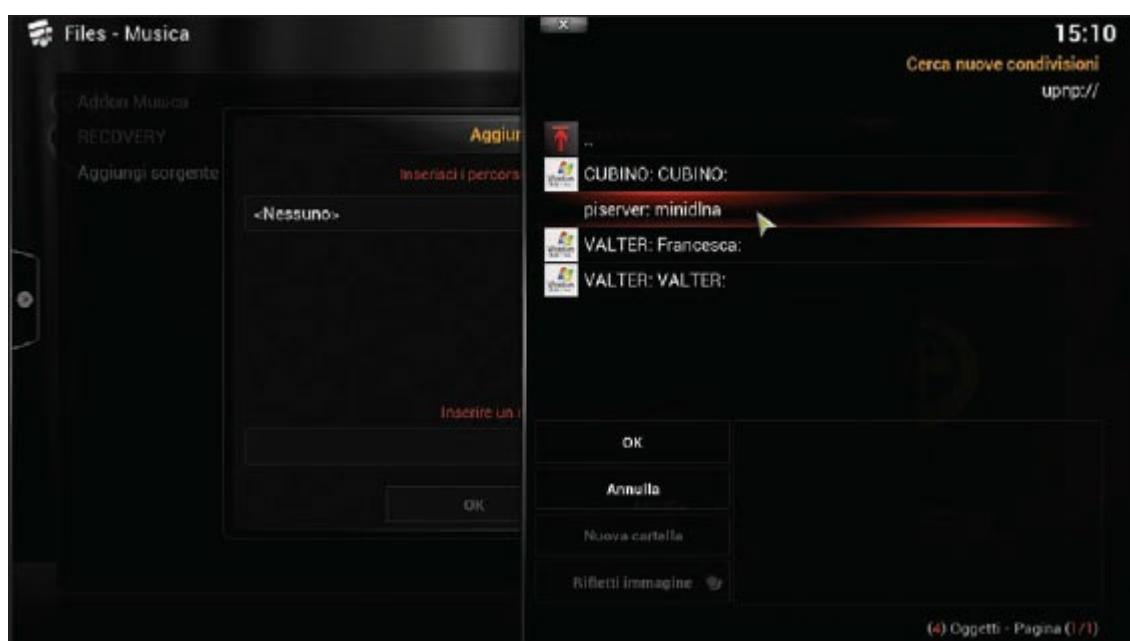


Figura 5.10 - Raspbmc - selezione di un server DLNA in rete.

I server DLNA all'interno della nostra rete locale vengono elencati automaticamente e ci basterà quindi selezionare quello che ci interessa nella lista che ci verrà mostrata per esplorare le diverse cartelle. Una volta raggiunta la cartella da aggiungere, potremo selezionarla premendo il pulsante **OK**.

È possibile aggiungere alla sorgente più cartelle, anche su dispositivi di tipo diverso e questo può essere utile se il nostro materiale è sparso tra diversi sistemi.

Una volta aggiunte tutte le risorse necessarie è possibile salvare la sorgente premendo **OK**. Ora il nostro server comparirà nella lista di sorgenti mostrate dall'opzione **Files**.

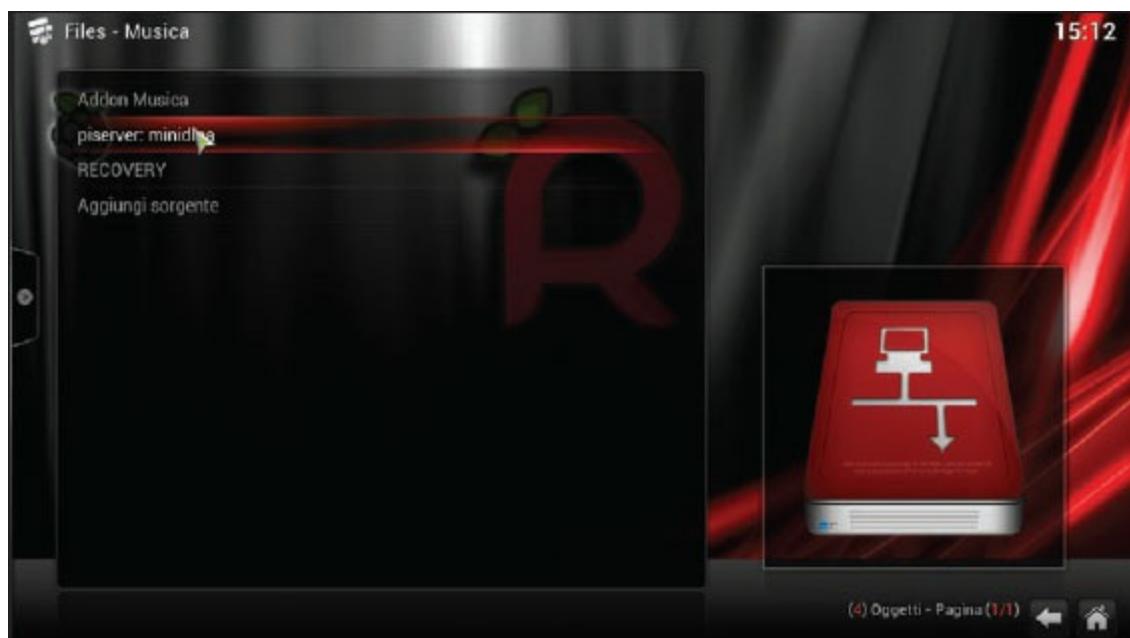


Figura 5.11 - Raspbmc - la nuova lista delle sorgenti.

Allo stesso modo potremo aggiungere come sorgenti cartelle locali o periferiche USB. Poi potremo selezionarle e navigare all'interno dei contenuti scorrendo le nostre canzoni divise per album, per artista, per genere ecc.



Figura 5.12 - Raspbmc - i contenuti sul nostro server DLNA.

Arrivati a una canzone potremo selezionarla e ascoltarla, anche mentre navighiamo tra i menu di XBMC.

Spostando il mouse sulla linguetta che appare nella parte sinistra dello schermo apparirà un menu che ci consente di modificare l'ordinamento della playlist e di passare in modalità schermo intero per il playback della musica.

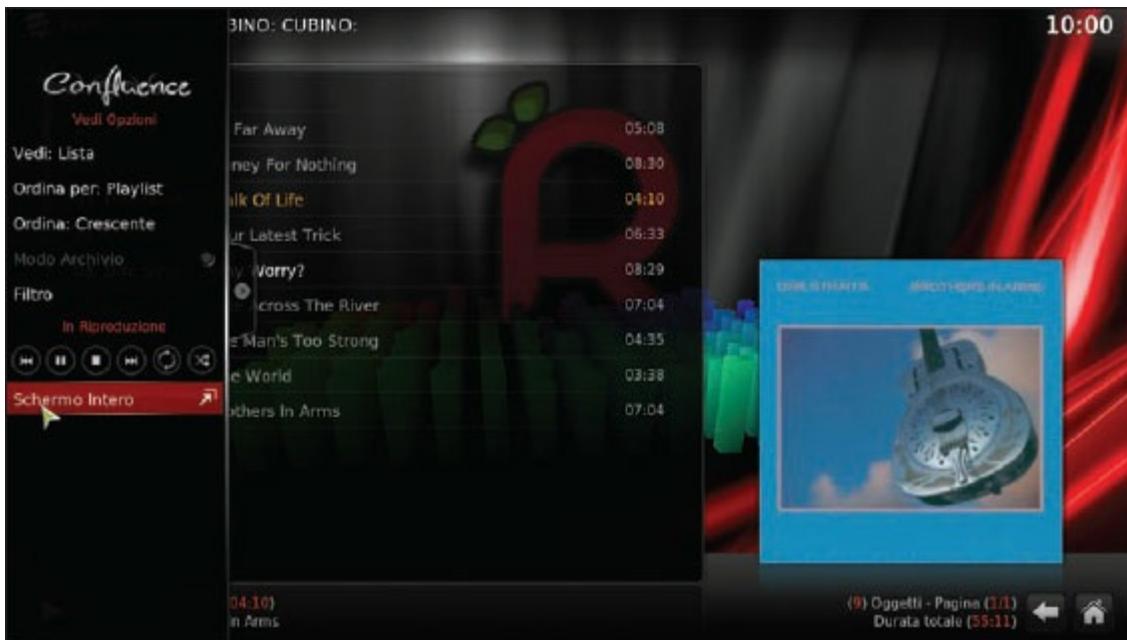


Figura 5.13 - Raspbmc - il menu playback.

Passando in modalità schermo intero, oltre alle informazioni sul brano che stiamo ascoltando, verranno mostrate animazioni grafiche sincronizzate con la musica, un modo piacevole di “animare” il nostro televisore in mancanza di altre immagini. Muovendo il mouse verranno mostrati, al di sotto delle informazioni sul brano corrente, i tradizionali controlli per il playback (pausa, stop, avanti e indietro ecc.) e una serie

di icone che ci permetteranno di vedere, se disponibili, i testi della canzone che stiamo ascoltando, di cambiare la visualizzazione, di configurarla o di resettarla al default.

Possiamo chiudere la visualizzazione a tutto schermo premendo il tasto **Esc** della nostra tastiera o selezionando con il mouse il tasto di chiusura in alto a destra nello schermo.

Gli elementi delle varie liste (album, artisti, singole canzoni) possono essere selezionati con un clic destro; apparirà un menu contestuale che ci consentirà di aggiungere un elemento alla playlist, di inserirlo tra i preferiti e così via.

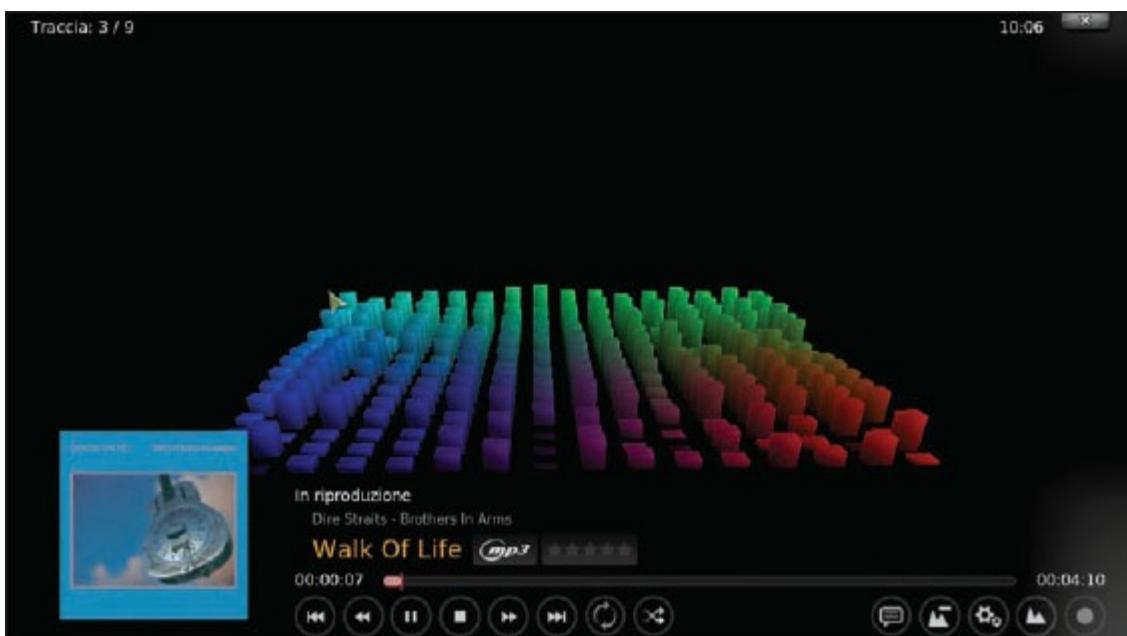


Figura 5.14 - Raspbmc - playback a tutto schermo.

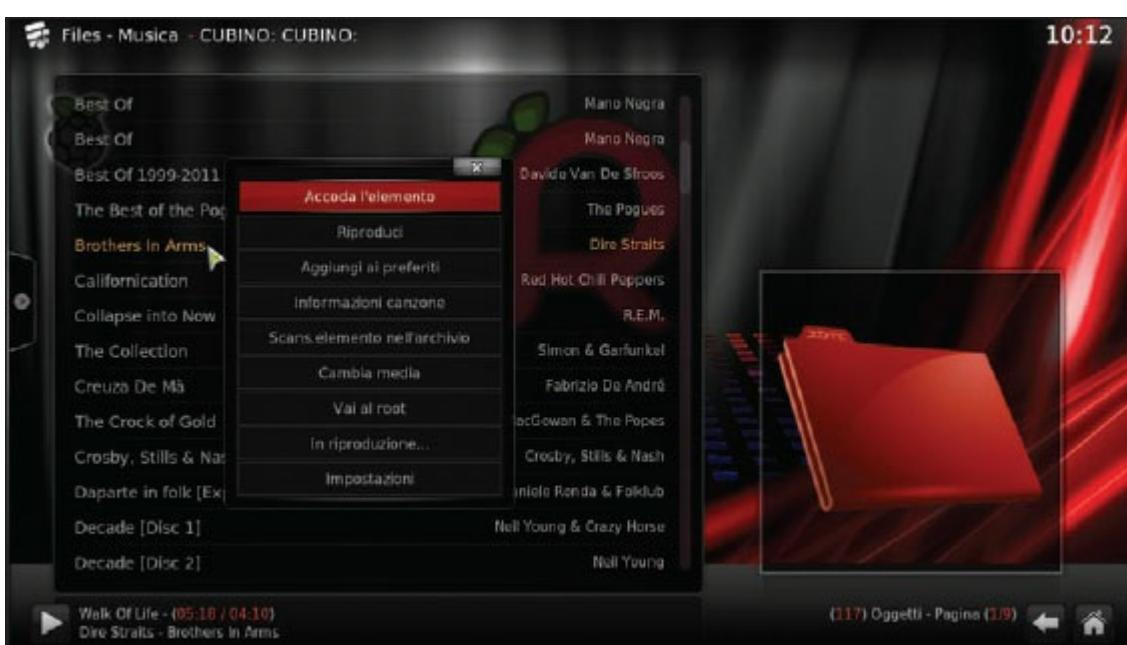


Figura 5.15 - Raspbmc - il menu contestuale relativo a un album.

Allo stesso modo potremo aggiungere sorgenti di video o immagini, sfruttando anche in questo caso le diverse modalità di connessione dei contenuti fornite da XBMC.



Figura 5.16 - Raspbmc - tipologie di sorgenti video.

Nel caso dei video possiamo aggiungere una sorgente scegliendo una tra le modalità disponibili. Possiamo, per esempio, usare il protocollo SMB per accedere a cartelle condivise con le modalità viste nel capitolo precedente. Una volta selezionato il server SMB da utilizzare ci verrà chiesto di autenticarci inserendo username e password.

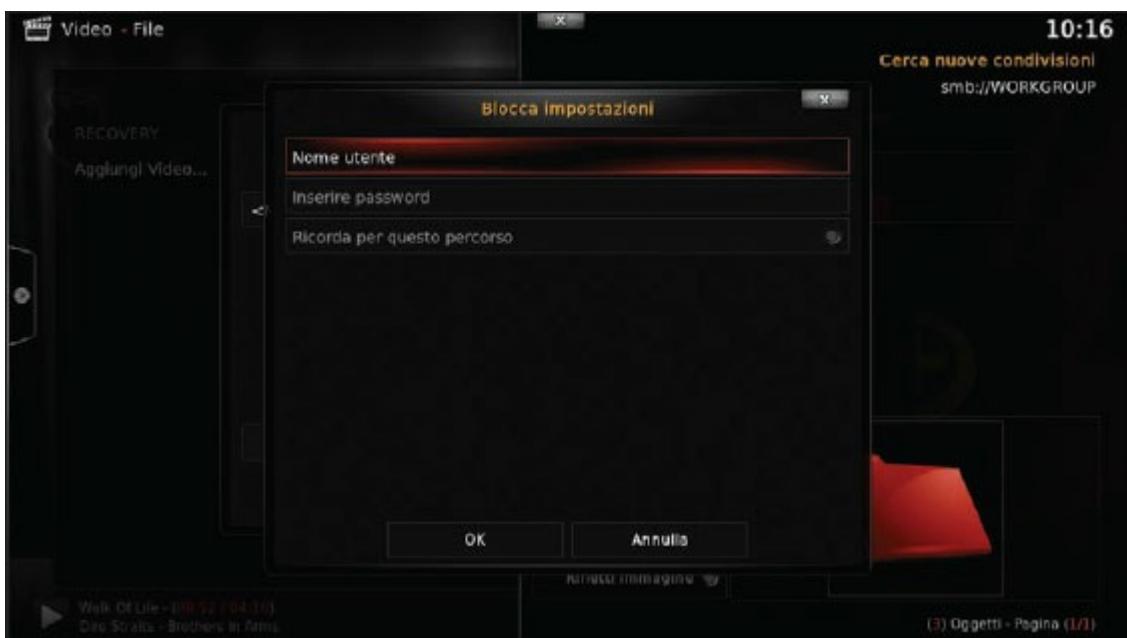


Figura 5.17 - Raspbmc - autenticazione per accedere a risorse SMB.

Una volta selezionata la cartella che vogliamo aggiungere alla sorgente, il sistema ci chiederà di identificare il tipo di contenuti (film, serie TV o altro) e di selezionare uno “scraper”. Lo **scraper** è un add-on che consente di ricavare informazioni aggiuntive (immagine del manifesto, informazioni su attori e regista ecc.) da un database internet. Esistono diversi scraper, a seconda del tipo di contenuti.



Figura 5.18 - Raspbmc - configurazione di una sorgente video.

Una volta configurata la nostra sorgente video potremo navigare e visualizzare i diversi video in modo analogo a quanto visto per la musica.

Recuperare o rielaborare i contenuti con gli add-on

XBMC consente di recuperare o rielaborare i contenuti tramite una serie di add-on. Questi componenti consentono di estendere e personalizzare il nostro media center e, anche se imparare a installarli e configurarli richiede un po' di pazienza, le nostre fatiche saranno ricompensate con ore di relax davanti a una TV più ricca e interattiva rispetto a quella che siamo abituati a vedere.

Per esempio, nella sezione Musica possiamo aggiungere add-on per ascoltare la radio via internet, ottimi per tenersi aggiornati con le ultime notizie o per ascoltare musica che non faccia già parte della nostra collezione digitale.

Un ottimo add-on per ascoltare radio online è ListenLiveEU, che

consente di ascoltare le radio di molti paesi europei.

Per aggiungerlo alla nostra installazione dobbiamo tornare alla sezione **Musica** e selezionare **Add-on**. Apparirà una lista, per ora vuota, degli add-on installati e potremo aggiungere un add-on selezionando l'opzione **Altro**.

La lista degli add-on disponibili è molto ricca e possiamo selezionarne uno per avere più informazioni sulle sue funzionalità.

Una volta trovato l'add-on da installare, nel nostro caso ListenLiveEU, possiamo selezionarlo e poi scegliere **Installa** dalla finestra di dialogo.

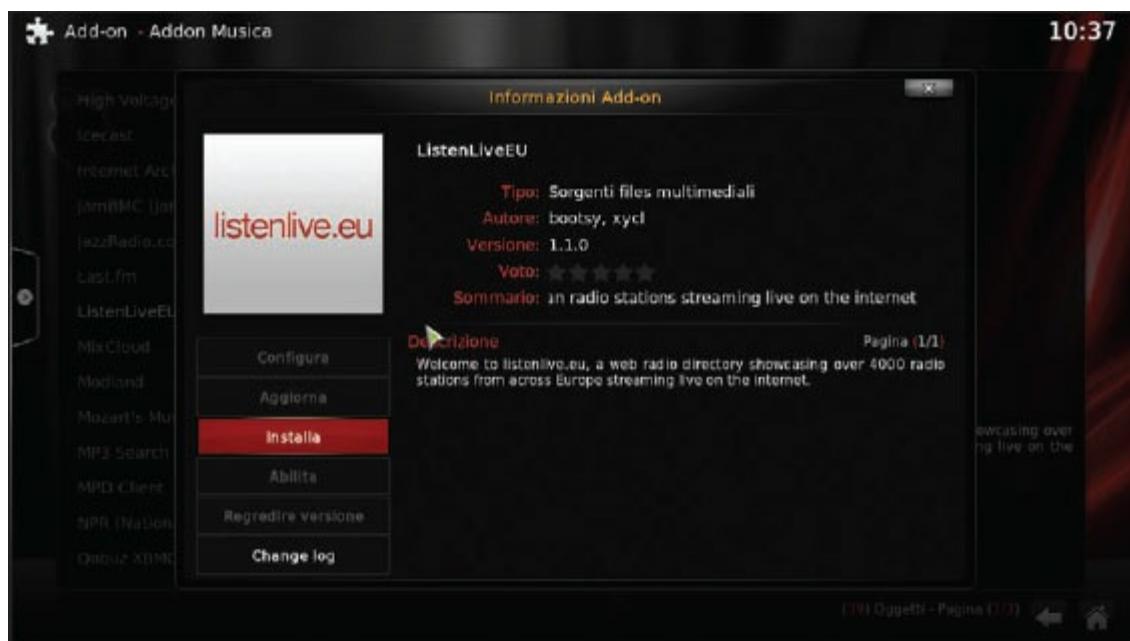


Figura 5.19 - Raspbmc - installazione di un add-on.

Una volta installato, il nostro nuovo add-on comparirà nella lista degli add-on disponibili. Selezionando **ListenLiveEU** potremo scegliere la radio da ascoltare.

Visto che il numero di stazioni è davvero notevole dovremo prima selezionare un Paese e poi la radio che vogliamo ascoltare.

Un clic sulla stazione desiderata ci permetterà di ascoltarla in diretta, anche mentre utilizziamo altre funzioni di XBMC, ovviamente se queste non hanno bisogno del canale audio!

Un altro add-on interessante è quello che ci consente di accedere ai contenuti di **YouTube**, collegato alla sezione **Video**.

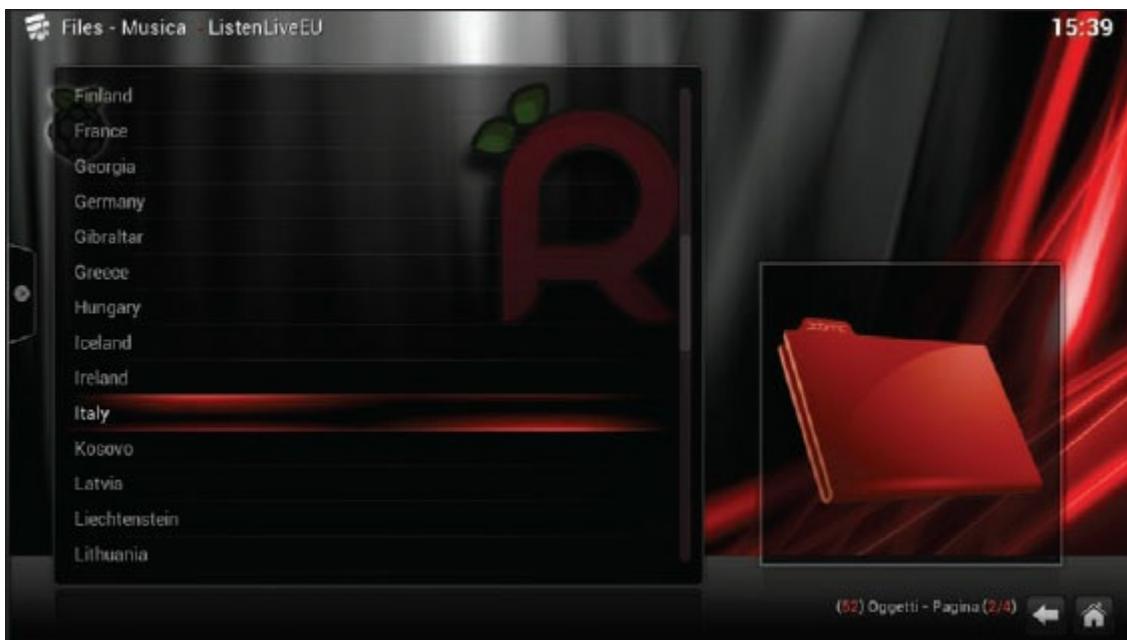


Figura 5.20 - ListenLiveEU - la lista dei Paesi.



Figura 5.21 - ListenLiveEU - la lista delle radio italiane.

Anche in questo caso per installare l'add-on occorre selezionare l'opzione **Add-on** della sezione **Video** di XBMC e poi l'opzione **Altro** nella lista degli add-on installati.

Una volta selezionato l'add-on YouTube, bastano un clic per visualizzarne le informazioni estese e il pulsante **Installa** per completare l'installazione.

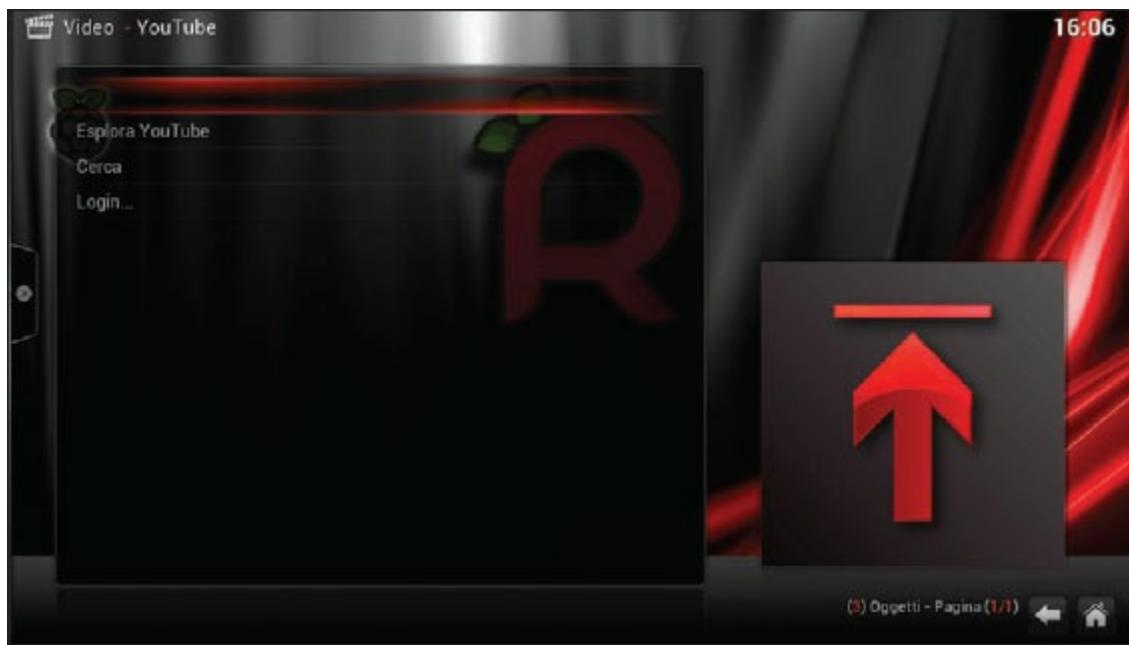


Figura 5.22 - Raspbmc - la schermata di installazione dell'add-on YouTube.

Una volta tornati sulla lista degli add-on video potremo lanciare l'add-on YouTube. Al primo avvio ci verrà mostrata una finestra di impostazioni in cui potremo inserire le nostre credenziali per l'accesso a YouTube (il login è facoltativo, ma usando un account potremo vedere i nostri video e quelli dei canali YouTube che abbiamo deciso di seguire), scegliere la lingua per i sottotitoli e una cartella in cui memorizzare i video scaricati.

Una volta completate queste informazioni possiamo navigare tra i video, organizzati nei vari canali, selezionando l'opzione **Esplora YouTube** oppure cercare i video per un determinato argomento tramite l'opzione **Cerca**.

Esplorando i contenuti di YouTube possiamo navigare tra le diverse categorie, guardare i video più popolari o, se le ore di utilizzo del nostro media center ci hanno fatto venire la voglia di goderci un film su grande schermo, selezionare l'opzione Top 100 Trailer di YouTube per decidere cosa andare a vedere nel nostro cinema preferito!

Una volta scelto un canale, oppure eseguita una ricerca, apparirà un elenco con le miniature dei diversi video; ci basterà selezionarne uno per potercelo godere a tutto schermo sul nostro nuovo media center.

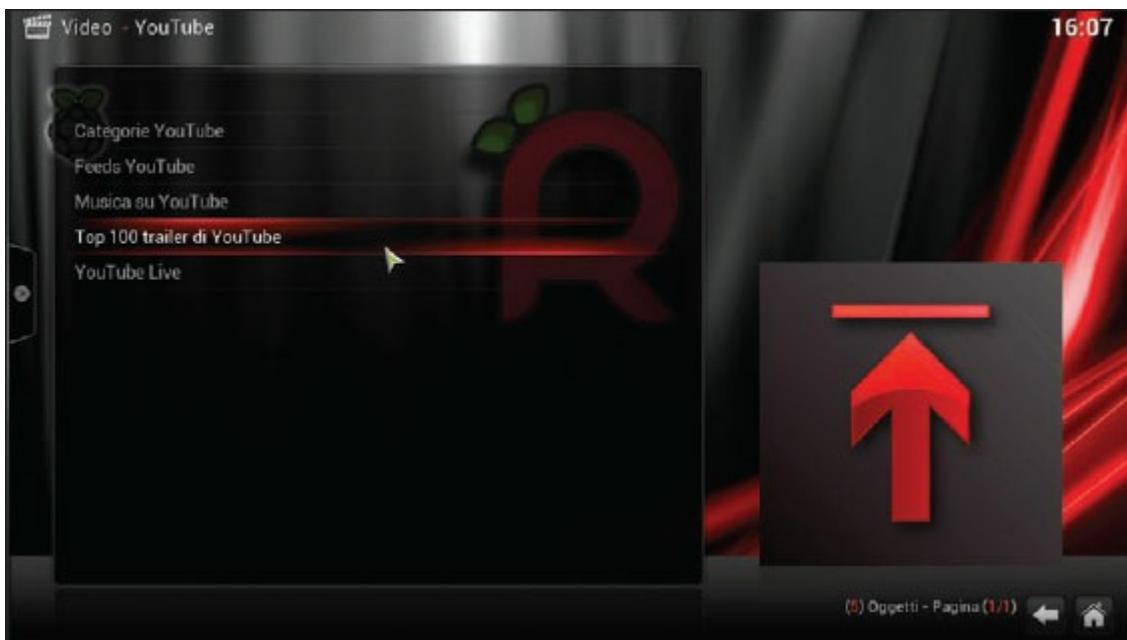


Figura 5.23 - Add-on YouTube - Esplora YouTube.

Configurare un telecomando

Se avete seguito i passaggi descritti in questo capitolo con il vostro Raspberry Pi già collegato alla TV vi sarete resi conto che utilizzare tastiera e mouse quando si è seduti in poltrona (o comodamente spaparanzati sul divano domestico) non è molto comodo. La mancanza di un piano di appoggio rende le cose complicate e anche utilizzare piccole tastiere senza cavi non migliora troppo la situazione, soprattutto se, come l'autore, madre natura vi ha dotato di mani over-size!

Anni di utilizzo, più o meno intensivo, della TV domestica vi avranno insegnato ad apprezzare la comodità del telecomando, soprattutto se, come l'autore, avete avuto modo di vivere nell'epoca in cui le TV erano in bianco e nero o a colori molto incerti, ci mettevano minuti prima di mostrare un'immagine decente e richiedevano scomodi viaggi per il salotto ogni volta che si voleva fare zapping tra i (pochi) canali a disposizione. Il telecomando è uno degli accessori che si sono evoluti meno nel corso degli anni. Si sono moltiplicati tasti e funzioni, ma la modalità di interazione è rimasta sempre la stessa, a prova della bontà del design iniziale. E se generazioni di utenti hanno usato un telecomando, perché forzarsi ad adoperare tastiera e mouse?

Raspbmc consente di collegare diversi tipi di telecomando al nostro media center domestico. Alcuni hanno bisogno di un ricevitore via USB e sono abbastanza costosi. Noi, fedeli alla filosofia dei maker, cercheremo invece di ottenere lo stesso risultato a costi molto bassi, a prezzo di un po' di fatica in più che ci permetterà però di imparare

qualcosa di nuovo.

Sfruttando i pin di General Purpose Input Output (GPIO) di Raspberry Pi possiamo infatti collegare direttamente un ricevitore a infrarossi e utilizzare un qualsiasi telecomando per controllare il nostro nuovo media center.

I normali telecomandi TV sono dotati di uno o più LED (Light Emitting Diode) che emettono luce infrarossa, non percepibile quindi dall'occhio umano. Il nostro telecomando si limita a tradurre la pressione di un tasto in una sequenza di impulsi luminosi, un po' come avveniva agli albori delle telecomunicazioni, quando un testo veniva tradotto in impulsi usando il codice Morse. Questi impulsi luminosi raggiungono un ricevitore che li traduce in impulsi elettrici, più facilmente gestibili da un processore.

Il nostro Raspberry Pi è perfettamente in grado di ricevere segnali sui suoi pin di GPIO. Quello che ci manca, e possiamo acquistare online o in un qualsiasi negozio di elettronica per un paio di euro, è un ricevitore a infrarossi. Ne esistono diversi tipi; nel nostro esempio utilizzeremo il modello 4838 di Vishay, uno dei più comuni. L'importante è che il vostro ricevitore utilizzi per l'alimentazione e i segnali una tensione di 3,3 V, che è quella utilizzata dai pin del nostro Raspberry Pi. Utilizzare tensioni diverse potrebbe danneggiarlo. Ci servirà inoltre un telecomando. Il telecomando di una TV, un DVD player o un videoregistratore non più in uso andrà benissimo, così come un telecomando universale, anche questo disponibile nella maggior parte dei negozi di elettronica.

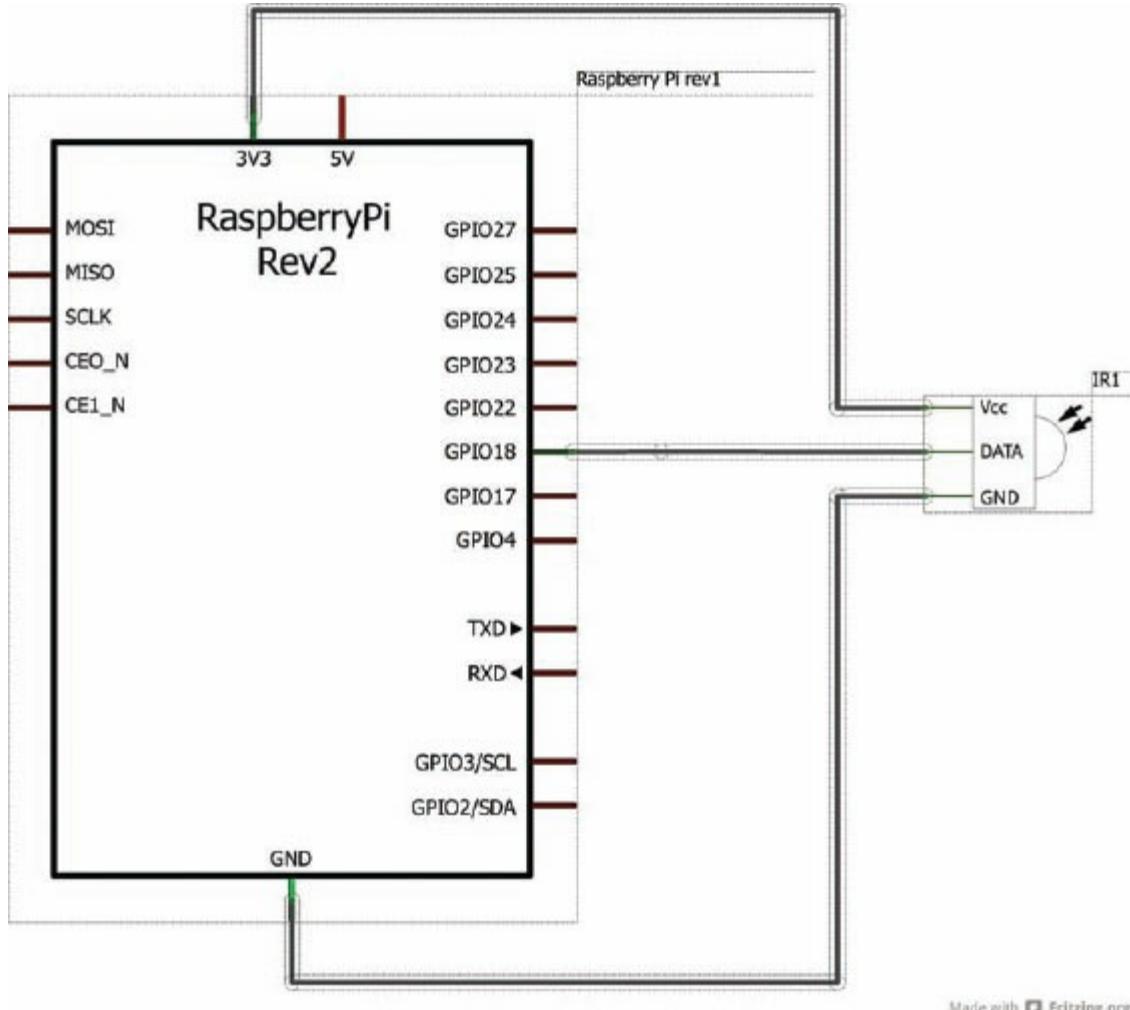
I ricevitori IR hanno tre piedini (che chiameremo pin, usando il termine inglese che si usa molto comunemente anche nella documentazione tecnica in italiano): due servono per l'alimentazione e uno ritornerà il segnale corrispondente agli impulsi a infrarossi ricevuti.

Il collegamento può essere rappresentato con il diagramma schematico che vedete in [Figura 5.24](#).

Questo e gli altri diagrammi di questo libro sono stati realizzati con un'applicazione gratuita e open source che si chiama Fritzing, e che vi consiglio caldamente di scaricare dal sito www.fritzing.org se deciderete di sperimentare con l'elettronica.

Il diagramma usa dei simboli convenzionali per rappresentare i diversi componenti elettronici. Sono molto utili per dare una rappresentazione formale e comprensibile agli addetti ai lavori, ma possono risultare ostici per chi non ha mai avuto a che fare con l'elettronica a livello scolastico

o professionale. Per fortuna Fritzing ci consente anche di generare schemi più visuali, in cui i componenti sono rappresentati come appaiono nella realtà. Questo formato è sicuramente più “digeribile” per chi non hai mai affrontato l’elettronica e vuole conoscerla sperimentando.



Made with Fritzing.org

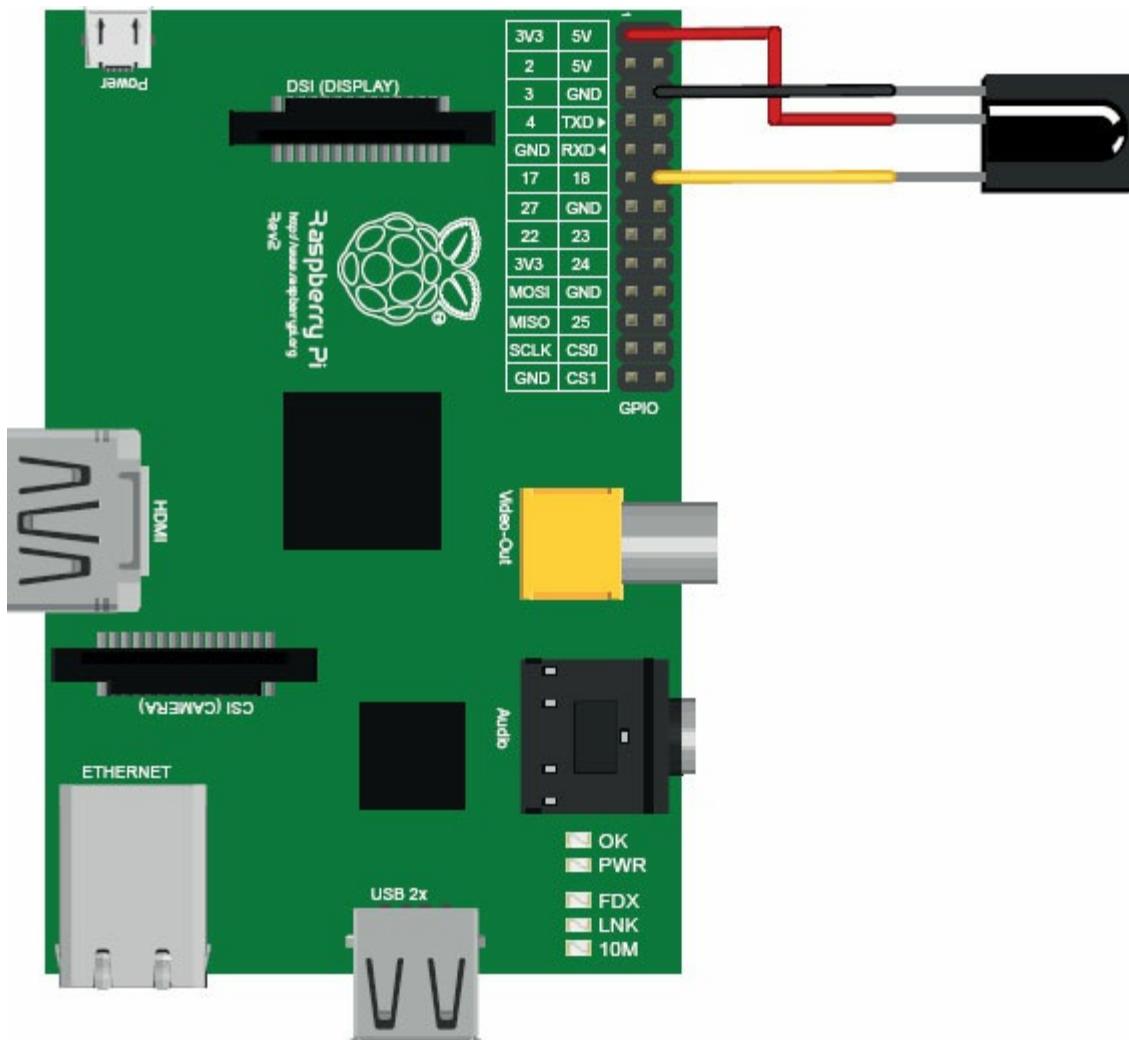
Figura 5.24 - Diagramma schematico di collegamento per il ricevitore IR.

La [Figura 5.25](#) rappresenta il collegamento tra il nostro Raspberry Pi e il ricevitore IR.

Se guardate il vostro Raspberry Pi con il connettore di alimentazione verso l’alto, i pin del connettore GPIO sono numerati dall’alto in basso e da sinistra a destra. Il ricevitore normalmente ha un lato piatto e uno leggermente bombato, che è quello che riceve il segnale IR. Prendete il sensore con il lato bombato verso di voi.

Il pin numero 1 del connettore GPIO fornisce l’alimentazione a 3,3 V e deve essere collegato al pin centrale del nostro ricevitore IR (questo nel caso del ricevitore Vishay 4838; verificate la documentazione del vostro ricevitore se il modello non corrisponde). Il pin più a sinistra del

ricevitore IR deve essere collegato alla massa (GND), che corrisponde al pin numero 6 del connettore GPIO. Il segnale, infine, deve essere collegato al pin numero 12, che corrisponde al pin GPIO 18 del processore (purtroppo le numerazioni dei pin non sono le stesse tra il connettore e il processore).



Made with Fritzing.org

Figura 5.25 - Diagramma visuale di collegamento del ricevitore IR.

Una volta collegato il ricevitore potrete alimentare il vostro Raspberry Pi. Tenete un dito sul ricevitore e, se incomincia a scaldarsi, spegnete subito tutto! C'è sicuramente qualche errore di connessione. Per esperienza diretta vi posso dire che bruciare un ricevitore IR è abbastanza difficile e per ora i miei ricevitori sono sopravvissuti a diversi errori di connessione, quindi non preoccupatevi e ricontrollate i collegamenti.

Per provare che il sensore sia funzionando correttamente possiamo utilizzare la command line.

Anche dietro l'interfaccia grafica di XBMC è “nascosta” una command line e possiamo usarla in locale, con la combinazione di tasti **Ctrl+Alt+F1**, oppure collegandoci da remoto via SSH.

Una volta connessi dovremo attivare il modulo LIRC (Linux Infrared Remote Control), in grado di interpretare i segnali in arrivo dal nostro ricevitore IR e tradurli in codici tasto, equivalenti a quelli generati da una normale tastiera o da un telecomando USB. LIRC non è una normale applicazione, ma un modulo kernel. Questi moduli vengono caricati all'interno del kernel e possono accedere all'hardware.

Per caricare LIRC possiamo utilizzare il comando `modprobe`:

```
pi@pivalter ~ $ sudo modprobe lirc_pi  
pi@pivalter ~ $
```

Eseguiremo `modprobe` tramite `sudo` perché, per ragioni di sicurezza, solo l'utente root può caricare moduli nel kernel. Nel nostro caso il modulo LIRC è in versione custom per Raspberry Pi, quindi si chiama `lirc_pi`.

Il caricamento di LIRC causa l'attivazione di un demone in user mode chiamato `lircd`; questa applicazione riceve i dati dal modulo kernel e li mette a disposizione delle altre applicazioni. Per provare che il nostro telecomando stia funzionando non abbiamo bisogno di `lircd` e lo possiamo quindi terminare utilizzando il comando `kill`, sempre lanciato in modalità super-user:

```
pi@pivalter ~ $ sudo kill $(pidof lircd)  
pi@pivalter ~ $
```

`lircd`, che riattiveremo tra poco, ha bisogno di essere configurato per poter “tradurre” gli impulsi inviati dal telecomando e, prima di iniziare a configurarlo, è meglio verificare che funzioni la trasmissione a più basso livello dei segnali. Per farlo possiamo utilizzare il comando `mode2`, che si limita a mostrare la durata degli impulsi ricevuti. Lanciando:

```
pi@pivalter ~ $ mode2 -d /dev/lirc0
```

e puntando il telecomando verso il nostro ricevitore IR dovremmo vedere che a ogni pressione di un tasto verranno mostrati una serie di messaggi di questo tipo:

```
pulse 340  
space 700  
pulse 325  
space 670
```

Questo indica che la ricezione dei dati dal sensore IR è corretta. Se non viene mostrato alcun messaggio, controllate bene il collegamento del sensore ai pin di GPIO, in particolare il pin di segnale. E controllate anche che il vostro sensore sia in grado di funzionare a 3,3 V, come richiesto da Raspberry Pi.

Una volta verificato che il collegamento fisico funziona, possiamo passare a configurare il nostro telecomando.

`lircd` traduce gli impulsi letti dal sensore IR e intercettati nel kernel da `lirc` in codici tasto. Per farlo ha bisogno di un file di configurazione. Purtroppo i codici tasto variano a seconda del produttore e non esiste una codifica valida per tutti i modelli di telecomando. Anche i telecomandi universali possono essere configurati per emulare diverse tipologie di telecomandi, di produttori diversi.

La strada più semplice è quella di cercare in rete, tramite il vostro motore di ricerca preferito, un file di configurazione per il vostro telecomando. Cercare “lirc configuration file <modello telecomando>” dovrebbe portarvi sulla buona strada. Una volta trovato il file potete copiarlo nella cartella /home/pi con il nome **`lircd.conf`**.

In alternativa, se non conoscete il modello di telecomando o non trovate in rete un file già pronto, è possibile, con un po' di pazienza, configurare manualmente il nostro telecomando. Per farlo possiamo utilizzare l'utility `irrecord`.

Per prima cosa, lanciando:

```
pi@pivalter ~ $ irrecord -list-namespace
```

verranno mostrati i codici tasto supportati dalla versione corrente di `lircd`.

Fortunatamente non è necessario configurare tutti questi codici: basterà configurare i tasti del telecomando che abbiamo a disposizione, rimappandoli su quelli più usati dentro XBMC. Oltre ai tasti numerici è utile avere dei tasti di navigazione (cursori), un pulsante **OK** e un pulsante **Back**; questo ci consentirà di navigare tranquillamente dentro i menu del sistema. Questa tabella riporta alcuni dei tasti più utili.

Codice tasto	Funzione
KEY_0	
KEY_1	
KEY_2	

KEY_3 KEY_4 KEY_5 KEY_6 KEY_7 KEY_8 KEY_9	Tasti numerici, utili se si utilizza un sintonizzatore TV per selezionare i canali
KEY_UP KEY_DOWN KEY_LEFT KEY_RIGHT	Tasti cursore. Molti telecomandi recenti ne sono dotati e il loro utilizzo semplifica molto la navigazione nei menu
KEY_OK	Tasto di conferma. Spesso è al centro dei tasti cursore
KEY_VOLUMEUP KEY_VOLUMEDOWN	Controllo del volume
KEY_CHANNELUP KEY CHANNELDOWN	Navigazione tra i canali, ma anche all'interno delle liste
KEY_EXIT	Annulla una selezione, esce da un menu
KEY_HOME	Torna alla schermata principale
KEY_POWER	Spegne il sistema
KEY_PLAY KEY_PAUSE KEY_STOP KEYREWIND KEY_FASTFORWARD	Controllano il playback di musica e filmati
KEY_RECORD	Attiva la registrazione (modalità TV)
KEY_MENU	Apre il menu relativo alla selezione corrente
KEY_SETUP	Apre la schermata di configurazione
KEY_BACK KEY_NEXT	Passa al brano/video precedente o successivo

Per configurare il nostro telecomando dovremo rilanciare `irrecord` e associare i vari tasti ai codici tasto che ci interessano.

```
pi@raspbmc:~$ irrecord -d /dev/lirc0 /home/pi/lircd.conf
```

```
irrecord - application for recording IR-codes for usage with lirc
```

```
Copyright (C) 1998,1999 Christoph Bartelmus(lirc@bartelmus.de)
```

```
This program will record the signals from your remote control  
and create a config file for lircd.
```

A proper config file for lircd is maybe the most vital part of this package, so you should invest some time to create a working config file. Although I put a good deal of effort in this program it is often not possible to automatically recognize all features of a remote control. Often short-comings of the receiver hardware make it nearly impossible. If you have problems to create a config file READ THE DOCUMENTATION of this package, especially section "Adding new remote controls" for how to get help.

If there already is a remote control of the same brand available at <http://www.lirc.org/remotes/> you might also want to try using such a remote as a template. The config files already contain all parameters of the protocol used by remotes of a certain brand and knowing these parameters makes the job of this program much easier. There are also template files for the most common protocols available in the remotes/generic/ directory of the source distribution of this package. You can use a template files by providing the path of the file as command line parameter.

Please send the finished config files to <lirc@bartelmus.de> so that I can make them available to others. Don't forget to put all information that you can get about the remote control in the header of the file.

Press RETURN to continue.

A questo punto *irrecord* attenderà la pressione di un tasto per continuare.

Now start pressing buttons on your remote control.

It is very important that you press many different buttons and hold them down for approximately one second. Each button should generate at least one dot but in no case more than ten dots of output.

Don't stop pressing buttons until two lines of dots (2x80) have been generated.

Press RETURN now to start recording.

Il primo step che è richiesto è quello di premere i vari tasti del telecomando per circa un secondo ciascuno. Questo permetterà a *irrecord* di riconoscere la modalità in cui il nostro telecomando invia i dati. L'acquisizione inizia premendo il tasto **Invio**.

A mano a mano che premiamo i tasti del telecomando verranno mostrati dei puntini sulla console. Una volta riconosciuta la modalità di codifica del nostro telecomando *irrecord* mostrerà delle informazioni sulla modalità di codifica riconosciuta.

.....
Found const length: 43842

Please keep on pressing buttons like described above.

.....
Space/pulse encoded remote control found.
Signal length is 25.

```
Found possible header: 2429 546
Found trail pulse: 693
No repeat code found.
Signals are pulse encoded.
Signal length is 12
Now enter the names for the buttons.
```

A questo punto ci chiederà di inserire il nome del primo tasto che vogliamo registrare e poi di premere quel tasto.

```
Please enter the name for the next button (press <ENTER> to finish recording)
KEY_0
```

```
Now hold down button "KEY_0".
```

```
Please enter the name for the next button (press <ENTER> to finish recording)
```

Dovremo ripetere l'operazione per ognuno dei tasti che vogliamo memorizzare. Il tutto è sicuramente un po' noioso ma, una volta completato, ci consentirà di utilizzare il media center comodamente seduti sulla nostra poltrona preferita.

Una volta passati in rassegna tutti i pulsanti del nostro telecomando, possiamo completare l'inserimento premendo **Invio** senza inserire alcun nome di tasto.

Prima di correre sul sofà dovremo compiere un'ultima operazione.

```
Checking for toggle bit mask.
Please press an arbitrary button repeatedly as fast as possible.
Make sure you keep pressing the SAME button and that you DON'T HOLD
the button down!.
If you can't see any dots appear, then wait a bit between button presses.
```

```
Press RETURN to continue.
Una volta premuto il tasto enter dovremo premere rapidamente lo stesso tasto più volte.
```

```
....
No toggle bit mask found.
```

```
Successfully written config file.
```

A questo punto il nostro telecomando è configurato correttamente e possiamo abilitarne la gestione da parte di Raspbian tornando nell'interfaccia grafica.

Per configurare il telecomando è necessario lanciare l'applicazione **RaspbmcSettings** che si trova sotto la voce **Programmi** del menu principale.



Figura 5.26 - Raspbmc - configurazione del telecomando IR.

Una volta lanciata l'opzione, nella cartella **IR Remote** potremo attivare il ricevitore IR selezionando **Enable TSOP IR Receiver** e selezionando **Custom** come tipologia di telecomando nella casella GPIO IR Remote Profile.

A questo punto il sistema ci chiederà di riavviare e, una volta completato il restart, potremo navigare tra i menu con il nostro fedele telecomando!

Se qualche tasto non funziona o non corrisponde a quanto vi aspettavate, non preoccupatevi: basta rilanciare `irrecord` per modificare la configurazione dei tasti o aggiungerne di nuovi.

Pilotare il media center con uno smartphone

Se non vi sentite ancora pronti a “pasticciare” con l’hardware o non avete a portata di mano i componenti necessari. O se, più banalmente, il telecomando è disperso tra i cuscini del sofà e avete bisogno di controllare il vostro media player, non disperate! Esistono applicazioni di controllo remoto sulle più diffuse piattaforme mobili che consentono di utilizzare il vostro smartphone come un “super-telecomando” per XBMC.

Per poterle utilizzare dovete abilitare il controllo remoto all’interno del menu **Sistema\Servizi**.

Una volta spuntata l’opzione potrete cercare nel marketplace della vostra piattaforma mobile “XBMC remote” per trovare le diverse applicazioni che supportano il controllo remoto di XBMC. Esistono anche

applicazioni ufficiali realizzate dal team di XBMC per Android e iOS, ma vi consiglio di provarne diverse per scegliere quella che meglio si adatta alle vostre esigenze.

Ora potete godervi il vostro film preferito o ascoltare un po' di musica prima di passare al prossimo capitolo e imparare a usare un altro strumento molto interessante: il linguaggio Python.

Programmare con Python

Il linguaggio Python combina semplicità e potenza e ci permetterà di muovere i primi passi nel mondo della programmazione scoprendo, al contempo, le funzionalità del nostro Raspberry Pi.

Python è stato creato da Guido van Rossum sul finire degli anni '80. La sua idea era quella di realizzare un linguaggio dalla sintassi chiara e semplice, non troppo rigido e formale, ma abbastanza strutturato per consentire di realizzare applicazioni complesse e non solo semplici script.

Come già detto nell'introduzione, Python è un linguaggio interpretato. Pur con qualche svantaggio in termini di performance, questo lo rende molto semplice da utilizzare.

Per realizzare e provare i nostri programmi Python possiamo utilizzare indifferentemente la command line o l'interfaccia grafica. In entrambi i casi dovremo lanciare un'interprete Python. La versione a command line può essere lanciata con il comando `python3` (questa è la versione che utilizzeremo nei nostri esempi), la versione grafica lanciando **IDLE 3** dall'icona sul desktop. In entrambi i casi ci verrà mostrato un prompt e potremo inserire direttamente dei comandi.

Per scoprire la filosofia di Python è interessante esplorare uno degli "easter eggs" (sorprese nascoste nel software) dell'interprete Python.

Per scoprirlo basta digitare quanto riportato nell'esempio qui di seguito.

```
pi@piserver ~ $ python3
Python 3.2.3 (default, Mar  1 2013, 11:53:50)
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
>>>
```

Come potete leggere, a chi ha sviluppato Python piacciono le cose semplici e belle e, per quanto può valere l'opinione di chi scrive, l'obiettivo è stato raggiunto.

The screenshot shows the Python Shell window in the IDLE application. The title bar reads "Python Shell". The menu bar includes "File", "Edit", "Shell", "Debug", "Options", "Windows", and "Help". The main window displays the "Zen of Python" poem, starting with "Beautiful is better than ugly." and ending with "Namespaces are one honking great idea -- let's do more of those!". The bottom right corner of the window shows "Ln: 26 Col: 4".

```
Python 3.2.3 (default, Mar 1 2013, 11:53:50)
[GCC 4.6.3] on linux2
Type "copyright", "credits" or "license()" for more information.
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
>>>
```

Figura 6.1 - IDLE - lo Zen di Python.

Questo capitolo non vuole essere una descrizione completa delle funzioni e delle capacità del linguaggio Python. Esistono ottimi testi in commercio dedicati a questo linguaggio che possono sicuramente fornire informazioni più ampie ed esaustive. “Assaggeremo” invece un po’ di Python apprendendo le basi che ci consentiranno di utilizzarlo per continuare a sperimentare con Raspberry Pi negli ultimi capitoli di questo libro.

I primi passi

Nel [Capitolo 2](#) abbiamo scritto un piccolissimo programma Python; possiamo riproporlo ora. Se usate la command line potrete scriverlo utilizzando qualsiasi editor; se usate IDLE potrete selezionare l’opzione **New Window** del menu **File**, in modo da aprire una seconda finestra in cui inserire il codice del vostro programma.

Il codice è molto semplice:

```
#!/usr/bin/env python3
print("Ciao mondo!")
```

Per lanciare il nostro programma da IDLE è sufficiente selezionare l’opzione **Run Module** del menu **Run** oppure semplicemente premere il tasto **F5**. IDLE ci chiederà di salvare il file prima di lanciarlo.

Per lanciarlo dalla command line, invece, dovremo per prima cosa renderlo eseguibile con `chmod` e poi eseguirlo.

```
pi@piserver ~ $ chmod a+x helloworld.py
pi@piserver ~ $ ./helloworld.py
Ciao mondo!
```

Nell'esecuzione da command line il messaggio viene stampato sulla console; utilizzando IDLE il messaggio apparirà nella finestra principale del programma.

La prima linea del nostro programma sarà ripetuta in testa a tutti i nostri programmi Python. Serve per indicare al sistema quale interprete utilizzare per eseguire il programma, in modo analogo a quanto avviene per gli script di shell descritti nel [Capitolo 3](#).

In entrambi i casi il programma ha stampato il messaggio "Ciao mondo!". Questo è lo scopo della seconda linea. Questa linea esegue una chiamata alla funzione `print()` (con la lettera iniziale minuscola, anche Python è case sensitive!), che è una funzione della libreria base di Python.

Ogni funzione ha un nome univoco e per richiamarla è necessario passarle tra parentesi tonde gli argomenti, nel nostro caso il messaggio "Ciao mondo!". Se una funzione non ha bisogno di argomenti è comunque necessario usare le parentesi dopo il suo nome, per consentire a Python di interpretare l'istruzione come chiamata a funzione.

In questo testo indicheremo i nomi di funzione sempre facendoli seguire da parentesi tonde, in modo da renderne subito chiaro il significato.

E se volessimo capire meglio come funziona la funzione `print()`? Possiamo usare un'altra funzione, la funzione `help()`!

Se al prompt di Python (nella finestra principale di IDLE; oppure, lanciando `python3` sulla command line, verrà mostrato lo stesso prompt `>>>`) richiamiamo la funzione `help()` passando come argomento la funzione `print` (questa volta senza parentesi); otteniamo una succinta descrizione dei suoi parametri.

```
>>> help(print)
Help on built-in function print in module builtins:

print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)
```

```
Prints the values to a stream, or to sys.stdout by default.  
Optional keyword arguments:  
file: a file-like object (stream); defaults to the current sys.stdout.  
sep: string inserted between values, default a space.  
end: string appended after the last value, default a newline.  
flush: whether to forcibly flush the stream.
```

>>>

Nel caso della command line è necessario premere il tasto **Q** per tornare al prompt.

Purtroppo il messaggio `fi help` fa riferimento a diversi concetti che non abbiamo ancora affrontato e risulterà abbastanza criptico. Non preoccupatevi; alla fine di questo capitolo avrete le idee un po' più chiare e sarà più facile interpretare questo tipo di messaggi.

Il messaggio "Ciao mondo!" è una stringa. Con questo termine nei linguaggi di programmazione si indicano le sequenze di caratteri. Nel nostro caso la stringa è stata scritta direttamente nel codice del nostro programma, semplicemente scrivendola tra doppi apici. Avremmo potuto anche usare singoli apici, Python non è "schizzinoso" in merito, ma l'autore arriva da anni di sviluppo in C/C++ ed è abituato a usare i doppi apici per le stringhe. Nelle prossime pagine vedremo come chiedere dati all'utente, come leggerli e come memorizzarli in file e variabili.

Tipi e variabili

Normalmente, durante la loro esecuzione, i programmi manipolano e modificano dati di vario tipo.

Questi dati vengono normalmente memorizzati all'interno di variabili.

Alcuni linguaggi di programmazione, tipicamente quelli compilati, prevedono un controllo stretto sui tipi di dati e richiedono che lo sviluppatore specifichi il tipo di dato per ogni singola variabile.

Python, come molti linguaggi di scripting, non è invece così restrittivo ed è in grado di riconoscere e gestire i dati numerici (interi e a virgola mobile), le stringhe e i caratteri in modo automatico.

Possiamo assegnare delle variabili dichiarandole direttamente sul prompt dell'interprete:

```
>>> stringa='ciao'  
>>> intero=10  
>>> frazione=10/3
```

Come vedete dall'ultima operazione, a una variabile può anche essere assegnato il risultato di un'operazione o, come vedremo più avanti, di una chiamata a funzione.

La funzione `type()` consente di determinare il tipo di una variabile; possiamo utilizzarla per capire come Python gestisce i diversi tipi di dati.

```
>>> type(stringa)
<class 'str'>
>>> type(intero)
<class 'int'>
>>> type(frazione)
<class 'float'>
```

La prima variabile è una stringa (`str` in Python), la seconda contiene un valore intero (`int`), la terza un valore decimale a virgola mobile (`float`, abbreviazione di floating point, la modalità con cui vengono rappresentati i numeri non interi all'interno del nostro processore).

Possiamo anche stampare i valori delle tre variabili usando una sola chiamata alla funzione `print()`:

```
>>> print(stringa,intero,frazione)
ciao 10 3.333333333333335
```

Come avete appena visto, `print()` può prendere più argomenti e, indipendentemente dal loro tipo, li stampa in un formato adatto.

E cosa succede quando proviamo a utilizzare tipi di dati diversi?

```
>>> risultato=intero+frazione
>>> type(risultato)
<class 'float'>
>>> print(risultato)
13.3333333333334
```

I tipi numerici possono essere combinati tra loro. Un numero intero verrà convertito in un numero a virgola mobile in automatico, perché i numeri interi possono essere rappresentati in virgola mobile mentre non è vero il contrario.

Con i tipi numerici possiamo effettuare tutte le classiche operazioni matematiche: addizione (+), sottrazione (-), divisione (/) e moltiplicazione (*).

```
>>> 10+3
13
>>> 10-3
7
>>> 10/3
```

```
3.333333333333335
>>> 10*3
30
>>>
```

Come dimostrano gli esempi qui sopra, possiamo far eseguire un'operazione matematica all'interprete Python semplicemente scrivendola sul prompt. Utile quando non si ha una calcolatrice a portata di mano!

Nel caso di operazioni complesse possiamo determinare la precedenza tra le operazioni mediante le parentesi tonde.

```
>>> 10+3*2
16
>>> (10+3)*2
26
```

Python introduce un paio di operatori che non fanno parte della classica notazione matematica. L'operatore `%` restituisce il resto di una divisione:

```
>>> 10%3
1
```

L'altro operatore "particolare" è `**`, che consente di elevare un valore a potenza:

```
>>> 3**3
27
```

Molte altre funzioni matematiche sono disponibili nel modulo `math`. Per esempio, possiamo calcolare la radice quadrata di un numero con la funzione `math.sqrt()`:

```
>>> import math
>>> math.sqrt(25)
5.0
```

Come vedete nell'esempio, prima di usare le funzioni di un modulo è necessario importarlo con l'istruzione `import`.

Le funzioni di Python possono essere estese utilizzando i moduli. Alcuni moduli, come `math`, fanno parte della libreria standard e sono quindi disponibili in tutte le implementazioni di Python.

Altri moduli, che impareremo a installare e a utilizzare a breve, ci consentiranno di accedere a funzionalità estese come l'accesso ai pin di GPIO del nostro Raspberry Pi.

Se invece proviamo a sommare stringhe e interi Python ci segnala un errore, perché i due tipi non sono compatibili o, meglio, perché non può convertire direttamente una stringa in un intero.

Usando la funzione `str()` possiamo convertire il valore intero in una stringa:

```
>>> risultato=stringa+str(intero)
>>> print(risultato)
ciao10
```

Allo stesso modo possiamo convertire in intero una stringa, con la funzione `int()`:

```
>>> stringa="10"
>>> risultato=int(stringa)+intero
>>> print(risultato)
20
```

Ovviamente la conversione funziona solo se stringa contiene la rappresentazione di un numero, anche in basi diverse da 10; in quel caso è necessario passare un parametro extra a `int()`:

```
>>> int("aa",16)
170
```

Usando invece una stringa non numerica verrà generata un'eccezione. Impareremo a gestire le eccezioni nel corso di questo capitolo.

```
>>> stringa="ciao"
>>> risultato=int(stringa)+intero
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: 'ciao'
```

Fino a ora abbiamo utilizzato stringhe su una sola riga, ma se volessimo usare stringhe più complesse, divise in più righe? In questo caso (e nel caso in cui la stringa contenga i caratteri " e ` che servono a delimitare le stringhe) dobbiamo racchiudere il nostro testo tra tripli doppi apici o tripli apici singoli:

```
>>> stringa="""questa è una
... stringa su
... più linee"""
>>> print (stringa)
questa è una
stringa su
più linee
>>>
```

Sulle stringhe possiamo effettuare anche altre operazioni interessanti. Le funzioni `upper()` e `lower()` consentono di convertire i caratteri della nostra stringa in maiuscolo o minuscolo:

```
>>> stringa="Ciao"  
>>> stringa.upper()  
'CIAO'  
>>> stringa.lower()  
'ciao'
```

In questo caso, invece di chiamare la funzione passando come argomento la stringa, la funzione viene chiamata prefissandone il nome con la variabile da utilizzare. Questo avviene perché Python è un linguaggio a oggetti, quindi le stringhe sono oggetti che contengono dati (il testo) e hanno dei metodi (delle funzioni) in grado di operare sui dati stessi. La programmazione a oggetti si è enormemente diffusa a partire dalla fine degli anni '80, affiancando e in molti casi rimpiazzando la programmazione procedurale.

Java, C#, C++ sono linguaggi a oggetti, mentre il C è un linguaggio procedurale.

Uno dei vantaggi principali della programmazione a oggetti è la possibilità di incapsulare in un oggetto i dati e le funzioni che manipolano i dati stessi, come avviene appunto per le stringhe di Python.

Questo, in linea teorica, consente di strutturare l'applicazione in modo più pulito e manutenibile. Ma non dimentichiamoci che la qualità del software dipende più da chi lo ha realizzato che dagli strumenti usati per la sua realizzazione!

Il metodo `find()` consente di trovare una stringa all'interno di un'altra, ritornando l'indice del primo carattere oppure `-1` se non è stato possibile trovare la stringa:

```
>>> stringa="uno due tre quattro cinque"  
>>> stringa.find("uno")  
0  
>>> stringa.find("due")  
4  
>>> stringa.find("sette")  
-1  
>>>
```

Anche il metodo `index()` cerca una stringa all'interno di un'altra, ma genera un'eccezione se non la trova:

```
>>> stringa.index("due")
4
>>> stringa.index("sette")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: substring not found
```

Usando gli indici possiamo accedere a singoli caratteri o a parti della nostra stringa. Per usare gli indici dobbiamo adoperare le parentesi quadre `[]` e includere l'indice, o l'indice di inizio e quello di fine separati da due punti:

```
>>> stringa[0]
'u'
>>> stringa[1]
'n'
>>> stringa[4:7]
'due'
>>>
```

Una porzione di stringa è chiamata “slice”.

Noteate che i caratteri sono numerati partendo da zero e che l'indice di fine è escluso dalla slice generata. La stringa ritornata dall'espressione `stringa[4:7]` contiene cioè solo il quinto, sesto e settimo carattere della nostra stringa (non dimentichiamoci che la numerazione parte da zero!).

Omettendo l'indice iniziale la nostra slice partirà dall'inizio della stringa, omettendo l'indice di fine arriverà fino alla fine:

```
>>> stringa[:3]
'uno'
>>> stringa[3:]
' due tre quattro cinque'
>>> stringa[:]
'uno due tre quattro cinque'
```

La funzione `split()` consente di dividere la nostra stringa in parti usando un separatore. Se il separatore non viene specificato verrà usato lo spazio.

```
>>> stringa.split()
['uno', 'due', 'tre', 'quattro', 'cinque']
```

In questo caso avrete notato che le cinque stringhe in cui è stata divisa la stringa originale vengono stampate tra parentesi quadre. Questo avviene perché la funzione `split()` ritorna un oggetto di tipo `list`.

Le liste sono un tipo di dato molto importante in Python e anch'esse

possono essere assegnate a una variabile:

```
>>> lista=stringa.split()  
>>>
```

Possiamo stampare la nostra lista:

```
>>> print(lista)  
['uno', 'due', 'tre', 'quattro', 'cinque']
```

O accedere a singoli elementi o a gruppi usando la stessa notazione con le parentesi quadre utilizzata per le stringhe:

```
>>> lista[0]  
'uno'  
>>> lista[1:3]  
['due', 'tre']  
>>> lista[2:]  
['tre', 'quattro', 'cinque']  
>>> lista[:3]  
['uno', 'due', 'tre']
```

Come già visto per le stringhe, anche le liste sono indicizzate partendo da zero e le slice non contengono l'elemento con l'indice di fine.

La funzione `len()` consente di recuperare la lunghezza di una lista o di una stringa:

```
>>> len(lista)  
5  
>>> len(stringa)  
26
```

La funzione `sort()` consente di ordinare la nostra lista di stringhe in ordine alfabetico:

```
>>> lista.sort()  
>>> lista  
['cinque', 'due', 'quattro', 'tre', 'uno']
```

Fino a ora abbiamo lavorato sulla lista che ci ha tornato la funzione `split()`, ma possiamo anche dichiarare una lista specificando gli elementi tra parentesi quadre:

```
>>> lista=[1,2,3,4,5]  
>>> lista  
[1, 2, 3, 4, 5]
```

Gli elementi della nostra lista non devono necessariamente essere tutti dello stesso tipo, anche se questo è il caso più comune:

```
>>> lista=[1,"due",3,"quattro",5]
>>> lista
[1, 'due', 3, 'quattro', 5]
>>> type(lista[0])
<class 'int'>
>>> type(lista[1])
<class 'str'>
```

Come potete vedere dall'esempio, ogni elemento della lista mantiene un suo tipo ed è quindi perfettamente lecito usare le funzioni specifiche del suo tipo:

```
>>> lista[1].upper()
'DUE'
>>> lista
[1, 'due', 3, 'quattro', 5]
>>>
```

È altrettanto importante notare come la funzione `upper()` non modifichi la stringa originale, ma ne ritorni una nuova con i caratteri maiuscoli.

E se invece volessimo modificare l'elemento della nostra lista?

È sufficiente assegnargli un nuovo valore:

```
>>> lista[1]=lista[1].upper()
>>> lista
[1, 'DUE', 3, 'quattro', 5]
>>>
```

Possiamo aggiungere elementi alla lista usando l'operatore `+` e concatenando tra loro due o più liste:

```
>>> lista=lista+[6,7,8]
>>> lista
[1, 'DUE', 3, 'quattro', 5, 6, 7, 8]
```

Per inserire elementi nel mezzo della lista è possibile utilizzare la funzione `insert()`:

```
>>> lista.insert(1,1.5)
>>> lista
[1, 1.5, 'DUE', 3, 'quattro', 5, 6, 7, 8]
```

Per rimuovere un elemento dato il suo valore possiamo usare la funzione `remove()`:

```
>>> lista.remove("DUE")
>>> lista
[1, 1.5, 3, 'quattro', 5, 6, 7, 8]
>>> lista.remove("milleottocentocinquantanove")
Traceback (most recent call last):
```

```
File "<stdin>", line 1, in <module>
ValueError: list.remove(x): x not in list
```

Per rimuovere un elemento data la posizione possiamo sommare tra loro le slice con i valori che vogliamo conservare:

```
>>> lista[:1]+lista[2:]
[1, 3, 'quattro', 5, 6, 7, 8]
```

Per verificare se un elemento è parte di una lista possiamo usare la keyword **in**:

```
>>> 'quattro' in lista
True
>>> 42 in lista
False
```

Notate che `in` è stata definita keyword e non funzione, perché è una parola chiave del linguaggio Python e non un nome arbitrario di funzione che richiede degli argomenti.

È inoltre importante notare che le espressioni che abbiamo scritto con la keyword `in` ritornano un nuovo tipo di dato, il valore booleano. I valori booleani possono assumere solo due stati: vero (`True`) o falso (`False`).

Qualsiasi espressione che verifica una condizione ritorna un valore booleano. Anche i valori booleani possono essere memorizzati in variabili e apprezzeremo la loro importanza tra poco, quando vedremo come controllare l'esecuzione delle istruzioni all'interno del nostro programma Python.

```
>>> booleano=42 in lista
>>> booleano
False
```

Istruzioni condizionali e funzioni

È raro che un programma debba sempre essere eseguito in sequenza, dalla prima all'ultima istruzione.

Spesso capita di dover eseguire operazioni diverse a seconda dell'esito di una funzione o del confronto tra valori.

L'istruzione più semplice per gestire un'esecuzione condizionata è `if`. Partiamo da un semplicissimo programma di esempio:

```
#!/usr/bin/env python3
numero=21932822384
```

```
if (numero%2)==0 :  
    print("il numero è pari")
```

In questo caso l'istruzione `if` calcola il resto della divisione di `numero` per 2 e, se questo è uguale a zero, ci segnala che il numero è pari. In queste poche righe di codice ci sono diversi spunti interessanti.

Il primo è che la condizione deve sempre ritornare un valore booleano. Possiamo utilizzare diversi operatori condizionali e tutti ritornano `True` per segnalare che la condizione è rispettata e `False` in caso contrario.

Possiamo provarli scrivendo direttamente al prompt:

```
>>> 3>4  
False  
>>> 3<4  
True  
>>> 3==4  
False  
>>> 3!=4  
True  
>>> 3<=4  
True  
>>> 3>=4  
False  
>>>
```

Come potete vedere dagli esempi qui sopra Python supporta alcuni dei classici operatori di confronto matematici (`>`, `<`, `>=` e `<=`), mentre altri operatori sono definiti con una simbologia diversa. L'uguaglianza, che normalmente è rappresentata dal simbolo di uguale, si confonderebbe con l'assegnazione di un valore a una variabile e viene quindi rappresentata con un doppio uguale (`==`). La differenza, come avviene in C/C++, Java, JavaScript e altri linguaggi, si indica con l'operatore `!=`.

Per le condizioni è possibile utilizzare anche la keyword `in`, sia per le liste sia per le stringhe:

```
>>> "uno" in [ "uno", "due", "tre"]  
True  
>>> "ciao" in "ciao mondo"  
True
```

Esistono poi funzioni che ritornano un valore booleano, quindi possono essere utilizzate direttamente all'interno di una condizione:

```
>>> stringa="12345"  
>>> stringa.isdigit()  
True
```

In questo caso la funzione `isdigit()` chiamata su una stringa ritorna `True` se all'interno della stringa ci sono solo numeri e quindi potrà essere convertita senza errori in un intero.

È possibile invertire una condizione con la keyword `not`:

```
>>> not 3>4
```

```
True
```

Inoltre più condizioni possono essere concatenate mediante le keyword `and` (che ritorna `True` se tutte le condizioni sono `True`) o `or` (che ritorna `True` se almeno una delle condizioni è `True`):

```
>>> 3>4 and 5<6
```

```
False
```

```
>>> 3>4 or 5<6
```

```
True
```

La sintassi dell'istruzione `if` può essere definita come:

```
if <condizione> :  
    <istruzioni>
```

Nel nostro esempio l'unica istruzione condizionata dall'`if` è la chiamata alla funzione `print()`.

Ma come possiamo fare a condizionare più istruzioni? Le istruzioni condizionate devono essere raggruppate in un blocco.

La maggior parte dei linguaggi prevedono caratteri o keyword apposite per identificare l'inizio e la fine di un blocco (le parentesi graffe in C, C++, Java, JavaScript o le keyword "begin" e "end" in Pascal). Python, invece, fa affidamento sull'indentazione del codice. Per indentazione si intende l'inserimento di spazi vuoti prima di cominciare una linea di testo. Nei linguaggi di programmazione moderni l'indentazione si usa per rendere più leggibile il codice, per esempio indentando un blocco di istruzioni condizionato da un'istruzione `if` anche se il blocco è già definito mediante i caratteri speciali o le keyword.

Per far capire quanto l'indentazione renda più leggibile il codice basta vedere questi due piccoli esempi in codice C (entrambi sintatticamente corretti):

```
if ((numero%2)==0) { print("il numero è pari"); }
```

e:

```

if ((numero%2)==0)
{
    print("il numero è pari");
}

```

Nel secondo caso, che utilizza la normale formattazione del codice adoperata dai programmati C balza subito all'occhio che la chiamata a `print()` dipende in qualche modo dall'istruzione `if` che la precede.

Python sfrutta direttamente l'indentazione come metodo per delimitare i blocchi, rendendo il codice più compatto e leggibile.

Se prendiamo questo semplice esempio:

```

#!/usr/bin/env python3

if False:
    print("uno")
print("due")
print("tre")

```

e proviamo a eseguirlo otterremo questo risultato:

```

pi@pivalter ~ $ ./esempio02.py
due
tre

```

Se spostiamo le ultime due istruzioni allineandole con quella che segue l'`if`:

```

#!/usr/bin/env python3

if False:
    print("uno")
    print("due")
    print("tre")

```

il risultato cambierà:

```

pi@pivalter ~ $ ./esempio02.py
pi@pivalter ~ $

```

A questo punto, infatti, tutte e tre le chiamate a `print()` faranno parte del blocco condizionato dall'istruzione `if` e, visto che la condizione è sempre falsa, non verranno eseguite.

In molti casi può essere utile eseguire delle operazioni sia quando la condizione è valida, sia quando non lo è. In questo caso potremo utilizzare, oltre a `if`, anche la keyword `else`.

```

#!/usr/bin/env python3

```

```

numero=12439304904

if (numero%2)==1:
    print("il numero e' dispari")
else:
    print("il numero e' pari")

```

Se invece dobbiamo verificare più condizioni alternative tra loro, allora possiamo inserire nel nostro `if` delle istruzioni `elif` (`else if`); la condizione `else` potrà essere utilizzata alla fine delle varie `elif`.

```

#!/usr/bin/env python3

numero=12439304904

if (numero%2)==1:
    print("il numero e' dispari")
elif numero==0:
    print("il numero e' zero")
else:
    print("il numero e' pari")

```

I computer sono bravissimi a ripetere la stessa sequenza di operazioni più e più volte senza sbagliare e senza lamentarsi per la noia.

L'istruzione `for` permette di ripetere delle operazioni su ogni elemento di una sequenza. Possiamo provarlo direttamente dal prompt di Python:

```

>>> sequenza=(1,2,3,4,5,6,7)
>>> somma=0
>>> for numero in sequenza:
...     somma=somma+numero
...
>>> somma
28

```

Notate che dopo l'istruzione `for` l'interprete inserisce automaticamente dei puntini di sospensione (...) per indicare che è necessario scrivere altro codice. Per terminare il blocco di istruzioni all'interno del ciclo `for` è necessario inserire una riga vuota. Questo vale solo quando si digita il codice direttamente al prompt; non siamo obbligati invece a lasciare righe vuote nei nostri sorgenti, anche se lasciarle non causa alcun errore e in qualche caso potrebbe migliorare la leggibilità del codice.

Nel ciclo `for` viene dichiarata una variabile (chiamata `numero` nell'esempio precedente) che assume il valore dell'elemento corrente della sequenza su cui avviene l'iterazione. Spesso capita che il ciclo `for` debba semplicemente incrementare una variabile numerica per ripetere le operazioni un numero predeterminato di volte. In questo caso è

possibile usare la funzione `range()`. Questa funzione ritornerà una sequenza di numeri secondo i criteri indicati. Se richiamata con un solo parametro ritornerà i valori interi da zero a quello precedente il numero indicato:

```
>>> for val in range(10):
...     print(val)
...
0
1
2
3
4
5
6
7
8
9
>>>
```

Se chiamata con due argomenti ritornerà i valori tra il primo e il secondo escluso:

```
>>> for val in range(5,10):
...     print(val)
...
5
6
7
8
9 >>>
```

Un terzo parametro, sempre opzionale, consente di impostare l'incremento tra un valore e il successivo:

```
>>> for val in range(0,10,2):
...     print(val)
...
0
2
4
6
8
>>>
```

Non sempre è possibile conoscere a priori il numero di ripetizioni di un ciclo.

L'esecuzione di una nuova ripetizione potrebbe essere condizionata al risultato delle operazioni eseguite durante l'iterazione precedente, e queste potrebbero non prevedere una sequenza già definita. In questo

caso è possibile utilizzare l'istruzione `while`. Con questa istruzione è possibile specificare una condizione che determina la ripetizione o meno del ciclo.

Se la condizione è vera, il ciclo verrà eseguito e verrà poi valutata di nuovo la condizione e così via, fino a che la condizione non si rivelerà falsa.

```
>>> somma=0
>>> while somma<10:
...     somma+=1
...     print(somma)
...
1
2
3
4
5
6
7
8
9
10
```

In questo caso le istruzioni di incremento e stampa verranno ripetute fino a quando il valore del variabile `somma` non arriva a 10.

Notate come l'operatore `+=` ci consenta di scrivere in modo compatto e leggibile l'espressione:

```
somma=somma+1
```

Allo stesso modo, per sottrarre un valore da una variabile, potremo scrivere:

```
somma-=1
```

Altri operatori dello stesso tipo supportano la moltiplicazione (`*=`) e divisione (`/=`).

Un altro caso molto comune quando si sviluppa software è quello di dover eseguire operazioni molto simili, ma con parametri diversi. Abbiamo già visto che la libreria di Python mette a disposizione moltissime funzioni e che, proprio passando loro i giusti parametri, possiamo riutilizzarle più volte all'interno del nostro codice.

Prendiamo questo codice di esempio:

```
>>> fattoriale=1
>>> numero=10
>>> for contatore in range(1,numero+1):
```

```

...
    fattoriale*=contatore
...
>>> print("il fattoriale di "+str(numero)+" e' "+str(fattoriale))
il fattoriale di 10 e' 3628800

```

Questo codice calcola il fattoriale di 10. Se volessimo calcolare il fattoriale di 12 dovremmo re-inizializzare la variabile `numero` e la variabile `fattoriale` e riscrivere il ciclo `for`. È sicuramente più pratico raggruppare queste istruzioni in una **funzione**. Le funzioni possono avere parametri e ritornare un valore; entrambe le cose sono opzionali e potremo avere, quindi, funzioni che non ritornano alcun valore e altre che invece non hanno parametri.

Nel nostro caso possiamo dichiarare una funzione `fattoriale()` che prenderà come argomento il numero intero di cui vogliamo calcolare il fattoriale e ritornerà il valore calcolato.

```

>>> def fattoriale(numero=0):
...     fattoriale=1
...     for contatore in range(1,numero+1):
...         fattoriale*=contatore
...     return fattoriale
...
>>>

```

Anziché come variabile, `numero` è stato dichiarato come argomento della nostra funzione e gli è stato assegnato un valore di default (0) che sarà utilizzato se non verrà passato alcun argomento. La variabile `fattoriale` viene inizializzata e dichiarata dentro la nostra funzione e non c'è bisogno che sia dichiarata altrove. È una variabile locale, ossia è valida solamente dentro la funzione in cui viene utilizzata.

La keyword `return`, infine, consente di ritornare un valore dalla nostra funzione.

Ora possiamo calcolare un po' di fattoriali:

```

>>> print(fattoriale())
1
>>> print(fattoriale(10))
3628800
>>> print(fattoriale(20))
2432902008176640000
>>> print(fattoriale(1000))
402387260077093773543702433923003985719374864210714632543799910429938512398629020
044208486969404800479988610197196058631666872994808558901323829669944590997424504
073759918823627727188732519779505950995276120874975462497043601418278094646496291
393887437886487337119181045825783647849977012476632889835955735432513185323958463
557409114262417474349347553428646576611667797396668820291207379143853719588249808
86783837455973174613608537953452422158659320192809087829730843139284403281231558
036976801357304216168747609675871348312025478589320767169132448426236131412508780

```

Come vedere la funzione `fattoriale()` è un piccolo blocco di codice che ora possiamo riutilizzare più volte.

Ma se volessimo consentire a un qualsiasi utente di calcolare il fattoriale, senza dover entrare nell'interprete Python per lanciare la nostra funzione?

Nel prossimo paragrafo scopriremo come interagire con l'utente e come gestire gli errori.

Interazione con l'utente e gestione degli errori

Per poter utilizzare le nostre funzioni Python senza bisogno di appoggiarci al prompt dell'interprete dobbiamo realizzare un programma che sia in grado di chiedere i valori da calcolare direttamente all'utente. Per farlo possiamo utilizzare la funzione `input()`. Questa funzione consente di mostrare all'utente un messaggio e di attendere che l'utente inserisca un valore e prema il tasto **Invio**.

```
#!/usr/bin/env python3

def fattoriale(numero):
    fattoriale=1
    for contatore in range(1,numero+1):
        fattoriale*=contatore
    return fattoriale
```

```
valore=input("Inserisci il valore di cui vuoi calcolare il fattoriale : ")
risultato=fattoriale(int(valore))
print("Il fattoriale e' :" +str(risultato))
```

Se proviamo a eseguire il programma potremo inserire direttamente sulla console il valore da calcolare (in questo caso il programma è stato salvato come `esempiofattoriale.py` ed eseguito da console):

```
pi@pivalter ~ $ ./esempiofattoriale.py
Inserisci il valore di cui vuoi calcolare il fattoriale : 10
Il fattoriale e' :3628800
pi@pivalter ~ $ ./esempiofattoriale.py
Inserisci il valore di cui vuoi calcolare il fattoriale : 20
Il fattoriale e' :2432902008176640000
```

In realtà il nostro programma non è molto solido:

```
pi@pivalter ~ $ ./esempiofattoriale.py
Inserisci il valore di cui vuoi calcolare il fattoriale : ciao
Traceback (most recent call last):
  File "./esempiofattoriale.py", line 10, in <module>
    risultato=fattoriale(int(valore))
ValueError: invalid literal for int() with base 10: 'ciao'
```

Se il valore inserito non è un numero valido, il sistema genera un errore e termina il programma.

In Python gli errori vengono gestiti generando delle eccezioni. Le eccezioni vengono generate quando l'interprete o una delle funzioni di libreria riconosce che un dato non è corretto o si verifica qualche altro tipo di errore; se non vengono intercettate dal nostro codice, arrivano all'interprete che a quel punto termina il programma con un messaggio di errore come quello generato nel caso precedente.

Per intercettare le eccezioni generate all'interno dei nostri programmi possiamo utilizzare le keyword `try` ed `except`.

Queste keyword ci permettono di definire un blocco di istruzioni (a seguito dell'istruzione `try`) e di fare in modo che le eccezioni sollevate possano essere intercettate dalle clausole `except` immediatamente successive.

Le eccezioni Python sono organizzate gerarchicamente con il tipo `Exception`, che è il tipo base da cui derivano diverse eccezioni specializzate. Vedremo più avanti come derivare un nostro tipo di eccezione e come organizzare il nostro codice mediante la programmazione a oggetti.

Vediamo come può essere modificato il nostro codice per gestire meglio eventuali errori di inserimento.

```

#!/usr/bin/env python3

def fattoriale(numero):
    fattoriale=1
    for contatore in range(1,numero+1):
        fattoriale*=contatore
    return fattoriale

try:
    valore=int(input("Inserisci il valore di cui vuoi calcolare il fattoriale"))
    risultato=fattoriale(valore)
except ValueError:
    print("Il valore inserito non è valido.")
    quit()
print("Il fattoriale e' :" +str(risultato))

```

Utilizzando le keyword `try` e `except` il nostro programma intercetta l'eccezione `ValueError`, che viene generata quando un numero non può essere convertito in una stringa. A questo punto, invece di terminare con il generico messaggio di errore di Python, l'applicazione segnalerà che il dato inserito non è corretto e terminerà chiamando la funzione `quit()`.

```

pi@pivalter ~ $ ./esempiofattoriale.py
Inserisci il valore di cui vuoi calcolare il fattoriale : ciao
Il valore inserito non è valido.

```

C'è ancora un piccolo problema. Se inseriamo un numero negativo, il programma ci ritornerà comunque un valore per il fattoriale.

```

pi@pivalter ~ $ ./esempiofattoriale.py
Inserisci il valore di cui vuoi calcolare il fattoriale : -1
Il fattoriale e' :1

```

Il calcolo del fattoriale per numeri negativi non è definito e sarebbe più corretto che la nostra funzione di calcolo del fattoriale verificasse che il parametro passatogli sia un intero positivo, generando un errore in caso contrario.

```

#!/usr/bin/env python3

def fattoriale(numero):

    numero=int(numero)

    if numero<0:
        raise ValueError

    fattoriale=1
    for contatore in range(1,numero+1):

```

```

fattoriale*=contatore
return fattoriale

try:
    valore=int(input("Inserisci il valore di cui vuoi calcolare il fattoriale"))
    risultato=fattoriale(valore)
except ValueError:
    print("Il valore inserito non è valido.")
    quit()

print("Il fattoriale e' :" +str(risultato))

```

La conversione esplicita di numero in intero fatta dalla funzione `int()` genererà un’eccezione se il dato passato non è un numero intero, ma non fallirà nel caso di interi negativi. È quindi necessario aggiungere una condizione `if` che controllerà se il nostro parametro è minore di zero e, in caso affermativo, genererà un’eccezione usando la keyword `raise`.

La gestione degli errori è fondamentale per riuscire a realizzare applicazioni stabili e robuste. Il meccanismo delle eccezioni di Python consente di farlo in modo molto chiaro e leggibile.

Se volessimo utilizzare il nostro piccolo programma da uno degli script di cui abbiamo parlato nel [Capitolo 3](#), dover inserire interattivamente il valore per cui calcolare il fattoriale potrebbe essere limitante e scomodo. Meglio passare il valore direttamente sulla command line, come argomento del nostro programma.

Per gestire i parametri a linea di comando possiamo utilizzare l’oggetto `sys.argv`. Quando il nostro programma viene lanciato dalla command line in `sys.argv` avremo una lista degli argomenti che gli sono stati passati.

Il primo argomento, come al solito con indice zero, è sempre presente, e contiene il percorso del nostro script.

Se state usando IDLE per provare gli esempi di questo capitolo, dovete comunque lanciare lo script dalla command line, dopo averlo reso eseguibile con `chmod`. Non è possibile passare argomenti da command line quando si lancia un programma Python dentro IDLE.

```

#!/usr/bin/env python3

import sys

def fattoriale(numero):
    numero=int(numero)

    if numero<0:
        raise ValueError

```

```

fattoriale=1
for contatore in range(1,numero+1):
    fattoriale*=contatore
return fattoriale

if len(sys.argv)!=2:
    print("Utilizzo: esempiofattoriale.py valore")
    quit()

try:
    risultato=fattoriale(sys.argv[1])
except ValueError:
    print("Il valore inserito non è valido.")
    quit()

print("Il fattoriale e' :" +str(risultato))

```

Per utilizzare le funzioni e gli oggetti del modulo `sys` è necessario importarlo mediante la keyword `import`.

A questo punto il programma verificherà che sia stato passato un solo parametro sulla command line (la lunghezza della lista argomenti sarà `2` perché l'argomento `argv[0]` è sempre valido e contiene il path del programma) e, in caso negativo, stamperà un messaggio per aiutare l'utente a correggere il suo errore. Il valore `argv[1]` viene passato direttamente alla funzione di calcolo del fattoriale, che lo validerà con le modalità descritte nel passaggio precedente.

```

pi@pivalter ~ $ ./esempiofattoriale.py 10
Il fattoriale e' :3628800
pi@pivalter ~ $ ./esempiofattoriale.py ciao
Il valore inserito non è valido.
pi@pivalter ~ $ ./esempiofattoriale.py -1
Il valore inserito non è valido.
pi@pivalter ~ $ ./esempiofattoriale.py 10 20
Utilizzo: esempiofattoriale.py valore

```

Ora che abbiamo realizzato il nostro primo programma completo possiamo passare a scoprire come utilizzare Python per consentire al nostro Raspberry Pi di interagire con il mondo esterno. Ovviamente queste poche pagine non hanno la pretesa di essere una descrizione esaustiva di tutte le capacità di Python e ne scopriremo ancora molte con gli esempi pratici dei prossimi capitoli. Ora però è tempo di tradurre in pratica quello che abbiamo visto in queste pagine.

Rimboccatevi le maniche perché nel prossimo capitolo ci sarà da lavorare!

Interagire con sensori e attuatori

Interfacciare sensori e attuatori con Raspberry Pi è semplice. Sfruttando il linguaggio Python e le librerie che ci consentono di controllare i segnali del connettore GPIO possiamo fare interagire il nostro piccolo computer con il mondo reale.

Nell'introduzione abbiamo notato che il connettore GPIO di Raspberry Pi non ha un equivalente sui normali PC, tablet e smartphone. Questo connettore permette di collegare direttamente al nostro piccolo personal computer dispositivi esterni, permettendoci di acquisire informazioni sull'ambiente circostante tramite i sensori e di interagire con oggetti reali tramite gli attuatori.

Il connettore GPIO

Il connettore di General Purpose Input Output di Raspberry Pi è composto da 26 piedini, più comunemente chiamati pin. Questi pin distano tra loro 2,54 mm, un decimo di pollice. Questa era la misura standard dei connettori prima che la miniaturizzazione dei componenti portasse a ridurre le dimensioni di tutto, connettori compresi.

Le funzioni di alcuni dei pin di questo connettore cambiano tra le due revisioni di Raspberry Pi.

La prima revisione, presentata al lancio, è stata infatti rivista per risolvere alcuni problemi e migliorare le funzionalità (nel caso del Model B monta 512 MB di RAM invece di 256).

Per controllare la revisione hardware della nostra board è sufficiente aprire una linea di comando e stampare il file `/proc/cpuinfo`.

```
pi@pivalter ~ $ cat /proc/cpuinfo
Processor      : ARMv6-compatible processor rev 7 (v6l)
BogoMIPS       : 697.95
Features       : swp half thumb fastmult vfp edsp java tls
CPU implementer: 0x41
CPU architecture: 7
CPU variant    : 0x0
CPU part       : 0xb76
CPU revision   : 7

Hardware      : BCM2708
Revision       : 000e
Serial         : 000000004edafaaf
```

La versione del nostro Raspberry Pi è riportata nel campo **Revision**, evidenziato nell'esempio precedente.

Tutti i valori tra 0003 e 000f (si tratta di valori esadecimali) indicano che il vostro dispositivo è delle revisione 2.0 che, al momento della stesura di questo libro, è l'ultima disponibile.

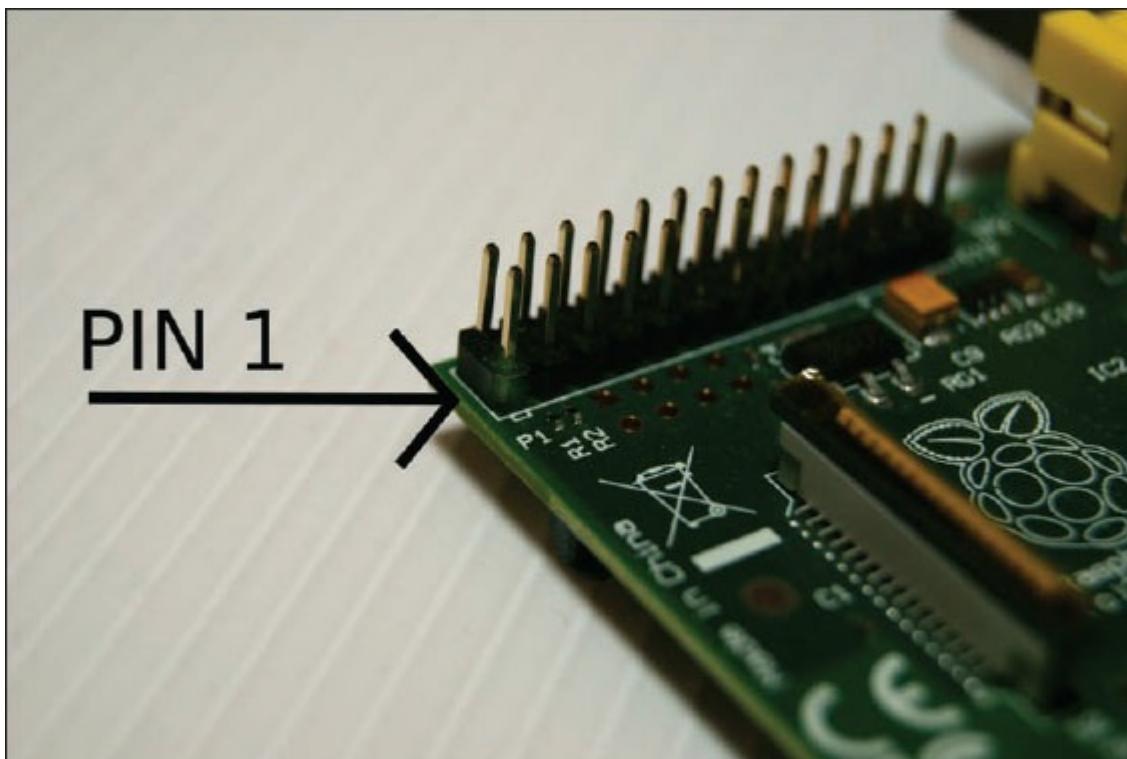


Figura 7.1 - Il connettore GPIO.

Le singole revisioni indicano la quantità di memoria RAM installata

(256 o 512 MB) e l'impianto in cui è stata prodotta la vostra board (board prodotte in impianti diversi sono funzionalmente equivalenti). Questo capitolo è basato sulla revisione 2 dell'hardware di Raspberry Pi, rilasciata a settembre 2012 e più diffusa. Se il vostro Raspberry Pi è revisione 1 non preoccupatevi: le differenze sono minime e vi basterà cambiare i pin utilizzati negli esempi per poter seguire comunque questo capitolo.

Il connettore GPIO si trova in uno degli angoli della board, a fianco del connettore RCA. Nella [Figura 7.1](#) è evidenziato il pin numero 1. I pin sono numerati progressivamente con i pin dispari nella fila verso il centro della board e quelli pari verso l'esterno.

In questa tabella sono riportate le funzionalità dei vari pin, sia per la revisione 1 sia per la revisione 2 di Raspberry Pi.

Pin	Funzione rev. 1	Funzione rev. 2
1	Alimentazione 3,3 V	
2	Alimentazione 5 V	
3	GPIO 0/I2C0 SDA	GPIO 2/I2C1 SDA
4	Non collegato	Alimentazione 5 V
5	GPIO 1/I2C0 SCL	GPIO 3/I2C1 SCL
6		GND
7		GPIO 4
8		GPIO 14/UART TXD
9	Non collegato	GND
10		GPIO 15/UARD RXD
11		GPIO17
12		GPIO18
13	GPIO21	GPIO27
14	Non collegato	GND
15		GPIO22
16		GPIO23
17	Non collegato	Alimentazione 3,3 V
18		GPIO24
19		GPIO 10/SP10 MOSI
20	Non collegato	GND

21	GPIO 9/SP10 MISO	
22	GPIO25	
23	GPIO 11/SP10 SCLK	
24	GPIO 7/SP10 CE0 N	
25	Non collegato	GND
26	GPIO 8/SP10 CE1 N	

Il connettore GPIO ha dei pin di alimentazione, a 3,3 V e a 5 V, con i relativi pin di massa (GND).

Nella seconda revisione diversi pin non utilizzati sono stati dedicati, intelligentemente, ad alimentazioni aggiuntive, per semplificare i collegamenti nel caso di più dispositivi collegati al connettore.

Per collegare dispositivi e sensori ci sono diverse alternative. Si possono collegare direttamente ai pin utilizzando cavi jumper femmina-femmina oppure si possono usare delle basette sperimentali (breadboard). In questo caso i segnali del connettore andranno riportati sulla basetta, sempre usando dei cavi jumper (femmina-maschio in questo caso) oppure mediante un adattatore apposito.

Esistono diversi adattatori dedicati a Raspberry Pi come il Pi Cobbler di Adafruit (www.adafruit.com), potrete trovare anche una lista di distributori) o quelli reperibili sul sito MODMYPI (www.modmypi.com).

Se pensate di sperimentare parecchio con il vostro Raspberry Pi questi cavi potrebbero semplificarvi la vita, sia durante la realizzazione del prototipo sia quando monterete, con l'aiuto di un saldatore, il vostro circuito definitivo su una basetta millefori o su un circuito stampato.

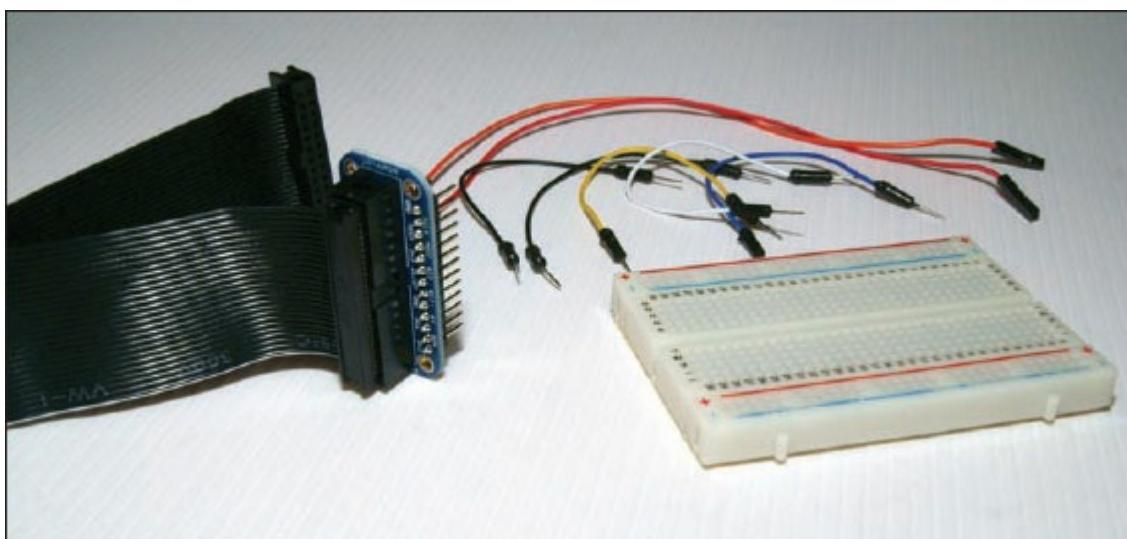


Figura 7.2 - Cavi jumper, basetta sperimentale e PI Cobbler.

Le basette sperimentali sono basette in plastica e metallo con già pronti dei buchi per inserire i pin dei vostri componenti. In questo modo si può evitare di saldare i componenti, è possibile realizzare prototipi rapidamente e, soprattutto, correggere eventuali errori senza troppo lavoro.

Le basette sono divise in piste e tutti i fori della stessa pista sono collegati tra loro.

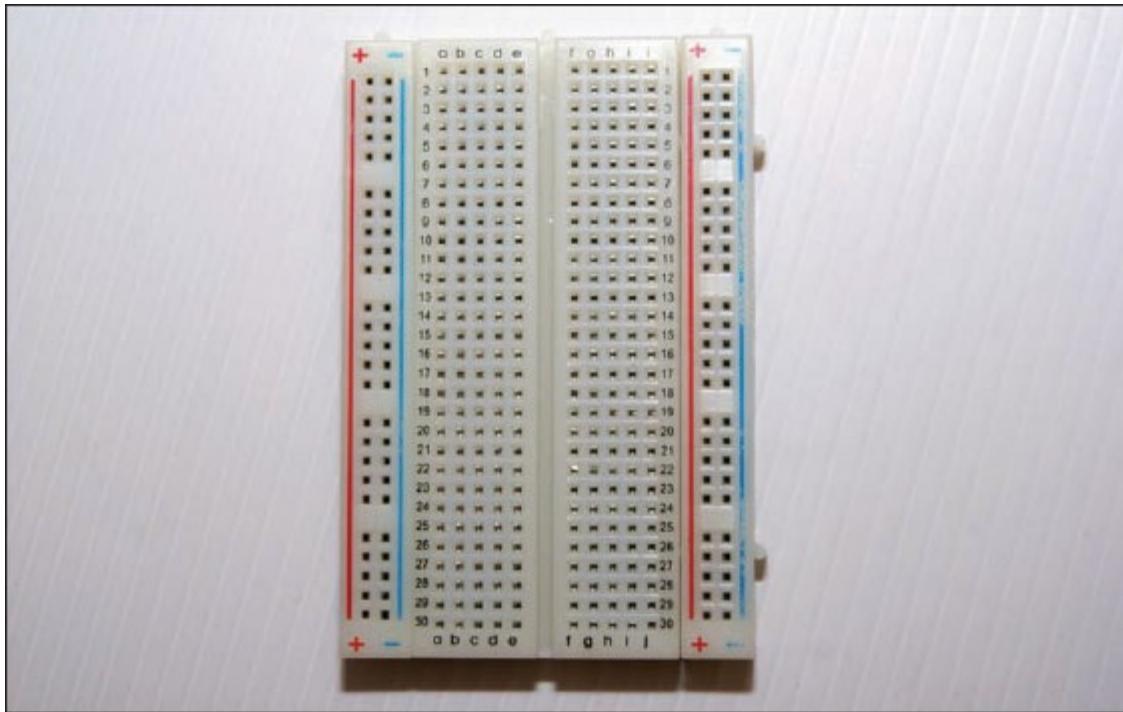


Figura 7.3 - Basetta sperimentale - le piste sono numerate da 1 a 30 e sono presenti piste verticali per l'alimentazione.

Normalmente hanno una scanalatura nel mezzo che separa le piste, in modo che a cavallo della scanalatura stessa possano essere montati dei chip che hanno pin sui due lati. Alcune basette, inoltre, hanno delle piste esterne che possono essere utilizzate per collegare l'alimentazione e la massa, facilitandone il trasporto ai diversi componenti.

Usando i cavi jumper si possono eseguire collegamenti tra le piste e quindi realizzare i circuiti senza bisogno di fare collegamenti permanenti. Negli esempi che faremo in questo capitolo i diagrammi presenteranno delle basette sperimentali, per aiutarvi nel cablaggio.

Come avrete sicuramente notato scorrendo i nomi dei pin, solo alcuni di essi sono marcati soltanto come GPIO. Per GPIO si intende un pin di ingresso/uscita digitale. I pin del connettore non seguono una numerazione progressiva, ma mantengono la stessa numerazione dei pin di GPIO del processore. Questo spiega perché abbiamo GPIO 4, 17

o 22 ma non 12 o 13.

Oltre ai pin di GPIO sono riportati sul connettore i pin di tre interfacce: UART (porta seriale), I2C (Inter-Integrated Circuit) e SPI (Serial Peripheral Interface).

Questi pin possono essere utilizzati, come vedremo in seguito, per far comunicare Raspberry Pi con altri dispositivi. Possono anche essere adoperati come normali pin di GPIO e questo spiega la doppia nomenclatura nella tabella dei pin.

È importante notare che tutti i pin del connettore di GPIO funzionano a 3,3 V (a eccezione dell'alimentazione a 5 V che troviamo sul pin 2). Non è quindi possibile collegarli a dispositivi che funzionino a tensioni diverse, pena il danneggiamento del vostro adorato piccolo personal computer o dei dispositivi collegati.

È possibile ovviare a questo inconveniente con dei circuiti ad-hoc ma, in linea generale, prima di collegare qualsiasi cosa al vostro Raspberry Pi, controllate che operi a 3,3 V. L'esperienza diretta dell'autore consiglia inoltre di effettuare sempre i collegamenti quando il vostro Raspberry Pi è spento e di ricontrollare le singole connessioni prima di dare tensione. Se non avete trovato alcun errore, ricontrollate fino a quando non lo trovate prima di accendere!

Se sentite puzza di bruciato o qualche componente scalda in modo sensibile, togliete subito l'alimentazione. Pochi secondi in più o in meno possono fare la differenza tra un componente caldo e uno bruciato per sempre.

Le correnti e le tensioni che utilizzeremo negli esempi di questo testo non sono pericolose, quindi non fatevi prendere dal panico se qualche cosa dovesse andare storto. Sbagliando si impara.

Funzionamento dei pin di GPIO e primi esperimenti

Se la definizione di GPIO come input/output digitale vi ha lasciati un po' perplessi, non preoccupatevi, è giunto il momento di chiarire meglio questo concetto.

Come sicuramente saprete i computer e tutti i sistemi basati su microprocessori lavorano in logica digitale. Tutte le informazioni sono memorizzate in bit binari che possono valere 0 o 1, a seconda del livello di tensione. Allo stesso modo anche i segnali sul connettore GPIO possono assumere solo due stati: alto (on, tensione a 3,3 V) o basso

(off, tensione a zero). I pin di GPIO possono essere configurati come ingressi (input) o uscite (output); nel resto di questo testo userò i termini inglesi. Mi scuso con chi preferisce l'uso dell'italiano, ma questi sono i termini che si trovano più spesso sulla documentazione tecnica, anche in quella in italiano.

Quando un pin di GPIO è configurato come output, allora è la CPU di Raspberry Pi a controllarne lo stato. Potremo impostare lo stato on, quindi avere una tensione in uscita, oppure lasciarlo off e quindi non avere tensione in uscita. Questo ci permetterà di pilotare dei dispositivi esterni che possono essere accesi o spenti.

In realtà la corrente che può essere erogata dai pin di GPIO è molto limitata, quindi non sarà possibile collegare direttamente dispositivi che richiedono potenza ai nostri pin. Vedremo in seguito come ovviare anche a questo inconveniente.

Se invece configuriamo il pin come ingresso, allora Raspberry Pi misurerà la tensione sul pin e lo leggerà nello stato on se questa è a 3,3 V o nello stato off se questa è a 0 V.

Come potete vedere, trattandosi di ingressi digitali, non è possibile avere misure diverse da on e off.

Quando si lavora con microcontrollori o dispositivi che non hanno un sistema operativo evoluto l'equivalente del piccolo programma che stampa "Ciao Mondo!" sulla console è un programma che fa lampeggiare un LED (Light Emitting Diode).

I LED sono piccoli diodi emettitori di luce. Nei diodi la corrente può fluire in una sola direzione (e infatti li si utilizza spesso per proteggere i circuiti). Nel caso dei LED il flusso di corrente genera anche emissione luminosa.

Da sempre i LED sono gli indicatori per antonomasia e, fin dagli albori dell'informatica, in ogni film di fantascienza non poteva mancare l'immagine di una sala di comando piena di quadri con lucine lampeggianti.

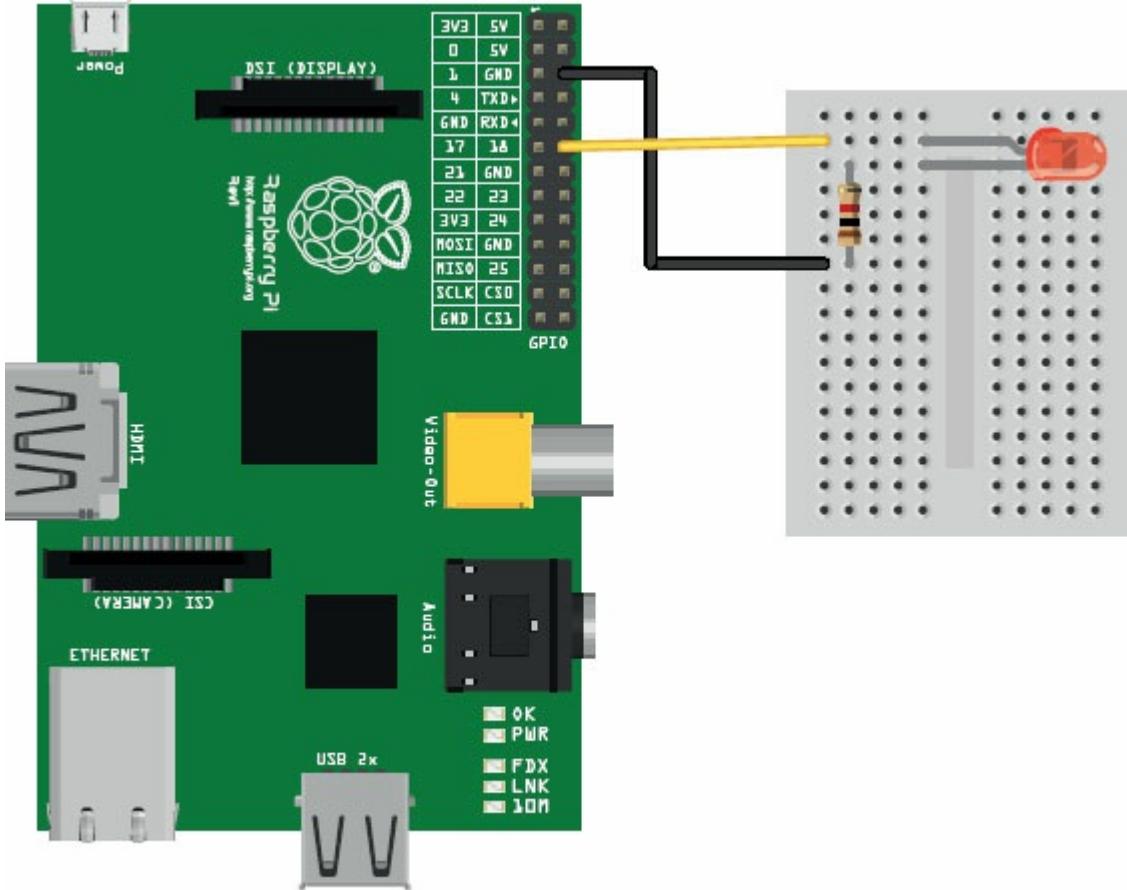
Per collegare un LED al nostro Raspberry Pi dobbiamo collegarlo a un pin di GPIO utilizzabile come output e fare in modo che l'attivazione del pin di uscita vada ad accendere il LED. Non basta però collegare il LED al pin. È necessario inserire nel circuito anche una resistenza. Le resistenze si oppongono al flusso della corrente e, nel nostro caso, limiteranno la corrente nel circuito, evitando di danneggiare i nostri componenti. La resistenza usata in questo esempio è da 1 K Ω ; più alto

è il valore di resistenza più sarà bassa la corrente e meno luminoso il LED.

NOTA

Le resistenze sono componenti che, come indica il nome, si oppongono al passaggio della corrente. Sono utili per limitare la quantità di corrente che passa nei circuiti ed evitare di danneggiare altri componenti con correnti troppo alte. Il valore di resistenza si misura in Ohm (abbreviato con la lettera greca omega, Ω). Il valore delle resistenze da utilizzare nei nostri esperimenti dipende dai componenti coinvolti ed è tipicamente tra 1 K Ω e 10 K Ω . Nei negozi fisici o online per gli hobbysti elettronici potrete trovare a pochi euro set con una decina o più di resistenze nei valori più comunemente utilizzati. Nei dispositivi moderni si utilizzano resistenze minuscole che sono piccoli blocchi rettangolari (ne potete vedere diversi anche sul vostro Raspberry Pi), mentre per i nostri esperimenti utilizzeremo resistenze tradizionali, adatte per essere utilizzate con le breadboard. Queste resistenze sono piccoli "salsicciotti" lunghi poco meno di un centimetro con due piedini metallici lunghi 2-3 centimetri, che possono essere facilmente inseriti nei fori delle basette sperimentali. Il valore della resistenza è indicato con una serie di bande colorate, tipicamente tre per il valore e una per la tolleranza (una percentuale di errore tra valore reale e nominale del componente). Potrete trovare la descrizione della codifica nella pagina di Wikipedia dedicata ai resistori: <http://it.wikipedia.org/wiki/Resistore#Codifica>.

Ricorriamo, come già nel [Capitolo 5](#), a Fritzing per disegnare uno schema di collegamento per il nostro LED.



Made with Fritzing.org

Figura 7.4 - Collegamento del LED.

Come si può vedere dallo schema il LED è collegato al pin GPIO 18 (pin 12 del connettore GPIO) e, passando per la resistenza, al pin di massa (pin 6 del connettore GPIO). I LED devono essere montati nel verso giusto per poter funzionare. Osservando il vostro LED vi accorgerete che uno dei lati del contenitore è smussato; la smussatura corrisponde al pin che deve essere montato verso massa (Gnd). A volte l'altro pin, quello che andrà collegato al pin di GPIO è più lungo di quello verso Gnd e anche questo è un buon metodo per capire la direzione in cui va montato il LED.

Comunque non c'è da preoccuparsi. In caso di errori il nostro componente non subirà danni; basterà girarlo per farlo funzionare correttamente.

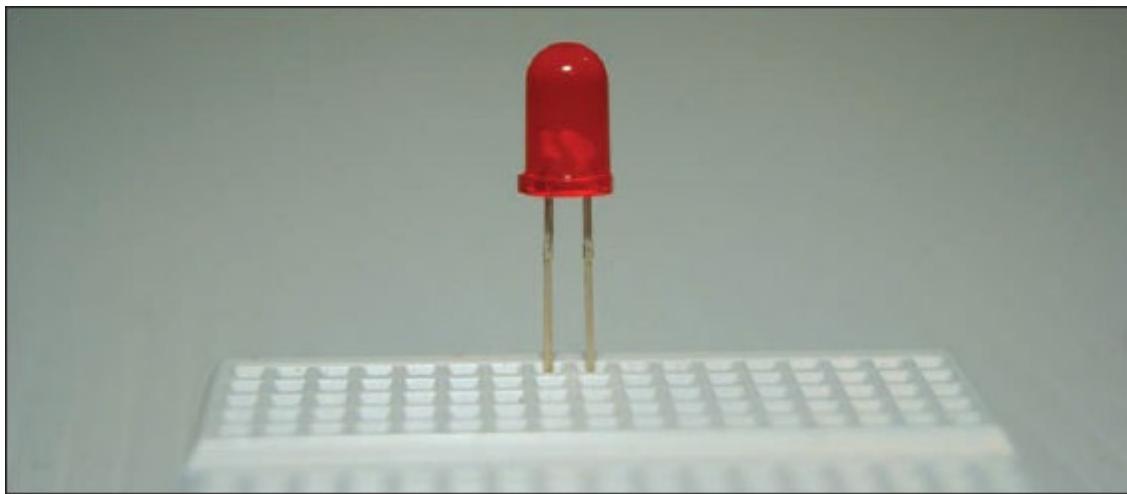


Figura 7.5 - Un LED inserito nella basetta.

Una volta collegato il nostro LED potremo controllarlo via software e per farlo torneremo a utilizzare Python.

Prima di poter utilizzare i nostri GPIO da Python, dovremo installare alcuni componenti. Per prima cosa possiamo installare la libreria che ci permetterà di controllare i pin di GPIO: quick2wire (www.quick2wire.com).

Questa libreria viene sviluppata insieme a una serie di schede di espansione che semplificano il collegamento di dispositivi esterni a Raspberry Pi, ma può anche essere utilizzata per controllare i segnali del connettore GPIO senza alcuna board collegata, come faremo nei nostri esempi.

La libreria è open source e per installarla dovremo scaricarne i sorgenti. Per prima cosa torniamo alla cartella home e creiamo una sotto-cartella per la libreria:

```
pi@pivalter ~/python-distribute $ cd  
pi@pivalter ~ $ mkdir quick2wire  
pi@pivalter ~ $ cd quick2wire/  
pi@pivalter ~/quick2wire $
```

Per scaricare i sorgenti useremo `git`. `Git` è lo strumento che Linus Torvalds ha sviluppato per gestire i sorgenti del kernel di Linux ed è poi diventato uno dei tool più diffusi per gestire sorgenti condivisi tra più programmatori, sia in ambito open source sia in ambito commerciale.

Per prima cosa dobbiamo creare un repository `git` nella nostra nuova cartella:

```
pi@pivalter ~/quick2wire $ git init  
Initialized empty Git repository in /home/pi/quick2wire/.git/
```

Questo ci consentirà di sincronizzare la cartella con i sorgenti del progetto quick2wire-python-api; sempre utilizzando git e dopo aver collegato il nostro Raspberry Pi alla rete, possiamo lanciare l'operazione di `clone` e scaricare i sorgenti aggiornati:

```
pi@pivalter ~/quick2wire $ git clone https://github.com/quick2wire/quick2wire-pyt
Cloning into 'quick2wire-python-api'...
remote: Counting objects: 1674, done.
remote: Compressing objects: 100% (659/659), done.
remote: Total 1674 (delta 939), reused 1614 (delta 884)
Receiving objects: 100% (1674/1674), 3.08 MiB | 23 KiB/s, done.
Resolving deltas: 100% (939/939), done.
```

L'operazione genererà una nuova sotto-cartella con i sorgenti; potete controllarne il contenuto usando `ls`:

```
pi@pivalter ~/quick2wire $ ls quick2wire-python-api/
```

Per funzionare la libreria ha bisogno di un'utility chiamata `gpio-admin`; possiamo scaricare anche questa con `git`:

```
pi@pivalter ~/quick2wire $ git clone https://github.com/quick2wire/quick2wire-gpi
Cloning into 'quick2wire-gpio-admin'...
remote: Counting objects: 184, done.
remote: Compressing objects: 100% (78/78), done.
remote: Total 184 (delta 90), reused 182 (delta 88)
Receiving objects: 100% (184/184), 39.62 KiB, done.
Resolving deltas: 100% (90/90), done.
```

In questo caso si tratta di un'applicazione in C. Dobbiamo limitarci a compilarla e installarla.

```
pi@pivalter ~/quick2wire $ cd quick2wire-gpio-admin/
pi@pivalter ~/quick2wire/quick2wire-gpio-admin $ make
gcc -Wall -O2 src/gpio-admin.c -c -o out/Linux-armv6l/gpio-admin.o
gcc out/Linux-armv6l/gpio-admin.o -o out/Linux-armv6l/gpio-admin
gzip -c man/gpio-admin.1 > out/gpio-admin.1.gz
pi@pivalter ~/quick2wire/quick2wire-gpio-admin $ sudo make install
mkdir -p /usr/local/bin/
install out/Linux-armv6l/gpio-admin /usr/local/bin/
mkdir -p /usr/local/share/man/man1/
install out/*.1.gz /usr/local/share/man/man1/
groupadd -f --system gpio
chgrp gpio /usr/local/bin/gpio-admin
chmod u=rwxs,g=rx,o= /usr/local/bin/gpio-admin
pi@pivalter ~/quick2wire/quick2wire-gpio-admin $
```

GPIO-admin consente ai semplici utenti di accedere ai pin di GPIO, a patto di essere inclusi nel gruppo `gpio`.

Prima di usare la libreria dobbiamo quindi aggiungere il nostro utente

al gruppo `gpio`. Per farlo possiamo usare il comando `adduser`, già visto nel Capitolo 3:

```
pi@pivalter ~/quick2wire/quick2wire-gpio-admin $ sudo adduser pi gpio
Aggiunta dell'utente «pi» al gruppo «gpio» ...
Aggiunta dell'utente pi al gruppo gpio
Fatto.
```

Ora possiamo riavviare il nostro Raspberry Pi e apprestarci a controllare i pin di GPIO dal nostro codice Python, lanciando l'interprete (IDLE3 o `python3`).

Per prima cosa dobbiamo aggiungere la cartella dei sorgenti della libreria quick2wire ai percorsi in cui Python cerca le librerie.

Per farlo dobbiamo usare l'oggetto `sys.path`:

```
>>> import sys
>>> sys.path.append("/home/pi/quick2wire/quick2wire-python-api/")
```

A questo punto possiamo importare le funzioni di `quick2wire.gpio` nell'interprete usando la keyword `import`:

```
>>> import quick2wire.gpio as GPIO
```

In questo caso la keyword `import`, già vista nel capitolo precedente, è utilizzata insieme alla keyword `as` per consentirci di usare le funzioni importate referenziandole con il prefisso `GPIO` al posto del prefisso originale.

I pin del connettore GPIO sono gestiti dall'oggetto `pi_header_1`.

Per controllare un pin dobbiamo ricavare un oggetto Python associato al pin stesso e lo possiamo fare chiamando il metodo `pin()` di `pi_header_1`:

```
>>> ledpin=GPIO.pi_header_1.pin(12,GPIO.OUT)
```

Notate che il pin non è indicizzato rispetto alla CPU (`GPIO18`), ma rispetto alla sua posizione sul connettore GPIO. Passando `GPIO.OUT` come secondo parametro andremo a configurare il pin come uscita.

L'oggetto che ci è stato ritornato è di classe `GPIO.Pin`.

Per usare il pin dovremo richiamare il suo metodo `open()` e ricordarci di chiamare il metodo `close()` quando non lo utilizzeremo più.

```
>>> ledpin.open()
```

A questo punto possiamo cambiare lo stato del pin usando la funzione `set()`:

```
>>> ledpin.set(1)
>>> ledpin.set(0)
```

In alternativa possiamo utilizzare anche la proprietà `value`:

```
>>> ledpin.value=1
>>> ledpin.value=0
```

Dovreste vedere il LED accendersi quando impostate lo stato del pin a 1 e spegnersi quando lo impostate a 0. Se non succede, controllate i collegamenti e, in particolare, di aver montato il LED nella direzione giusta.

Prima di uscire dall'interprete è necessario chiamare la funzione `close()` del nostro pin.

```
>>> ledpin.close()
```

Questa funzione rilascia le risorse allocate dal modulo GPIO e riporta tutti i pin nella condizione di input, quella più sicura e che consuma meno energia.

Non richiamare la funzione di `close` potrebbe creare problemi se si eseguono altre applicazioni che usano i pin di GPIO.

Abbiamo detto che l'equivalente di "Ciao Mondo!" nel mondo dei dispositivi "fisici" è un piccolo programma che fa lampeggiare un LED.

Vediamo il suo sorgente:

```
#!/usr/bin/env python3

import sys

sys.path.append("/home/pi/quick2wire/quick2wire-python-api/")

import time
import quick2wire.gpio as GPIO

ledpin=GPIO.pi_header_1.pin(12,GPIO.Out)
ledpin.open()

try:

    stato=1

    while True:
        ledpin.set(stato)
        stato=0 if stato==1 else 1
        time.sleep(0.5)

finally:
    ledpin.close()
```

Anche questo piccolo esempio fornisce una serie di spunti interessanti.

Per prima cosa viene aggiornato l'oggetto `sys.path`, in modo che possa essere trovata la libreria `quick2wire`.

Oltre al modulo `quick2wire.gpio` viene importato anche il modulo `time`. Questo modulo contiene diverse funzioni collegate alla misura e alla gestione del tempo. Potete, come al solito, saperne di più usando la funzione `help()`.

In particolare nel nostro modulo viene usata la funzione `time.sleep()`, che consente di attendere per un tempo definito in secondi.

A questo punto il programma ricava l'oggetto per controllare il pin a cui è collegato il LED, usando il metodo `pin()` dell'oggetto `pi_header_1`, e chiama la sua funzione `open()`. Tutte le istruzioni successive sono all'interno di un blocco delimitato dalle keyword `try/finally`. Questo blocco è simile al blocco `try/except` visto nel capitolo precedente. La differenza è che le istruzioni specificate all'interno del blocco `finally` vengono eseguite sia in caso di eccezione sia se tutto termina correttamente. Usare questo accorgimento ci assicura che la funzione `close()` dell'oggetto `ledpin` venga richiamata sempre e comunque prima di uscire dal programma.

È possibile usare le keyword `except` e `finally` insieme; il blocco `finally` deve essere l'ultimo della lista e verrà eseguito anche se un'eccezione è stata intercettata da uno dei blocchi `except` precedenti.

A questo punto il codice entra in un ciclo infinito (la condizione del loop `while` è sempre vera) che cambia lo stato del LED fisico e di una variabile (`stato`) che lo mantiene all'interno del programma. Poi attende mezzo secondo e commuta di nuovo facendo così lampeggiare il LED.

Anche l'assegnazione della variabile `stato` è scritta in un modo un po' particolare rispetto a quanto visto fino a ora. L'istruzione `if` può essere utilizzata anche per eseguire un assegnamento condizionale. In questo caso la sintassi è diversa da quella vista nel capitolo precedente:

```
<variabile>=valore if <condizione> else <valore alternativo>
```

Questo consente di assegnare un valore se è verificata la condizione o un'altro se non lo è, tutto in una sola linea di codice. Come al solito Python privilegia la compattezza e la leggibilità.

L'istruzione:

```
stato=0 if stato==1 else 1
```

fa in modo che la variabile `stato` valga 0 se al momento dell'esecuzione dell'istruzione valeva 1 e viceversa. Un sistema molto comodo per cambiare lo stato a ogni ripetizione del ciclo! A questo punto potete lanciare il vostro script e vedere il LED lampeggiante. Siete ufficialmente diventati programmati di sistemi embedded!

Per terminarlo potete premere **Ctrl+C**, inviando un segnale di break. Questo genererà un messaggio di errore sulla console, ma assicurerà comunque l'esecuzione delle istruzioni di terminazione.

```
pi@pivalter ~ $ sudo ./lampeggia.py
^CTraceback (most recent call last):
  File "./lampeggia.py", line 16, in <module>
    time.sleep(0.5)
KeyboardInterrupt
```

In realtà possiamo scrivere il nostro programma in una forma ancora più compatta:

```
#!/usr/bin/env python3

import sys

sys.path.append("/home/pi/quick2wire/quick2wire-python-api/")

import time
import quick2wire.gpio as GPIO

ledpin=GPIO.pi_header_1.pin(12,GPIO.Out)
with ledpin:

    stato=1

    while True:
        ledpin.set(stato)
        stato=0 if stato==1 else 1
        time.sleep(0.5)
```

L'istruzione `with` consente di eseguire un blocco di codice garantendo che all'inizio del blocco venga richiamato un metodo di inizializzazione (`open` nel caso del nostro Pin) e all'uscita, normale o con eccezione, un metodo di chiusura (`close`). Questa sintassi è ancora più chiara di quella del blocco `try/ finally`, ma si può utilizzare solo con alcune tipologie di oggetti; nel caso la terminazione richieda operazioni più complesse sarà necessario tornare a utilizzare `try/finally`.

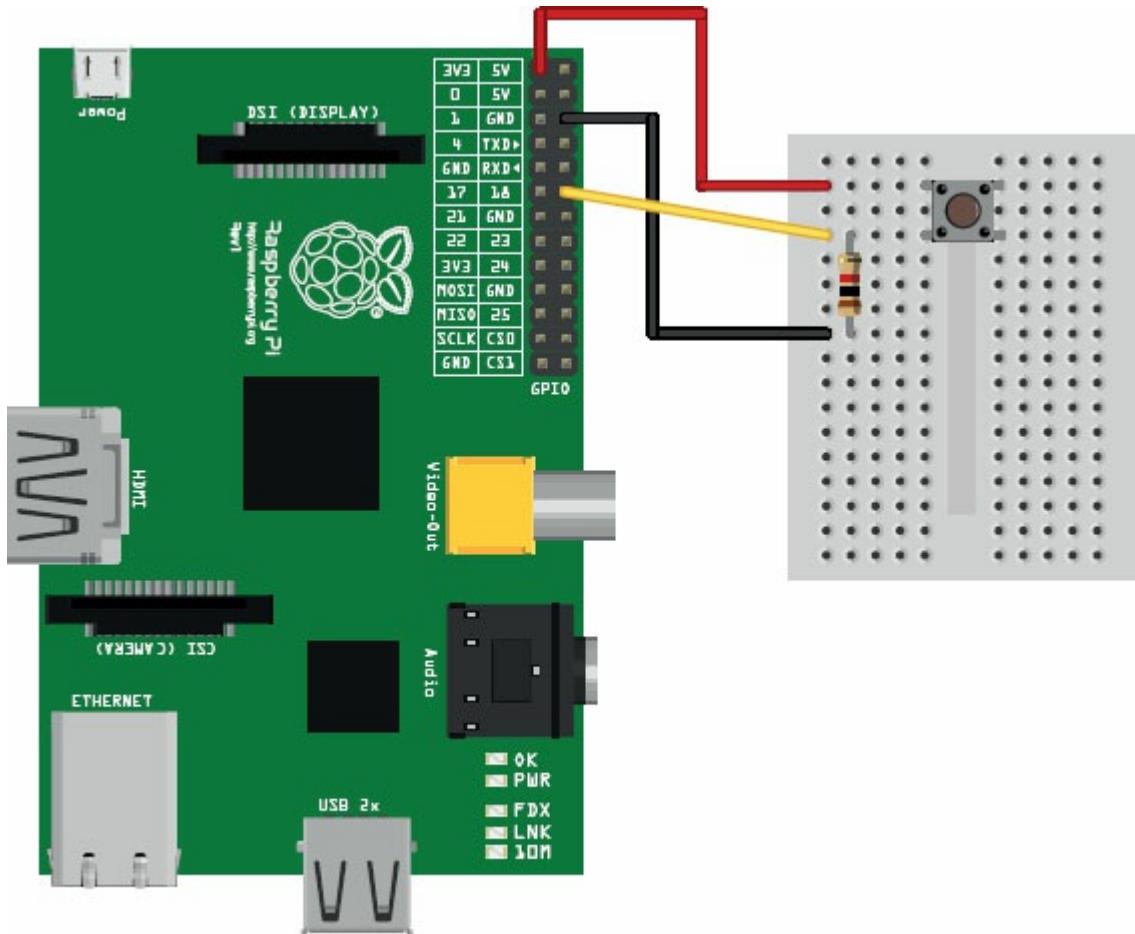
Muovere un'uscita digitale, seppure con una potenza limitata, consentirà di controllare molti dispositivi. Combinata con transistor e relè ci permetterà di pilotare anche potenze più elevate arrivando, per

esempio, alla possibilità di accendere e spegnere un elettrodomestico. Per ora restiamo nel mondo dei dispositivi a bassa potenza: meglio non rischiare fino a quando non avremo preso dimestichezza con il sistema!

Dopo aver completato l'esempio "canonico" possiamo ora esplorare le altre funzioni dei pin di GPIO. Possiamo utilizzare questi pin come ingressi e non più come uscite. Quando è usato come ingresso il nostro pin di GPIO legge la tensione tra il piedino e la massa e riporta lo stato on (1) quando questa è a 3,3 V e lo stato off (0) quando questa è a zero.

Il modo più semplice per cambiare lo stato di un ingresso è collegarci un pulsante. Anche in questo caso, oltre al pulsante, utilizzeremo una resistenza per limitare la corrente nel nostro circuito quando il pulsante viene chiuso.

La corrente scorre tra l'alimentazione (3,3 V nel nostro caso) e la massa. Il pulsante, quando non è premuto, interrompe il circuito, quindi non fa fluire corrente. Il nostro pin di GPIO non sentirà quindi nessuna tensione perché sarà collegato direttamente al pin GND. Quando il pulsante viene premuto, invece, il circuito si chiude e la corrente scorre. Questo farà sì che il nostro ingresso legga uno stato on.



Made with Fritzing.org

Figura 7.6 - Collegamento del pulsante.

Collegate pulsante e resistenza al vostro Raspberry Pi come mostrato in [Figura 7.6](#) e poi lanciate come super-user l'interprete Python.

Anche in questo caso dovremo importare quick2wire.GPIO con i passaggi visti in precedenza.

```
Type "help", "copyright", "credits" or "license" for more information.
>>> import sys
>>> sys.path.append("/home/pi/quick2wire/quick2wire-python-api/")
>>> import quick2wire.GPIO as GPIO
```

Dovremo anche configurare l'oggetto per controllare il pin, questa volta come input (`GPIO.In`):

```
>>> pulsante=GPIO.pi_header_1.pin(12,GPIO.In)
```

Dovremo poi chiamare la funzione `open()`:

```
>>> pulsante.Open()
```

La proprietà `value` ci indicherà lo stato del pulsante:

```
>>> pulsante.value
```

Tenendo premuto il tasto e rilanciando la lettura delle proprietà (un po' di esercizio fisico non fa male!), possiamo vedere come lo stato rifletterà il nuovo stato del pulsante:

```
>>> pulsante.value
1
```

Prima di terminare i nostri esperimenti dovremo chiamare la funzione `close()`:

```
>>> pulsante.close()
```

Leggere un ingresso digitale è utilissimo per poter interfacciare a Raspberry Pi tantissime tipologie di sensori. I sensori di movimento, i sensori magnetici ecc. generalmente cambiano stato da on a off per segnalare un evento e possono quindi essere collegati al nostro Raspberry Pi in maniera analoga a quanto fatto per il pulsante. Sempre tenendo conto della tensione a 3,3 V!

IQuadroCI

Zero e uno sono gli stati alla base di tutti i sistemi digitali che conosciamo.

Ma non sempre gli oggetti e i sensori che vogliamo collegare al nostro Raspberry Pi ritornano informazioni così semplici. Per scambiare informazioni più complesse, sempre in formato digitale, è possibile utilizzare i tre protocolli di comunicazione supportati dai pin del nostro connettore di GPIO: UART, SPI e I2C.

Si tratta di protocolli seriali. I protocolli seriali inviano su una linea dati le informazioni un bit alla volta, cambiando lo stato da zero a uno in sincrono con un temporizzatore (clock). Questo consente di far transitare grandi quantità di dati su poche linee fisiche. L'approccio opposto è quello delle interfacce parallele, dove si trasferiscono più bit ma servono molte linee dati. Molti dei protocolli più diffusi, anche in ambito PC, sono seriali. Pensiamo a USB (Universal Serial Bus), SATA (Serial ATA) ecc.

I protocolli seriali possono essere sincroni quando anche il clock viene trasmesso insieme ai dati (è il caso di SPI e I2C) o asincroni quando si stabilisce una frequenza di trasmissione che tutti devono rispettare e

che quindi non viene propagata sulla linea dati (è il caso delle seriali UART come le RS-232 usate fino a qualche anno fa anche sui PC).

Ci concentreremo in particolare sul protocollo I2C perché è molto semplice, supportato da tantissimi dispositivi e consente di collegare alle stesse due linee più device, rendendo possibili anche configurazioni complesse.

Il bus I2C (pronunciato I quadro C perché il 2 rappresenta I^2 , visto che la sigla è l'acronimo di Inter-Integrated Circuit) è stato inventato da Philips negli anni '80 e adottato da sempre più numerosi produttori hardware.

Il protocollo si basa su due sole linee fisiche e questo consente di ridurre al minimo il cablaggio necessario. Una linea viene usata per scambiare dati e l'altra per fornire il segnale di sincronizzazione (clock); normalmente le linee vengono chiamate SDA e SCK, e questi sono anche i nomi di due dei pin del nostro connettore GPIO.

Nella maggior parte dei casi il bus I2C è controllato da un master, che decide modi e tempi della comunicazione, e da una serie di slave che possono soltanto rispondere alle richieste del master.

Il nostro Raspberry Pi, forte del suo status di piccolo computer, fungerà da master e potrà controllare i dispositivi collegati.

Visto che più dispositivi possono essere collegati alle stesse linee, ogni dispositivo avrà un indirizzo univoco sul bus, da 1 a 127.

Avendo una sola linea per i dati è necessario che un solo dispositivo alla volta si attivi in trasmissione, mentre tutti gli altri saranno in ricezione. Il protocollo prevede che il master possa inviare richieste di lettura o scrittura in qualsiasi momento e che i dispositivi slave possano invece occupare il bus solo per rispondere alle richieste del master. Questo evita "collisioni", cioè accessi contemporanei al bus da parte di più dispositivi. Il master dovrà inviare le richieste di scrittura o di lettura dati, sempre precisando l'indirizzo del dispositivo con cui vuole comunicare. E i dispositivi parleranno solo quando interrogati dal master.

La maggior parte dei dispositivi I2C prevede una serie di registri interni (identificati da un indirizzo numerico) che possono essere letti e/o scritti dal master per configurare il dispositivo e per ricavarne lo stato. La specifica di questi registri è normalmente contenuta nel datasheet, il documento che ne descrive le funzionalità, che i produttori mettono normalmente a disposizione gratuitamente sul loro sito.

Visto che le linee utilizzate normalmente dal protocollo I2C sono all'interno del circuito stampato o, nel nostro caso, in collegamenti di pochi centimetri con frequenze di trasmissione relativamente basse, il sistema non prevede controlli di parità o altre tecniche per rilevare gli errori. Proprio per la sua semplicità I2C è stato implementato in tantissimi dispositivi, dai controllori per motori elettrici a diverse tipologie di sensori.

Alcuni dispositivi I2C, soprattutto quelli meno recenti, lavorano solamente a 5 V e non sono quindi compatibili con Raspberry Pi, se non utilizzando dei chip adattatori, ma non è difficile trovare dispositivi compatibili con la tensione a 3,3 V.

Il bus I2C non è normalmente abilitato in Raspbian. Per abilitarlo è necessario modificare alcuni file di configurazione.

Per prima cosa, dalla command line, modificare il file `/etc/modprobe.d/raspi-blacklist.conf`. Questo file contiene l'elenco dei moduli kernel disabilitati e uno di questi è il modulo I2C.

È necessario modificare il file come super-user:

```
pi@pivalter ~ $ sudo nano /etc/modprobe.d/raspi-blacklist.conf
```

e commentare con il simbolo `#` la riga relativa al modulo I2C:

```
# blacklist spi and i2c by default (many users don't need them)

blacklist spi-bcm2708
#blacklist i2c-bcm2708
```

Ora il nostro modulo potrà essere caricato.

Modificando il file `/etc/modules` possiamo fare in modo che venga caricato all'avvio. Anche questo file può essere modificato solo dall'utente root:

```
pi@pivalter ~ $ sudo nano /etc/modules
```

Questo è il file modificato:

```
# /etc/modules: kernel modules to load at boot time.
#
# This file contains the names of kernel modules that should be loaded
# at boot time, one per line. Lines beginning with "#" are ignored.
# Parameters can be specified after the module name.

snd-bcm2835

# carica i moduli I2C
i2c-bcm2708
```

Questo farà in modo che i moduli vengano caricati automaticamente al prossimo riavvio.

Dobbiamo anche aggiungere il nostro utente al gruppo degli utenti abilitati ad accedere alle funzioni i2c:

```
pi@pivalter ~ $ sudo addgroup pi i2c
```

A questo punto possiamo riavviare Raspberry Pi per fare in modo che le nostre modifiche alla configurazione diventino operative:

```
pi@pivalter ~ $ sudo reboot
```

Una volta che il sistema è ripartito, il bus I2C sarà attivo. Per poterlo provare e utilizzare con Python è necessario installare alcuni pacchetti aggiuntivi di Raspbian utilizzando il comando `apt-get`.

```
pi@pivalter ~ $ sudo apt-get install i2c-tools
Lettura elenco dei pacchetti... Fatto
Generazione albero delle dipendenze
Lettura informazioni sullo stato... Fatto
Pacchetti suggeriti:
  libi2c-dev
I seguenti pacchetti NUOVI saranno installati:
  i2c-tools
0 aggiornati, 2 installati, 0 da rimuovere e 13 non aggiornati.
È necessario scaricare 71,0 kB di archivi.
Dopo quest'operazione, verranno occupati 317 kB di spazio su disco.
Scaricamento di:1 http://mirrordirector.raspbian.org/raspbian/ wheezy/main i2c-to
armhf 3.1.0-2 [59,5 kB]
Scaricamento di:2 http://mirrordirector.raspbian.org/raspbian/ wheezy/main python
smbus armhf 3.1.0-2 [11,5 kB]
Recuperati 71,0 kB in 0s (73,8 kB/s)
Selezionato il pacchetto i2c-tools non precedentemente selezionato.
(Lettura del database... 62506 file e directory attualmente installati.)
Estrazione di i2c-tools (da .../i2c-tools_3.1.0-2_armhf.deb)...
Elaborazione dei trigger per man-db...
Configurazione di i2c-tools (3.1.0-2)...
```

Una volta installati questi pacchetti potremo testare il bus I2C mediante l'utility `i2cdetect`:

```
pi@pivalter ~ $ i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- --
10:          -- -- -- -- -- -- -- -- -- -- -- --
20:          -- -- -- -- -- -- -- -- -- -- -- --
30:          -- -- -- -- -- -- -- -- -- -- -- --
40:          -- -- -- -- -- -- -- -- -- -- --
50:          -- -- -- -- -- -- -- -- -- --
60:          -- -- -- -- -- -- -- -- -- -- -- --
```

Se avete un Raspberry Pi revisione 1.0 dovete usare il bus I2C0 invece di I2C1:

```
pi@pivalter ~ $ i2cdetect -y 0
```

Ovviamente non verranno rilevati dispositivi I2C, ma l'esecuzione senza errori dell'utility indica che i driver sono caricati correttamente.

A questo punto possiamo provare a collegare il nostro primo dispositivo I2C.

In questo esempio utilizzeremo un I/O expander, un chip che ci consente di pilotare ingressi e uscite digitali tramite il bus I2C. Aggiungere ingressi e/o uscite può essere utile per aumentare le capacità di espansione di Raspberry Pi. Inoltre questo chip è estremamente semplice e ci permetterà di dimostrare l'utilizzo delle funzioni I2C senza scrivere troppo codice.

Il chip in questione è il PCF8574AP di NXP. Sul sito del produttore (http://www.nxp.com/documents/data_sheet/PCF8574.pdf) è possibile trovare il datasheet, che ci sarà utile per capire meglio il funzionamento di questo componente.

Il PCF8574AP consente di aggiungere 8 ingressi/uscite digitali supplementari al nostro Raspberry Pi. Per prima cosa vediamo lo schema logico del nostro circuito (per Raspberry Pi è riportato il solo connettore GPIO).

Lo schema può sembrare complesso, ma leggendolo con calma sarà poi facile "tradurlo" in collegamenti con la nostra basetta sperimentale.

Il PCF8574AP ha 16 pin. Il pin 16 deve essere collegato all'alimentazione (3,3 V) e il pin 8 a massa (GND).

Il BUS I2C è gestito dai pin SDA e SCL, che andranno quindi collegati ai corrispondenti pin del connettore GPIO.

Visto che è possibile collegare più dispositivi di questo tipo allo stesso bus I2C, per avere 16, 24, 32 o più pin di I/O, è possibile configurare l'indirizzo del dispositivo mettendo a massa (GND) o a 3,3 V i pin A0, A1 e A2. L'indirizzo (a 7 bit) è formato da una parte fissa (i primi quattro bit) e una determinata dai pin A0-A2:

Parte fissa				Parte variabile		
0	1	1	1	A2	A1	A0

Mettendo tutti e tre i pin a GND (quindi al livello logico 0), come nel nostro esempio, l'indirizzo configurato sarà 38h (esadecimale; per gli indirizzi di componenti e registri useremo questa notazione, che è quella comunemente usata anche nei datasheet e negli esempi).

A questo punto restano un pin di Interrupt, che non utilizzeremo, e 8 pin di ingresso/uscita P0-P7.

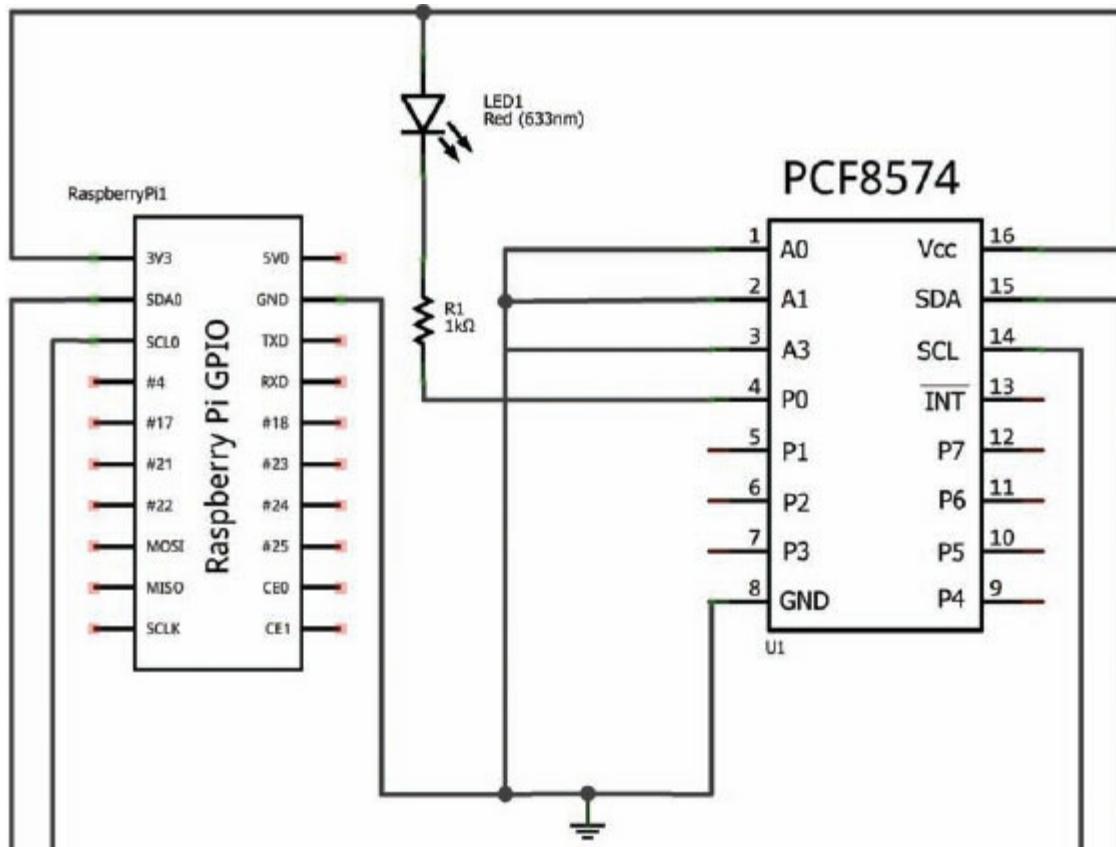


Figura 7.7 - Schema logico di collegamento per il PCF8574AP.

Al pin P0 collegheremo un LED, usandolo però in modo diverso rispetto a quanto fatto nell'esempio precedente. Il LED sarà infatti collegato all'alimentazione e, attraverso una resistenza, al pin.

Il PCF8574 non è in grado di erogare molta corrente e il LED collegato come nell'esempio precedente emetterebbe una luce molto fioca. Collegato in questo modo il LED si accenderà quando l'uscita sarà al livello logico 0 perché ci sarà tensione tra l'alimentazione e il pin. Quando invece il pin è a 1, il LED resterà spento perché la tensione ai suoi capi sarà nulla (quella di alimentazione e quella del pin saranno equivalenti).

Il cablaggio, sfruttando una basetta sperimentale con piste per alimentazione e massa, è più complesso di quelli visti in precedenza, ma con un po' di pazienza e l'aiuto di Fritzing, è riproducibile senza troppi

problemi.

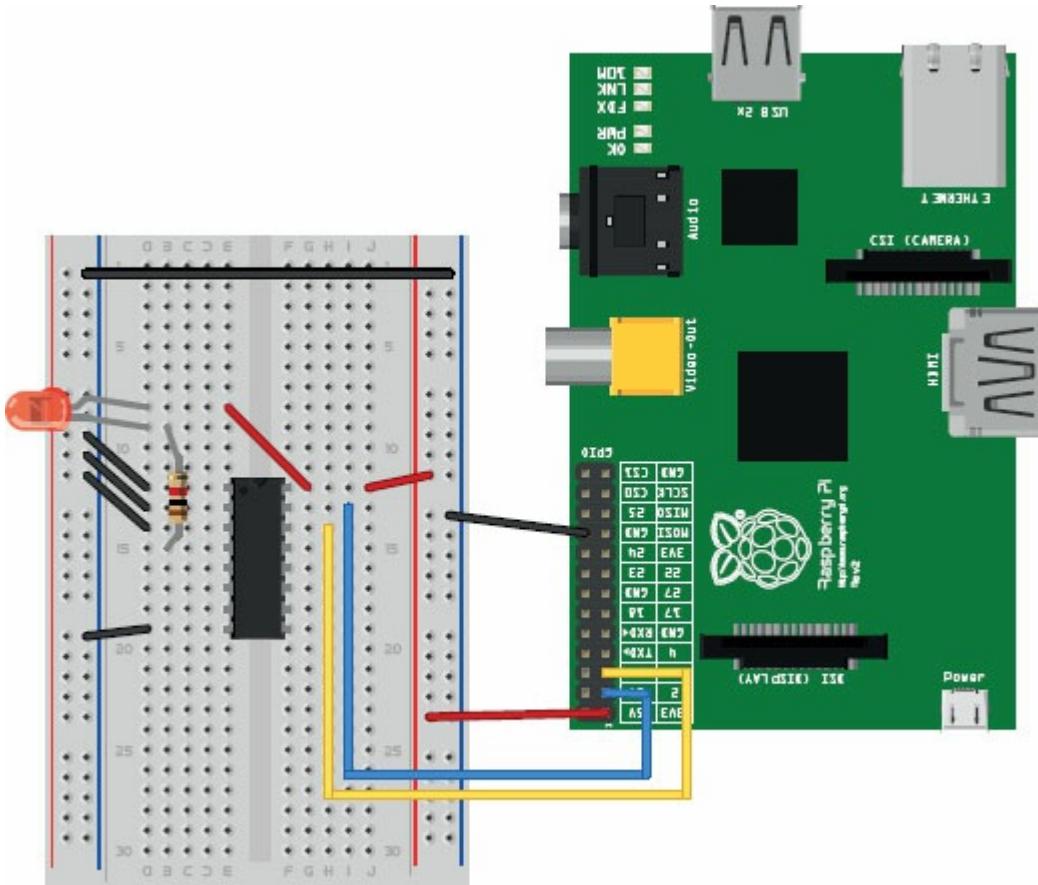


Figura 7.8 - Collegamento per il PCE8574P.

Per riconoscere il pin 1 del chip dovete osservarlo tenendo i pin in giù. Uno dei due lati corti ha un'incisione semi-circolare al centro. Tenendo il chip con questo lato verso l'alto i pin saranno numerati in senso antiorario partendo da quello in alto a sinistra.

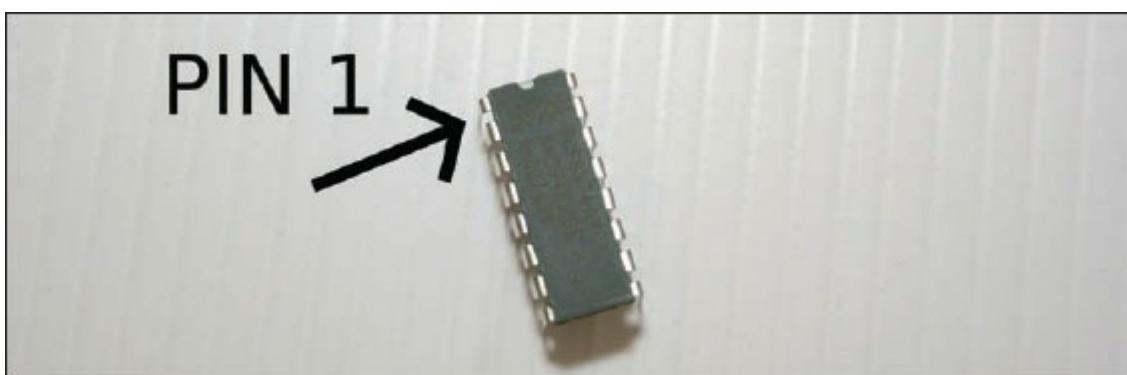


Figura 7.9 - Il PCF8574 con evidenziato il pin 1.

Una volta fatti e verificati i collegamenti possiamo caricare Python e provare a comunicare con il nostro chip.

Anche stavolta, dopo aver lanciato l'interprete, dovremo configurare il

percorso delle librerie e importare un modulo di `quick2wire`. In questo caso utilizzeremo il modulo `I2C`:

```
>>> import sys
>>> sys.path.append("/home/pi/quick2wire/quick2wire-python-api/")
>>> import quick2wire.i2c as I2C
```

Il modulo consente di comunicare con i dispositivi collegati al bus.

Il PCF8574AP è un dispositivo molto semplice. È possibile cambiare lo stato dei suoi pin semplicemente scrivendo un byte al suo indirizzo I2C. Il bit meno significativo corrisponderà al pin P0, il successivo a P1 ecc. allo stesso modo si potrà leggere lo stato degli ingressi semplicemente leggendo un byte. Non c'è nemmeno bisogno di configurare quali pin sono utilizzati come ingressi e quali come uscite.

Vediamo un semplice programma che ci consente di far lampeggiare il LED collegato al nostro chip esterno:

```
#!/usr/bin/env python3

import sys
sys.path.append("/home/pi/quick2wire/quick2wire-python-api/")

import quick2wire.i2c as I2C
import time

i2cbus=I2C.I2CMaster()

with i2cbus:

    stato=1

    while True:
        i2cbus.transaction(I2C.writing_bytes(0x38,stato,stato))
        stato=0 if stato==1 else 1
        time.sleep(0.5)
```

Le prime righe corrispondono a quelle dell'esempio precedente ma, al posto di importare il modulo `gpio` viene importato il modulo `I2C`, anche in questo caso adoperando la keyword `as` per usare un prefisso più semplice per gli oggetti.

L'oggetto ritornato dalla funzione `I2C.I2CMaster()` ci permette di comunicare sul bus I2C e viene gestito tramite un blocco `with` come nell'esempio precedente, per essere sicuri che venga in ogni caso richiamata la funzione `close()`.

La struttura del loop con la commutazione della variabile di stato e il ritardo di mezzo secondo sono immutati, ma in questo caso, per

accendere o spegnere il LED, dovremo inviare dei dati sul bus I2C. Questi dati verranno inviati all'interno di una transazione.

Per transazione si intende una sequenza di operazioni che non può essere interrotta da altre operazioni richieste da programmi diversi. Questo è necessario perché spesso le operazioni di scrittura e lettura devono avere un ordine preciso e non possono essere inframmezzate con altre, anche rivolte a device diversi.

L'oggetto `I2CMaster` consente di racchiudere più operazioni in una transazione passandole alla funzione `transaction()`.

Nel nostro caso l'operazione è molto semplice. Viene scritto lo stato delle uscite, usando l'indirizzo del nostro chip (0x38 in notazione esadecimale con la sintassi di Python) e scrivendo il valore `1` per attivare l'uscita P0 o `0` per spegnerla. Lo stato delle 8 uscite cambia quando al chip viene inviato un nuovo dato in scrittura; questo è il motivo per cui la alla funzione `I2C.writing_bytes()`, oltre all'indirizzo, viene passato due volte il valore della variabile `stato`, in modo che alla fine della prima scrittura le uscite cambino effettivamente il loro stato.

Ora che abbiamo visto come gestire un oggetto molto semplice collegato al bus `I2C` possiamo provare a complicare un po' le cose e controllare oggetti più complessi, ma anche più utili. Questo ci servirà per compensare una delle lacune di Raspberry Pi: l'assenza di ingressi analogici.

Leggere dati analogici

Oggi sempre più informazioni viaggiano in formato digitale. La musica è passata dai vinili ai CD e ai file mp3, la fotografia dalle pellicole alle memory card, il video dal VHS ai DVD e infine anche la televisione ha abbandonato il segnale analogico per i nuovi standard digitali satellitari e terrestri.

Il mondo reale, però, rimane ostinatamente analogico. Per misurare e utilizzare nei nostri programmi grandezze prese dal mondo reale è necessario convertirle in un formato digitale tramite un hardware apposito.

Questo hardware si chiama convertitore analogico-digitale (ADC, abbreviazione di Analog to Digital Converter) e, per nostra fortuna, è un dispositivo sempre più comune ed economico. Esistono molti convertitori ADC controllabili dal bus I2C in commercio. Purtroppo i

componenti sono sempre più miniaturizzati e questo li rende inutilizzabili per progetti amatoriali, a meno che non abbiate la manualità e la pazienza di un mastro orafo.

Per fortuna diverse aziende come Sparkfun (www.sparkfun.com), Adafruit (www.adafruit.com), Seeedstudio (www.seeedstudio.com) e altre hanno realizzato piccoli circuiti stampati che riportano i segnali di questi minuscoli chip in un formato gestibile anche sulle basette millefori. Le schede di questo tipo vengono comunemente chiamate breakout board e proprio con una di queste, prodotta da Adafruit, collegheremo il nostro Raspberry Pi a un semplice sensore analogico.

Il chip è prodotto da Texas Instruments e si chiama ADS1015, lo stesso nome della breakout board di Adafruit. Potete, come al solito, trovare il datasheet sul sito del produttore, all'indirizzo <http://www.adafruit.com/datasheets/ads1015.pdf>.

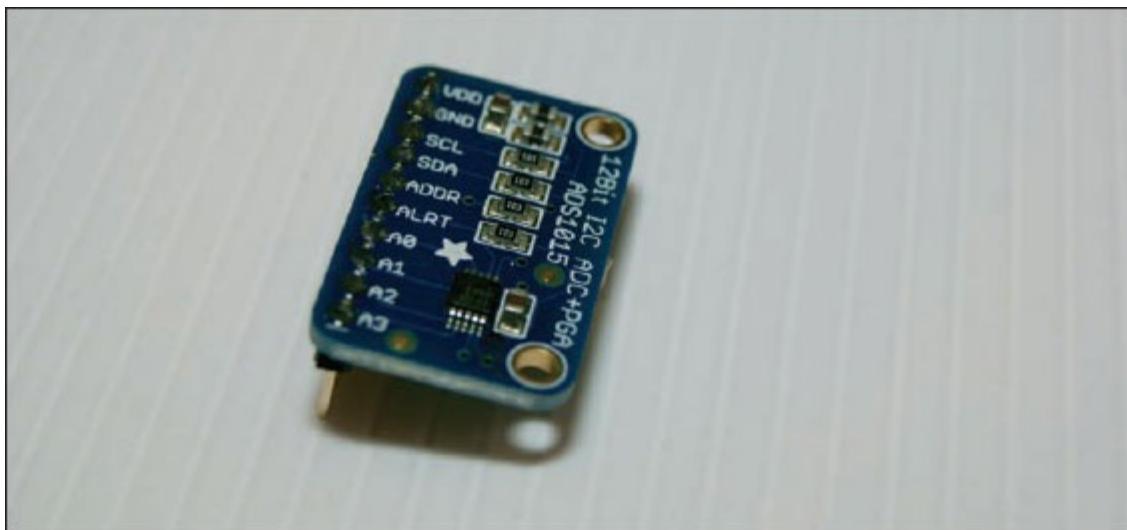


Figura 7.10 - Breakout board ADS1015.

Il chip è un ADC a 12 bit e 4 ingressi. È quindi in grado di misurare, anche se non contemporaneamente, quattro grandezze analogiche con 4096 valori possibili. In realtà un bit viene utilizzato per il segno e, se si usano tutti e 4 gli ingressi per misurare la tensione tra l'ingresso e la massa, non è possibile misurare valori negativi. In questa configurazione, quindi, i possibili valori sono "solamente" 2048. In alternativa è possibile usare il chip in modalità differenziale, misurando la differenza tra un ingresso e un altro. In questo modo si possono leggere anche valori negativi, ma il numero di grandezze misurabili si dimezza riducendosi a 2. Per la maggior parte delle applicazioni, soprattutto amatoriali, questa è una precisione più che sufficiente.

Per provare il nostro ADS1015 avremo bisogno di un sensore analogico. Uno dei sensori più semplici è la fotoresistenza.

Questo componente ha una resistenza al passaggio della corrente che è inversamente proporzionale alla quantità di luce che lo colpisce. Misurando la tensione ai suoi capi e mettendo in serie una resistenza fissa per limitare la corrente è possibile calcolare la quantità di luce che colpisce il sensore. La tensione potrà essere tradotta in un dato numerico dal nostro ADC converter.

Anche in questo caso partiamo dallo schema logico.

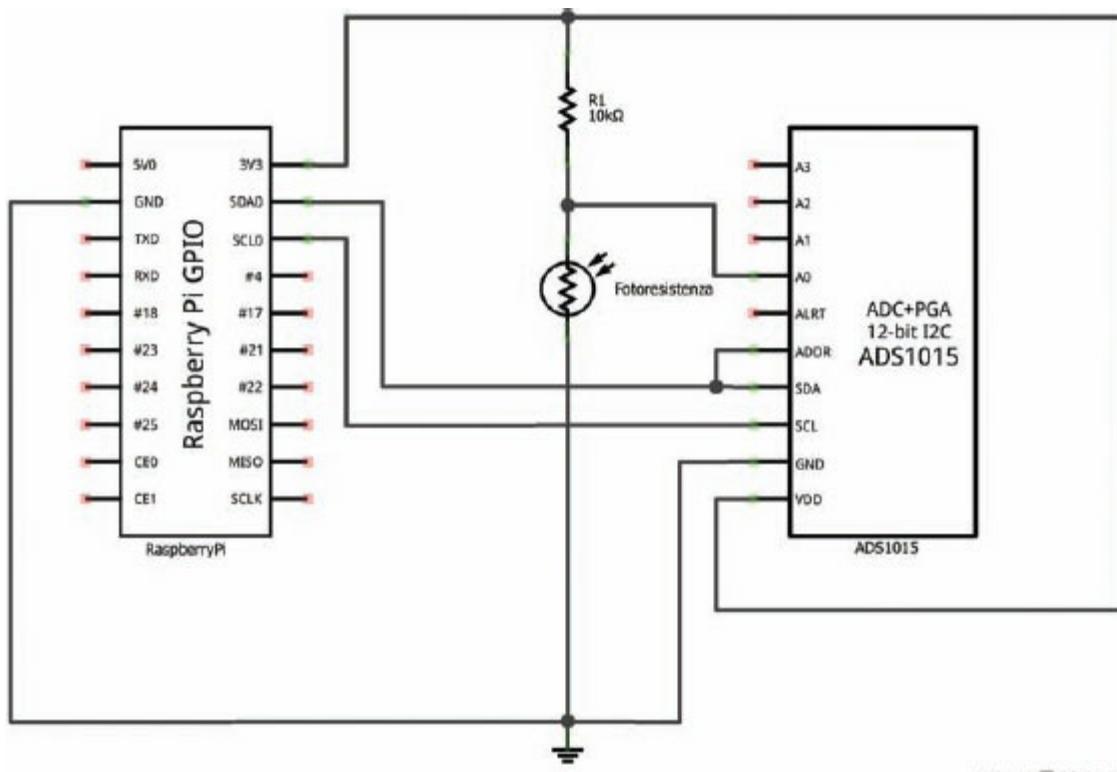


Figura 7.11 - Schema logico dell'ADS1015.

Come negli esempi precedenti, colleghiamo le alimentazioni e i segnali del bus I2C.

Anche l'ADS1015 ha un pin che consente di configurare l'indirizzo I2C, fornendo fino a 4 possibili indirizzi (collegandolo a 3,3 V, GND, SDA o SCL). Collegandolo al pin SDA avremo l'indirizzo 4Ah.

La fotoresistenza viene collegata all'alimentazione e poi, passando per una seconda resistenza, alla massa. Il valore della seconda resistenza dovrà essere calcolato in funzione dei valori che assume la fotoresistenza al buio e al massimo della luminosità; nel nostro caso 10 K Ω corrisponde più o meno al valore minimo di resistenza che potrà raggiungere.

Il pin A0, che verrà usato per misurare la tensione, sarà collegato tra le due resistenze, in modo da poter registrare i cambiamenti di tensione.

Il cablaggio è abbastanza complicato per via dei segnali che si incrociano, ma con un po' di pazienza e di tentativi si risolverà senza problemi.

Una volta completato il nostro hardware e collegato il tutto al Raspberry Pi, possiamo iniziare a sperimentare con il software.

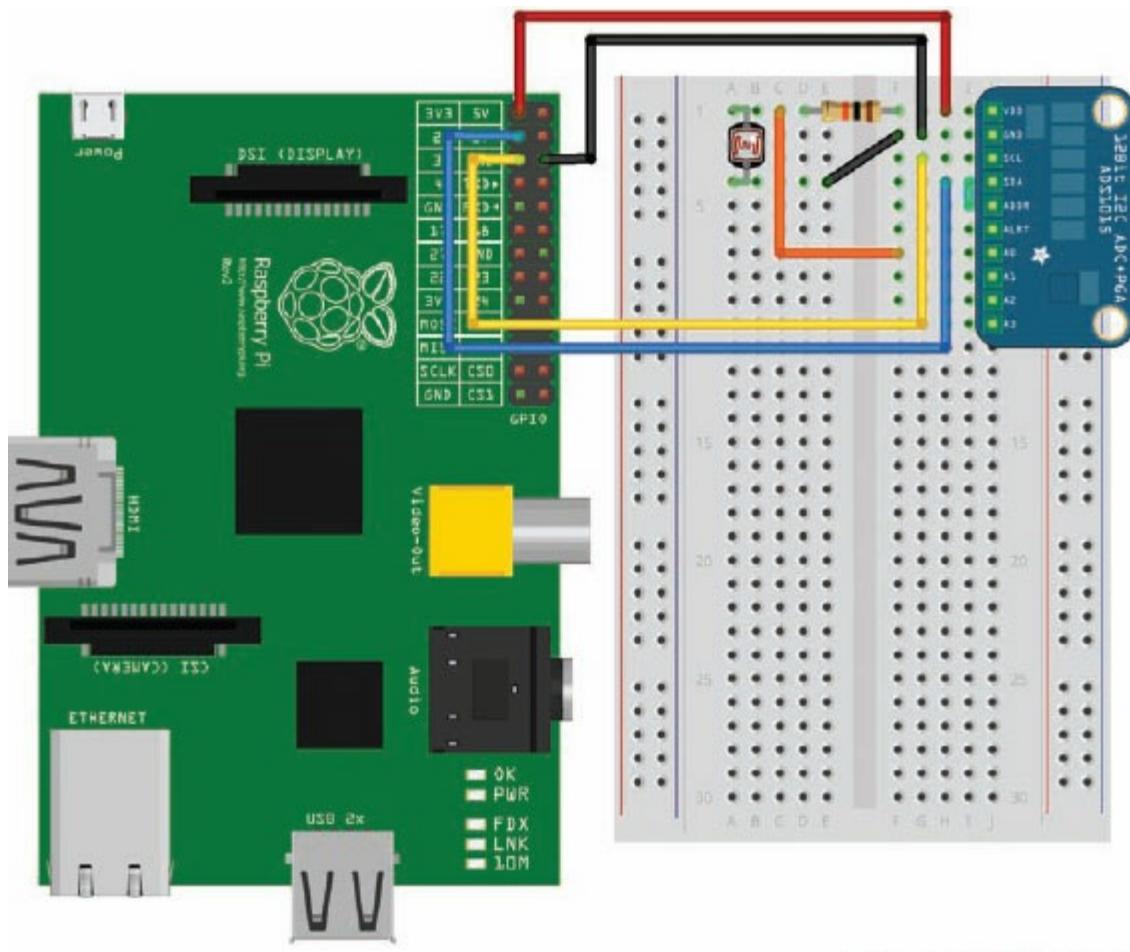


Figura 7.12 - Collegare l'ADS1015

L'ADS1015 ha quattro registri, ma per il nostro esempio ne utilizzeremo soltanto due:

- il registro di conversione (indirizzo 0), dove avremo il valore letto dall'ADC converter. Il registro è a 16 bit, ma i dati soltanto a 12. I 4 bit meno significativi di questo registro non sono quindi utilizzati (dobbiamo scartarli dividendo il valore letto per 16);
 - il registro di controllo (indirizzo 1), che utilizzeremo per configurare il nostro convertitore e per verificare il risultato delle operazioni di conversione.

I registri sono a 16 bit e nel registro di controllo ogni singolo bit o gruppo di bit ha una funzione.

I bit sono descritti nella tabella qui in seguito:

Nome	Bit	Significato
OS	15	Quando viene messo a 1 questo bit attiva il chip ed esegue una conversione. Normalmente il chip è in una condizione a basso consumo, per non sprecare energia
MUX	14:12	Configura tra quali ingressi deve essere effettuata la misura: 000 - A0 e A1 001 - A0 e A3 010 - A1 e A3 011 - A2 e A3 100 - A0 e GND 101 - A1 e GND 110 - A2 e GND 111 - A3 e GND Nel nostro caso utilizzeremo il valore 100 per misurare la tensione tra il pin A0 e la massa
PGA	11:9	Il segnale in ingresso può essere amplificato per misurare in modo preciso anche segnali molto piccoli. Nel nostro caso la tensione massima è quella di alimentazione, 3,3 V. 000 - misura segnali fino a 6,144 V 001 - misura segnali fino a 4,096 V 010 - misura segnali fino a 2,048 V 011 - misura segnali fino a 1,024 V 100 - misura segnali fino a 0,512 V 101,110,111 - misura segnali fino a 0,256 V
MODE	8	Configura la modalità di funzionamento. Se impostato a 1 l'ADC effettuerà una singola misura (la modalità che utilizzeremo nel nostro esempio); se impostato a 0 il chip convertirà continuamente i dati in ingresso
DR	7:5	Velocità di campionamento. È possibile configurarla nel caso di lettura continua; nel nostro caso (lettura singola) verrà lasciato il valore di default 100 (1600 letture al secondo)
		Questi bit configurano il comparatore. Quando il

COMP_MODE	4	comparatore è attivo il segnale ALRT viene attivato se il dato letto è all'interno o al di fuori di due soglie di comparazione configurate tramite registri.
COMP_POL	3	
COMP_LAT	2	Non utilizzeremo questa funzionalità nel nostro esempio
COMP_QUE	1:0	Questi due bit abilitano il comparatore in diverse modalità oppure lo disabilitano. Nel nostro caso saranno configurati a 11 per disabilitare il comparatore

Vediamo ora il codice del semplice programma che leggerà dall'ADC e stamperà continuamente sulla console lo stato dell'ingresso A0.

```
#!/usr/bin/env python3

# aggiungiamo la libreria quick2wire al path di sistema
import sys
sys.path.append("/home/pi/quick2wire/quick2wire-python-api/")

# importiamo le librerie necessarie
import quick2wire.i2c as I2C
import time

# valori del campo MUX a seconda dell'ingresso utilizzato
mux=[ 0b0100000000000000, 0b0101000000000000, 0b0110000000000000, 0b0111000000000000]

# la funzione legge il valore corrente dell'ingresso richiesto
# sul chip AD1015 all'indirizzo specificato
def leggiADS1015(indirizzo, ingresso):

    if (ingresso<0) or (ingresso>3):
        raise ValueError

    i2cbus=I2C.I2CMaster()

    with i2cbus:

        # configura il chip per eseguire una lettura dall'ingresso
        controllo=0b1000001110000011
        # OS - 15 - 1 attiva conversione
        # MUX - 14:12 - verranno scritti nell'istruzione successiva
        # PGA- 11:9 - 001 amplificazione segnale (max 4.096V)
        # MODE - 8 - 1 modalità singola conversione
        # DR - 7:5 - 000 non considerato
        # COMP_MODE - 4 - 0 non usato
        # COMP_POL - 3 - 0 non usato
        # COMP_LAT - 2 - 0 non usato
        # COMP_QUE - 1:0 - 11 disabilita comparatore

        # il valore di mux (bit 14-12) cambia a seconda dell'ingresso
        # selezionato e viene preso dalla lista mux
        controllo|=mux[ingresso]
```

```

# ora possiamo scrivere il registro di controllo per avviare
# la conversione
i2cbus.transaction(
    I2C.writing_bytes(indirizzo,
        0x01,
        (controllo>>8) & 0xFF,controllo & 0xFF)
)

# attendiamo 100ms per lasciar completare la conversione
time.sleep(.1)

# una volta completata la conversione è possibile leggere
# il valore
bytes=i2cbus.transaction(
    I2C.writing_bytes(indirizzo,0x00),
    I2C.reading(indirizzo,2))

valore=(bytes[0][0]<<8)+bytes[0][1]

# il valore deve essere shiftato di 4 bit a destra
# perché solo 12bit sono significativi
return valore>>4

while True:
    val=leggiADS1015(0x4A,0)
    print(str(val))

```

Per prima cosa notiamo che ci sono dei commenti all'interno del codice. Commentare il codice è importante, soprattutto quando le cose iniziano a diventare complicate! Per aggiungere un commento in Python basta inserire il carattere `#` (diesis o cancelletto); tutto quanto scritto dal carattere alla fine della riga verrà considerato un commento e ignorato ai fini della sintassi del linguaggio.

Il nostro piccolo programma importa i moduli necessari con le stesse modalità già viste per l'esempio precedente.

Poi viene dichiarata una funzione `leggiADS1015()` che ritorna il valore corrente di un ingresso passatole come parametro. Come parametro viene passato anche l'indirizzo I2C del chip da interrogare.

La funzione per prima cosa controlla che il parametro `ingresso` sia compreso tra `0` e `3` (gli indici degli ingressi del nostro chip). Poi inizializza l'oggetto `i2cbus` in maniera analoga a quella utilizzata nell'esempio precedente.

Il primo passaggio necessario per leggere un valore analogico è configurare l'ADC scrivendo nel registro di controllo.

L'assegnazione del registro:

```
controllo=0b1000001110000011
```

è scritta usando un numero in notazione binaria.

Questa notazione è molto utile quando, come nel nostro caso, è necessario vedere il valore di ogni singolo bit.

Una parte del valore del registro di controllo dipende dall'ingresso selezionato. I quattro possibili valori, sempre in formato binario, sono memorizzati nella lista `mux` e il valore giusto viene combinato con il resto dei bit della variabile `controllo` usando l'operatore or binario (`|`). Anche questo operatore, come quelli matematici, può essere anteposto al simbolo uguale per fare l'or tra una variabile e un valore, mettendo il risultato nella variabile stessa:

```
controllo |= mux[ingresso]
```

A questo punto il programma è pronto a inviare il suo primo comando I2C. Scriverà all'indirizzo passato come parametro prima l'indice del registro di configurazione (`01h`), poi il valore della variabile `controllo`. Il valore, però, deve essere "spezzato" nei singoli byte. Il primo byte deve contenere gli 8 bit più significativi. Per ottenerli è sufficiente utilizzare l'operatore di shift a destra (`>>`) e far scorrere il nostro valore di 8 posizioni, eliminando gli otto bit meno significativi. Per ottenere la parte bassa viene utilizzato l'operatore and binario (`&`) con il valore massimo di un byte (`FFh`); questa operazione rimuove gli otto bit più alti del valore.

Una volta scritto il registro di controllo, il nostro ADC effettuerà la conversione. La funzione `time.sleep()` gli darà il tempo di completare l'operazione.

La transazione I2C successiva legge il risultato. Per leggere un dato sul bus I2C è per prima cosa necessario scrivere l'indirizzo del registro da leggere. Per questo la prima operazione della transazione è una scrittura dell'indirizzo `00h`.

La seconda operazione legge due byte che saranno ritornati dalla funzione `transaction()`. La funzione ritorna una lista di liste di byte (riprendete fiato per un attimo!). Ogni elemento della lista costituisce la risposta a un'operazione di lettura. Nel nostro caso c'è una sola chiamata alla funzione `i2c.reading()` all'interno della transazione, quindi la lista avrà un solo elemento.

Questo elemento sarà la lista dei byte letti, cioè due.

Anche in questo caso è necessario usare le operazioni di shift (questa volta a sinistra!) e di or binario (`|`) per ricombinare i due byte in un valore a 16 bit. Questo valore verrà infine fatto scorrere a destra di

quattro posizioni per eliminare i bit non significativi e verrà ritornato. Le ultime istruzioni del programma, infine, richiamano la funzione di lettura in un loop infinito.

Se lanciate l'esempio vedrete stampare rapidamente dei valori sulla console. Coprendo la fotoresistenza con un dito i valori scenderanno, mentre illuminandola con una lampada o con la luce dello schermo del vostro cellulare il valore salirà. Questo ci consentirà di avere una misura della luce che colpisce il nostro sensore. Ed è un esempio di come Raspberry Pi può interagire con il mondo "analogico" che lo circonda.

Visto che la funzione che abbiamo appena scritto può tornare utile in altri esempi, possiamo far diventare il nostro piccolo programma un modulo.

Per farlo ci basta togliere le istruzioni al di fuori della funzione, cioè:

```
while True:  
    val=leggiADS1015(0x4A, 0)  
    print(str(val))
```

Se salviamo il modulo con il nome `ADS1015.py`, potremo poi importarlo in un nuovo, e più semplice, programma di test che a questo punto importerà il modulo e implementerà solo il loop di lettura:

```
#! /usr/bin/env python3  
import ADS1015  
  
while True:  
    val=ADS1015.leggiADS1015(0x4A, 0)  
    print(str(val))
```

Aggregare le nostre funzioni in moduli è un buon modo di riutilizzare il codice.

Un bug sistemato nel modulo sarà "magicamente" sistemato in tutti i programmi che ne fanno uso. Se invece copiassimo il codice da un programma all'altro dovremmo poi andare a correggere lo stesso bug in tutte le copie.

Fare un po' di lavoro per organizzare bene il vostro codice vi farà risparmiare tantissimo lavoro noioso in seguito. Non siate pigri!

La telecamera

I "sensi" di Raspberry Pi possono essere estesi con tantissimi sensori. Un esempio interessante è il camera module.

Verso la metà del 2013 è stato reso disponibile un modulo fotocamera

per Raspberry Pi. Questo modulo integra un sensore a 5 Megapixel e si connette direttamente al processore con un connettore dedicato (che è tra il connettore HDMI e il connettore Ethernet).

In precedenza l'unico sistema per acquisire immagini con Raspberry Pi era costituito dalle webcam USB, che però non garantivano un adeguato frame rate e una risoluzione elevata.

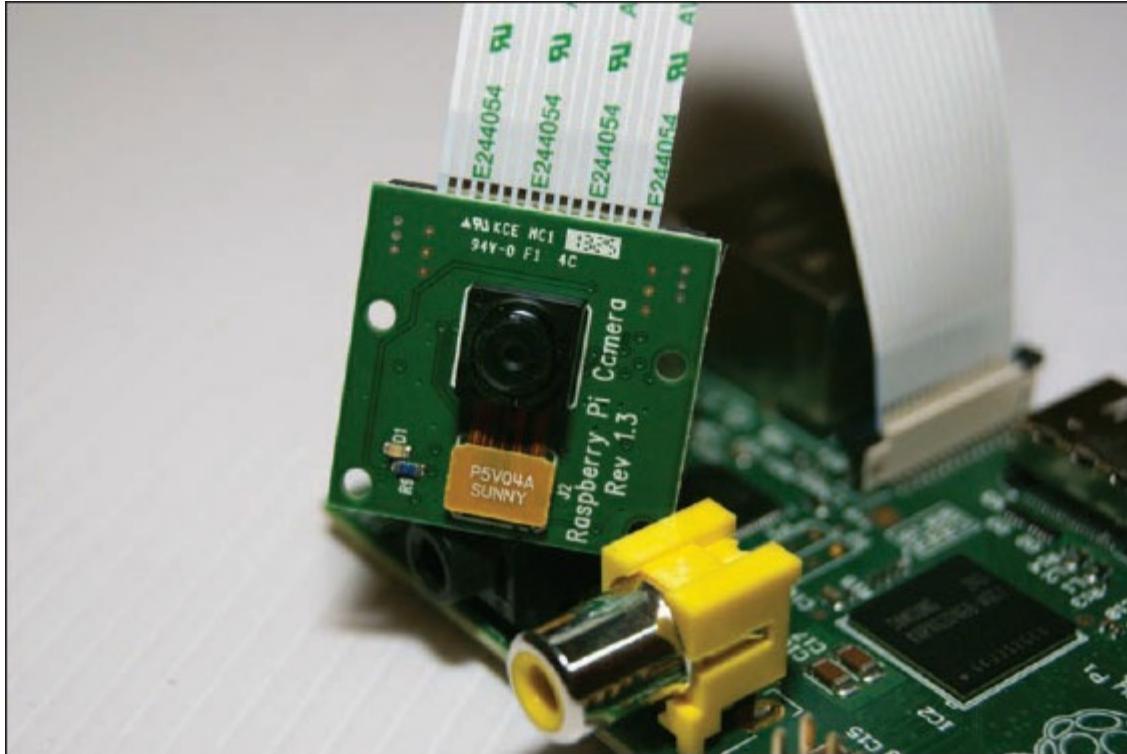


Figura 7.13 - Il modulo telecamera connesso a Raspberry Pi.

Anche il modulo fotocamera è acquistabile da molti dei distributori che vendono Raspberry Pi e ha un costo decisamente abbordabile.

Il suo successo è stato rapidissimo, tanto che fino a settembre 2013 era difficile acquistarne uno perché i pochi pezzi in possesso dei distributori finivano regolarmente esauriti dopo pochi giorni dall'annuncio di disponibilità.

Dopo aver collegato il modulo fotocamera (con Raspberry Pi spento!) è necessario abilitare il supporto per la camera usando `raspi-config`.

È possibile lanciare l'utility digitando:

```
pi@pivalter ~ $ sudo raspi-config
```

Il supporto per la fotocamera si attiva selezionando l'opzione **Enable camera** del menu di `raspi-config`; dopo l'abilitazione, è necessario un reboot prima di poter utilizzare il componente.

Una volta che il sistema si è riavviato è possibile attivare la fotocamera

e vedere in tempo reale il video catturato sul nostro monitor lanciando l'utility `raspivid`:

```
pi@pivalter ~ $ raspivid -d
```

Dopo alcuni secondi il sistema tornerà al desktop o alla console.
Usando l'opzione `--help` `raspivid` ci mostra tutte le sue potenzialità:

```
pi@pivalter ~ $ raspivid --help
Display camera output to display, and optionally saves an H264 capture at request
```

usage: `raspivid` [options]

Image parameter command

<code>-?, --help</code>	: This help information
<code>-w, --width</code>	: Set image width <size>. Default 1920
<code>-h, --height</code>	: Set image height <size>. Default 1080
<code>-b, --bitrate</code>	: Set bitrate. Use bits per second (e.g. 10MBits/s would be -b 10000000)
<code>-o, --output</code>	: Output filename <filename> (to write to stdout, use '-o -')
<code>-v, --verbose</code>	: Output verbose information during run
<code>-t, --timeout</code>	: Time (in ms) to capture for. If not specified, set to 5s. Ze
<code>-d, --demo</code>	: Run a demo mode (cycle through range of camera options, no c
<code>-fps, --framerate</code>	: Specify the frames per second to record
<code>-e, --penc</code>	: Display preview image *after* encoding (shows compression ar
<code>-g, --intra</code>	: Specify the intra refresh period (key frame rate/GoP size)

Preview parameter commands

<code>-p, --preview</code>	: Preview window settings <'x,y,w,h'>
<code>-f, --fullscreen</code>	: Fullscreen preview mode
<code>-op, --opacity</code>	: Preview window opacity (0-255)
<code>-n, --nopreview</code>	: Do not display a preview window

Image parameter commands

<code>-sh, --sharpness</code>	: Set image sharpness (-100 to 100)
<code>-co, --contrast</code>	: Set image contrast (-100 to 100)
<code>-br, --brightness</code>	: Set image brightness (0 to 100)
<code>-sa, --saturation</code>	: Set image saturation (-100 to 100)
<code>-ISO, --ISO</code>	: Set capture ISO
<code>-vs, --vstab</code>	: Turn on video stablisation
<code>-ev, --ev</code>	: Set EV compensation
<code>-ex, --exposure</code>	: Set exposure mode (see Notes)
<code>-awb, --awb</code>	: Set AWB mode (see Notes)
<code>-ifx, --imxfx</code>	: Set image effect (see Notes)
<code>-cfx, --colfx</code>	: Set colour effect (U:V)
<code>-mm, --metering</code>	: Set metering mode (see Notes)
<code>-rot, --rotation</code>	: Set image rotation (0-359)
<code>-hf, --hflip</code>	: Set horizontal flip
<code>-vf, --vflip</code>	: Set vertical flip

Notes

```
Exposure mode options :  
off,auto,night,nightpreview,backlight,spotlight,sports,snow,beach,verylong,fixedf  
ntishake,fireworks  
AWB mode options :  
off,auto,sun,cloud,shade,tungsten,fluorescent,incandescent,flash,horizontal  
  
Image Effect mode options :  
none,negative,solarise,sketch,denoise,emboss,oilpaint,hatch,gpen,pastel,watercolor  
film,blur,saturation,colourswap,washedout,posterise,colourpoint,colourbalance,car  
  
Metering Mode options :  
average,spot,backlit,matrix
```

L'utility permette, per esempio, di impostare diversi parametri della fotocamera come la risoluzione, il bilanciamento del bianco, la modalità di misura dell'esposizione. Consente anche di impostare effetti grafici che verranno generati in tempo reale.

Per esempio, lanciando:

```
pi@pivalter ~ $ raspivid -p 0,0,640,480 -ifx oilpaint -t 0
```

potrete vedervi all'interno di una finestra posizionata in alto a destra dello schermo (configurata dal parametro `p`) raffigurati come vi avrebbe ritratto un maestro fiammingo con i suoi colori a olio (opzione `ifx`), ma animati in tempo reale!

La visualizzazione in questo caso non termina dopo 5 secondi perché il parametro `t` ha impostato un timeout infinito. Dovremo terminare `raspivid` premendo Ctrl+C sulla nostra console.

Divertitevi a sperimentare con i diversi parametri e a scoprire tutte le funzioni di `raspivid`. Fino a qualche anno fa questo tipo di effetti grafici era impensabile anche per i normali PC, quindi poterli utilizzare con un dispositivo così semplice ed economico apre a un mare sterminato di possibili implementazioni!

`Raspivid` può essere utilizzata anche per salvare video in formato mp4, trasformando il nostro Raspberry Pi in una videocamera.

Se invece ci interessa acquisire immagini statiche possiamo utilizzare `raspistill`.

Questa seconda utility condivide diverse delle opzioni di `raspivid` e ha le stesse capacità in termini di effetti.

```
pi@pivalter ~ $ raspistill  
raspistill Camera App v1.1
```

```
Runs camera for specific time, and take JPG capture at end if requested  
usage: raspistill [options]
```

Image parameter commands

```
-?, --help      : This help information
-w, --width    : Set image width <size>
-h, --height   : Set image height <size>
-q, --quality  : Set jpeg quality <0 to 100>
-r, --raw      : Add raw bayer data to jpeg metadata
-o, --output   : Output filename <filename> (to write to stdout, use '-o -').  
specified, no file is saved
-v, --verbose   : Output verbose information during run
-t, --timeout   : Time (in ms) before takes picture and shuts down (if not spec  
set to 5s)
-th, --thumb    : Set thumbnail parameters (x:y:quality)
-d, --demo     : Run a demo mode (cycle through range of camera options, no ca
-e, --encoding  : Encoding to use for output file (jpg, bmp, gif, png)
-xx, --exif    : EXIF tag to apply to captures (format as 'key=value')
-tl, --timelapse : Timelapse mode. Takes a picture every <t>ms
```

Preview parameter commands

```
-p, --preview   : Preview window settings <'x,y,w,h'>
-f, --fullscreen : Fullscreen preview mode
-op, --opacity   : Preview window opacity (0-255)
-n, --nopreview  : Do not display a preview window
```

Image parameter commands

```
-sh, --sharpness : Set image sharpness (-100 to 100)
-co, --contrast  : Set image contrast (-100 to 100)
-br, --brightness : Set image brightness (0 to 100)
-sa, --saturation : Set image saturation (-100 to 100)
-ISO, --ISO      : Set capture ISO
-vs, --vstab     : Turn on video stablisation
-ev, --ev        : Set EV compensation
-ex, --exposure  : Set exposure mode (see Notes)
-awb, --awb      : Set AWB mode (see Notes)
-ifx, --imfx     : Set image effect (see Notes)
-cfx, --colfx    : Set colour effect (U:V)
-mm, --metering  : Set metering mode (see Notes)
-rot, --rotation : Set image rotation (0-359)
-hf, --hflip     : Set horizontal flip
-vf, --vflip     : Set vertical flip
```

Notes

Exposure mode options :

off,auto,night,nightpreview,backlight,spotlight,sports,snow,beach,verylong,fixedf
ntishake,fireworks

AWB mode options :

off,auto,sun,cloud,shade,tungsten,fluorescent,incandescent,flash,horizon

Image Effect mode options :

none,negative,solarise,sketch,denoise,emboss,oilpaint,hatch,gpen,pastel,watercolo
film,blur,saturation,colourswap,washedout,posterise,colourpoint,colourbalance,car

Metering Mode options :
average, spot, backlit, matrix

La differenza è che alla fine del tempo di cattura (impostabile e con tanto di preview a schermo) verrà catturata un'immagine statica in formato Jpeg a 5 Megapixel.

Per catturare un'immagine con `raspistill` è necessario specificare il nome di un file da salvare.

Lanciando:

```
pi@pivalter ~ $ raspistill -ifx watercolour -o selfshot.jpg
```

l'autore ha ottenuto l'autoritratto di Raspberry Pi che vedete in [Figura 7.14](#).

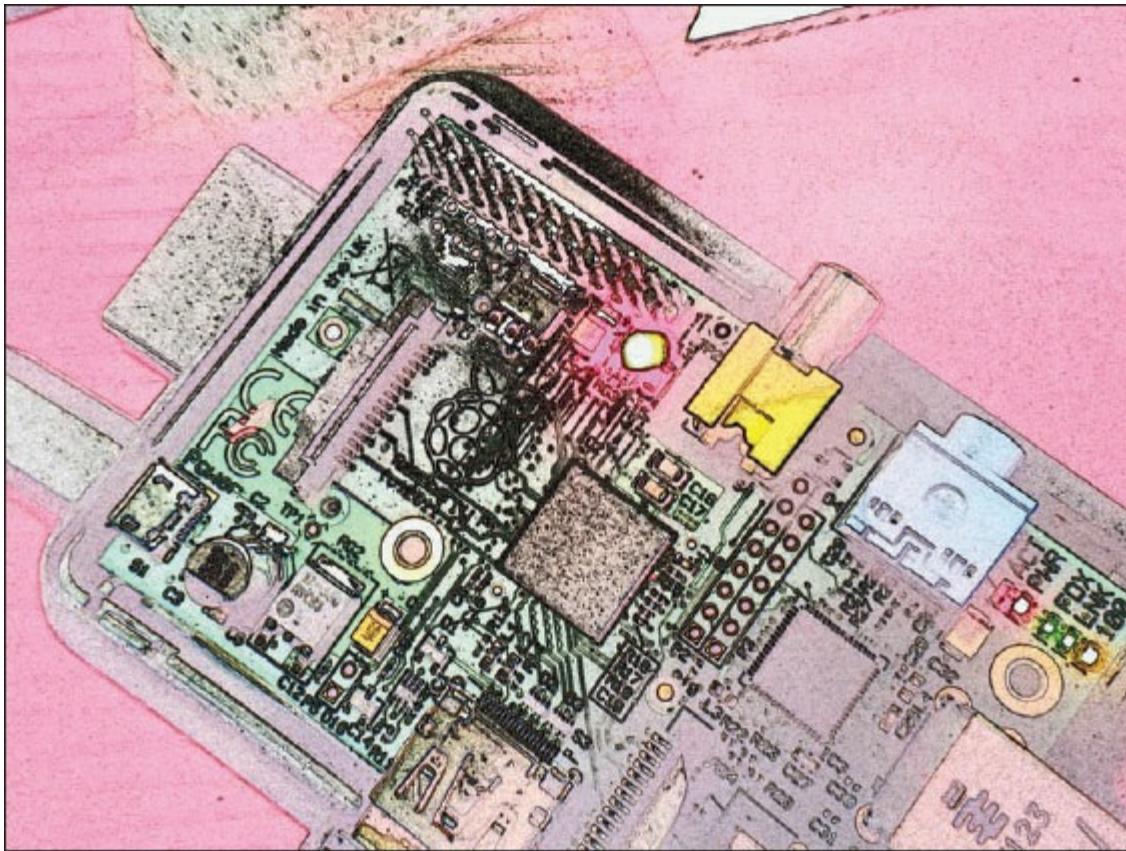


Figura 7.14 - "Autoritratto" di Raspberry Pi

In realtà la fotocamera mette a fuoco soggetti a circa 60 cm di distanza; per ottenere la foto è stato necessario interporre tra la fotocamera e Raspberry Pi una lente di ingrandimento, e il riflesso del LED rosso del modulo telecamera sulla lente si può notare nella foto, vicino al connettore GPIO.

Purtroppo al momento non sono disponibili API in Python per acquisire direttamente immagini o video dal modulo camera; possiamo però

sfruttare la funzione `os.system()` di Python per lanciare raspivid o raspistill e salvare o mostrare a video il flusso di immagini della fotocamera.

```
#! /usr/bin/env python3

import os

def catturaImmagine(filename,width=2592,height=1944,quality=100):
    comando="raspistill -o " + filename
    comando+=" -w "+str(width)
    comando+=" -h "+str(height)
    comando+=" -q "+str(quality)
    comando+=" -n" # disabilita il preview a video
    comando+=" -t 0"# acquisisce immediatamente l'immagine

    os.system(comando)

catturaImmagine("prova.jpg",640,480)
```

In questo semplice esempio la funzione `catturaImmagine` non fa altro che preparare la command line di `raspistill` per poi passarla a `os.system()`, che lancerà `raspistill`. Gestisce solo una piccola parte dei parametri di `raspistill`, ma può essere una base di partenza per aggiungere opzioni e, tra l'altro, offre la possibilità di scattare la foto quando viene premuto un pulsante collegato a un pin di GPIO, come in una vera fotocamera!

In questo capitolo abbiamo solo sfiorato un argomento complesso come quello dei sensori e del loro interfacciamento con Raspberry Pi. Avremo modo di giocare ancora un po' con i sensori nelle prossime pagine, ma spero che questi pochi esempi abbiano dimostrato il grandissimo potenziale di Raspberry Pi e stuzzicato la vostra curiosità e immaginazione per sfruttarlo per gli utilizzi più disparati!

Raspberry Pi e la nuvola

Cloud computing e internet delle cose sono solo termini marketing? Oppure sono concetti che possiamo applicare al nostro Raspberry Pi per farlo diventare un dispositivo connesso, in grado di memorizzare e scambiare informazioni nella rete delle reti? In questo capitolo vedremo come collegare il nostro piccolo PC alle infrastrutture più complesse di internet.

Il cloud computing

Negli ultimi anni i servizi su internet si sono evoluti in modo impressionante, con una gestione di conseguenza sempre più complessa. Esistono servizi che devono memorizzare e distribuire quantità enormi di dati; pensiamo ai social network e a servizi come Gmail o DropBox. Questi servizi devono anche gestire un numero molto alto di accessi, magari con punte estemporanee di traffico significative (pensiamo ai portali di e-commerce che praticano saldi nel Black-Friday americano). Le tradizionali architetture client-server si dimostrano limitate nel rispondere a questo tipo di esigenze. È necessario sovrdimensionare i server per gestire i picchi di richieste, distribuire il carico e mantenere coerenti i dati tra i diversi server e così via.

Per rispondere a queste esigenze diverse aziende tra cui Amazon, Google e Microsoft, hanno sviluppato infrastrutture per il cloud computing. Questi sistemi distribuiscono il lavoro tra diversi server, localizzati in datacenter sparsi per il mondo, e gestiscono in modo trasparente la copia dei dati e il bilanciamento del traffico. Il numero di server dedicati a un'applicazione (o a un sito web) può variare dinamicamente e i costi sono generalmente calcolati in funzione del traffico e della quantità di elaborazioni effettuate. L'esistenza di queste infrastrutture, inizialmente utilizzate direttamente dalle aziende che le hanno sviluppate per i loro servizi e ora aperte come "piattaforme" per chiunque voglia svilupparci applicazioni, consente anche a piccole realtà di fornire servizi in grado di gestire una crescita delle richieste e un servizio sempre attivo.

L'internet delle cose

Raspberry Pi è la dimostrazione che, allo stato attuale della tecnologia, è possibile realizzare dispositivi che, oltre a discrete capacità di elaborazione e memorizzazione locale, hanno la possibilità di collegarsi in rete. Fino a pochi anni fa l'idea di collegare in rete dispositivi a microcontrollore sembrava un'utopia. Oggi anche oggetti da pochi euro hanno la capacità di accedere alla rete e condividere informazioni.

La possibilità di comunicare offre ai dispositivi due importanti opportunità: possono condividere i dati che acquisiscono localmente tramite sensori e possono accedere a dati nella rete.

Pensiamo, per esempio, a un comune navigatore satellitare. Il navigatore può (ovviamente nel rispetto della privacy!) comunicare la posizione del veicolo e ricavare le informazioni sul traffico dalla rete.

Qui qualcuno si domanderà: ma non è utile solo la seconda parte? Avere le informazioni sul traffico mi consente di risparmiare tempo. Ma qual è il vantaggio di comunicare la posizione in rete?

Il vantaggio si intuisce quando si relaziona l'idea dei device sempre connessi con il cloud computing. La posizione dei singoli veicoli non è un dato così significativo (se non per violare la privacy del guidatore!), ma le posizioni di migliaia o milioni di veicoli potrebbero consentire di capire immediatamente eventuali criticità (tutti i veicoli su un certo tratto di strada si fermano, probabilmente c'è un incidente!), di prevenirle (molti veicoli hanno impostato la stessa destinazione e la

raggiungeranno alla stessa ora, saturando la viabilità) o anche di elaborare trend più complessi arrivando, per esempio, a individuare gli orari in cui una strada è trafficata o libera basandosi su dati di percorrenza di mesi o anni. Ovviamente il processore del nostro Raspberry Pi, che non è troppo diverso da quelli che si trovano sui navigatori in commercio, non sarebbe in grado di ricevere ed elaborare tutte queste informazioni in tempo reale. E non avrebbe nemmeno senso che tutti i navigatori conoscano la posizione di ogni altro veicolo! Inviare i dati acquisiti in locale, lasciare che “la nuvola” li elabori e prendersi i risultati che ci interessano (la situazione del traffico sulle strade che andremo a percorrere) è l’approccio migliore per ottenere il risultato con il minimo dispendio di forze.

Un esempio di questa filosofia è il cronotermostato Nest (www.nest.com). Questo cronotermostato è molto simile a quelli tradizionali, ma è in grado di apprendere le abitudini dell’utente durante l’uso e anche di recuperare informazioni dalla rete sia direttamente sul dispositivo (previsioni meteo) sia tramite il sito web o applicazioni per smartphone e tablet, che consentono di verificare i consumi e anche di controllare il sistema da remoto.

Dispositivi sempre più “intelligenti” e facili da usare potrebbero migliorare il nostro modo di vivere e diventare parte delle nostre abitudini di tutti i giorni, come lo sono diventati, in epoche diverse, l’energia elettrica, il telefono, internet.

Raspberry Pi e il cloud

A questo punto vi starete domandando: cosa c’entra il mio Raspberry Pi con tutto questo?

L’accesso al cloud computing non è prerogativa solo di grandi aziende o di prodotti all’avanguardia. Anche con il nostro Raspberry Pi possiamo sperimentare e sfruttare alcuni dei servizi cloud per realizzare applicazioni interessanti e, perché no, divertenti. Vedremo alcuni servizi cloud e apprenderemo le basi per accedere alle funzioni che esportano, tipicamente tramite delle Application Programming Interfaces (API).

Il primo servizio che prenderemo in considerazione è il popolare servizio di condivisione file DropBox. DropBox supporta moltissime piattaforme mobili e non, e consente di scambiare e sincronizzare file e cartelle, aiutandoci ad avere sempre sotto mano i nostri file

indipendentemente dal dispositivo che stiamo utilizzando.

DropBox fornisce delle API per Python e questo non dovrebbe stupirci visto che Guido Van Rossum, il “papà” di Python, lavora proprio per DropBox!

DropBox

Prima di poter utilizzare DropBox da Python è necessario compiere alcuni passi preparatori.

Il primo passo, se già non abbiamo un account, è quello di registrarcici come utenti DropBox sul sito www.dropbox.com. La registrazione è gratuita e ci dà diritto a 2 GB di spazio per i nostri file. È possibile acquistare più spazio e più funzionalità acquistando le offerte a pagamento, ma per ora 2 GB basteranno.

Una volta registrati come utenti dovremo registrare la nostra applicazione. Ogni applicazione che accede a DropBox è registrata con delle chiavi di accesso specifiche, per tracciare possibili utilizzi non corretti e per consentire agli utenti di avere sotto controllo le applicazioni che possono accedere ai propri dati. Ogni applicazione dovrà essere autorizzata dall’utente proprietario dei dati, ma vedremo questo passaggio in uno step successivo.

Per ora possiamo andare sul sito developer.dropbox.com e creare la nostra applicazione (dopo esserci registrati e autenticati sul sito principale di DropBox).

Per creare la nostra applicazione dobbiamo scegliere l’opzione **App Console** nel menu a sinistra della homepage.

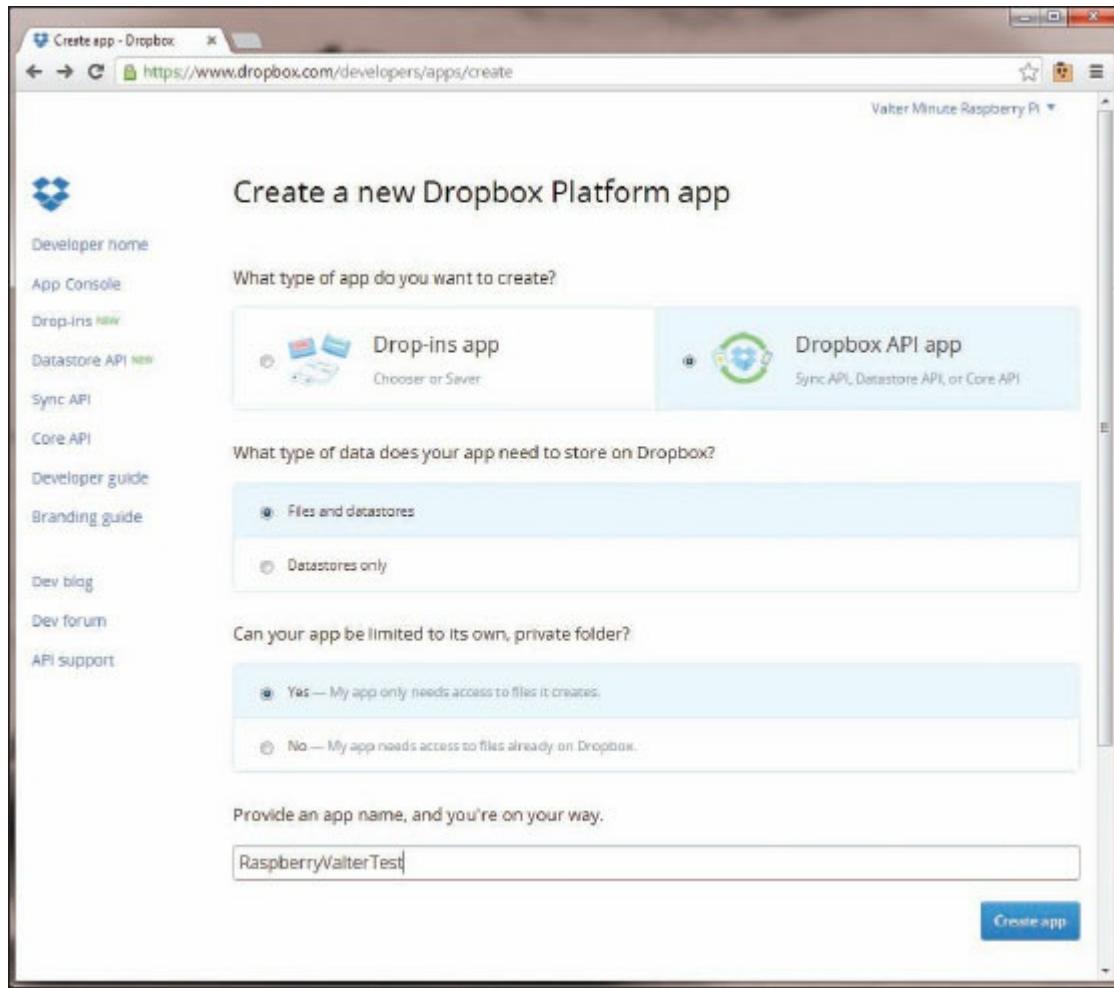


Figura 8.1 - DropBox - App Console.

Nella console scegliamo l'opzione **DropBox API app** per creare una nuova applicazione che faccia uso delle API, selezioniamo l'accesso a file e datastore e facciamo in modo che la nostra applicazione possa accedere solo ai file che andrà a creare (giusto per evitare danni accidentali in caso di errori durante le prime prove). A questo punto dovremo assegnare un nome alla nostra applicazione e potremo premere il pulsante **Create App** per crearla.

Una volta creata la nuova applicazione ci verranno mostrati i suoi dati salienti.

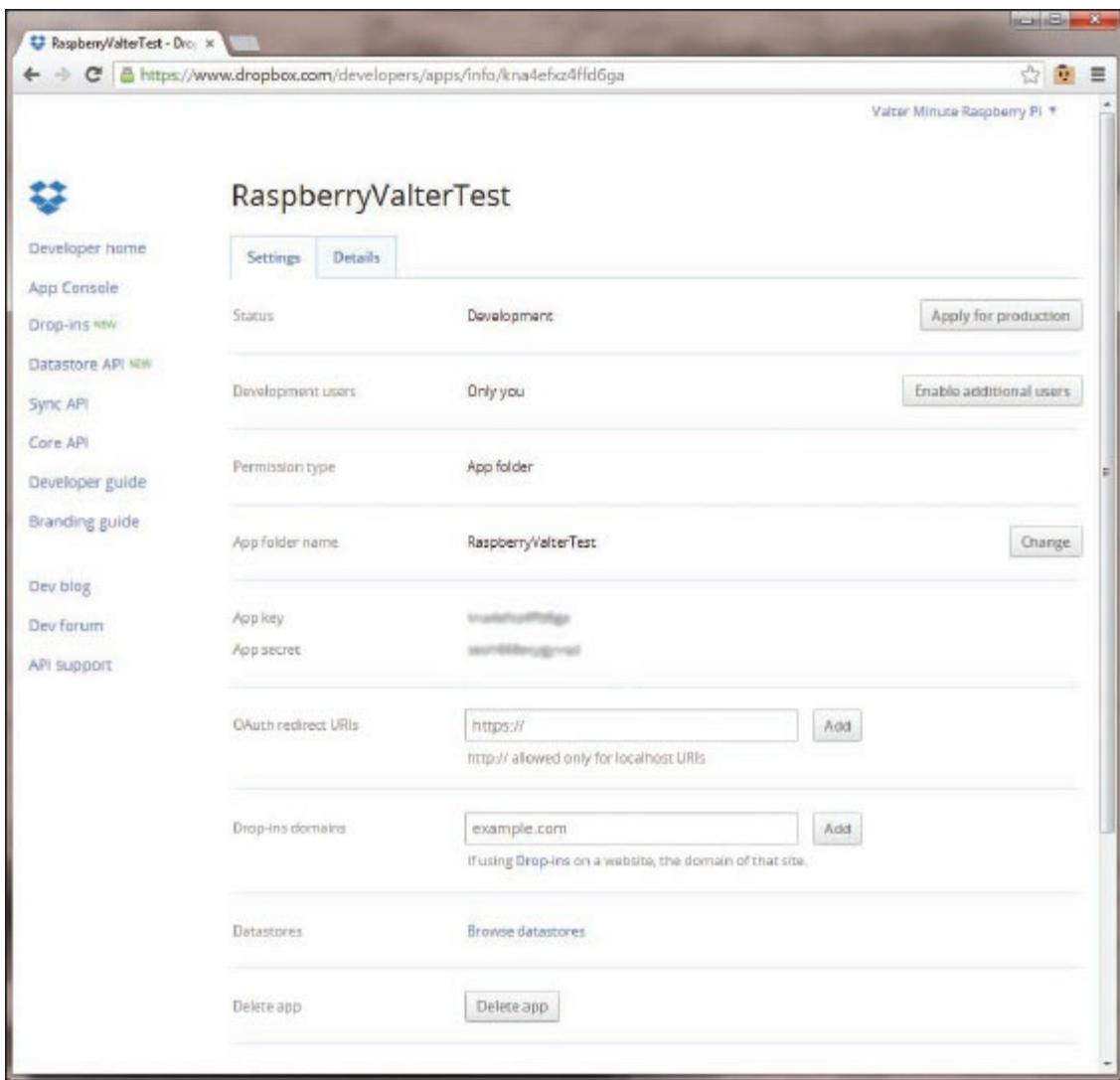


Figura 8.2 - DropBox - informazioni sulla app.

È importante annotarsi i dati **App key** e **App secret**. Questi dati serviranno per identificare la nostra applicazione e non devono essere condivisi con altri, perché questo viola le condizioni di licenza di DropBox e potrebbe portare alla cancellazione della nostra app. Questo è il motivo per cui le informazioni sono state nascoste in [Figura 8.2](#).

Tornando nella App Console vedremo la nostra nuova applicazione e potremo, nel caso, recuperare i dati necessari.

Il secondo step è l'installazione delle API di DropBox per Python. Esistono diversi set di API per le varie funzioni. Noi utilizzeremo le Core API e per prima cosa dovremo scaricare il relativo SDK sul nostro Raspberry Pi che dovrà essere, come per il resto di questo capitolo, collegato in rete:

```
pi@pivalter ~ $ wget https://www.dropbox.com/static/developers/dropbox-python-sdk  
--2013-09-23 12:10:05-- https://www.dropbox.com/static/developers/dropbox-python  
Risoluzione di www.dropbox.com (www.dropbox.com) ... 199.47.217.171  
Connessione a www.dropbox.com (www.dropbox.com) |199.47.217.171|:443... connesso.  
Richiesta HTTP inviata, in attesa di risposta... 200 OK
```

```
Lunghezza: 812016 (793K) [application/zip]
Salvataggio in: "dropbox-python-sdk-1.6.zip"
```

```
100% [=====] 812.016 221K/s in 3,6s
```

```
2013-09-23 12:10:17 (221 KB/s) - "dropbox-python-sdk-1.6.zip" salvato [812016/812
```

Il file è compresso e dovremo decomprimerlo mediante il comando `unzip`.

```
pi@pivalter ~ $ unzip dropbox-python-sdk-1.6.zip
Archive: dropbox-python-sdk-1.6.zip
  creating: dropbox-python-sdk-1.6/
  inflating: dropbox-python-sdk-1.6/CHANGELOG
  inflating: dropbox-python-sdk-1.6/conf.py
...
  inflating: dropbox-python-sdk-1.6/tests/test_rest.py
  inflating: dropbox-python-sdk-1.6/tests/test_session.py
```

Ora possiamo installare le API di DropBox. Prima di installarle, andremo a configurare la variabile `PYTHONPATH`. Questa variabile consente di specificare le cartelle da cui caricare i moduli. Terremo le nostre librerie nella cartella `pythonlibs` sotto la nostra home.

```
pi@pivalter ~ $ cd dropbox-python-sdk-1.6/
pi@pivalter ~/dropbox-python-sdk-1.6 $ export PYTHONPATH=$HOME/pythonlibs/lib/
python3.2/site-packages
pi@pivalter ~/dropbox-python-sdk-1.6 $ python3 setup.py install --prefix $HOME/py
running install
running bdist_egg
running egg_info
...
Installed /home/pi/pythonlibs/lib/python3.2/site-packages/dropbox-1.6-py3.2.egg
Processing dependencies for dropbox==1.6
Finished processing dependencies for dropbox==1.6
```

Visto che andremo a utilizzare spesso la nostra libreria, conviene mettere la configurazione della variabile `PYTHONPATH` all'interno del file `.bashrc`, nella nostra home. Questo file viene eseguito a ogni login e aggiungere il comando:

```
export PYTHONPATH=$HOME/pythonlibs/lib/python3.2/site-packages
```

alla fine di esso ci garantirà che la variabile venga impostata ogni volta che riavviamo il nostro Raspberry Pi.

Ora possiamo provare che tutto funzioni correttamente lanciando l'interprete Python e digitando:

```
pi@pivalter ~ $ python3
Python 3.2.3 (default, Mar 1 2013, 11:53:50)
```

```
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import dropbox
>>>
```

Se l'istruzione `import` non segnala errori, siamo pronti per utilizzare le API di DropBox! Il primo step per utilizzare DropBox dalla nostra applicazione è ottenere dall'utente il consenso ad accedere ai suoi dati. Anche se in questo caso utente e sviluppatore dell'applicazione coincidono, dovremo compiere questo passaggio.

Per prima cosa possiamo creare un oggetto di autenticazione chiamando il metodo `DropboxOAuth2FlowNoRedirect()` delle API di DropBox.

```
>>> autenticazione=dropbox.client.DropboxOAuth2FlowNoRedirect("***APPKEY***",
"***APPSECRET***")
```

I due parametri del metodo sono l'app key e l'app secret che abbiamo ottenuto nel passaggio precedente dal sito di DropBox.

Per poter dare il suo consenso l'utente deve accedere a una pagina web il cui indirizzo è ritornato dal metodo `start()` dell'oggetto di autenticazione.

```
>>> urlautenticazione=autenticazione.start()
>>> urlautenticazione
'https://www.dropbox.com/1/oauth2/authorize?response_type=code&client_id=knf4efxz'
```

A questo punto dovremo copiare questo indirizzo nel browser e usarlo per autorizzare l'applicazione.

La pagina di autorizzazione riporta i dati principali dell'applicazione e il tipo di accesso richiesto (in questo caso solo l'accesso ai dati nella cartella dedicata all'applicazione).

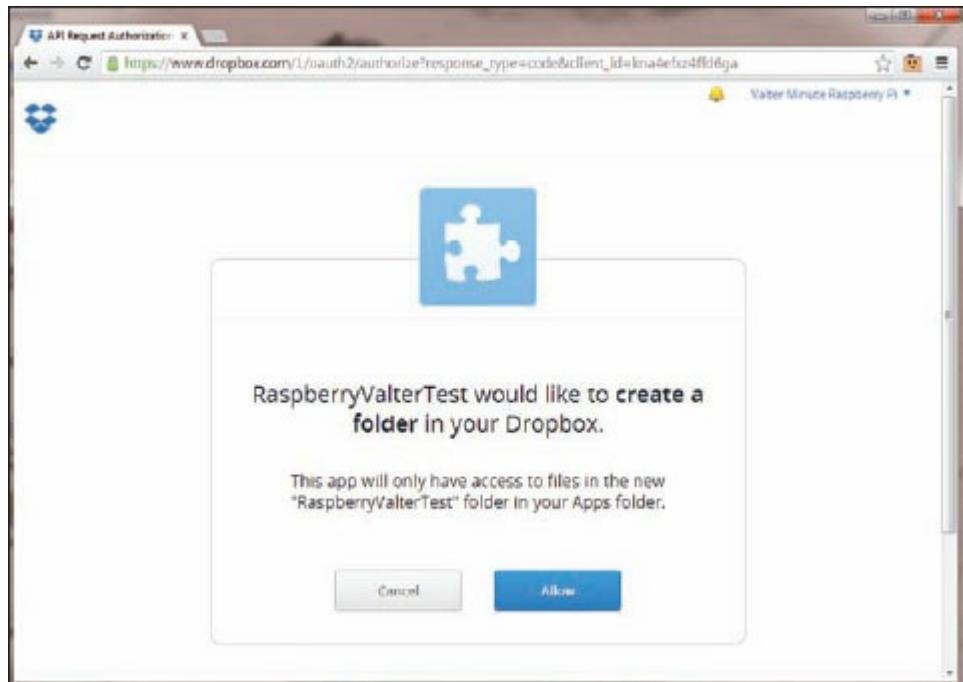


Figura 8.3 - DropBox - autorizzazione dell'applicazione.

Premendo il pulsante **Allow** l'applicazione verrà autorizzata e verrà generato un codice di accesso.

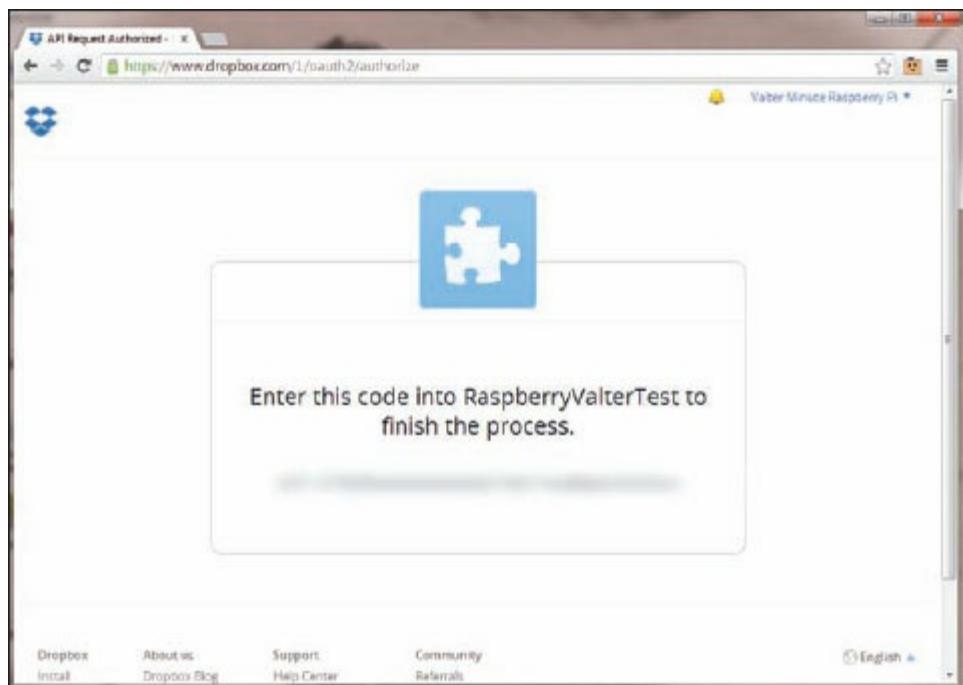


Figura 8.4 - DropBox - codice di accesso.

Una volta ottenuto questo codice l'applicazione potrà usarlo per generare un token di autenticazione. Questo token potrà essere memorizzato dall'applicazione e utilizzato per accedere ai dati, senza ripetere il processo di autenticazione.

```
>>> token=autenticazione.finish("**CODICE**")
```

Ora che l'applicazione è stata autorizzata, il token può essere utilizzato per accedere alle funzioni di DropBox client. Per farlo dobbiamo ottenere un oggetto `DropboxClient` passandogli il token:

```
>>> client=dropbox.client.DropboxClient(token[0])
```

Noteate che viene passato l'elemento `0` di `token`, questo perché `token` è in realtà un **tuple**. I tuple in Python sono liste di oggetti che però, a differenza delle liste viste in precedenza, non sono modificabili. Con un tuple è possibile rappresentare dati aggregati, costituiti cioè da più valori combinati.

```
>>> token  
('dpwQfv2G4zsAAAAAAAAAdBVZtLNvwnuZvbWrKhsDdTyJYyW5U7xBkeJrg9PsEDF', '217158070')
```

Il tuple viene stampato tra parentesi tonde (a differenza della lista che utilizza le quadre) ed è, in questo caso, costituito da un token e da un identificativo utente.

A questo punto è possibile copiare file nella nostra cartella DropBox. Per copiare un file dobbiamo per prima cosa aprirlo chiamando la funzione `open()`. Questa funzione non è parte di DropBox, ma è la funzione standard di Python per ottenere un oggetto che potrà poi essere utilizzato per leggere e scrivere dati nel file. In questo caso ci limiteremo a passarlo alla funzione `put_file` del client di DropBox.

```
>>> file=open("ADS1015.py","rb")  
>>> client.put_file("ADS1015.py",file)  
{'size': '1.8 KB', 'rev': '4146e99ef', 'thumb_exists': False, 'bytes': 1832,  
'modified': 'Mon, 23 Sep 2013 12:31:50 +0000', 'mime_type': 'text/x-python', 'pat  
'/ADS1015 (1).py', 'is_dir': False, 'icon': 'page_white_code', 'root': 'app_fold  
'client_mtime': 'Mon, 23 Sep 2013 12:31:50 +0000', 'revision': 4}
```

La funzione `put_file()` ci torna dei metadati relativi al nostro file. Per metadati si intendono informazioni accessorie, oltre ai dati contenuti nel file, quali data e ora di creazione, dimensione ecc.

Visitando la pagina principale di DropBox (www.dropbox.com), dopo aver fatto il login apparirà l'elenco dei file nella nostra cartella.

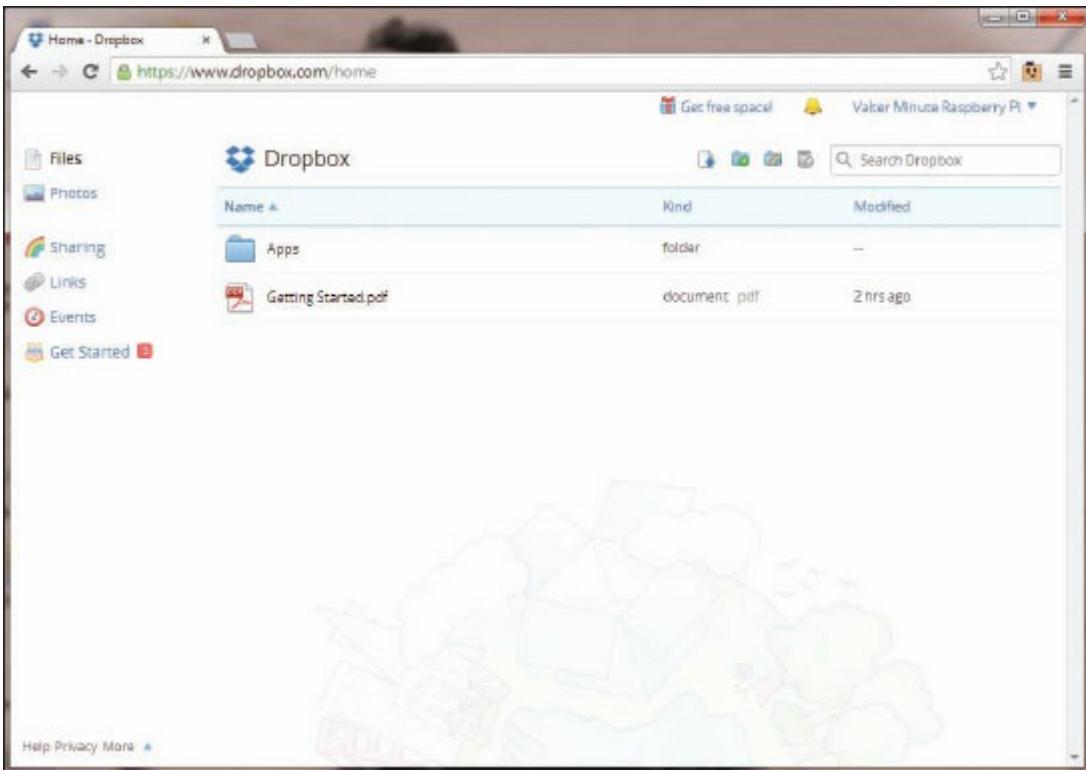


Figura 8.5 - DropBox - l'elenco dei file presenti nella nostra cartella.

Ci sarà una cartella Apps con al suo interno una cartella con il nome della nostra applicazione e, all'interno di questa, il file che abbiamo appena caricato.

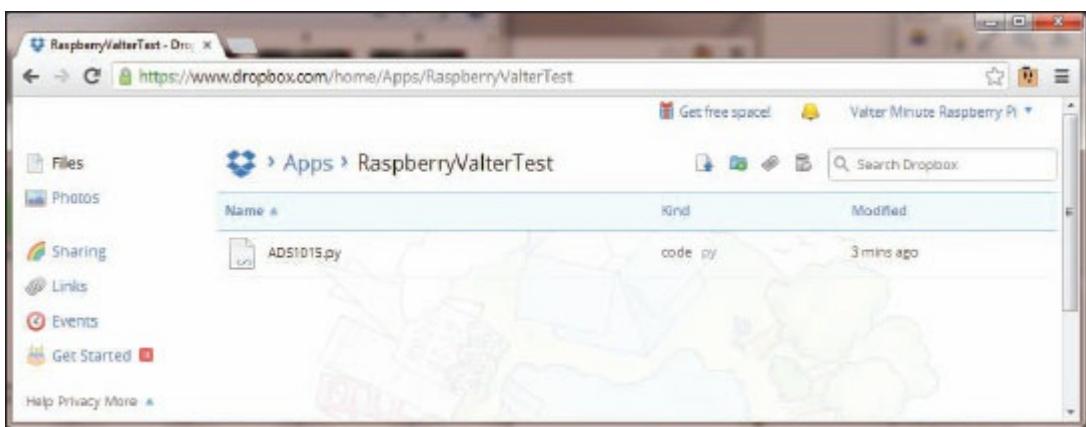


Figura 8.6 - DropBox - il nostro primo upload!

Come abbiamo visto DropBox consente di utilizzare uno storage in rete, condiviso tra tanti dispositivi e aggiornato su PC, tablet e smartphone in modo automatico. Immaginate quali possibilità per le nostre applicazioni.

Pensiamo, per esempio, a come trasformare il nostro Raspberry Pi in una piccola telecamera di sorveglianza. Se non avete il camera module di Raspberry Pi, ma avete una webcam compatibile con lo standard

USB video (si collega a Windows e Mac senza driver) potete utilizzarla per catturare immagini. Invece di `raspistill` potrete utilizzare un'altra utility: `fswebcam`.

Questa utility va installata usando, come al solito, `apt-get`:

```
pi@pivalter ~ $ sudo apt-get install fswebcam
sudo: unable to resolve host pivalter
Lettura elenco dei pacchetti... Fatto
Generazione albero delle dipendenze
Lettura informazioni sullo stato... Fatto
I seguenti pacchetti NUOVI saranno installati:
  fswebcam
[...]
Elaborazione dei trigger per man-db...
Configurazione di fswebcam (20110717-1) ...
```

Una volta installata l'applicazione potremo catturare un'immagine lanciandola da command line:

```
pi@pivalter ~ $ fswebcam provawebcam.jpg
--- Opening /dev/video0...
Trying source module v4l2...
/dev/video0 opened.
No input was specified, using the first.
Adjusting resolution from 384x288 to 352x288.
--- Capturing frame...
Captured frame in 0.00 seconds.
--- Processing captured image...
Writing JPEG image to 'provawebcam.jpg'.
```

Come al solito, l'opzione `--help` ci spiegherà tutti i parametri e la loro funzione:

```
pi@pivalter ~ $ fswebcam --help
Usage: fswebcam [<options>] <filename> [<options>] <filename> ... ]
```

Options:

<code>-?, --help</code>	Display this help page and exit.
<code>-c, --config <filename></code>	Load configuration from file.
<code>-q, --quiet</code>	Hides all messages except for errors.
<code>-v, --verbose</code>	Displays extra messages while capturing
<code>--version</code>	Displays the version and exits.
<code>-l, --loop <seconds></code>	Run in loop mode.
<code>-b, --background</code>	Run in the background.
<code>-o, --output <filename></code>	Output the log to a file.
<code>-d, --device <name></code>	Sets the source to use.
<code>-i, --input <number/name></code>	Selects the input to use.
<code>-t, --tuner <number></code>	Selects the tuner to use.
<code>-f, --frequency <number></code>	Selects the frequency use.
<code>-p, --palette <name></code>	Selects the palette format to use.
<code>-D, --delay <number></code>	Sets the pre-capture delay time. (seconds)
<code>-r, --resolution <size></code>	Sets the capture resolution.

--fps <framerate>	Sets the capture frame rate.
-F, --frames <number>	Sets the number of frames to capture.
-S, --skip <number>	Sets the number of frames to skip.
--dumpframe <filename>	Dump a raw frame frame to file.
-s, --set <name>=<value>	Sets a control value.
--revert	Restores original captured image.
--flip <direction>	Flips the image. (h, v)
--crop <size>[,<offset>]	Crop a part of the image.
--scale <size>	Scales the image.
--rotate <angle>	Rotates the image in right angles.
--deinterlace	Reduces interlace artifacts.
--invert	Inverts the images colours.
--greyscale	Removes colour from the image.
--swapchannels <c1c2>	Swap channels c1 and c2.
--no-banner	Hides the banner.
--top-banner	Puts the banner at the top.
--bottom-banner	Puts the banner at the bottom. (Default)
--banner-colour <colour>	Sets the banner colour. (#AARRGGBB)
--line-colour <colour>	Sets the banner line colour.
--text-colour <colour>	Sets the text colour.
--font <[name] [:size]>	Sets the font and/or size.
--no-shadow	Disables the text shadow.
--shadow	Enables the text shadow.
--title <text>	Sets the main title. (top left)
--no-title	Clears the main title.
--subtitle <text>	Sets the sub-title. (bottom left)
--no-subtitle	Clears the sub-title.
--timestamp <format>	Sets the timestamp format. (top right)
--no-timestamp	Clears the timestamp.
--gmt	Use GMT instead of local timezone.
--info <text>	Sets the info text. (bottom right)
--no-info	Clears the info text.
--underlay <PNG image>	Sets the underlay image.
--no-underlay	Clears the underlay.
--overlay <PNG image>	Sets the overlay image.
--no-overlay	Clears the overlay.
--jpeg <factor>	Outputs a JPEG image. (-1, 0 - 95)
--png <factor>	Outputs a PNG image. (-1, 0 - 10)
--save <filename>	Save image to file.
--exec <command>	Execute a command and wait for it to complete.

Ora possiamo vedere il nostro semplicissimo programma di video-sorveglianza.

```
#! /usr/bin/env python3

import dropbox
import os
import time

FILE_CONFIGURAZIONE = "sorveglianza_token"
FILE_IMMAGINE = "sorveglianza_immagine.jpg"

def scattaFoto(path):
# se state usando una webcam decommentate la riga qui sotto e commentate
# l'altra chiamata a system
```

```

# os.system("fswebcam "+path)
os.system("raspistill -n -t 0 -w 640 -h 480 -q 90 -o "+path)

#controlla se esiste il file dove viene salvato il token di sicurezza
token=""

if not (os.path.isfile(FILE_CONFIGURAZIONE)):
# il file non c'è, dobbiamo creare un token
autenticazione=dropbox.client.DropboxOAuth2FlowNoRedirect("****APPKEY****", "****AP"
url=autenticazione.start()
print("L'applicazione non è ancora stata autorizzata.")
print("E' necessario aprire l'URL")
print(url)
print("Con il browser e inserire il codice ritornato da DropBox")
codice=input("Codice : ")
tokentuple=autenticazione.finish(codice)
token=tokentuple[0]

file=open(FILE_CONFIGURAZIONE, "w")
file.write(token)
file.close()

else:

# se il file esiste, allora legge il token
file=open(FILE_CONFIGURAZIONE, "r")
token=file.read()
file.close()
print("token caricato")

# crea l'oggetto client
client=dropbox.client.DropboxClient(token)

while True:
# acquisisce l'immagine
scattaFoto(FILE_IMMAGINE)
print("foto acquisita")

# effettua l'upload forzando la sovrascrittura
fileimmagine=open(FILE_IMMAGINE, "rb")
try:
    client.put_file(FILE_IMMAGINE, fileimmagine, overwrite=True)
finally:
    fileimmagine.close()

print("upload completato")

time.sleep(60)

```

La struttura del programma è molto semplice e i commenti dovrebbero aiutarvi a comprenderlo abbastanza chiaramente. Vediamo, come al solito, qualche spunto interessante.

Il token viene memorizzato in un file. Il nome del file e quello dell'immagine che viene salvata e copiata su DropBox sono scritti tutti

in carattere maiuscolo. Questa è una convenzione per indicare che si tratta di costanti. Valori, cioè, che non dovrebbero cambiare nel corso dell'esecuzione del programma. Python non supporta costanti con un nome, quindi i nostri due valori sono, a tutti gli effetti, delle variabili, ma averli dichiarati in quel modo dovrebbe farci capire il loro scopo. Usare delle costanti è utile perché ci consentono di modificare i nomi o i valori di default usati nei nostri programmi senza dover ogni volta "spulciare" tutto il codice.

La funzione `open()` apre un file dato il suo nome. Il secondo parametro definisce la modalità di accesso (`r` per Read, lettura, `w` per Write, scrittura) e può essere seguito dalla lettera `b` che indica un accesso in modalità binaria e non testuale.

Nel nostro caso la funzione viene prima usata per scrivere il token in un file `o`, se il file esiste già (condizione controllata tramite `os.path.isfile()`), per leggerlo e poi per aprire il file immagine per l'upload su DropBox.

Il programma attende un minuto prima di catturare la nuova immagine.

Visto il ritmo "blando" di acquisizione, dei messaggi, stampati con chiamate a `print()`, ci informano dei vari passaggi effettuati.

La cattura dell'immagine è confinata nella funzione `scattaFoto` ed eseguendo `fswebcam` al posto di `raspistill` potremo utilizzare anche una webcam USB.

Ovviamente questo programma è solo un abbozzo di quello che potrebbe essere un vero programma di videosorveglianza. Non gestisce gli errori, non accetta parametri che configurino il tempo di acquisizione, la tipologia di camera da utilizzare ecc.

Avere l'ultima immagine catturata (o, con qualche modifica sfruttando le API di DropBox per cancellare e rinominare file, le ultime immagini) su un server nella "nuvola" ci consentirà di recuperare le informazioni anche se il nostro Raspberry Pi non è fisicamente accessibile (o noi siamo fuori casa!) o se è stato spento o addirittura distrutto.

E potremo vedere comodamente le immagini anche dal nostro smartphone; fate solo attenzione a non sincronizzare troppo spesso l'immagine se avete una connessione dati limitata.

Potrete prendere questo piccolo programma a modello per sviluppare soluzioni più complesse per tenere sotto controllo il vostro acquario, un bimbo che dorme, il tempo fuori dalla finestra di casa... fate solo

attenzione a non violare mai la privacy di nessuno!

Xively

Xively è un servizio che consente ai dispositivi di pubblicare informazioni che poi possono essere utilizzate dai dispositivi stessi oppure da applicazioni in grado di aggregare i dati dei diversi dispositivi. Come vedete da questa brevissima (e incompleta!) descrizione, Xively è un esempio di utilizzo del cloud computing unito ai dispositivi.

Con Xively è possibile pubblicare dei datastream con le informazioni lette in funzione del tempo. Questo consentirà di vedere, per esempio, i dati letti dai nostri sensori in formato grafico.

Per utilizzare i servizi di Xively è necessaria una registrazione gratuita. Per registrarsi basta visitare il sito [www.xively.com](https://xively.com) e selezionare il pulsante **Get Started**.

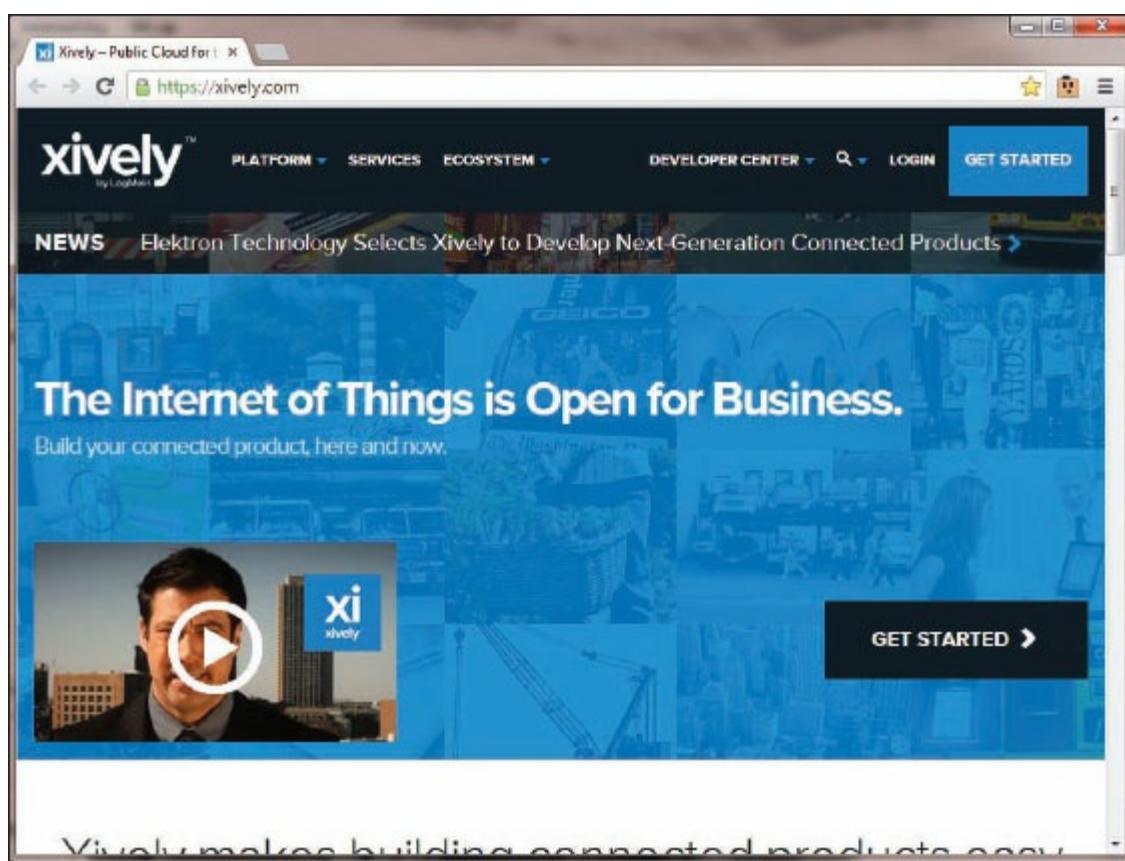


Figura 8.7 - Xively - la pagina principale.

Per poter utilizzare i servizi di Xively gratuitamente dovremo selezionare l'opzione **Sign up for a free Developer Account**. Xively fornisce anche servizi professionali alle aziende, ma per sperimentare non sono necessari.

Una volta completata la fase di registrazione potremo effettuare il

login e utilizzare i web tools.

Selezionate l'opzione **Add Device** della pagina **Develop** per poter pubblicare informazioni dal nostro Raspberry Pi

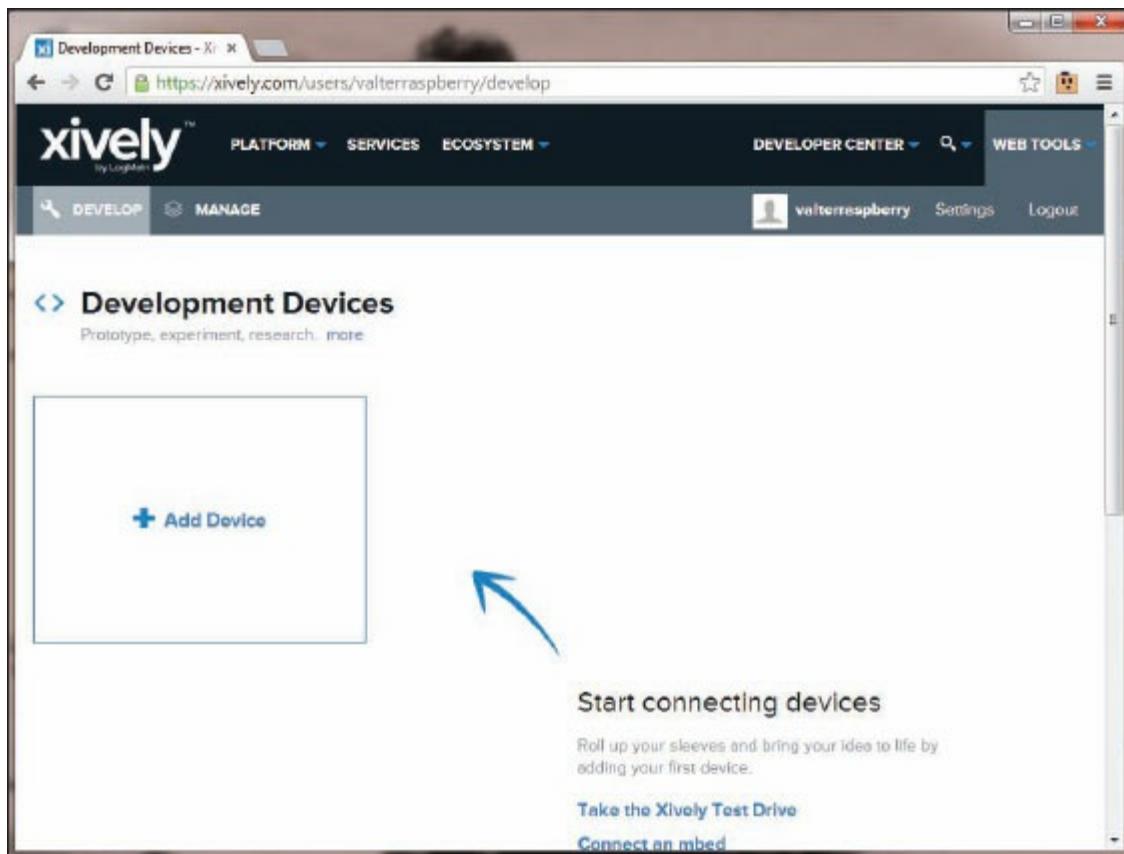


Figura 8.8 - Xively - Web tools - Develop.

Per creare un nuovo device dovremo inserire un nome, una descrizione e decidere se il nostro device sarà privato (visibile solo a noi dopo aver fatto il login) oppure pubblico. Per iniziare possiamo tenere privato il device; potremo cambiare questa impostazione in un secondo tempo.

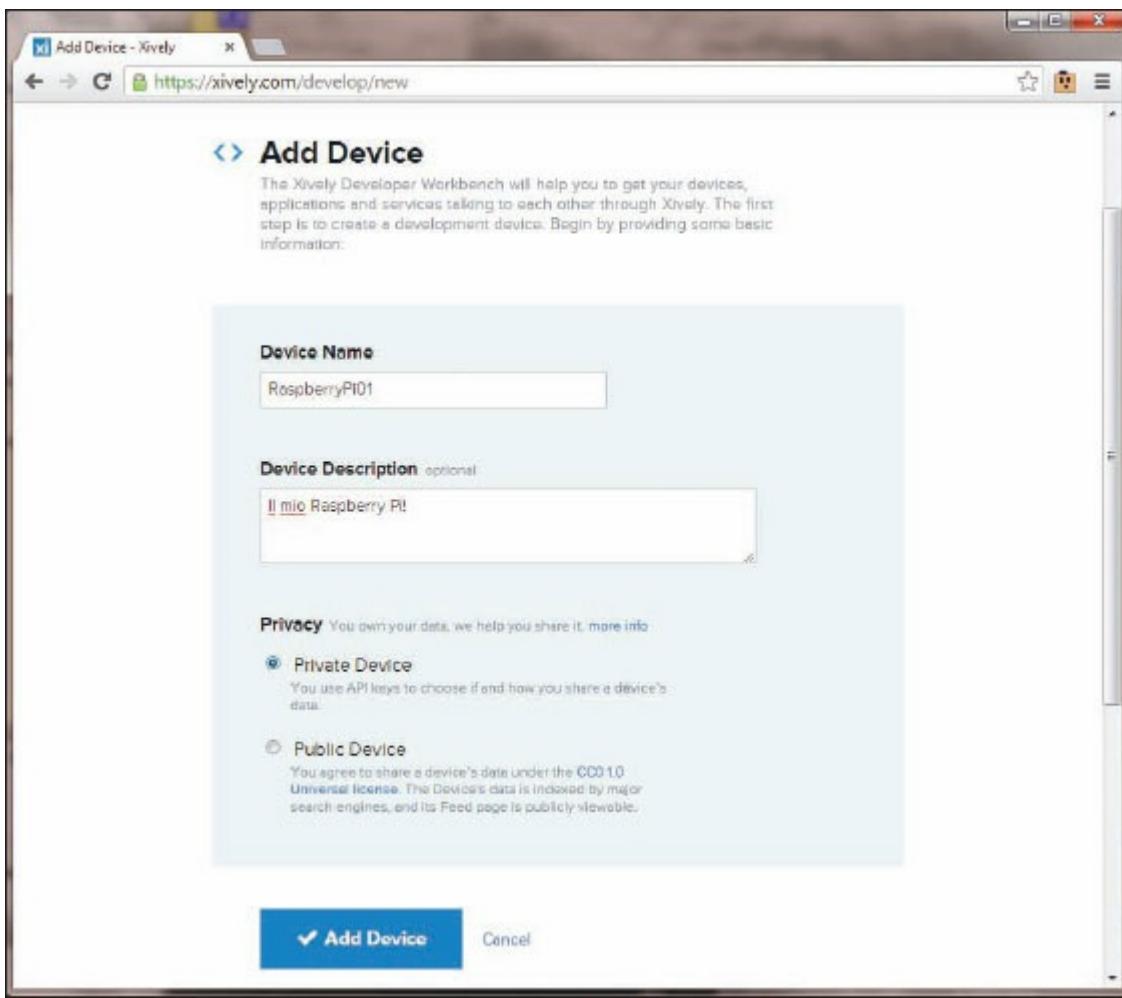


Figura 8.9 - Xively - registrazione di un nuovo dispositivo.

Una volta creato il dispositivo potremo selezionarlo e, nella pagina di informazioni relativa, trovare la chiave (alfanumerica) che ci servirà per utilizzare le API di Xively e l'ID (numerico) del nostro feed, che consentirà di associare le informazioni che trasferiremo a Xively al nostro Raspberry Pi. Queste informazioni sono personali e associate all'account Xively, motivo per sono state nascosta in [Figura 8.10](#); dovremo utilizzarle nei nostri programmi tra poche pagine.

Ogni feed può essere costituito da più canali. In questo esempio useremo i sensori integrati in Raspberry Pi per misurare la temperatura della CPU e della GPU. Possiamo quindi usare il pulsante [Add Channel](#) per creare due nuovi canali che chiameremo **TemperaturaCPU** e **TemperaturaGPU**, inserendo le informazioni come mostrato in [Figura 8.11](#).

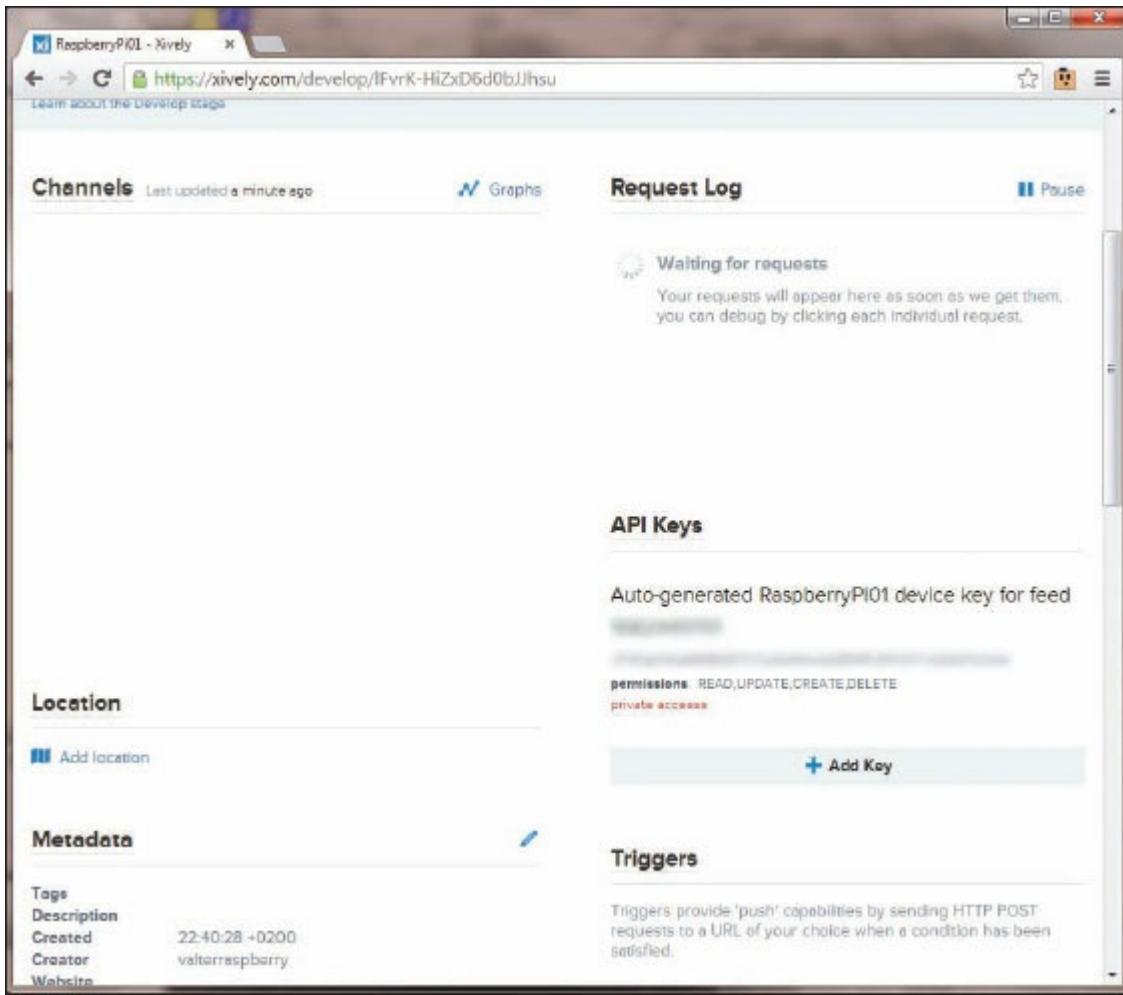


Figura 8.10 - Xively - la chiave alfanumerica per utilizzare le API e l'ID del nostro feed.

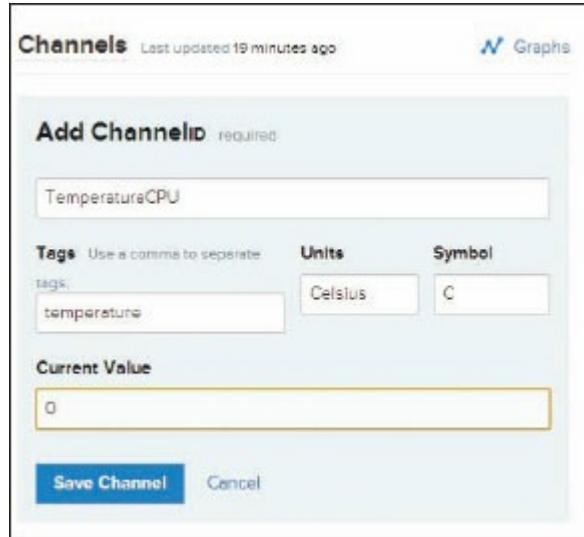


Figura 8.11 - Xively - configurazione di un canale dati.

Una volta configurato il nostro dispositivo sul sito di Xively possiamo scaricare e installare le librerie Python in modo analogo a quanto fatto per DropBox e quick2wire:

```
pi@pivalter ~ $ mkdir xively
pi@pivalter ~ $ cd xively
```

```

pi@pivalter ~/xively $ git init
Initialized empty Git repository in /home/pi/xively/.git/
pi@pivalter ~/quick2wire $ git clone https://github.com/xively/xively-python.git
Cloning into xively-python'...
...
Resolving deltas: 100% (939/939), done.
pi@pivalter ~/xively $ cd xively-python
pi@pivalter ~/xively/xively-python $ python3 setup.py install --prefix $HOME/pyth

```

Prima di scrivere il nostro primo programma che utilizza **Xively** dobbiamo scoprire come recuperare le informazioni che ci servono, cioè la temperatura di CPU e GPU. I due sensori espongono, purtroppo, le informazioni in formato diverso, ma nulla di troppo complicato.

La temperatura della CPU può essere letta nel file virtuale `/sys/class/thermal/thermal_zone0/temp`. Questo file è gestito dal kernel e consente di leggere in tempo reale la temperatura utilizzando il sensore interno della CPU:

```

pi@pivalter ~ $ cat /sys/class/thermal/thermal_zone0/temp
46540

```

La temperatura in questo caso è di 46,54 °C (il valore è in millesimi di grado Celsius).

Per leggere la temperatura della GPU il procedimento è leggermente più complicato. È necessario lanciare l'utility `/opt/vc/bin/vcgencmd`:

```

pi@pivalter ~ $ /opt/vc/bin/vcgencmd measure_temp
temp=46.0'C

```

In questo caso il dato ci sarà ritornato direttamente in gradi Celsius.

Ora che sappiamo come leggere la temperatura dai nostri due sensori, possiamo implementare un piccolo programma Python per inviare questi dati a Xively.

Ormai dovremmo avere familiarità con Python, quindi gli spunti interessanti del nostro programma sono evidenziati nel codice usando dei commenti.

```

#!/usr/bin/env python3

# importa le librerie necessarie (in particolare Xively!)
import xively
import time
import subprocess
import datetime

# legge la temperatura della CPU
def leggiTemperaturaCPU():

```

```

# apre il file, ne legge il contenuto, lo converte in intero e
# lo divide per 1000 per avere un valore in gradi Celsius

file=open("/sys/class/thermal/thermal_zone0/temp")
temperatura=file.read()
file.close()
return int(temperatura)/1000

# legge la temperatura della GPU
def leggiTemperaturaGPU():
    # in questo caso dobbiamo catturare l'output su console
    # la funzione subprocess.check_output() ci consente di farlo
    # memorizzando il valore in una stringa
    output=subprocess.check_output(["/opt/vc/bin/vcgencmd","measure_temp"])
    # eliminiamo le parti che non ci servono (temp= e l'unita' di misura)
    temperatura=output[5:-3]
    return float(temperatura)

# qui dovete sostituire i valori che avete ottenuto registrandovi
# come sviluppatori sul sito Xively
API_KEY="*****"
FEED_ID="*****"

# l'oggetto XivelyApiClient ci consente di interfacciarsi con Xively
api=xively.XivelyApiClient(API_KEY)

# recuperiamo il nostro feed
feed=api.feeds.get(FEED_ID)

# loop infinito
while True:
    # recuperiamo i datastream relativi ai canali che abbiamo
    # creato per il nostro dispositivo
    # (I nomi dei datastream sono case sensitive!)
    temperaturacpu=feed.datastreams.get("TemperaturaCPU")
    temperaturagpu=feed.datastreams.get("TemperaturaGPU")

    # impostiamo il valore della temperatura corrente
    # e la data e ora corrente
    temperaturacpu.current_value=leggiTemperaturaCPU()
    temperaturacpu.at=datetime.datetime.utcnow()

    # debug
    print("Temperatura CPU letta : " + str(temperaturacpu.current_value))

    # aggiornando il datastream i dati saranno inviati a Xively
    temperaturacpu.update()

    # leggiamo anche la temperatura della GPU
    temperaturagpu.current_value=leggiTemperaturaGPU()
    temperaturagpu.at=datetime.datetime.utcnow()

    print("Temperatura GPU aggiornata : " + str(temperaturagpu.current_value))

    # e aggiorniamo il datastream corrispondente
    temperaturagpu.update()

```

```
# aspettiamo un minuto prima di aggiornare i dati
time.sleep(60)
```

Ora, dopo aver verificato di essere connessi a internet, possiamo lanciare il nostro programma e questo stamperà ogni minuto la temperatura corrente di CPU e GPU.

```
pi@pivalter ~ $ ./xivelytest.py
Temperatura CPU letta : 46.54
Temperatura GPU aggiornata : 47.1
```

La cosa interessante, però, è tornare sul sito Xively e riaprire la pagina relativa al nostro device.

Nella finestra **Request Log** verranno mostrate tutte le operazioni effettuate dal device e i dati dei due canali si popoleranno a mano a mano con i dati letti dal nostro device!

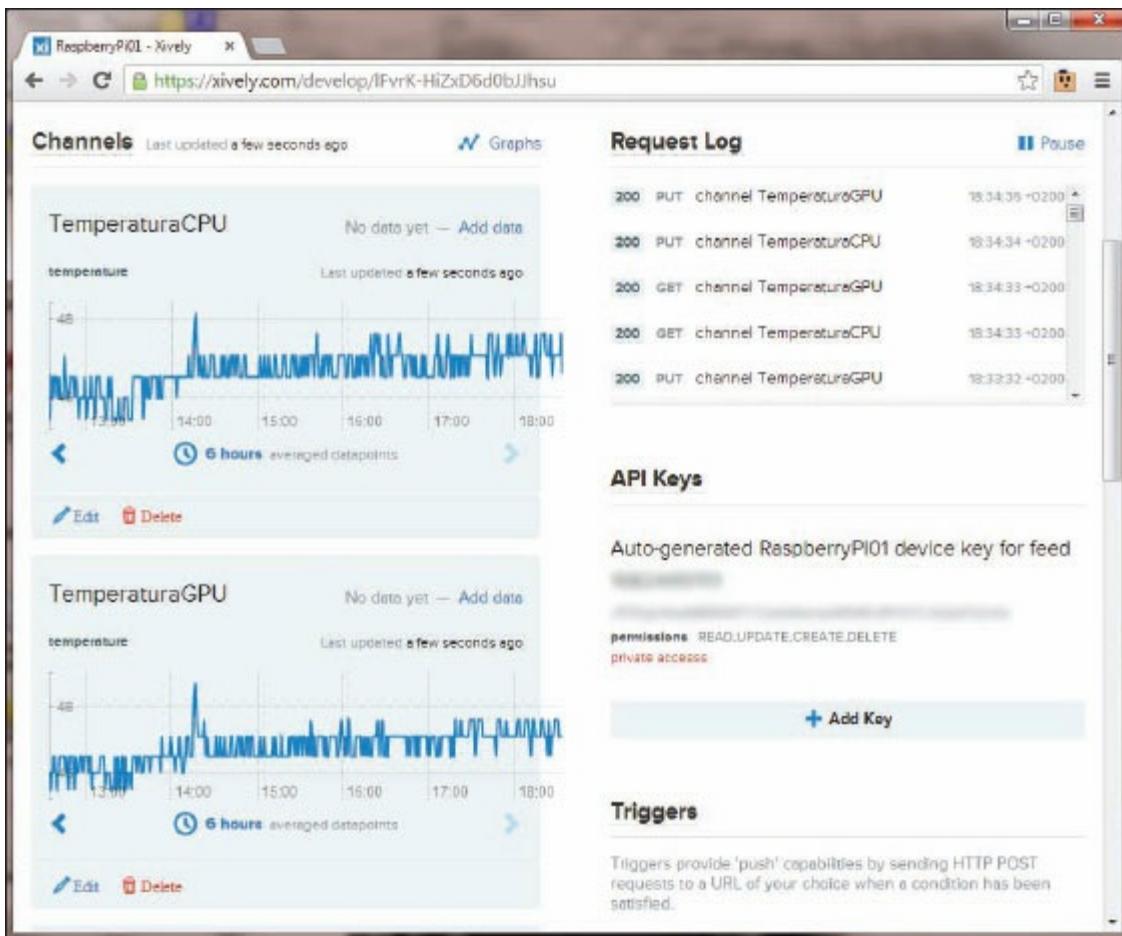


Figura 8.12 - Xively - acquisizione dati.

Il nostro Raspberry Pi sta pubblicando informazioni nella rete delle reti. Lo fa comunicando attraverso il protocollo HTTP, lo stesso che utilizza il nostro browser quando navighiamo su internet. Le funzioni di Xively sono accessibili tramite richieste HTTP e questo ne rende molto

semplice l'utilizzo con diversi strumenti di programmazione, non soltanto con Python. Lo stesso vale per le API di DropBox e di moltissimi altri servizi cloud-based.

Il nostro dispositivo viene però ancora visto come in fase di sviluppo. Per poter collegare più dispositivi dovremo passare alla fase di "produzione". Per farlo possiamo selezionare l'opzione **Management** della finestra Web tools e scegliere l'opzione **Add Product Batch** che ci permetterà di creare un "lotto" di oggetti dello stesso tipo.

Questo consentirà di configurare uno o più dispositivi che invieranno i dati separatamente.

Possiamo inserire un nome e una descrizione per il nostro batch e poi crearlo selezionando **Add Batch**.

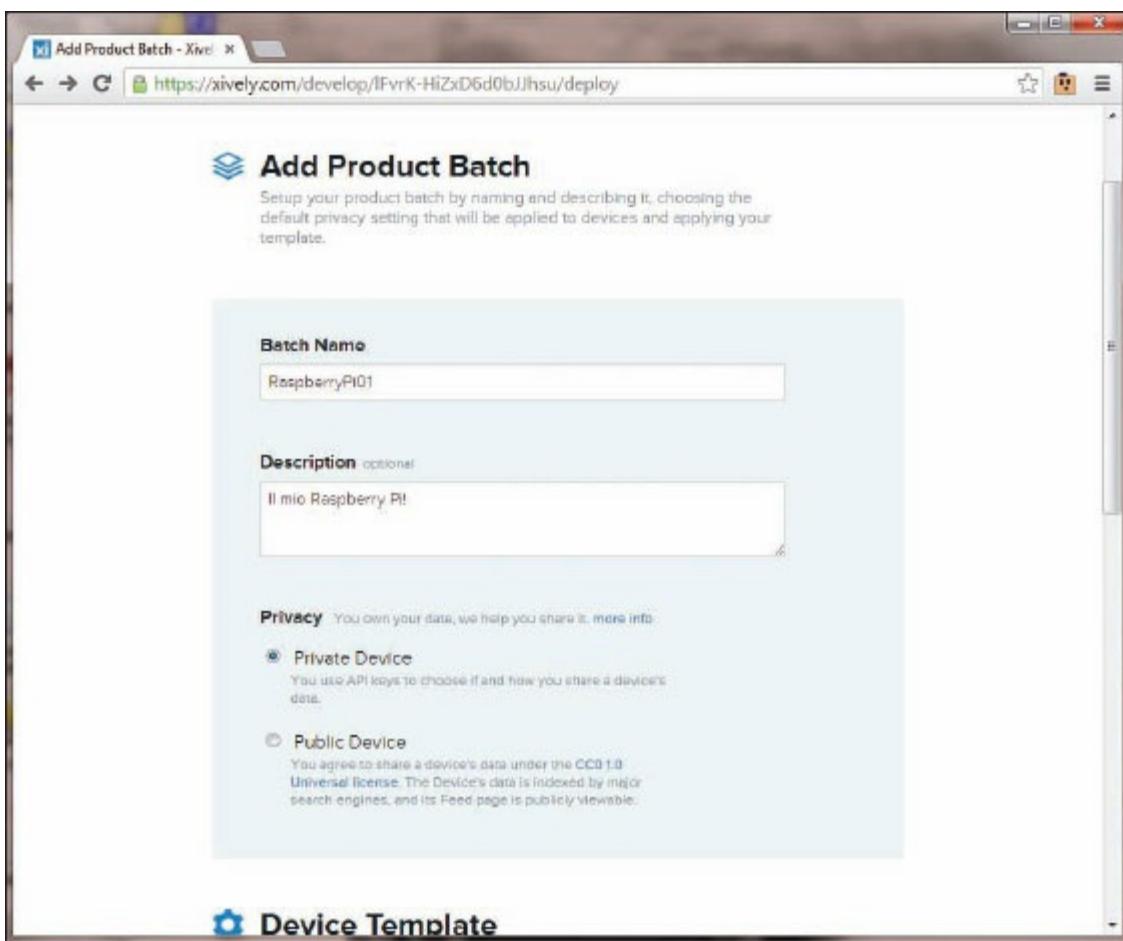


Figura 8.13 - Xively - creazione di un lotto di dispositivi.

A questo punto il nostro dispositivo si vedrà assegnare un ID di prodotto e una chiave (**Product Secret**). Questa informazione ci servirà per modificare il nostro programma di prova in modo da farlo girare su diversi dispositivi.

Ogni dispositivo dovrà avere un numero di serie. I numeri di serie possono essere assegnati selezionando **Add Serial Numbers**.

È possibile caricare più numeri di serie contemporaneamente eseguendo l'upload di un file, ma nel nostro caso, visto il numero ridotto di device, possiamo selezionare l'opzione **Manually** e inserire il nostro numero di serie (001).

Una volta selezionato **Add Serial Numbers**, il nostro dispositivo verrà mostrato in elenco ma marcato come non ancora attivato. Lo dovremo attivare modificando il programma originale per farlo funzionare con un dispositivo di produzione.

Le modifiche al codice sono evidenziate nel listato qui di seguito e riguardano principalmente l'attivazione del dispositivo.

```
#! /usr/bin/env python3

# importa le librerie necessarie (in particolare Xively!) import xively
import time
import subprocess
import datetime

# legge la temperatura della CPU
def leggiTemperaturaCPU():
    # apre il file, ne legge il contenuto, lo converte in intero e
    # lo divide per 1000 per avere un valore in gradi Celsius

    file=open("/sys/class/thermal/thermal_zone0/temp")
    temperatura=file.read()
    file.close()
    return int(temperatura)/1000

# legge la temperatura della GPU
def leggiTemperaturaGPU():
    # in questo caso dobbiamo catturare l'output su console
    # la funzione subprocess.check_output() ci consente di farlo
    # memorizzando il valore in una stringa
    output=subprocess.check_output(["/opt/vc/bin/vcgencmd","measure_temp"])
    # eliminiamo le parti che non ci servono (temp= e l'unita' di misura)
    temperatura=output[5:-3]
    return float(temperatura)

# ora il nostro prodotto avrà un numero di serie e una chiave
# la chiave dovrà essere quella creata da Xively quando avete
# aggiunto il lotto di dispositivi
# notate che le due stringhe hanno il prefisso b prima dei doppi
# apici questo farà sì che Python le consideri come sequenze di
# byte e non come normali stringhe
SERIAL_NUMBER=b"001"
PRODUCT_SECRET=b"*****"
# dovremo memorizzare le credenziali in un file di configurazione
FILE_CONFIGURAZIONE="xively_conf"

# prima di usare le API è necessario registrare il prodotto
# se il prodotto è già attivato le credenziali saranno nel memorizzate nel
# file di configurazione
```

```

credenziali=""

# se il file di configurazione non esiste e' necessario attivare il dispositivo
if not os.path.isfile(FILE_CONFIGURAZIONE):
    # converte il product secret in formato esadecimale
    productsecret=binascii.a2b_hex(PRODUCT_SECRET)
    # queste funzioni della libreria Python generano una "firma" derivata da
    # codice segreto del prodotto e numero di serie e la convertono in una
    # stringa di caratteri esadecimali
    codiceattivazione=hmac.new(productsecret, SERIAL_NUMBER, hashlib.sha1).

    # il prodotto andrà registrato inviando una richiesta HTTP al sito xively
    # l'indirizzo (url) conterrà la chiave generata nelle righe precedenti
    connessione=http.client.HTTPConnection("api.xively.com")
    url=str("/v2/devices/"+codiceattivazione+"/activate")
    connessione.request("GET",url)

    # il server ci risponderà inviandoci le credenziali da utilizzare per accedere
    # a xively dal nostro dispositivo. Queste vengono tradotte in una stringa
    # la funzione decode() e salvate nel file di configurazione
    credenziali=connessione.getresponse().read().decode("utf-8")
    connessione.close()

    file=open(FILE_CONFIGURAZIONE,"w")
    file.write(credenziali)
    file.close()
else:
    # se l'attivazione è già stata fatta ci basta rileggere le credenziali
    file=open(FILE_CONFIGURAZIONE,"r")
    credenziali=file.read()
    file.close()

# le credenziali sono state ritornate in fase di attivazione, non serve più
# la chiave per le API, le informazioni sono in formato json (un formato molto
# usato in ambito web), dobbiamo interpretarle e Python ci aiuta a farlo con la
# funzione json.loads() che converte la stringa caricata dal file in un
# dizionario
json=json.loads(credenziali)
api=xively.XivelyAPIClient(json["apikey"])

feed=api.feeds.get(json["feed_id"])

# loop infinito
while True:
    # recuperiamo i datastream relativi ai canali che abbiamo
    # creato per il nostro dispositivo
    #(I nomi dei datastream sono case sensitive!)
    temperaturacpu=feed.datastreams.get("TemperaturaCPU")
    temperaturagpu=feed.datastreams.get("TemperaturaGPU")

    # impostiamo il valore della temperatura corrente
    # e la data e ora corrente
    temperaturacpu.current_value=leggiTemperaturaCPU()
    temperaturacpu.at=datetime.datetime.utcnow()

    # debug
    print("Temperatura CPU letta : " + str(temperaturacpu.current_value))

```

```

# aggiornando il datastream i dati saranno inviati a Xively
temperaturacpu.update()

# leggiamo anche la temperatura della GPU
temperaturagpu.current_value=leggiTemperaturaGPU()
temperaturagpu.at=datetime.datetime.utcnow()

print("Temperatura GPU aggiornata : " + str(temperaturagpu.current_value))

# e aggiorniamo il datastream corrispondente
temperaturagpu.update()

# aspettiamo un minuto prima di aggiornare i dati
time.sleep(60)

```

Come potete vedere confrontando i due listati la parte principale del programma, il loop di lettura dei sensori e di aggiornamento delle informazioni su Xively, non cambia. Quello che cambia è la fase di autenticazione del nostro programma, che ora non userà solo la chiave per l'accesso alle API ma sarà associato anche a un numero di serie e quindi a un dispositivo preciso. Se vogliamo confrontare la temperatura del nostro Raspberry Pi con quella del Pi di un amico, basterà passargli il programma cambiando il numero di serie dopo aver registrato un nuovo numero di serie sul sito.

Una volta lanciato il programma modificato, l'output su console non cambierà:

```

pi@pivalter ~ $ ./xivelytest.py
Temperatura CPU letta : 46.54
Temperatura GPU aggiornata : 47.1

```

Se però ricarichiamo la pagina di configurazione il nostro dispositivo verrà marcato come attivato (con la data di attivazione) e, selezionandolo, potremo andare a vedere i suoi dati.

Impostando il dispositivo come pubblico potremo condividere la pagina con i grafici e i dati raccolti.

Xively consente di fare anche molto altro: leggere i dati di un dispositivo, generare richieste a un server quando i valori di un canale o di più canali superano determinate soglie (in positivo o in negativo) ecc.

Esistono anche molti altri servizi simili a Xively, ognuno con il suo set di API, spesso disponibili gratuitamente, perlomeno per la sperimentazione.

In rete, sia sul sito Xively sia sui blog di tanti appassionati, troverete diversi esempi interessanti per continuare a esplorare l'interazione tra il vostro piccolo personal computer e... le nuvole!

The screenshot shows a web browser window for the Xively platform. The URL is <https://xively.com/manage/KYIHMOJytkN82FnYup5t>. The top navigation bar includes links for PLATFORM, SERVICES, ECOSYSTEM, DEVELOPER CENTER, WEB TOOLS, and user account information (valterraspberry, Settings, Logout). A search bar and a 'Add Serial Numbers' button are also present.

The main content area is titled "RaspberryPi01". It shows a "Private Product" with the following details:

- Product ID: KYIHMOJytkN82FnYup5t
- Product Secret: 647c8f61272816982b00db017224829d69f3f09

A section titled "Devices" shows 1 of 1 devices, 100% activated. The table lists the device information:

Serial Number	Activated	Date Added
001	✓ 2013-09-27 09:26:07	2013-09-25

Figura 8.14 - Xively - elenco dei dispositivi; il nostro dispositivo risulta attivato.

Che tempo fa Raspberry Pi?

Dopo aver fatto un bel po' di **esperimenti**, passiamo alla **pratica** e realizziamo una piccola **stazione meteo** con il nostro Raspberry Pi. Forse non ci aiuterà a diventare famosi e infallibili come il colonnello **Edmondo Bernacca**, ma impareremo ancora meglio a utilizzare il nostro piccolo personal computer.

Chi, come l'autore, ha passato la quarantina, ricorda sicuramente le previsioni del tempo del colonnello Edmondo Bernacca. In pochi minuti prima del telegiornale un signore dai modi antichi spiegava, destreggiandosi tra anticlioni e millibar, cosa avremmo dovuto aspettarci in termini meteorologici dalla giornata seguente. Non azzeccava sempre le previsioni, ma non perse mai la stima e il rispetto del pubblico perché le sue spiegazioni erano chiare e comprensibili. Questo consentì a una generazione di italiani di prendere dimestichezza con i concetti base di una scienza che prima era solo per iniziati.

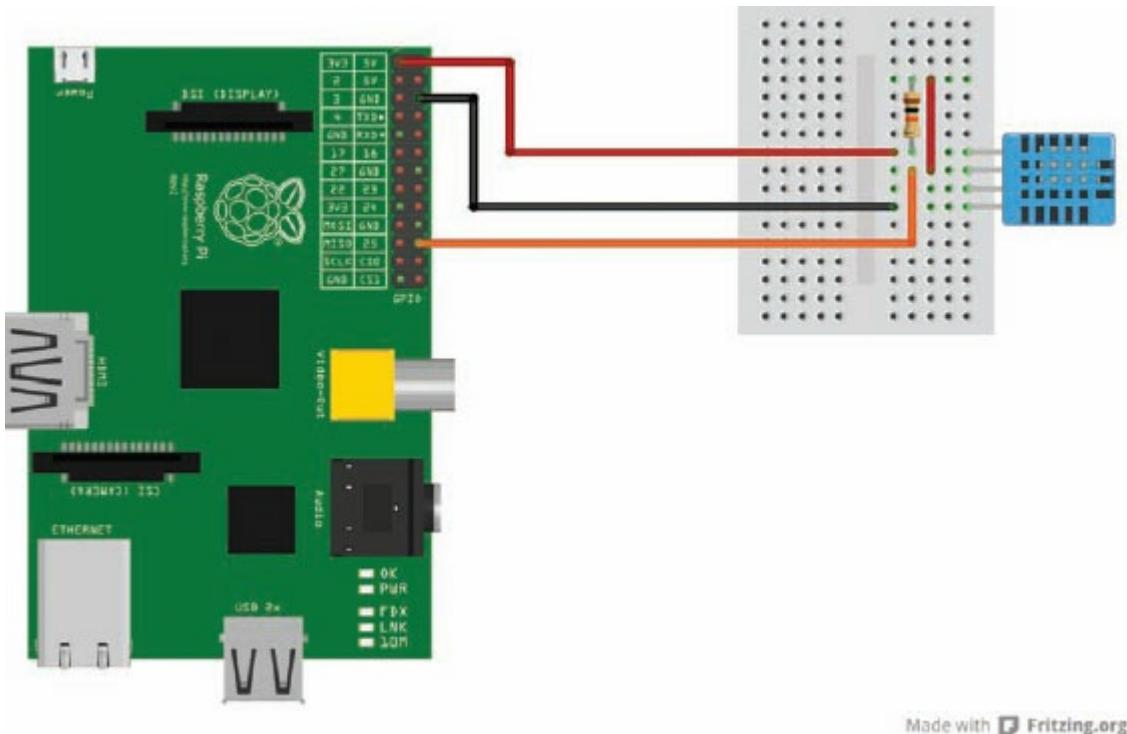
Negli anni '70 la meteorologia non si poteva avvalere dei satelliti e questo rendeva il lavoro dei meteorologi assai complicato, e le loro previsioni non sempre azzeccate.

Anche noi non possiamo avvalerci di un satellite collegato al nostro

Raspberry Pi ma, fortunatamente, la tecnologia moderna ci consente di avere sensori meteo abbastanza precisi a costi molto abbordabili. La nostra stazione meteo memorizzerà i suoi dati in un database, per consentirci di rivedere i dati storici, e li pubblicherà in rete. Ma partiamo, come al solito, dalle basi, cioè dall'hardware che ci servirà per realizzare la nostra stazione meteo.

Il sensore

Esistono molte tipologie di sensori che consentono di misurare temperatura, umidità, pressione, velocità del vento ecc. Nel nostro esperimento ci limiteremo alle prime due grandezze e utilizzeremo un sensore molto comune, il DHT11. Questo sensore, analogamente al "fratellone" DHT22 che è più preciso e ha un range di temperature più esteso, ma anche un costo maggiore, è in grado di leggere i dati e tradurli in formato digitale. Non servirà quindi un convertitore analogico-digitale. Il sensore utilizza un protocollo particolare e non è compatibile con I2C. Ma vedremo come installare e utilizzare delle librerie che consentiranno di utilizzarlo in modo semplice tramite Python. Per prima cosa dobbiamo collegare il sensore. Oltre al sensore ci servirà anche una resistenza da $10\text{ K}\Omega$ da collegare tra l'alimentazione e il pin usato per i dati. Il sensore DHT11 ha quattro pin, ma solo tre sono utilizzati. Due per alimentazione (3,3 V e GND) e uno per i dati. Lo schema di collegamento di [Figura 9.1](#) è equivalente per DHT11 e DHT22. I cavi di connessione possono essere anche abbastanza lunghi, fino a una decina di metri, e questo ci consentirà di posizionare il sensore all'esterno lasciando il nostro Raspberry Pi al calduccio e al sicuro dentro casa.



Made with Fritzing.org

Figura 9.1 - Schema di collegamento del sensore DHT11.

Una volta collegato il sensore possiamo passare a gestirlo dal nostro codice in Python. Per farlo dobbiamo però installare un'applicazione sviluppata in linguaggio C. Il protocollo DHT richiede tempi precisi e non è possibile garantirli controllando i pin tramite Python.

Il linguaggio C è più complesso di Python ma, come potete vedere da questo esempio, consente di realizzare applicazioni o librerie che non è possibile implementare con un linguaggio interpretato. Scendere nei dettagli del C va molto oltre le prerogative di questo testo ma, se vi farete tentare dal “lato oscuro” e deciderete di sperimentare anche con altri linguaggi oltre a Python, Raspberry Pi è un’ottima palestra anche per il C e il suo “cugino” C++.

Per gestire il protocollo DHT11 ci servirà un’utility sviluppata da Adafruit (www.adafruit.com) che, tra le altre cose, sviluppa e vende moltissimi accessori interessanti per Raspberry Pi, già menzionati nei capitoli precedenti.

Per scaricare il codice di Adafruit useremo `git`.

Per prima cosa torniamo nella cartella `home`, creiamo una nuova cartella e un nuovo repository `git`:

```
pi@pivalter ~ /bcm2835/bcm2835-1.29 $ cd
pi@pivalter ~ $ mkdir Adafruit
pi@pivalter ~ $ cd Adafruit
pi@pivalter ~/Adafruit $ git init
Initialized empty Git repository in /home/pi/Adafruit/.git/
```

Ora possiamo, come già fatto per altre librerie in precedenza, lanciare `git clone` e scaricare il codice:

```
pi@pivalter ~/Adafruit $ git clone https://github.com/adafruit/Adafruit-Raspberry  
Pi-Python-Code.git  
Cloning into 'Adafruit-Raspberry-Pi-Python-Code'...  
remote: Counting objects: 449, done.  
remote: Compressing objects: 100% (283/283), done.  
remote: Total 449 (delta 213), reused 347 (delta 145)  
Receiving objects: 100% (449/449), 131.58 KiB | 128 KiB/s, done.  
Resolving deltas: 100% (213/213), done.
```

Il programma che ci interessa è nella cartella `Adafruit-Raspberry-Pi-Python-Code/Adafruit_DHT_Driver/` e possiamo usare il comando `cd` per entrarci:

```
pi@pivalter ~/Adafruit $ cd Adafruit-Raspberry-Pi-Python-Code/Adafruit_DHT_Driver
```

Per provare il nostro sensore possiamo lanciare l'applicazione `Adafruit_DHT` che troviamo già compilata nella cartella con i sorgenti. Per consentirle di leggere i dati dovremo passare sulla command line il tipo di sensore (¹¹ per DHT11, ²² per DHT22) e il pin di GPIO a cui abbiamo collegato il segnale in arrivo dal sensore (²⁵ nel nostro caso):

```
pi@pivalter ~/Adafruit/Adafruit-Raspberry-Pi-Python-Code/Adafruit_DHT_Driver $ su  
. ./Adafruit_DHT 11 25  
Using pin #25  
Data (40): 0x26 0x0 0x19 0x0 0x3f  
Temp = 25 *C, Hum = 38 %
```

Se il programma ritorna le informazioni senza errori, allora possiamo proseguire. Se invece si è verificato un errore e il programma non ritorna i dati, verificate bene le connessioni tra il Raspberry Pi e il vostro sensore!

Come avete visto, abbiamo dovuto utilizzare `sudo` per poter lanciare l'applicazione. Questo perché il codice dell'applicazione va ad accedere all'hardware di Raspberry Pi e questo è consentito solo al super-user.

Nei sorgenti di Adafruit c'è anche una libreria per consentire al codice Python di accedere al sensore DHT11. Purtroppo, al momento della stesura di questo testo, la libreria non è stata aggiornata per supportare Python 3. Useremo la tecnica già vista nel capitolo precedente per leggere i dati del sensore DHT, lanciando l'utility che abbiamo appena utilizzato per provarne il funzionamento.

Per comodità possiamo copiarla nella cartella home:

```
pi@pivalter ~/Adafruit/Adafruit-Raspberry-Pi-Python-Code/Adafruit_DHT_Driver $ cp  
Adafruit_DHT $HOME
```

Ora possiamo implementare la funzione che lancia l'utility, ne cattura l'output e lo interpreta. Per poterla poi riutilizzare più semplicemente nel nostro codice, la salveremo un file di nome DHT.py.

Come al solito i commenti nel codice ci aiuteranno a capire meglio le operazioni eseguite.

```
import subprocess

# percorso dell'utility da lanciare
ADAFRUIT_DHT="/home/pi/Adafruit_DHT"

# la funzione legge i dati dal sensore DHT usando
# l'utility di Adafruit
# parametri:
# tipo - tipo di sensore 11 = DHT11 22 = DHT22
# pin - pin di GPIO a cui e' collegato il sensore
# ritorna tre valori:
# booleano - a True se i dati sono validi
# float - temperatura
# float - umidita
def leggiSensore(tipo,pin):

    # lancia l'utility usando sudo e passando come parametri tipo e pin
    outputbytes=subprocess.check_output(["sudo",ADAFRUIT_DHT,str(tipo),str(pi)

        # l'output viene ritornato come sequenza di byte
        # la funzione decode ci consente di tradurlo in

        # una stringa
    output=outputbytes.decode("utf-8")

    # la funzione find() dell'oggetto stringa viene utilizzata per
    # ricavare il valore della temperatura
    indice=output.find("Temp = ")
    indice2=output.find(" *C")

    # se non sono state trovate le stringhe l'utility non è
    # riuscita a leggere dati validi
    if indice===-1 or indice2===-1:
        return False,0.0,0.0

    # ricava la temperatura come sotto-stringa e la converte
    # in un valore a virgola mobile
    temperatura=float(output[indice+7:indice2])

    # cerca il valore di umidita'
    indice=output.find("Hum = ")

    # controlla che l'output sia corretto
    if indice===-1:
```

```

        return False,0.0,0.0

# ricava anche l'umidita' e la converte in
# numero a virgola mobile
umidita=float(output[indice+6:-2])

# il valore True indica che i parametri sono validi
return True, temperatura, umidita

```

Ora possiamo lanciare l’interprete e provare a leggere i dati usando Python.

```

>>> import DHT
>>> print(DHT.leggiSensore(11,25))
(True, 25.0, 38.0)

```

Possiamo quindi “parcheggiare” per un attimo il nostro sensore e scoprire come possiamo memorizzare i dati che leggeremo, in modo da poterli poi rapidamente recuperare e consultare. Per farlo ci serve un database.

Database

Esistono tantissimi sistemi database, alcuni free e molti altri proprietari. Ci sono database in grado di gestire miliardi di informazioni e milioni di richieste al secondo, ma per il nostro progetto possiamo accontentarci di qualcosa di più semplice. Utilizzeremo SQLite. Si tratta di un motore database molto utilizzato e che, con un utilizzo di risorse ridotto, consente di gestire basi di dati abbastanza grandi in modo semplice. Nel nostro caso i dati si limiteranno alle informazioni di temperatura e umidità collegate all’orario in cui sono state lette.

Structured Query Language (SQL) è un linguaggio che serve per compiere operazioni sui database. È standardizzato e molti sistemi database lo utilizzano: Oracle, Microsoft SQL Server, MySQL ecc. Come si può intuire dal nome, anche SQLite utilizza questo linguaggio e, come gli altri database citati, è basato sul modello relazionale. I database sono normalmente organizzati in tabelle. Ogni tabella prevede dei campi che definiscono gli elementi base che costituiscono ogni elemento (record) memorizzato. Se, per esempio, volessimo memorizzare i contatti telefonici dei nostri amici potremmo avere una tabella di questo tipo:

Nome	Cognome	Numero
------	---------	--------

Will	Coyote	1234
Bip	Bip	5678
Bugs	Bunny	9012

La nostra tabella in questo esempio ha tre campi (Nome, Cognome e Numero) e tre record, corrispondenti a ognuno dei nostri amici.

Proviamo a “tradurla” in una tabella con SQLite.

Per prima cosa dovremo installare SQLite usando `apt-get`, come già fatto per altri componenti nei capitoli precedenti:

```
pi@pivalter ~ $ sudo apt-get install sqlite3
Lettura elenco dei pacchetti... Fatto
...
Configurazione di sqlite3 (3.7.13-1+deb7u1) ...
```

Una volta installato, possiamo lanciare SQLite e interagire usando il suo prompt; scopriremo poi come fare a utilizzarlo attraverso Python.

Passando il nome di un database come argomento a `sqlite3` il database verrà aperto o, come nel nostro caso, creato se non esiste ancora.

```
pi@pivalter ~ $ sqlite3 contatti.db
SQLite version 3.7.13 2012-06-11 02:05:22
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
```

Per prima cosa possiamo provare a creare una tabella. Il comando SQL da utilizzare è `CREATE TABLE`:

```
sqlite> CREATE TABLE CONTATTI(NOME TEXT, COGNOME TEXT, NUMERO TEXT);
sqlite>
```

Il linguaggio SQL in SQLite è case-insensitive, ossia l’uso di maiuscole o minuscole nei nomi e nei comandi è equivalente. Noi useremo comandi e nomi in caratteri maiuscoli perché questa è la convenzione che si vede utilizzata più spesso negli esempi e perché le keyword e la maggior parte dei nomi di oggetti e funzioni Python usa invece le lettere minuscole; questo ci permetterà quindi di distinguere più facilmente il codice SQL nei nostri programmi.

Le istruzioni SQL devono essere terminate con il carattere punto e virgola.

Per creare la tabella dovremo fornire a `CREATE TABLE` un nome e poi l’elenco dei suoi campi specificando per ognuno un tipo di dato. Nel

caso del nostro semplice elenco di contatti tutti i campi sono di tipo `TEXT`, equivalenti alle stringhe di Python, ma SQLite gestisce anche dati numerici (interi e a virgola mobile) o dati complessi come le date e gli orari.

Una volta creata la tabella possiamo inserire dei dati usando l'istruzione `INSERT`.

```
sqlite> INSERT INTO CONTATTI VALUES("Will","Coyote","1234");
sqlite> INSERT INTO CONTATTI VALUES("Bip","Bip","5678");
sqlite> INSERT INTO CONTATTI VALUES("Bugs","Bunny","9012");
sqlite>
```

Nell'istruzione `INSERT` dobbiamo specificare la tabella in cui inserire i dati (dopo la keyword `INTO`) e poi, mediante la keyword `VALUES` e le parentesi, i valori dei vari campi del nostro nuovo record.

Per rivedere i dati inseriti possiamo usare l'istruzione `SELECT`.

```
sqlite> SELECT * FROM CONTATTI;
Will|Coyote|1234
Bip|Bip|5678
Bugs|Bunny|9012
sqlite>
```

L'istruzione `SELECT` richiamata con il parametro `*` ritornerà tutti i campi della tabella (è possibile far ritornare solo una parte dei campi specificandone i nomi separati da virgolette dopo `SELECT`) specificata dopo la keyword `FROM`. Mediante la keyword `ORDER BY` possiamo ricavare i nostri dati ordinati in base a uno dei campi:

```
sqlite> SELECT * FROM CONTATTI ORDER BY COGNOME;
Bip|Bip|5678
Bugs|Bunny|9012
Will|Coyote|1234
```

La keyword `WHERE`, invece, ci permetterà di ottenere i dati filtrandoli in base a una serie di criteri di ricerca:

```
sqlite> SELECT NOME,NUMERO FROM CONTATTI WHERE COGNOME="Coyote";
Will|1234
```

SQLite permette anche di mettere in relazione tra loro diverse tabelle (da cui la definizione di database relazionale) ed eseguire ricerche su strutture dati anche molto complesse.

Per la nostra stazione meteo non avremo bisogno di sfruttare tutte queste potenzialità. Ci basterà essere in grado di inserire dati e ricercarli

su una singola tabella. Ma se l'argomento database vi interessa, cosa è meglio di SQLite e Raspberry Pi per sperimentare!

Ora possiamo creare un database per la nostra stazione meteo. Per farlo dovremo terminare SQLite e rilanciarlo con un nuovo database.

```
sqlite> .quit  
pi@pivalter ~ $ sqlite3 meteo.db  
SQLite version 3.7.13 2012-06-11 02:05:22  
Enter ".help" for instructions  
Enter SQL statements terminated with a ";"
```

Notate che il comando `quit` non fa parte del linguaggio SQL; viene quindi prefissato con un carattere punto e non terminato con il punto e virgola.

Ora possiamo creare una tabella in cui memorizzeremo le temperature che verranno lette dal sensore. Questo ci consentirà di memorizzare i nostri dati in modo permanente e di poterli poi ordinare e ricercare sfruttando i comandi SQL.

```
sqlite> CREATE TABLE VALORI (DATA DATETIME, TEMPERATURA REAL, UMIDITA REAL, TRASMESS  
sqlite> .quit
```

A questo punto siamo pronti per utilizzare il nostro database da un programma Python. Per farlo useremo il modulo `sqlite3`. Possiamo iniziare a impostare la struttura della nostra applicazione meteo salvando il codice seguente nel file `meteo.py` e rendendolo eseguibile con `chmod`.

```
#! /usr/bin/env python3  
  
import DHT  
import sqlite3 as sqlite  
import time  
import datetime  
  
# apre il database e ottiene un oggetto connessione  
connessione=sqlite.connect("meteo.db")  
  
# l'uso della keyword with ci assicura che la connessione verrà  
# chiusa anche in caso di eccezioni  
with connessione:  
  
    # loop infinito  
    while True:  
  
        # legge i dati dal sensore  
        dativalidi, temperatura, umidita = DHT.leggiSensore(11,25)  
  
        # se i dati non sono validi verrà stampato
```

```

# un messaggio di errore
if not dativalidi:
    print ("Errore nella lettura dei dati")
else:
    # per eseguire i comandi SQL dobbiamo usare un
    # oggetto cursore che sara' ritornato dalla funzione
    # cursor() dell'oggetto connessione
    cursole=connessione.cursor()

    # nel comando SQL possiamo inserire dei ? che verranno
    # poi sostituiti con i parametri passati a execute
    comandosql="INSERT INTO VALORI VALUES(?, ?, ?, 0)"

    # la funzione execute viene richiamata passando i
    # parametri nello stesso ordine in cui devono essere
    # inseriti nel comando SQL
    cursole.execute(comandosql, (datetime.datetime.utcnow(),
                                  temperatura,
                                  umidita))
    # la chiamata a commit() fa in modo che i dati
    # vengano salvati in modo permanente
    connessione.commit()

    print("Dati salvati - temperatura: "+str(temperatura)+""
umidita: " + str(umidita))

    # aspetta un minuto prima di fare una nuova lettura
    time.sleep(60)

```

Per utilizzare le funzioni di SQLite è necessario, per prima cosa, aprire una connessione al database, specificando il file da utilizzare.

I comandi SQL dovranno poi essere eseguiti su un oggetto cursore che viene ritornato dalla funzione `cursor()` dell'oggetto che gestisce la connessione.

Il cursore ci consentirà di eseguire dei comandi richiamandone il metodo `execute()` a cui passeremo il comando SQL (senza il punto e virgola finale). Visto che spesso questi comandi conterranno i valori delle nostre variabili, potremo indicarli con dei punti di domanda nel testo del comando e poi passare alla funzione `execute()` un tuple con i valori da sostituire (per dichiarare un tuple basta inserire i valori tra parentesi tonde invece delle quadre usate per le liste).

Lanciando il nostro programma vedremo, ogni minuto, i valori di temperatura letti:

```

pi@pivalter ~ $ ./meteo.py
sudo: unable to resolve host pivalter
Dati salvati - temperatura: 25.0 umidita: 38.0
sudo: unable to resolve host pivalter
Dati salvati - temperatura: 25.0 umidita: 39.0

```

```
sudo: unable to resolve host pivalter
Dati salvati - temperatura: 24.0 umidita: 40.0
```

Può capitare che ogni tanto la funzione di lettura dei dati dal sensore DHT fallisca: le tempistiche del protocollo sono molto strette e anche il codice C potrebbe sporadicamente non riuscire a leggere correttamente i dati.

Dopo qualche minuto possiamo interrompere l'esecuzione del programma premendo **Ctrl+C** e rilanciare SQLite per vedere che i nostri dati siano stati inseriti correttamente:

```
pi@pivalter ~ $ sqlite3 meteo.db
SQLite version 3.7.13 2012-06-11 02:05:22
Enter ".help" for instructions
Enter SQL statements terminated with a ";""
sqlite>SELECT * FROM VALORI;
2013-09-28 10:54:23.348532|24.0|40.0|0
2013-09-28 10:55:19.834110|25.0|39.0|0
2013-09-28 10:55:51.072261|25.0|38.0|0
2013-09-28 10:57:00.576244|25.0|39.0|0
2013-09-28 10:58:04.947722|24.0|40.0|0
sqlite>.quit
```

La nostra piccola stazione meteo inizia a prendere forma!

Ma consultare i dati via SQL non è il metodo più user friendly. Sarebbe molto comodo poter vedere questi dati usando il nostro PC o addirittura il nostro smartphone.

Il server HTTP

Il protocollo HTTP è uno dei più importanti tra quelli che fanno funzionare internet.

Consente a un client (un browser, ma anche le nostre applicazioni cloud del capitolo precedente) di inviare richieste a un server per scaricare pagine HTML (il linguaggio usato per sviluppare le pagine web che vengono mostrate dal nostro browser), ma anche dati o contenuti di altro tipo.

Attivando un server HTTP sul nostro Raspberry Pi potremo ricevere richieste da altri dispositivi della nostra rete locale dotati di un browser e questo ci consentirà di visualizzare le informazioni della nostra piccola stazione meteo usando il nostro PC, il nostro smartphone o il nostro tablet.

Anche in questo caso, per semplificarci la vita, possiamo appoggiarci a una libreria già pronta. La libreria Bottle (www.bottlepy.org) consente di

“ospitare” all’interno della propria applicazione Python un web server e di ritornare informazioni via HTTP in modo molto semplice.

Bottle è costituita da un solo file .py che possiamo scaricare nella cartella dei nostri sorgenti (la cartella home nel nostro caso):

```
pi@pivalter ~ $ wget http://bottlepy.org/bottle.py
--2013-09-28 14:29:39-- http://bottlepy.org/bottle.py
Risoluzione di bottlepy.org (bottlepy.org) ... 173.214.207.14
[...]
2013-09-28 14:29:55 (303 KB/s) - "bottle.py" salvato [144217/144217]
```

Implementare un semplice web server con Bottle è facilissimo.

```
#! /usr/bin/env python3

import bottle

@bottle.route("/")
def ciao():
    return "<html><body>Ciao!</body></html>"

bottle.run(host="0.0.0.0",port=8080)
```

Lanciando questo piccolo programma di esempio sul nostro Raspberry Pi vedremo comparire un messaggio, generato da Bottle:

```
Bottle v0.12-dev server starting up (using WSGIRefServer())...
Listening on http://0.0.0.0:8080/
Hit Ctrl-C to quit.
```

Il nostro web server è pronto e in attesa di richieste. Ma cosa succede quando arriva una richiesta?

Per saperlo possiamo aprire il browser e digitare nella barra degli indirizzi l’indirizzo IP del nostro Raspberry Pi (nell’esempio 192.168.99.20) separato con un due punti dal numero di porta (8080).

Il risultato è quello visibile in [Figura 9.2](#).

Non è sicuramente la pagina web più bella che vi è capitato di vedere negli ultimi anni, ma dovete ammettere che lo sforzo necessario per generarla non è stato così improbo! Per capire meglio cosa è successo e come Bottle ha potuto ritornare la pagina al nostro browser possiamo partire guardando la console su cui è in esecuzione il nostro programma.

```
192.168.99.112 - - [28/Sep/2013 15:10:58] "GET / HTTP/1.1" 200 31
```

Il messaggio indica che il server HTTP di Bottle ha ricevuto una richiesta (vedrete l’indirizzo IP del PC o del dispositivo da cui avete

lanciato il browser al posto di 192.168.99.112). La richiesta è di tipo `GET` (la richiesta più semplice e comune supportata dal protocollo HTTP) ed è relativa al percorso root (`/`).

Il server ha risposto con il codice di operazione terminata correttamente (200 nel protocollo HTTP) ritornando 31 byte.

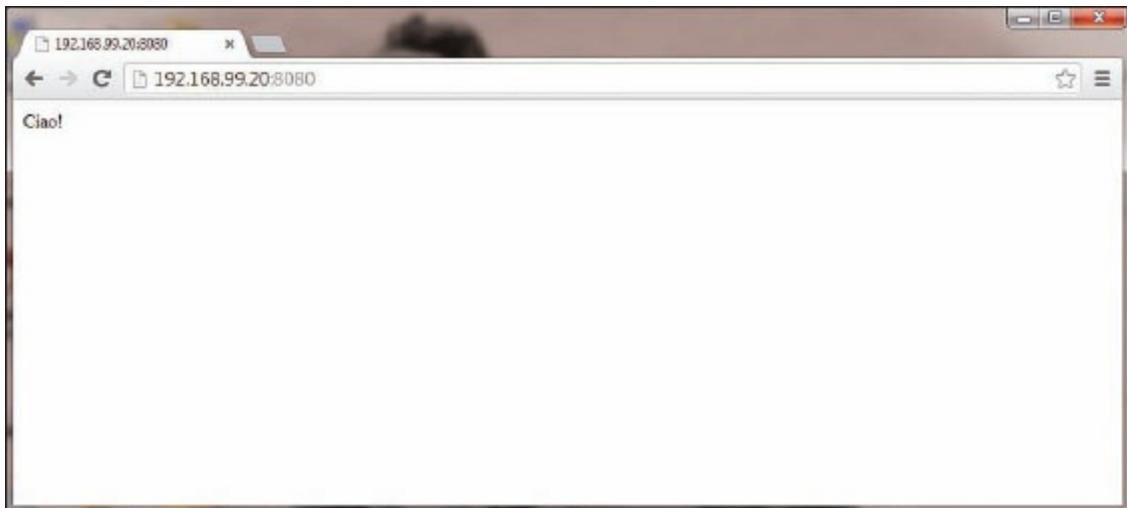


Figura 9.2 - La nostra prima pagina web generata tramite Bottle!

Ora possiamo riguardare il nostro piccolo esempio. Prima della funzione `ciao()` troviamo una riga con una sintassi diversa da quanto visto negli esempi precedenti.

```
@bottle.route("/")
def ciao():
    return "<html><body>Ciao!</body></html>"
```

`@bottle.route` è un decorator. Un decorator è un attributo che può essere agganciato a una funzione o a una classe Python. In questo caso il decorator è utilizzato da Bottle per associare la funzione `ciao()` al percorso `/` del nostro server. Quando un client richiede il percorso `/` Bottle invocherà la nostra funzione e ritornerà al client (il browser) la stringa ritornata dalla funzione stessa. Un meccanismo semplice ma molto potente, che ci consente di usare codice Python per generare dinamicamente le pagine che verranno viste dal nostro utente. Se provate a puntare il browser su un percorso diverso otterrete un errore 404 (risorsa non trovata). Questo succede perché solo il percorso `/` è stato associato a una funzione Python.

La stringa ritornata dalla nostra funzione è codice HTML. Spiegare in dettaglio HTML esula dallo scopo di questo libro, ma quello che ci basta sapere è che in HTML si utilizzano dei tag (le parole tra `<` e `>`) per

identificare varie parti della pagina e anche per inserire elementi di formattazione grafica. Se, per esempio, cambiamo la funzione `ciao()` in questo modo:

```
def ciao():
    return "<html><body><b>Ciao!</b></body></html>"
```

ricaricando la pagina il nostro messaggio verrà mostrato in grassetto.

In rete e in libreria troverete ottimi corsi e manuali su HTML. Qui ci limiteremo a descrivere le funzionalità essenziali per utilizzarlo come interfaccia utente della nostra stazione meteo.

Dover costruire le pagine HTML all'interno del codice non è molto comodo quando le pagine diventano più complesse del nostro semplice esempio. Per questo Bottle consente di creare dei template, mischiando codice Python e codice HTML.

Per applicare i template al nostro esempio dobbiamo creare una cartella `views` all'interno della cartella `home` e creare il nostro primo template.

```
%if valorivalidi:
<p>Orario di aggiornamento: <b>{{orario}}</b></p>
<p>Temperatura: <b>{{temperatura}}</b>C&deg;</p>
<p>Umidit&agrave: <b>{{umidita}}%</b></p>
%else:
<h2>Si &eacute; verificato un errore leggendo i dati dal sensore</h2>
%end
```

Il template deve essere salvato nella cartella `/home/pi/views` con il nome `valori.tpl`; questo consentirà a Bottle di utilizzarlo.

Come potete vedere il template unisce codice Python, nelle righe che iniziano con il simbolo `%`, e codice HTML nelle altre.

Questa formattazione rende complicato gestire l'indentazione del codice. Per questo motivo è necessario usare la keyword `end` per terminare i blocchi (nell'esempio il blocco `if/else`).

È anche possibile delimitare interi di blocchi come codice Python usando i tag `<%` e `%>`, per esempio riformattando il nostro template in questo modo:

```
<%
if valorivalidi:
%>
<p>Orario di aggiornamento: <b>{{orario}}</b></p>
<p>Temperatura: <b>{{temperatura}}</b>C&deg;</p>
<p>Umidit&agrave: <b>{{umidita}}%</b></p>
<%
```

```

else:
%
<h2>Si &eacute; verificato un errore leggendo i dati dal sensore</h2>
<%
end
%>
```

Per inserire il valore (convertito in stringa) di una variabile Python nel codice HTML è necessario includerne il nome tra doppie parentesi graffe come in questo esempio:

```
<p>Temperatura: <b>{{temperatura}}</b></p>
```

Potremo usare tutte le funzionalità di Python viste nei capitoli precedenti e quindi avere, per esempio, loop, condizioni o chiamate a funzioni.

Notate anche che i caratteri accentati o altri caratteri particolari come il simbolo ° devono essere inseriti in HTML usando appositi codici come à per la a accentata o ° per il simbolo °. Potete trovare elenchi di questi caratteri in rete o sui testi dedicati a HTML oppure usare il sito HTML Escape (http://www.htmlescape.net/htmlescape_tool.html) e lasciare che sia lui a fare le conversioni necessarie.

Possiamo modificare il nostro esempio iniziale per aggiungere il codice necessario a utilizzare il template e a leggere dal sensore i valori che saranno poi usati nella pagina HTML generata.

```

#!/usr/bin/env python3

import bottle
import DHT
import datetime

# funzione associata al percorso /valori
@bottle.route("/valori")
def paginaValori():

    # legge i valori dal sensore
    valorivalidi,temperatura,umidita = DHT.leggiSensore(11,25)

    # legge data e ora corrente
    orario=datetime.datetime.now()

    # chiama la funzione bottle.template passando il template
    # e le variabili usate all'interno del template
    return bottle.template("valori",
                           valorivalidi=valorivalidi,
                           temperatura=temperatura,
                           umidita=umidita,
                           orario=orario )
```

```
# avvia il web server  
bottle.run(host="0.0.0.0",port=8080)
```

Ora la funzione Python è associata al percorso `/valori`. La funzione legge i dati usando le funzioni del modulo DHT sviluppato in precedenza e non ritorna più codice HTML, ma invoca la funzione `bottle.template()` passandole il nome del template da utilizzare e poi, come parametri associati a un nome, le variabili con i dati utilizzati nel template. Questo approccio consente di isolare meglio il codice legato alle pagine HTML da quello più specifico della nostra applicazione.

Ora possiamo rilanciare il nostro programma e far puntare il browser al percorso `/valori`. Il risultato dovrebbe essere simile a quanto si vede in [Figura 9.3](#).



Figura 9.3 - La pagina con i valori correnti di temperatura e umidità.

E se volessimo poter leggere anche i valori più vecchi, prelevandoli dal nostro database? Per prima cosa dovremo definire un nuovo template per una pagina che ci mostri una tabella con i dati. Se la nostra stazione meteo girerà per parecchio tempo mostrare tutti i dati nella stessa pagina potrebbe non essere possibile; dovremo mostrare un certo numero di valori per pagina e consentire all'utente di muoversi tra le diverse pagine.

L'utente potrà collegarsi al percorso `/tabella` per vedere le prime righe e ai percorsi `/tabella/<indice>` per vedere le righe a partire da quella all'indice passato come percorso (per esempio, `/tabella/100` presenterà le righe dalla centesima in poi).

Il nostro template utilizzerà tre parametri: l'indice della prima riga mostrata, il numero di righe per pagina e una lista con i dati di ogni lettura (data, temperatura e umidità).

Ecco il template, da salvare come `tabella.tpl` nella cartella `/home/pi/views`, con evidenziato in grassetto il codice Python:

```
% if indice!=0:
```

```

<a href="/tabella/{{indice-valoriperpagina}}">Valori precedenti</a>
% end
% if len(righe)>=valoriperpagina:
<a href="/tabella/{{indice+valoriperpagina}}">Valori successivi</a>
% end

<table border="1">
<tr>
<th>Data</th><th>Temperatura</th><th>Umidit&agrave</th>
</tr>
% for riga in righe:
<tr>
<td>{{riga[0]}}</td>
<td>{{riga[1]}}</td>
<td>{{riga[2]}}</td>
</tr>
% end
</table>

```

In questo template sono utilizzati diversi nuovi tag HTML. Il primo è il tag `a`, che consente di inserire un link. Il link consente di muoversi tra una pagina e l'altra e, nel nostro caso, caricherà l'URL `/tabella/<indice>` spostando l'indice avanti o indietro di un certo numero di righe. Questo consentirà di avere due link `Valori precedenti` e `Valori successivi` nella nostra pagina che ci consentiranno di scorrere i dati nella tabella. Il codice Python a inizio pagina serve a inserire il link ai valori precedenti solo sulle pagine successive alla prima (nella prima `indice` vale zero e la condizione dell'`if` non è soddisfatta) e quello ai valori successivi in tutte le pagine tranne l'ultima.

Gli altri tag (`table`, `tr`, `th` e `td`), sono legati alla gestione delle tabelle in HTML. Il tag `table` consente di inserire una tabella e, nel nostro caso, di specificare un bordo con l'attributo `border`. Il tag `tr` consente di definire una riga all'interno della tabella. Una riga sarà composta di celle; le celle potranno essere celle normali (dichiarate con il tag `td`) o celle di intestazione, dichiarate con il tag `th`.

La nostra tabella avrà tre colonne (data, temperatura e umidità) e le righe saranno create da un loop `for` in Python. Ogni elemento della lista `righe` è un tuple che contiene i tre dati nell'ordine in cui sono dichiarati nel database (che corrisponde a quello utilizzato dalla nostra tabella). Ora vediamo cosa abbiamo aggiunto al nostro programma per gestire la lettura della tabella di valori.

Visto che il nostro codice inizia a diventare piuttosto complesso è utile estrapolare il codice legato al web server in un modulo che salveremo con il nome di `meteoserver.py`. Ecco il codice con evidenziati, come al

solti, i commenti:

```
import bottle
import DHT
import datetime
import sqlite3 as sqlite

# numero di valori per pagina nella pagina tabella
VALORI_PER_PAGINA=20

# funzione associata al percorso /valori e al template views/valori.tpl
@bottle.route("/valori")
def paginaValori():

    # legge i valori dal sensore
    valorivalidi,temperatura,umidita = DHT.leggiSensore(11,25)

    # legge data e ora corrente
    orario=datetime.datetime.now()

    # chiama la funzione bottle.template passando il template
    # e le variabili usate all'interno del template
    return bottle.template("valori",
                           valorivalidi=valorivalidi,
                           temperatura=temperatura,
                           umidita=umidita,
                           orario=orario )

# funzione associata al percorso /tabella ritorna le prime 20 righe
@bottle.route("/tabella")
def paginaTemperatureZero():
    return paginaTemperature(0)

@bottle.route("/tabella/<indice>")
def paginaTemperature(indice):

    # verifica che indice sia un valore intero
    indice=int(indice)

    # apre il database
    connessione = sqlite.connect("meteo.db")

    with connessione:
        # crea una lista in cui verranno inserite le prime VALORI_PER_PAGINA
        # righe o le righe rimanenti se sono meno di VALORI_PER_PAGINA
        righe=list()

        # ricava il cursore per eseguire operazioni sul database
        cursore=connessione.cursor()
        cursore.execute("SELECT * FROM VALORI ORDER BY DATA DESC")

        # salta le prime righe, fino a quella passata come parametro
        contatore=0
        riga=cursore.fetchone()

        while contatore<indice and riga!=None:
```

```

        riga=cursore.fetchone()
        contatore+=1

        # non ci sono piu' righe nel database
        if riga==None:
            return bottle.template("tabella",
                    righe=righe,
                    indice=indice,
                    valoriperpagina=VALORI_PER_PAGINA)

        # riazzera' contatore che ora servira' a contare le righe
        # lette dal database
        contatore=0

        # il loop arriva al massimo a VALORI_PER_PAGINA righe
        while contatore<VALORI_PER_PAGINA and riga!=None:
            # carica la riga
            riga=cursore.fetchone()

            # se e' vuota, termina qui il loop
            if riga==None:
                break

            # la inserisce nella lista
            righe.append(riga)
            contatore+=1

# chiama la funzione bottle.template passando il template
# e le variabili usate all'interno del template
return bottle.template( "tabella",
                    righe=righe,
                    indice=indice,
                    valoriperpagina=VALORI_PER_PAGINA )

# avvia il web server
def avviaServer():
    bottle.run(host="0.0.0.0",port=8080)

```

Al codice sono state aggiunte tre nuove funzioni. La prima, `avviaServer()`, consentirà di attivare il server HTTP che nell'esempio precedente veniva lanciato automaticamente al caricamento del modulo.

Le altre due funzioni servono a gestire la pagina con l'elenco dei valori. Il decoratore `@bottle.route` è usato per associare la funzione `paginaTemperatureZero()` al percorso `"/tabella"`, mentre la funzione `paginaTemperature()` è associata al percorso `"/tabella/<indice>"`. L'utilizzo dei caratteri `< e >` consente di definire un parametro che assumerà il valore del percorso (per esempio, se verrà richiesto il percorso `/tabella/40` il parametro `indice` varrà `40`). La funzione `paginaTemperatureZero()` si limita a richiamare la funzione `paginaTemperature()` passandole `0` come indice.

Il codice interessante è quindi tutto in questa seconda funzione.

La funzione legge i dati dal database che abbiamo popolato con l'esempio precedente. Per leggere dati da un database è necessario, come per la scrittura, aprire una connessione e ottenere un cursore. Il cursore verrà usato per eseguire un comando di `SELECT`:

```
SELECT * FROM VALORI ORDER BY DATA DESC
```

Questo comando ritorna tutti i valori ordinati in modo descendente (grazie alla keyword `DESC`) per data.

L'oggetto `cursor` è in grado di accedere ai dati una sola riga (record) alla volta. Per caricare i dati dobbiamo utilizzare la funzione `fetchone()`, che carica il record successivo (o il primo se abbiamo appena eseguito la query di `SELECT`).

Il record viene ritornato come tuple (lista costante) e conterrà i valori dei singoli campi del nostro record.

La funzione `tabellaTemperature()` esegue la query e poi scorre le righe con indice minore di quello richiesto. A questo punto salva nella lista `righe` un numero di righe definito dalla costante `VALORI_PER_PAGINA` e chiama la funzione `bottle.template()` per generare il codice HTML necessario.

Ora abbiamo le funzioni che ci servono per mostrare i dati al nostro utente via HTTP e possiamo integrarle nel programma `meteo.py` realizzato nel passaggio precedente per catturare i dati dal sensore e salvarli nel database. Le righe modificate sono evidenziate in grassetto.

```
#! /usr/bin/env python3

import DHT
import sqlite3 as sqlite
import time
import datetime
import meteoserver
import threading

# apre il database e ottiene un oggetto connessione
connessione=sqlite.connect("meteo.db")

# attiva il server HTTP per ricevere le richieste
# il server dovrà essere attivato in un thread in background
server=threading.Thread(target=meteoserver.avviaServer)
server.start()

# l'uso della keyword with ci assicura che la connessione verrà
# chiusa anche in caso di eccezioni
with connessione:
```

```

# loop infinito
while True:

    # legge i dati dal sensore
    dativalidi, temperatura, umidita = DHT.leggiSensore(11,25)

    # se i dati non sono validi verrà stampato
    # un messaggio di errore
    if not dativalidi:
        print ("Errore nella lettura dei dati")
    else:
        # per eseguire i comandi SQL dobbiamo usare un
        # oggetto cursore che sara' tornato dalla funzione
        # cursor() dell'oggetto connessione
        cursore=connessione.cursor()

        # nel comando SQL possiamo inserire dei ? che verranno
        # poi sostituiti con i parametri passati a execute
        comandosql="INSERT INTO VALORI VALUES(?, ?, ?, 0)"

        # la funzione execute viene richiamata passando i
        # parametri nello stesso ordine in cui devono essere
        # inseriti nel comando SQL
        cursore.execute(comandosql,(datetime.datetime.now(),
                                      temperatura,
                                      umidita))

        # la chiamata a commit() fa in modo che i dati
        # vengano salvati in modo permanente
        connessione.commit()

        print("Dati salvati - temperatura: "+str(temperatura)+""
umidita: " + str(umidita))

    # aspetta un minuto prima di fare una nuova lettura
    time.sleep(60)

```

La nuova versione importa le funzioni del modulo `meteoserver` e della libreria `threading` di Python.

Questo è necessario perché, se lanciassimo la funzione `bottle.run()` all'interno del nostro codice non potremmo più eseguire altre operazioni, visto che questa funzione ritorna solo quando l'utente termina l'applicazione con **Ctrl+C**.

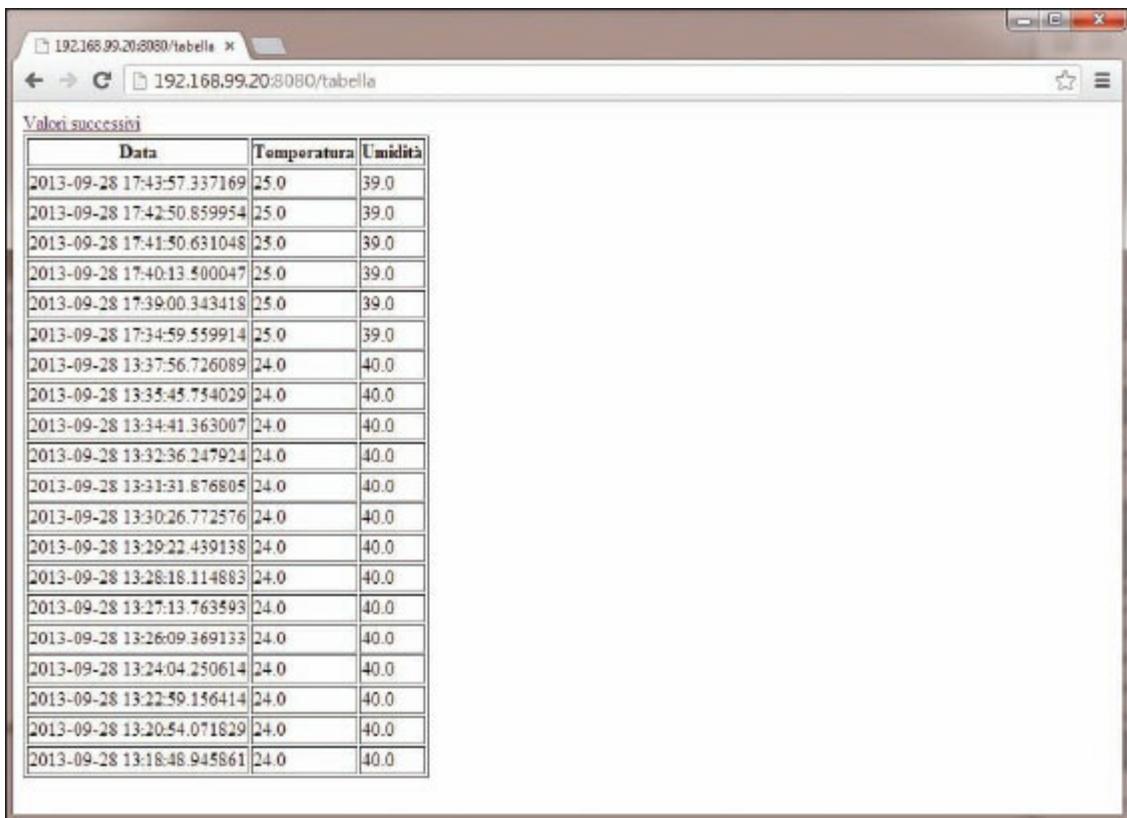
Creando un thread possiamo eseguire in parallelo la funzione di gestione del server HTTP e il loop che legge i dati dal sensore. Linux è un sistema multitasking e avere più thread all'interno della stessa applicazione è abbastanza comune.

Per creare un thread dobbiamo passare la funzione da eseguire a un nuovo oggetto `Thread` e chiamare poi il suo metodo `start()` per avviare l'esecuzione.

Ora lanciando il nostro programma vedremo contemporaneamente

sulla console i messaggi del server HTTP di Bottle e quelli del loop di lettura e salvataggio dei dati. La nostra stazione meteo inizia a funzionare!

Possiamo collegarci per vedere i valori di temperatura inserendo come indirizzo nel browser l'IP del nostro Raspberry Pi seguito dal percorso /tabella (192.168.99.20/tabella nell'esempio in [Figura 9.4](#)).



The screenshot shows a Windows-style window with a title bar "192.168.99.20:8080/tabella". The main content is a table titled "Valori successivi" with three columns: Data, Temperatura, and Umidità. The table contains 20 rows of data, each representing a measurement taken at a specific date and time.

Data	Temperatura	Umidità
2013-09-28 17:43:57.337169	25.0	39.0
2013-09-28 17:42:50.859954	25.0	39.0
2013-09-28 17:41:50.631048	25.0	39.0
2013-09-28 17:40:13.500047	25.0	39.0
2013-09-28 17:39:00.343418	25.0	39.0
2013-09-28 17:34:59.559914	25.0	39.0
2013-09-28 13:37:56.726089	24.0	40.0
2013-09-28 13:35:45.754029	24.0	40.0
2013-09-28 13:34:41.363007	24.0	40.0
2013-09-28 13:32:36.247924	24.0	40.0
2013-09-28 13:31:31.876805	24.0	40.0
2013-09-28 13:30:26.772576	24.0	40.0
2013-09-28 13:29:22.439138	24.0	40.0
2013-09-28 13:28:18.114883	24.0	40.0
2013-09-28 13:27:13.763593	24.0	40.0
2013-09-28 13:26:09.369133	24.0	40.0
2013-09-28 13:24:04.250614	24.0	40.0
2013-09-28 13:22:59.156414	24.0	40.0
2013-09-28 13:20:54.071629	24.0	40.0
2013-09-28 13:18:48.945861	24.0	40.0

Figura 9.4 - La tabella con i valori letti dal sensore.

Anche in questo caso l'aspetto grafico non è granché, ma spero sia appagante vedere come, sfruttando poco codice Python, un sensore semplicissimo e l'hardware e il software di Raspberry Pi sia possibile realizzare applicazioni complesse che coinvolgono accesso all'hardware, database e tecnologie web.

Se siete sviluppatori web sarete probabilmente inorriditi guardando l'interfaccia rudimentale implementata nell'esempio. Ma quale occasione migliore per fare un po' di pratica, modificando i template e arricchendoli o, perché no, studiando un po' la documentazione di Bottle al fine di modificare il codice Python per far ritornare, invece di HTML già formattato, dati in formato JSON che possono essere poi gestiti da codice JavaScript e magari presentati in formato grafico? Se invece non siete sviluppatori web il paragrafo precedente deve esservi sembrato scritto in ostrogoto. Ma gli spunti per imparare e sperimentare

sono tantissimi, anche in un esempio semplice come questo.

La stazione meteo sulla nuvola!

Non vi voglio suggerire di spedire il vostro piccolo personal computer in cielo, quanto piuttosto di utilizzare un altro servizio cloud, Google Documents, per pubblicare le informazioni catturate dalla nostra stazione meteo. In questo modo potremo vedere i dati ed elaborarli senza bisogno di collegarci direttamente al nostro Raspberry Pi.

Adafruit ha realizzato un ottimo tutorial in merito e questo esempio lo estende prelevando i dati dal nostro database e garantendo che questi siano aggiornati anche se la connettività internet dovesse essere intermittente. Questo è lo scopo del campo `TRASMESSO` che abbiamo inserito nella nostra tabella. Quando il dato viene salvato questo campo viene messo a `0` (`False`). La funzione di trasmissione proverà a inserire nel nostro foglio di calcolo tutti i dati che non sono ancora presenti. Per inserire i dati in un foglio di calcolo Google Documents possiamo usare la libreria `gspread`. Questa libreria consentirà di aprire e modificare i documenti dalle nostre applicazioni in Python in modo semplice. Come per tutte le librerie, è necessario per prima cosa installarla. I comandi per il download dei sorgenti (tramite `git`) e l'installazione sono simili a quelli già visti per le altre librerie:

```
pi@pivalter ~ $ mkdir gspread
pi@pivalter ~ $ cd gspread/
pi@pivalter ~/gspread $ git init
Initialized empty Git repository in /home/pi/gspread/.git/
pi@pivalter ~/gspread $ git clone https://github.com/burnash/gspread.git
Cloning into 'gspread'...
...
Resolving deltas: 100% (469/469), done.
pi@pivalter ~/gspread $ cd gspread/
pi@pivalter ~/gspread/gspread $ python3 setup.py install --prefix $HOME/pythonlib
running install
...
Finished processing dependencies for gspread==0.1.0
```

Una volta installata la libreria `gspread` possiamo utilizzarla per inserire i nostri dati nel foglio elettronico. Prima, però, dovremo creare il nostro foglio elettronico!

Per prima cosa, se già non lo avete, è necessario registrarsi per un account Google. La registrazione è gratuita e si può effettuare dal sito docs.google.com. Una volta fatta la registrazione ed effettuato il login

potremo creare un nuovo documento selezionando il pulsante **Create** nella pagina principale e poi **Spreadsheet** per creare un nuovo foglio di calcolo.

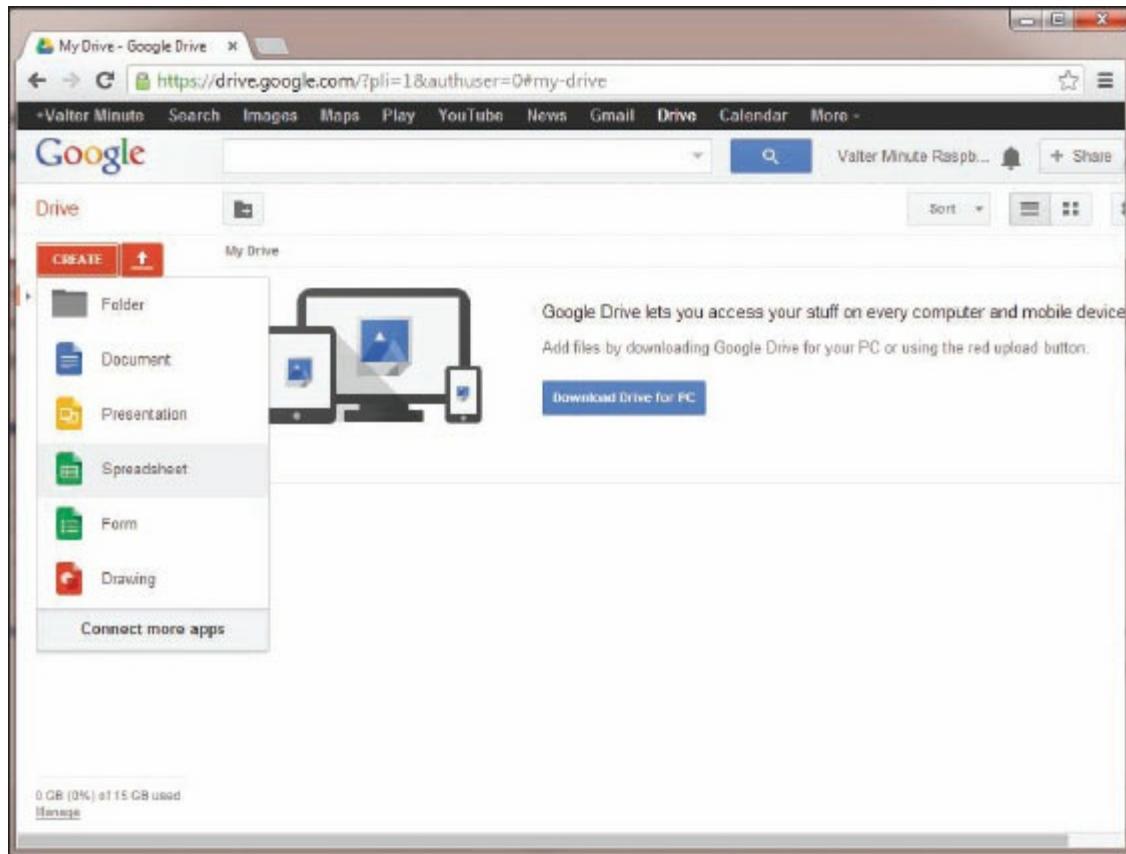


Figura 9.5 - Creazione di un nuovo foglio di calcolo in Google Documents.

A questo punto ci verrà mostrato un foglio di calcolo vuoto. Per prima cosa è necessario assegnare un nome al nostro foglio di calcolo; lo possiamo fare selezionando il titolo del documento e rinominandolo in **RaspberryPiMeteo**, come in [Figura 9.6](#).

Un foglio di calcolo di Google Documents può essere costituito da più fogli. per default il primo foglio creato si chiama "Sheet1"; possiamo rinominarlo selezionando il nome del foglio e scegliendo l'opzione **Rename**.

Possiamo assegnargli il nome "Valori". Il nome del documento e del foglio ci serviranno per poter inserire dei dati usando Python.

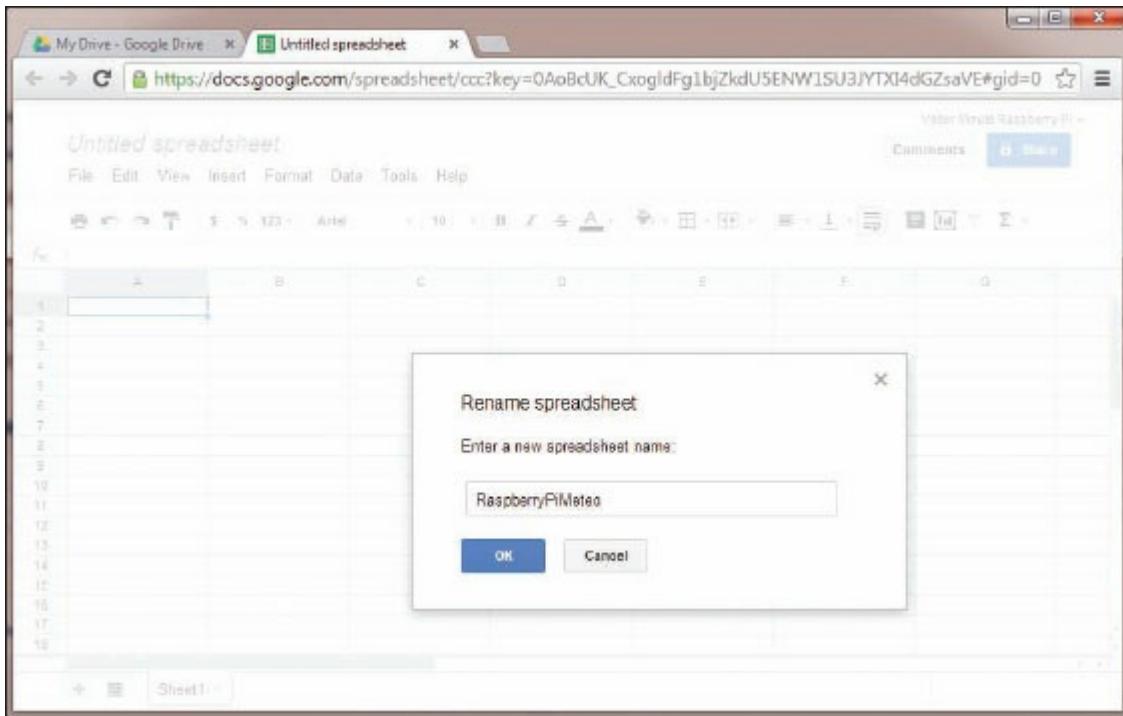


Figura 9.6 - Assegnare un nome al documento.

A screenshot of a Google Sheets window titled "RaspberryPiMeteo". The window title bar also includes the URL "https://docs.google.com/spreadsheet/ccc?key=0AoBcUK_CxogIdFg1bjZkdU5ENW1SU3jYTXI4dGZsaVE#gid=0". The spreadsheet contains a single row of headers in row 1: "Data" in column A, "Temperatura" in column B, and "Umidità" in column C. The rest of the rows (2 to 19) are empty. The toolbar at the top shows various spreadsheet functions like bold, italic, and font size. The status bar at the bottom indicates "Valori" (Values).

Data	Temperatura	Umidità

Figura 9.7 - Il foglio rinominato.

Ora possiamo inserire nella prima riga del foglio le intestazioni delle colonne, poi selezioniamo le righe rimanenti (da 2 a 100), facciamo un clic destro con il mouse e selezioniamo l'opzione **Delete**. In questo modo nel foglio resterà soltanto la prima riga. Questo ci serve perché la libreria `gspread` non è in grado di recuperare qual è la prima riga libera del foglio e rileggere i contenuti delle celle a ogni aggiornamento richiederebbe parecchio tempo. Eliminando le righe vuote `gspread` potrà

semplicemente aggiungere una riga alla fine del foglio e riempirla con i dati.

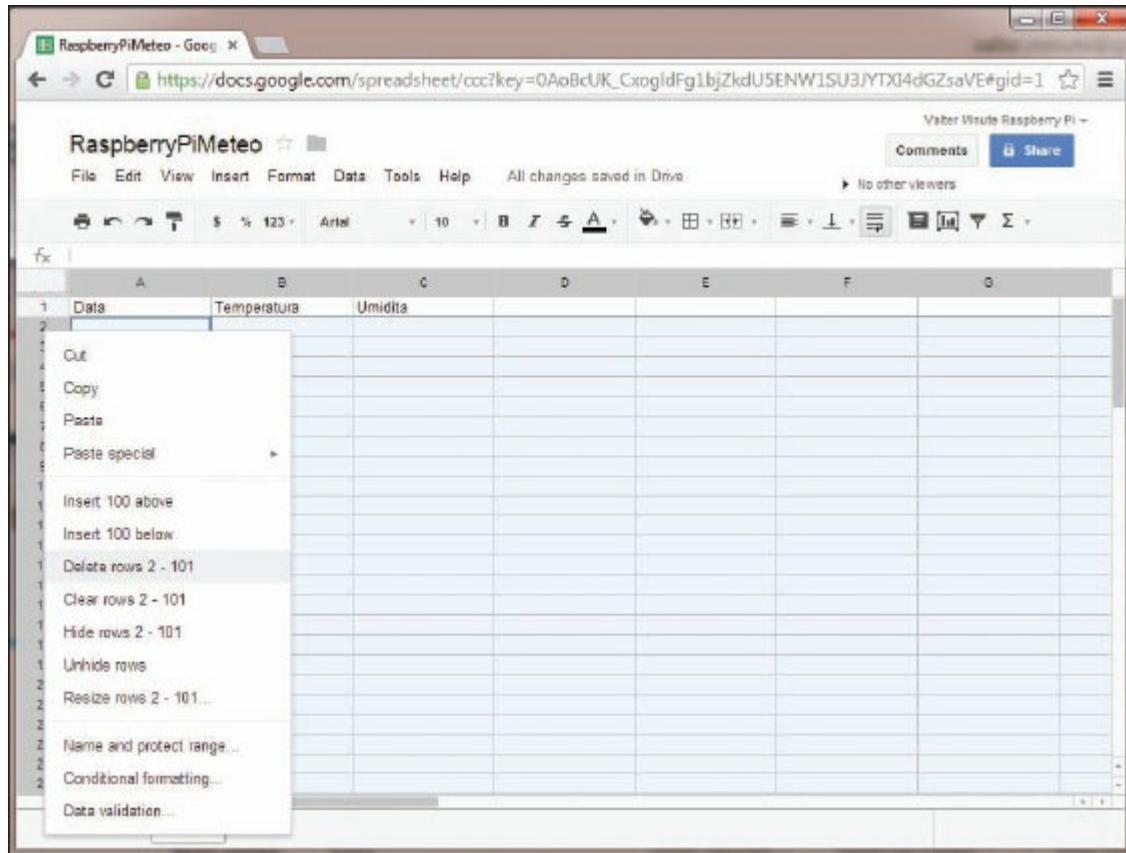


Figura 9.8 - Cancellazione dal foglio di calcolo di tutte le righe tranne la prima.

Ora che abbiamo creato e preparato il nostro foglio di calcolo e installato la libreria `gspread` possiamo mettere mano al codice Python.

Anche in questo caso è utile implementare le funzioni che aggiornano i dati in un modulo separato, che chiameremo `meteospread.py`. Questo è il codice del modulo:

```
import gspread
import sqlite3 as sqlite

def aggiungiRecord(user,password,data,temperatura,umidita):
    # apre una sessione effettuando il login
    sessione=gspread.login(user,password)

    # apre il foglio e la pagina per inserire i valori
    foglio=sessione.open("RaspberryPiMeteo")
    pagina=foglio.worksheet("Valori")

    # crea una nuova riga con i dati da inserire
    pagina.append_row([data,temperatura,umidita])

def aggiornaFoglioDiCalcolo(user,password,connessione):
```

```

cursore=connessione.cursor()
# visto che potrebbero esserci dei dati non salvati,
# il programma legge tutte le righe dove il
# campo TRASMESSO vale 0 e prova a trasmetterle
comandosql="SELECT * FROM VALORI WHERE TRASMESSO=0 ORDER BY DATA"
cursore.execute(comandosql)

riga=cursore.fetchone()

# il salvataggio dei dati può fallire; intercettiamo
# le eccezioni per evitare che facciano terminare il
# programma
while riga!=None:
    try:

        aggiungiRecord(user,
                       password,
                       riga[0],
                       riga[1],
                       riga[2] )

        # se la scrittura nel dato fallisce
        # esce dal loop e riproverà quando arriverà
        # un nuovo dato da salvare
    except:
        break

    # aggiorna la tabella per non ritrasmettere di nuovo il valore
    cursoreaggiornamento=connessione.cursor()

    comandosql="UPDATE VALORI SET TRASMESSO=1 WHERE DATA=? AND
    TEMPERATURA=? AND UMIDITA=?"
    cursoreaggiornamento.execute(comandosql,
                                 (riga[0],riga[1],riga[2]))


    print("Dato trasmesso - ",riga[0],riga[1],riga[2])

    # passa al dato successivo
    riga=cursore.fetchone()

    # salva le modifiche
    connessione.commit()

```

Nel modulo implementeremo due funzioni: `aggiungiRecord()` e `aggiornaFoglioDiCalcolo()`. La prima funzione è quella che andrà a scrivere i dati nel nostro documento. Per usare le funzioni di `gspread`, oltre a importare il modulo relativo, dobbiamo per prima cosa attivare una sessione usando il metodo `login()` a cui dovremo passare nome utente e password.

Una volta validate le nostre credenziali, potremo accedere al foglio di calcolo utilizzando la funzione `open()` dell'oggetto `sessione`. Il foglio di calcolo (anche se pare una contraddizione in termini...) è composto da

più pagine e con la funzione `worksheet()` recupereremo la pagina che abbiamo creato e chiamato "Valori". Nel foglio potremo aggiungere altre pagine e usarle, per esempio, per calcolare le temperature medie, lo sbalzo termico tra giorno e notte, o per mostrare in formato grafico l'andamento delle temperature. L'importante è non modificare la pagina "Valori". La funzione `append_row()` dell'oggetto `pagina` consentirà di aggiungere una nuova riga alla fine del foglio di calcolo inserendo, a partire dalla colonna numero 1, i dati passati come lista.

La seconda funzione del modulo, `aggiornaFoglioDiCalcolo()`, è un po' più complessa. Utilizza il database, come abbiamo già visto in precedenza, passando per l'oggetto `cursor`. La funzione utilizza due cursori, di cui uno serve a ricavare tutti i dati non ancora trasmessi eseguendo un comando di `SELECT` sulla tabella `VALORI`, per cercare tutti i record il cui campo `TRASMESSO` vale 0. Un comando SQL che ritorna un elenco di valori viene chiamato query, da cui la Q in SQL (Structured Query Language).

La funzione `fetchone()` dell'oggetto `cursor` ritornerà il primo record che corrisponde alla nostra ricerca oppure `None`, una costante Python che rappresenta un oggetto nullo, non esistente.

Una volta ricavati i dati da inserire nel foglio di calcolo (la query li ordina dal più vecchio al più nuovo, in modo da non alterare il normale ordine nelle celle), viene richiamata la funzione `aggiungiRecord()`. Se qualcosa va storto (e quando c'è una connessione a internet che passa da un continente all'altro qualcosa può andare storto!) le funzioni di `gspread` genereranno un'eccezione e il nostro codice si limiterà a intercettarle e a terminare l'aggiornamento del foglio di calcolo, usando la keyword `break` per uscire dal ciclo e, di conseguenza, dalla funzione. Il ciclo più esterno continuerà l'esecuzione e alla lettura dati successiva la funzione `aggiornaFoglioDiCalcolo()` sarà chiamata di nuovo e dovrà aggiornare due dati invece di uno soltanto.

Se invece il dato è stato inserito correttamente nel foglio di calcolo di Google Documents, allora dovremo impostare a un 1 il campo `TRASMESSO` del record corrispondente, per evitare di trasmetterlo più volte. Per modificare i dati nelle nostre tabelle il linguaggio SQL ci mette a disposizione il comando `UPDATE`.

```
UPDATE VALORI SET TRASMESSO=1 WHERE DATA=? AND TEMPERATURA=? AND UMIDITA=?
```

Dovremo specificare la tabella su cui operare e, dopo la keyword `SET`, i campi da modificare con i relativi valori. Anche per il comando `UPDATE` è

possibile specificare una clausola `WHERE` e quindi selezionare i record da modificare. Nel nostro caso, passando data, temperatura e valore di umidità siamo sicuri di modificare soltanto il record appena aggiornato.

Questo comando non verrà eseguito usando il cursore già in uso per leggere i record da spedire, ma creando un nuovo cursore.

Una volta eseguito il comando di `UPDATE` attraverso la funzione `execute()` del cursore appena creato, il nostro codice chiamerà `fetchone()` per caricare il record successivo e poi la funzione `commit()` della connessione al database per assicurarsi che i dati siano effettivamente aggiornati sul file.

Ora dovremo integrare le funzioni appena implementate nel programma principale. Eccolo qui di seguito con evidenziate le modifiche:

```
#! /usr/bin/env python3

import DHT
import sqlite3 as sqlite
import time
import datetime
import meteoserver
import threading
import meteospread

# inserite qui le vostre credenziali!
GOOGLE_USERNAME=<username>
GOOGLE_PASSWORD=<password>

# apre il database e ottiene un oggetto connessione
connessione=sqlite.connect("meteo.db")

# attiva il server HTTP per ricevere le richieste
# il server dovrà essere attivato in un thread in background
server=threading.Thread(target=meteoserver.avviaServer)
server.start()

# l'uso della keyword with ci assicura che la connessione verrà
# chiusa anche in caso di eccezioni
with connessione:

    # loop infinito
    while True:

        # legge i dati dal sensore
        dativalidi, temperatura, umidita = DHT.leggiSensore(11,25)

        # se i dati non sono validi verrà stampato
        # un messaggio di errore
        if not dativalidi:
            print ("Errore nella lettura dei dati")
```

```

else:

    # per eseguire i comandi SQL dobbiamo usare un
    # oggetto cursore che sara' tornato dalla funzione
    # cursor() dell'oggetto connessione
    cursore=connessione.cursor()

    # nel comando SQL possiamo inserire dei ? che verranno
    # poi sostituiti con i parametri passati a execute
    comandosql="INSERT INTO VALORI VALUES(?, ?, ?, 0)"

    # la funzione execute viene richiamata passando i
    # parametri nello stesso ordine in cui devono essere
    # inseriti nel comando SQL
    cursore.execute(comandosql, (datetime.datetime.now(),
                                  temperatura,
                                  umidita))

    # la chiamata a commit() fa in modo che i dati
    # vengano salvati in modo permanente
    connessione.commit()

    print("Dati salvati - temperatura: "+str(temperatura)+""
umidita: " + str(umidita))

    meteospread.aggiornaFoglioDiCalcolo(GOOGLE_USERNAME, GOOGLE_
PASSWORD, connessione)

    # aspetta dieci minuti prima di fare una nuova lettura
    time.sleep(600)

```

Come potete vedere le modifiche sono abbastanza limitate. Viene importato il modulo meteospread appena creato e, dopo aver salvato i dati nel database, viene richiamata la funzione per aggiornare il foglio di calcolo.

Inoltre il timeout per le letture è stato allungato a dieci minuti. La temperatura e l'umidità normalmente variano molto lentamente e fare letture molto frequenti serve solo a generare traffico inutile sulla nostra rete.

Ora possiamo rilanciare il nostro programma e dovremmo vedere che, dopo aver letto i dati, segnalerà anche di averli trasmessi correttamente:

```

pi@pivalter ~ $ ./meteo.py
Bottle v0.12-dev server starting up (using WSGIRefServer())...
Listening on http://0.0.0.0:8080/
Hit Ctrl-C to quit.

```

```

Dati salvati - temperatura: 24.0 umidita: 39.0
Dato trasmesso - 2013-09-29 13:30:49.570737 24.0 39.0

```

Se riaprite il vostro foglio di calcolo su Google Documents vedrete che

i dati sono aggiornati in tempo reale appena vengono trasmessi dalla nostra stazione meteo.

The screenshot shows a Google Sheets spreadsheet with the title 'RaspberryPiMeteo'. The data is organized into columns A through G. Column A contains dates and times from 9/28/2013 21:17:09 to 9/28/2013 21:39:33. Column B contains temperature values ranging from 24 to 26. Column C contains humidity values ranging from 38 to 40. The rest of the columns (D, E, F, G) are empty. The status bar at the bottom right indicates 'Count: 3'.

	A	B	C	D	E	F	G
159	9/28/2013 21:17:09	25	39				
160	9/28/2013 21:19:15	25	39				
161	9/28/2013 21:20:20	25	38				
162	9/28/2013 21:21:25	25	39				
163	9/28/2013 21:23:30	25	39				
164	9/28/2013 21:24:38	25	39				
165	9/28/2013 21:25:42	25	39				
166	9/28/2013 21:27:47	25	39				
167	9/28/2013 21:28:53	25	39				
168	9/28/2013 21:31:58	25	39				
169	9/28/2013 21:33:04	25	39				
170	9/28/2013 21:34:11	25	38				
171	9/28/2013 21:35:15	24	40				
172	9/28/2013 21:36:20	25	39				
173	9/28/2013 21:37:24	24	40				
174	9/28/2013 21:38:28	25	39				
175	9/28/2013 21:39:33	24	40				

Figura 9.9 - I dati aggiornati in tempo reale.

È possibile utilizzare altre pagine per elaborare i nostri dati e anche per mostrarli in formato grafico, come si vede in [Figura 9.10](#).

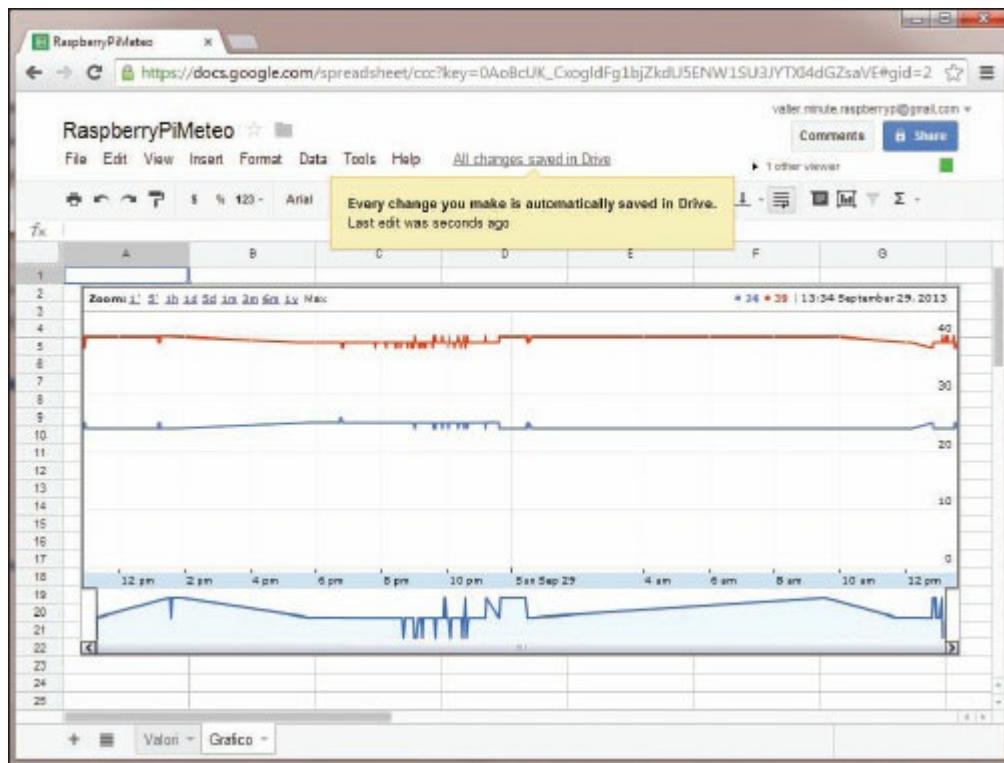


Figura 9.10 - Mostrare i dati in un grafico.

Pronti? Via!

Ora bisogna far sì che il nostro programma venga lanciato all'avvio, in

modo da non dover effettuare un login, in locale o via SSH, per doverlo lanciare.

Per fare questo passaggio dobbiamo capire un po' meglio come Raspbian decide quali programmi lanciare all'avvio.

Il sistema prevede una serie di "runlevel" che corrispondono a diverse modalità di funzionamento. Quando il sistema parte attiva il runlevel di default e, osservando i messaggi che vengono stampati a video durante il boot, ne noterete uno che dice:

```
INIT: Entering runlevel: 2
```

Questo è il runlevel di default di Raspbian. Lo possiamo vedere anche osservando il file `/etc/inittab`.

All'inizio del file troverete l'impostazione del runlevel di default:

```
pi@pivalter ~ $ cat /etc/inittab
# /etc/inittab: init(8) configuration.
# $Id: inittab,v 1.91 2002/01/25 13:35:21 miquels Exp $

# The default runlevel.
id:2:initdefault:
```

Il meccanismo di startup di Linux è piuttosto complesso. Quello che vogliamo sapere lo possiamo però recuperare rapidamente dal filesystem. Nella cartella `/etc` ci sono una serie di sotto cartelle chiamate `rc<runlevel>.d`. Queste cartelle contengono gli script di startup. Nel nostro caso ci interessa la cartella `/etc/rc2.d`. Se ne osserviamo il contenuto mediante il comando `ls` vedremo che tutti i file sono dei link:

```
pi@pivalter ~ $ ls /etc/rc2.d/ -la
totale 12
drwxr-xr-x  2 root root 4096 set 29 15:07 .
drwxr-xr-x 96 root root 4096 set 29 15:08 ..
lrwxrwxrwx  1 root root  17 set 29 14:39 K02lightdm -> ../init.d/lightdm
lrwxrwxrwx  1 root root  20 ago 27 17:47 K06nfs-common -> ../init.d/nfs-common
lrwxrwxrwx  1 root root  17 ago 27 17:47 K06rpcbind -> ../init.d/rpcbind
...
lrwxrwxrwx  1 root root  13 giu 19 12:57 S02ntp -> ../init.d/ntp
lrwxrwxrwx  1 root root  15 giu 19 13:33 S02rsync -> ../init.d/rsync
lrwxrwxrwx  1 root root  13 giu 19 14:38 S02ssh -> ../init.d/ssh
lrwxrwxrwx  1 root root  15 set 29 15:07 S03meteo -> ../init.d/meteo
lrwxrwxrwx  1 root root  18 giu 19 13:36 S04plymouth -> ../init.d/plymouth
lrwxrwxrwx  1 root root  18 giu 19 13:36 S04rc.local -> ../init.d/rc.local
lrwxrwxrwx  1 root root  19 giu 19 13:36 S04rmnologin -> ../init.d/rmnologin
```

Questi link iniziano tutti con gli stessi prefissi `S` o `K` e due cifre, e tutti puntano a degli script memorizzati nella cartella `/etc/init.d`.

Il prefisso serve per decidere se lanciare lo script linkato con il parametro start (quando il link inizia per `s`), per avviare un demone (il nome con cui il Linux vengono chiamati i servizi), oppure con stop. Visto che noi vogliamo lanciare il nostro programma ci servirà un link con il prefisso `s`. Le due cifre servono a determinare l'ordine di esecuzione degli script. Non è necessario che il numero usato sia univoco, ma il sistema garantisce che gli script con un numero più basso siano eseguiti prima di quelli con numeri più alti.

Quindi, per eseguire la nostra applicazione allo startup dovremo creare uno script nella cartella `/etc/init.d` e creare un link nella cartella `/etc/rc2.d` con un nome che inizi con `s` e due cifre.

Ma nello script non possiamo mettere semplicemente il comando di esecuzione di `meteo.py`. I comandi eseguiti all'avvio vengono eseguiti con i diritti di root e il nostro codice Python non troverebbe le librerie che sono nella home dell'utente `pi`. La soluzione è lanciare il nostro comando in modo che usi i diritti e le impostazioni dell'utente `pi`; per farlo useremo il comando `sudo` che, fino a ora, abbiamo utilizzato solo per lanciare comandi con i diritti di super-user. In realtà lanciando `sudo` con il parametro `-u` possiamo specificare l'utente da utilizzare per eseguire il comando passato come argomento.

Usare `sudo -u` non basta perché non tutte le impostazioni dell'utente `pi` vengono riportate. Per assicurarci che l'applicazione venga eseguita esattamente come nel caso in cui la lanciamo da una command line normale dovremo realizzare un piccolo script nella nostra cartella `home`, salvandolo con il nome `pimeteo`:

```
#!/bin/bash
export PYTHONPATH=/home/pi/pythonlibs/lib/python3.2/site-packages
cd /home/pi
/home/pi/meteo.py &
```

Ricordiamoci di renderlo eseguibile usando `chmod`:

```
pi@pivalter ~ $ chmod a+x pimeteo
```

Lo script riconfigura la variabile di environment con il percorso delle librerie Python, si posiziona nella cartella `home` (in modo che continuino a funzionare i path relativi nel nostro codice) e poi lancia il nostro programma `meteo.py` usando il carattere `&` alla fine del comando per fare in modo che questo venga eseguito in background.

Ora dobbiamo scrivere lo script di avvio. Per farlo dovremo lanciare

Nano usando l'utente super user:

```
pi@pivalter ~ $ sudo nano /etc/init.d/meteo
```

Anche in questo caso lo script è abbastanza semplice:

```
#!/bin/sh  
sudo -u pi /home/pi/pimeteo
```

Si limita a usare `sudo` per lanciare lo script `pimeteo` usando l'utente `pi`. Anche in questo caso dovremo rendere eseguibile lo script:

```
pi@pivalter ~ $ sudo chmod a+x /etc/init.d/meteo
```

Per fare in modo che lo script venga eseguito all'avvio dobbiamo per prima cosa creare un link nella cartella `/etc/rc2.d`:

```
pi@pivalter ~ $ sudo ln -s /etc/init.d/meteo /etc/rc2.d/S04meteo
```

Ora dobbiamo lanciare uno script che aggiornerà le informazioni di startup del sistema:

```
pi@pivalter ~ $ sudo update-rc.d meteo defaults  
sudo: unable to resolve host pivalter  
update-rc.d: using dependency based boot sequencing  
insserv: warning: script 'S04meteo' missing LSB tags and overrides  
insserv: warning: script 'meteo' missing LSB tags and overrides  
insserv: warning: current start runlevel(s) (2) of script 'meteo' overrides LSB  
defaults (2 3 4 5).  
insserv: warning: current stop runlevel(s) (empty) of script 'meteo' overrides LS  
defaults (0 1 6).
```

I warning ci segnalano che il nostro script non gestisce i parametri "start" e "stop"; dovendo soltanto lanciare il nostro programma all'avvio, questo non è un problema.

Ora siamo pronti a riavviare il nostro Raspberry Pi e, se tutto è stato configurato correttamente, la nostra stazione meteo inizierà a trasmettere dati appena completerà il boot.

```
pi@pivalter ~ $ sudo reboot
```

Il colonnello Bernacca sarebbe stato fiero di noi!

Hai voluto la bicicletta?

Un modo di dire, ma anche un incitamento ad andare avanti, portare a compimento un lavoro, un viaggio, un'avventura. Raspberry Pi è la vostra nuova bicicletta: potete usarla per imparare, per muovervi nella rete, per provare a realizzare progetti impossibili.

I miei genitori quando, dopo averli tormentati per ottenere qualche nuovo gioco, tornavo a tormentarli per farmi aiutare nell'utilizzarlo, ripetevano spesso questa frase: "Hai voluto la bicicletta? E adesso pedala!".

Non era una modo di scaricarmi, anche perché poi qualche aiuto inevitabilmente arrivava.

Era un modo di incoraggiarmi a provare, a sperimentare, a esplorare. Perché, come avrei capito solo molto tempo dopo, la soddisfazione di ottenere qualcosa da soli è incommensurabilmente più grande di quella che si prova a ottenere un risultato già bello e pronto.

Questo non vuol dire che non si possa condividere questa passione con altri e lavorare insieme.

La rete è piena di progetti e spunti interessanti e ci permette di collaborare e condividere esperienze e informazioni con persone che

magari vivono dall'altra parte del globo.

Ma limitarsi a prendere quanto già esiste, per quanto divertente, non sarà mai appagante come sapere di essere riusciti a fare un passo in più, da soli. E ancora più appagante sarà poi condividere quanto avete fatto o scoperto, per consentire poi ad altri di andare ancora più avanti.

Raspberry Pi è uno strumento che con il suo basso costo, la sua espandibilità, la sua potenza di calcolo e il suo modello di sviluppo aperto, vi mette in grado di fare moltissime cose interessanti.

Ma, come per ogni strumento, sono le mani e la testa di chi lo usa a farlo rendere al massimo.

Spero che questo testo vi abbia fornito degli spunti interessanti da approfondire. Spero vi abbia fatto conoscere strumenti e tecnologie nuove o modi nuovi di utilizzare tecnologie che conoscevate già.

E ora? Verso dove si può pedalare?

Se la filosofia dei maker vi attrae, perché non approfondirla leggendo Il Manuale del Maker di Andrea Maietta e Paolo Aliverti, pubblicato anch'esso da Edizioni FAG? Potrete scoprire nuove aree in cui sperimentare, come la stampa 3D e Arduino, o approfondire le vostre conoscenze in campo elettronico, magari integrando quanto imparate con quello che ora sapete fare con Raspberry Pi.

Oppure potrete scoprire la programmazione approfondendo la vostra conoscenza di Python (che in questo testo abbiamo davvero solo sfiorato!).

Potete adoperare Raspberry Pi come uno strumento per semplificarvi la vita quotidiana, per automatizzare la vostra casa, oppure come server domestico in modo ancora più completo di quanto descritto nel [Capitolo 4](#).

Potrete semplicemente sedervi sul divano a guardare film e ascoltare musica, personalizzando XBMC al massimo.

Potrete usarlo come strumento per studiare, magari aggiungendo funzioni alla piccola stazione meteo. Potete aggiungere sensori per misurare altre grandezze (la pressione, la velocità del vento...) o progettare un'interfaccia migliore, magari con una foto per ogni valore letto e la possibilità di vedere un "film" accelerato del clima fuori dalla vostra finestra. Oppure potete sfruttare i dati catturati elaborandoli in modo più complesso. O, ancora, potete dotare la stazione meteo di un'interfaccia grafica per vedere i dati direttamente sul vostro monitor o sulla vostra TV.

Come vedete gli unici limiti sono la fantasia e (purtroppo) il portafogli. Anche se, per il secondo punto, si diffondono sempre di più dispositivi, sensori e attuatori pensati per il mercato e le capacità di spesa degli hobbisti.

E se volete condividere la vostra passione, anche nel nostro Paese si stanno diffondendo i fab lab, laboratori che mettono a disposizione gli strumenti, ma anche una possibilità di condividere idee e informazioni. A Milano esiste Frankenstein Garage (www.frankensteingarage.it/), ma realtà simili si stanno diffondendo anche in altre città. Ci sono poi gli eventi come le Maker Faire, che consentono di toccare con mano le tecnologie di cui magari abbiamo solo sentito parlare in rete.

Senza dimenticare che la rete, seppure spesso in modo solo virtuale, consente di conoscere persone che condividono la nostra passione e magari di lavorare insieme su progetti che sarebbero troppo grandi e complessi per le nostre sole forze.

Oppure, per unire l'utile al dilettevole, potrete applicare quanto avete scoperto leggendo queste pagine al vostro lavoro, magari sfruttando qualcuna delle tecnologie con cui abbiamo "giocato", o scoprendo che in qualche caso un dispositivo dedicato può soddisfare alcune esigenze meglio di un generico PC o di uno smartphone o tablet commerciale.

Non dimentichiamoci che molti prodotti di successo sono nati proprio da progetti partiti in modo amatoriale.

Raspberry Pi non è adatto per essere utilizzato in produzione, ma esistono, anche in Italia, molti professionisti e aziende che possono far diventare il vostro prototipo un prodotto vero. Perché non provarci?

Se poi lavorate nel mondo della scuola spero abbiate avuto modo di valutare come un oggetto così semplice possa consentire agli studenti alle prese con i primi passi nel mondo della programmazione o dell'elettronica di imparare sperimentando e divertendosi.

Non fatevi scrupoli a contattarmi (raspberrypi@valterminute.com) se avete idee che possano aiutare i ragazzi ad avvicinarsi a questo mondo, magari "pasticciando" un po' tra elettronica e informatica, e se pensate che l'esperienza di chi scrive possa tornare utile.

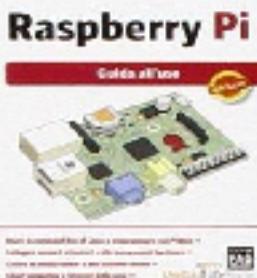
Ora potete posare il libro e prendere in mano la tastiera. E pedalare!

Appendice I componenti utilizzati

Una piccola lista della spesa con i componenti che sono stati utilizzati negli esempi proposti nel testo.

Componente	Produttore	Reperibilità
Cavi jumper, basette sperimentali, LED, pulsanti, resistenze, fotoresistenza	Vari	Qualsiasi negozio o sito online di componentistica elettronica. Spesso è possibile acquistare dei kit che includono le basette, i cavi e i componenti più comunemente usati
Ricevitore IR Vishay 4838	Vishay	Siti online di componenti elettronici, anche eBay
PiCobbler	Adafruit	www.adafruit.com ; sul sito esiste una sezione "distributori" con diversi distributori in Italia ed Europa. Cavi simili sono disponibili anche da Sparkfun (www.sparkfun.com), con distributori italiani, o sul sito www.modmypi.com che spedisce dal Regno Unito
		Negozi di componentistica online o su eBay.

		Negozi di componentistica online o su eBay.
AdS1015 breakout	Adafruit	www.adafruit.com e distributori
Sensore DHT11 o DHT22	Vari	Adafruit, Sparkfun e distributori, sia come componente singolo sia già montato su una board con i componenti aggiuntivi (resistenza da 10 K)



EDIZIONI
FAG
MILANO

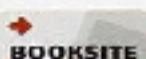
*pro
DigitalLifeStyle

Al centro non c'è la macchina, ci siamo noi. Con la nostra voglia di capire e di fare. Informazione essenziale ma completa, raccolta direttamente sul campo. Ecco i libri Digital LifeStyle Pro. Uno strumento nuovo, per nuovi lettori, che nel digitale trovano la loro passione o la loro professione.

Raspberry Pi

Guida all'uso

Raspberry Pi è un piccolo ed economico personal computer nato per facilitare l'apprendimento dell'informatica e della programmazione. Permette non solo di imparare a usare il software, ma anche di modificarlo, estenderlo e realizzarne di nuovo. Il suo costo ridotto e i suoi bassi consumi consentono inoltre di utilizzarlo come dispositivo dedicato per le esigenze più diverse. In questa guida, dopo aver familiarizzato con la command line di Linux e la programmazione con Python, faremo interagire Raspberry Pi con il mondo reale, utilizzando componenti hardware facilmente reperibili e del costo di poche decine di euro. Seguendo chiare spiegazioni passo passo, corredate da immagini a colori, impareremo a realizzare un server domestico, un media center, un sistema di videosorveglianza, una stazione meteorologica. Grazie al collegamento a servizi di cloud computing come DropBox e Xively, i dati raccolti da Raspberry Pi saranno quindi disponibili su ogni nostro device.



Elementi aggiuntivi a supporto del libro disponibili online:
www.valterminute.com/raspberrypi

Tra gli argomenti trattati

- Installare e configurare Raspbian
- Usare la command line di Linux e realizzare semplici script
- Programmare con Python
- Collegare sensori, attuatori e altri componenti hardware
- Realizzare un server domestico, un media center, un sistema di videosorveglianza, una stazione meteo
- Collegare Raspberry Pi a servizi di cloud computing

L'autore

Valter Minute ha iniziato a pasticciare con i computer quando gli regalarono un Commodore VIC-20 verso la metà degli anni '80 e, da allora, non si è mai stancato di sperimentare, programmare e scoprire le meraviglie nascoste in un pezzetto di silicio in grado di elaborare solo zero e uno.

Edizioni FAG srl

via Garibaldi, 5 - 20090 Assago (Mi)
tel. 02 4885241 - www.fag.it

€ 29,90

ISBN 978-88-6604-388-1



9 788866 043881