PostgreSQL gestisce i dati, LuaLaTeX li stampa

Roberto Giacomelli
Articolo sul blog *http://robitex.wordpress.com*
e-mail: giaconet dot mailbox at gmail dot com

22 febbraio 2011

Sommario

Complici PostgreSQL e il nuovissimo LuaLaTeX, dimostreremo come sia possibile eseguire interrogazione su un database direttamente da un sorgente .tex per produrre report come ricevute, prospetti, lettere e comunicazioni in serie e così via. Nell'articolo verrà mostrata la procedura passo passo in un caso semplice in cui si vuole generare nel formato pdf una ricevuta conseguente ad un pagamento periodico.

Indice

1 Argomentare sulla soluzione migliore

Vi sono moltissime situazioni gestionali in cui ricorriamo a fogli di calcolo o alle stesse directory di file per memorizzare dati e stampare report di analisi e di rendicontazione. Sappiamo però che lo strumento migliore per definizione per gestire i dati è un database ed il migliore tra i DBMS è PostgreSQL, mentre dal lato della stampa, il migliore programma di composizione tipografica è LaTeX.

In questo post illustreremo passo passo come unire i due mondi, gestione dati con PosgreSQL e stampa di report con LaTeX, il modo migliore possibile per svolgere il compito. Non solo, usufruiremo di un altro gioiello del software libero, il sistema operativo Linux Ubuntu 10.04.

2 Installazione di PostgreSQL 9

Non farò cenno alla procedura per installare una distribuzione TeX, per esempio una TeX Live perché già oggetto di altri post su questo stesso blog, per esempio il post Installare TeX Live 2010 in Ubuntu

Lucid Lynx, ne come si installa Ubuntu stesso. Cominceremo invece installando PostgreSQL nella versione più recente e potente: la 9.0.3 su un sistema della serie LTS 10.04.

Ad oggi, la versione 9 di PostgreSQL non è ancora disponibile nei repository ufficiali di Ubuntu 10.04 (lo sarà per Ubuntu 11.04 Natty Narwhal, quindi per il prossimo mese di aprile), principalmente per le policy di Debian. Noi possiamo installare tranquillamente la versione 8.4 oppure rivolgersi al repository su Launchpad di Martin Pitt, il maintainer Debian per PostgreSQL. Basterà qualche comando da terminale per portare a termine l'operazione.

Si aggiunge il repository di Martin Pitt tra le sorgenti software disponibili sul nostro sistema:

```
$ sudo add-apt-repository ppa:pitti/postgresql
$ sudo apt-get update
$ sudo apt-get upgrade
```

e poi si installa l'intero database server con il comando:

```
$ sudo apt-get install postgresql-9.0
```

2.1 Creare un nuovo utente

Con quest'ultimo semplice comando vengono eseguite numerose attività tra cui la creazione dell'utente ***postgres*** per la gestione in sicurezza del DB, a cui si può accedere soltanto se si è l'amministratore del sistema.

Ma perché si interagisce con PostgreSQL per mezzo di credenziali utente?

PostgreSQL si basa sulla struttura di rete *client-server*: un programma, detto *server*, accetta richieste di rete solo se provenienti da utenti autorizzati, mentre un programma, detto *client*, invia le richieste al recapito del server ed eventualmente riceve la risposta.

Nel nostro caso, il server ed il client girano sullo stesso PC ma, ovviamente, niente vieta che con le dovute impostazioni di sicurezza e di rete, il server con l'intero complesso di possa essere contattato da qualsiasi altro computer connesso al web (per di più indipendentemente dal sistema operativo della macchina client). In particolare è diffusissimo il caso in cui la tecnologia client-server venga sfruttata per fornire servizi dati ai PC di una rete locale LAN basata sul protocollo della rete internet.

Dunque creiamo un nuovo utente autorizzato per l'accesso al server di PostgreSQL, figura che ha un ruolo diverso da quello dell'amministratore ovvero il compito di leggere e scrivere dati da una postazione client e non quello di gestire il server.

Sia *luke* il nome dell'utente, allora apriamo il terminale e per prima cosa diventiamo l'admin *postgres* (attenzione, si tratta di un particolare account del sistema operativo), l'unico (per il momento) in grado di accedere al DB server:

```
$ sudo su postgres
```

e creiamo l'utente con password (-P) da conservare in forma criptata (-E) con un comando scorciatoia (digitate **man createuser** oppure **createuser –help** per la spiegazione delle opzioni). Ecco la sessione di lavoro con il comando *createuser* (confronta anche l'immagine sottostante):

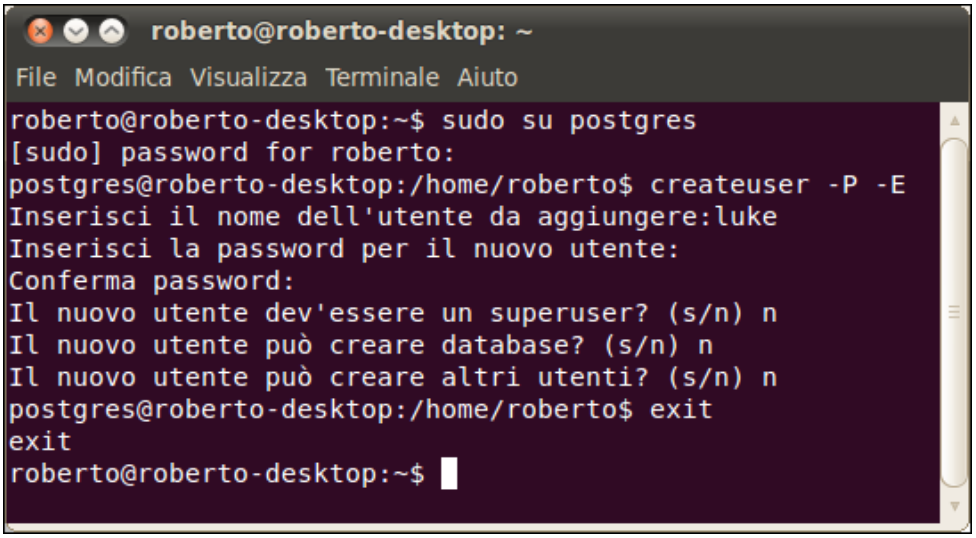

Figura 1: Adding a new user in PostgreSQL from terminal

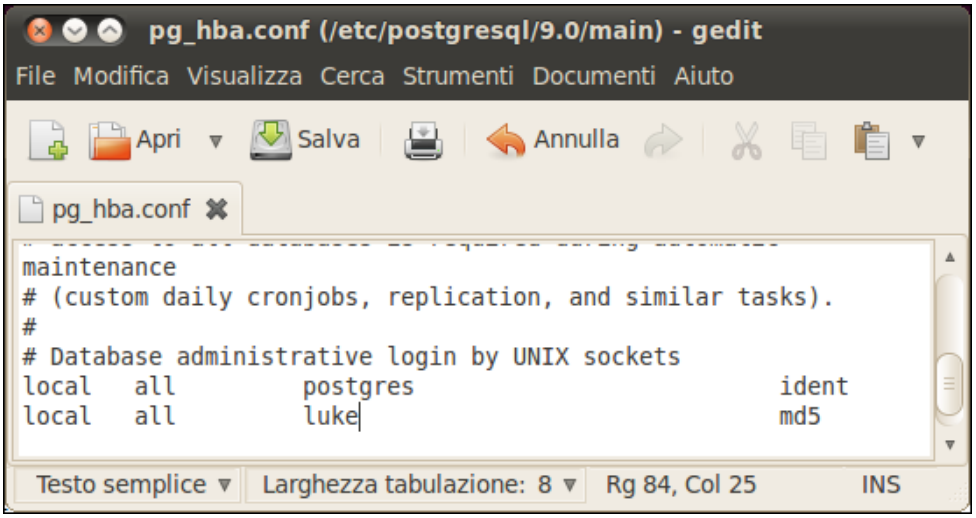

Figura 2: Editing pg_hba.conf

```
postgres$ createuser -P -E
Inserisci il nome dell'utente da aggiungere:luke
Inserisci la password per il nuovo utente:
Conferma password:
Il nuovo utente dev'essere un superuser? (s/n) n
Il nuovo utente può creare database? (s/n) n
Il nuovo utente può creare altri utenti? (s/n) n
postgres@roberto-desktop:/home/roberto$ exit
$
```

Adesso potete svestire i panni dell'utente 'postgres', digitando semplicemente 'exit' (se qualcosa va storto, potete eliminare l'utente con il comando **dropuser luke** e ricominciare da capo).

Il server riceve la richiesta e crea il nuovo utente di lavoro, ma non abbiamo ancora terminato perché al nuovo ruolo dobbiamo assegnare l'autorizzazione necessaria. Per farlo editiamo con i diritti di amministratore il file di configurazione **pg_hba.conf**:

```
$ cd /etc/postgres/9.0/main
$ sudo gedit pg_hba.conf
```

Attenzione, l'operazione di modifica è delicata! Aggiungete, come vedete nella schermata la riga 'local all luke md5' sotto la riga già presente per l'utente postgres come segue:

```
# Database administrative login by UNIX sockets
local   all         postgres                          ident
local   all         luke                              md5
```

Abbiamo reso possibile le connessioni da rete locale per tutti i database presenti sul server (chiave 'all') per l'utente 'luke'. Inoltre la password transiterà in rete locale in forma criptata con l'algoritmo md5 (consultare la documentazione di PostgreSQL al capitolo Client Authentication disponibile sul sito alla sezione Documentation).

Per far si che abbia effetto la modifica, riavviate il server da utente 'postgres' con i comandi (non può certo farlo un utente normale):

```
$ sudo su postgres
postgres$ /etc/init.d/postgresql restart
postgres$ exit
$
```

Se le cose, vi sembrano complicate è più che normale, il server di PostgreSQL non è certo un piccolo programma desktop, e porta con se molti nuovi concetti da imparare. In più si crea una certa confusione per il fatto che lavoriamo su un unico computer e con diverse identità utente.

3 Creare il nostro database

Il server può gestire molti database contemporaneamente. Per crearne uno è sufficiente sceglierne il nome e lanciare il comando seguente come utente 'postgres' ('createdb –help' per consultare le informazioni

di base):

```
$ sudo su postgres
postgres$ createdb ricDB
postgres$ exit
```

4 Il problema

Come esempio, ho scelto una situazione semplice in cui è necessario emettere molti documenti periodici: il rilascio di ricevute di pagamento. Ogni ricevuta ha un numero progressivo che si riazzera ogni anno, un ammontare, una data di emissione, e la data di scadenza. Esprimiamo tutto ciò in linguaggio SQL (per documentarsi un po' ancora la documentazione di PostgreSQL è un ottima fonte, al capitolo Tutorial):

```
CREATE TABLE receipts (
progrnum      integer,               -- numero mese ricevuta nell'anno
yearnum       integer,               -- anno ricevuta
amount        numeric(16,4) NOT NULL, -- importo (4 decimali e 16 cifre totali)
duedate       date          NOT NULL, -- data di scadenza
paymentdate   date,                  -- data di pagamento
receiptyn     boolean,               -- consegna al cliente avvenuta si/no
PRIMARY KEY (progrnum, yearnum)
);

ALTER TABLE receipts ADD
   CONSTRAINT month_great_than_zero
   CHECK (progrnum>0);

ALTER TABLE receipts ADD
   CONSTRAINT year_great_than_zero
   CHECK (yearnum>0);

-- inserimento di alcune ricevute di esempio
INSERT INTO receipts
(prognum, yearnum, duedate, paymentdate, amount, receiptyn)
VALUES
( 1, 2010, '2010-06-15', '2010-06-15',  159.87, 'y'),
( 2, 2010, '2010-06-15', '2010-06-16',  180.30, 'y'),
( 3, 2010, '2010-06-15', '2010-06-20',  780.99, 'y'),
( 4, 2010, '2010-07-10', '2010-08-10',  139.14, 'y'),
( 5, 2010, '2010-07-15', '2010-08-23',  800.90, 'y'),
( 6, 2010, '2010-08-01', '2010-08-01',  111.70, 'y'),
( 7, 2010, '2010-08-28', '2010-08-23',  202.50, 'y'),
( 8, 2010, '2010-09-30', '2010-09-27',  300.41, 'y'),
( 9, 2010, '2010-09-30', '2010-09-27',  100.91, 'y'),
(10, 2010, '2010-10-15', '2010-10-25',   59.87, 'y'),
(11, 2010, '2010-11-10', '2010-11-23', 1059.87, 'y'),
( 1, 2010, '2010-12-01', '2010-12-15',   19.10, 'y'),
( 2, 2011, '2011-01-08', '2011-01-12',   59.80, 'y');
```

Avremo dovuto in realtà prevedere anche un nominativo con una chiave esterna verso una seconda tabella, ma l'esempio si complicherebbe oltre i nostri obiettivi.

4.1 Usare il client psql

Copiate il codice SQL precedente in un file di testo e chiametelo 'run.sql'. Lo eseguiremo per intero in un colpo solo utilizzando il client a riga di comando fornito da PostgreSQL dal breve nome di **psql**. Questo è un compito per il nostro luke (la directory di lavoro del terminale deve essere quella contenente il file a meno di non speficiare per intero il path di 'run.sql', mentre il siglificato delle opzioni ricavabile da un banale 'psql –help' dovrebbe comunque essere chiaro):

```
$ psql -U luke -d ricDB -f run.sql
```

4.2 Subito qualche analisi sui dati

Il client 'psql' può essere anche in modalità interattiva. Dovremo specificare solamente l'utente con cui connettersi al server ed il database su cui vogliamo lavorare:

```
$ psql -U luke -d ricDB
```

Una volta entrati in modalità interattiva, possimao impartire direttamente i comandi SQL anche su più righe. Leggiamo subito la versione di PostgreSQL con cui stiamo lavorando digitando (attenzione, se l'output è troppo esteso per le dimensioni della finestra del terminale i risultati sono mostrati in uno speciale modo da cui si esce per tornare al prompt interattivo premendo il tasto '`q`', inoltre non dimenticatevi il punto e virgola finale):

```
ricDB=>SELECT version();
```

Per consultare l'intera tabella

```
ricDB=> select * from receipts;
  progrnum | yearnum |  amount   |   duedate  | paymentdate | receiptyn
-----------+---------+-----------+------------+-------------+-----------
         1 |    2010 |  159.8700 | 2010-06-15 | 2010-06-15  | t
         2 |    2010 |  180.3000 | 2010-06-15 | 2010-06-16  | t
         3 |    2010 |  780.9900 | 2010-06-15 | 2010-06-20  | t
         4 |    2010 |  139.1400 | 2010-07-10 | 2010-08-10  | t
         5 |    2010 |  800.9000 | 2010-07-15 | 2010-08-23  | t
         6 |    2010 |  111.7000 | 2010-08-01 | 2010-08-01  | t
         7 |    2010 |  202.5000 | 2010-08-28 | 2010-08-23  | t
         8 |    2010 |  300.4100 | 2010-09-30 | 2010-09-27  | t
         9 |    2010 |  100.9100 | 2010-09-30 | 2010-09-27  | t
        10 |    2010 |   59.8700 | 2010-10-15 | 2010-10-25  | t
        11 |    2010 | 1059.8700 | 2010-11-10 | 2010-11-23  | t
         1 |    2010 |   19.1000 | 2010-12-01 | 2010-12-15  | t
         2 |    2011 |   59.8000 | 2011-01-08 | 2011-01-12  | t
(13 rows)
ricDB=>
```

Per trovare il totale dei pagamenti digitate:

```
ricDB=> select sum(amount) as totale from receipts;
```

Per vedere i giorni di ritardo od anticipo rispetto alla data di scadenza, e per calcolarne la media digitare:

```
ricDB=> select duedate-paymentdate as diff from receipts;
ricDB=> select avg(duedate-paymentdate) as average_diff from receipts;
```

In ambiente reale, vi sarebbe la necessita di scrivere uno script interattivo per esempio per facilitare l'inserimento dei dati di una ricevuta, piuttosto che utilizzare direttamente il linguaggio SQL. Bene. Adesso possiamo uscire dal client '`psql`' digitando `\q`, e passare al reporting dei dati.

5 Creare un report con LuaLaTeX

5.1 Breve definizione di LuaLaTeX

Nel tempo sono stati sviluppati diversi motori di composizione tipografica a partire dal progenitore TeX. Questi programmi elaborano un file di testo detto *sorgente*, scritto nella corretta sintassi, in un processo chiamato *compilazione* per produrre un file di output contenente il *risultato tipografico*, normalmente in formato PDF. L'ultimo nato in casa TeX ed ancora in fase di sviluppo è LuaTeX che oltre a riconoscere il linguaggio TeX esteso comprende anche la capacità di eseguire codice Lua, un linguaggio di scripting semplice e generale. Ed è proprio questa la caratteristica che ci permette di connetterci al database direttamente da un sorgente come dimostreremo tra poco.

I motori di composizione fin qui citati, TeX, pdfTeX, LuaTeX, contengono istruzioni molto specializzate per compiti tipografici di dettaglio e quindi poco adatti per un uso di produzione, così ben presto sono nati ulteriori linguaggi macro di più alto livello, implementati con le funzioni primitive, il più conosciuto dei quali è senz'altro LaTeX. Ebbene il formato di alto livello equivalente di LaTeX per LuaTeX è stato chiamato LuaLaTeX. Anche qui, vi faccio notare che se non avete capito bene cosa sono questi motori tipografici, non sorprendetevi poiché vi ho appena condensato più di 32 anni di storia di sviluppo di questi programmi. Per un esposizione pratica potete ulteriormente riferirvi alle due pagine teoria e pratica, del blog, oltre che naturalmente al sito italiano del GuIT.

5.2 La regola fondamentale

L'idea di scrivere qualche comando in un file sorgente per LuaLaTeX per eseguire una query verso un database server è molto interessante e di fattibilità tanto recente che non è ancora chiaro quale ne sia il potenziale applicativo. Si deve tuttavia rispettare una regola, la regola che rispettano tutti i programmi di reporting e che in LuaLaTeX si può potenzialmente violare:

> ***Se si sta costruendo un report, eseguire sul database esclusivamente operazioni di lettura!***

Rispettate questa regola e avrete molti problemi in meno ed un flusso di lavoro più efficiente.

5.3 Installare e configurare LuaSQL

L'accesso al database da LuaLaTeX è possibile perché esiste una libreria che possiamo caricare all'interno del sorgente: LuaSQL, altrimenti dovremo scrivere parecchio codice in C. Per installarne la versione adatta per PostgreSQL, in Ubuntu c'è il solito immancabile comando da terminale:

```
$ sudo apt-get install liblua5.1-sql-postgres-2
```

Avvenuta l'installazione della libreria occorre indicare a LuaTeX, che entra in azione dietro le quinte quando si lancia LuaLaTeX, dove si trova la libreria LuaSQL, modificando (con i diritti di amministratore trovandosi in directory di sistema non modificabili dall'utente ordinario) un file di configurazione chiamato 'texmf.cnf'. Ipotizzando che sia installata una distribuzione TeX Live 2010 ecco i comandi di console per aprire in modifica il file con l'editor di sistema 'Gedit':

```
$ sudo gedit /usr/local/texlive/2010/texmf/web2c/texmf.cnf
```

Cercate la linea che definisce la variabile CLUAINPUTS (dovrebbe essere intorno alla riga 400) e modificatela per aggiungerci il percorso '/usr/lib/lua' (fate attenzione a non modificare il resto della definizione e non dimenticatevi di specificare il doppio slash finale che indica di esplorare anche le subdirectory):

```
% Lua needs to look for binary lua libraries distributed with packages.
CLUAINPUTS = .;/usr/lib/lua//;$SELFAUTOLOC/lib/{$progname,$engine,}/lua//
```

5.4 Finalmente pronti

Giunti fin qui dopo aver dettagliatamente descritto i passi necessari per la costruzione del sistema, ci dedichiamo finalmente al nostro sorgente LuaLaTeX. Comprendere il linguaggio Lua non è difficile grazie alla bravura e alla lungimiranza dei suoi creatori, il codice è semplice. Per saperne comunque di più non c'è niente di meglio che studiarsi il PiL, acronimo del titolo del libro 'Programming in Lua'.

Nel listato che segue, un sorgente elementare in LuaLaTeX, lo scopo è quello di creare un unica ricevuta, per esempio la n. 8 del 2010. All'interno del comando **\directlua** si trova il codice in Lua che per prima cosa si connette al database con le credenziali dell'utente 'luke' e poi esegue la query per ricavare i dati della ricevuta. Il risultato della query è una tabella Lua in cui i nomi delle chiavi corrispondono ai nomi dei campi della tabella *receipts* del database. L'accesso alla tabella restituita dalla query avviene tramite un oggetto **cursor**, al cui metodo fetch() è passata una tabella e un parametro che indica di creare le chiavi con i nomi dei campi e non con il loro semplice indice numerico. Per rendere disponibile a TeX i dati, vengono create con l'ausilio della funzione **tex.sprint()** interna alla funzione **makecmd()**, normalissime macro. Nel sorgente ricordatevi di sostituire alla riga 8 la 'password' che avete scelto al momento della creazione dell'utente 'luke', e pure di compilarlo con il comando **lualatex nomefile**, dove naturalmente 'nomefile' (senza estensione) è quello che gli avete assegnato.

```
\documentclass{minimal}

% connessione al server PostgreSQL
%
\directlua{
  require "luasql.postgres"
  local env = assert(luasql.postgres())
  local con = assert(env:connect("ricDB","luke","password"))

  local cur = assert(con:execute(
  "SELECT * FROM receipts WHERE mounth=1 AND year=2011"))
  local result = cur:fetch({},"a")

```

```
  cur:close()
  con:close()
  env:close()

  local function makecmd( cmd, val)
     local bs = "\string\\"
     tex.sprint(bs.."def"..bs..cmd.."{"..val.."}")
  end

  makecmd("anno",    result.yearnum)
  makecmd("numric",  result.prognum)
  makecmd("importo", result.amount)
  makecmd("dataric", result.paymentdate)
  makecmd("scadenza",result.duedate)
}

\begin{document}
\hspace*{5cm}Ricevuta n. \numric /\anno del \dataric

Si rilascia ricevuta di pagamento per la somma di euro \importo,
avvenuto in data \dataric{} con scadenza in data \scadenza.

Distinti Saluti
\end{document}
```

6 Conclusioni

I software utilizzati sono potenti e complessi, e pongono pochi limiti all'utente che, proprio per questo, deve investire parecchie risorse per padroneggiarli ma ciò non esclude un utilizzo della procedura descritta nell'ambito reale, anche se non si è un esperto Database Administrator o un guru di TeX. Vantaggi e limiti risiedono nel completo controllo sulla nostra soluzione.

Nell'ambito di sviluppo invece, il post dimostra come sia possibile implementare un linguaggio specifico per i documenti TeX per consentire all'utente nuove potenti applicazioni in ambito aziendale usufruendo di una gestione centralizzata dei dati sulla rete locale. Per esempio non è difficile immaginare servizi di gestione documentale, dove il team di colleghi condivide le informazioni producendo documenti PDF di elevata qualità con TeX e con il massimo di disponibilità della fonte dati con PostgreSQL. Efficienza, accuratezza, potenza di analisi e controllo.

7 Licenza ed informazioni varie

Questo articolo come tutto il materiale didattico/divulgativo del blog http://robitex.wordpress.com è rilasciato sotto licenza Creative Commons "Attribuzione-Non commerciale-Non opere derivate" 2.5 Italia, il cui testo integrale con valore legale è consultabile a questo indirizzo. Ciò significa che:

1. Bisogna sempre attribuire la paternità del materiale a http://robitex.wordpress.com;
2. Non si può usare il materiale per fini commerciali;
3. Non si può alterare o trasformare i contenuti, ne' usarne stralci per creare altre opere.

Se esplicitamente indicato nei commenti iniziali, il codice relativo a programmi software è rilasciato nella specifica licenza.

7.1 Distribuzione/Citazioni

Ogni volta che usi o distribuisci parti ridistribuibili quest'opera devi farlo secondo i termini con cui esse sono state rilasciate e avendo cura di comunicare tali termini con chiarezza. Ricorda di inserire sempre un hyperlink alla risorsa che ridistribuisci o citi.

Il modo migliore per dimostrarmi il vostro apprezzamento è semplicemente quello di linkare direttamente le pagine del blog, senza copiare gli articoli in altri siti, oltre naturalmente a lasciare un commento. Ci sono però casi in cui vorreste poter estrapolare alcune parole dai miei articoli per incuriosire i vostri lettori. In quel caso la prassi convenuta e che, grazie a tutti i blogger seri, viene coscienziosamente rispettata, è questa:

- Creare un “blockquote”, ossia un campo in cui è inserita una citazione;
- Inserire nel blockquote solo il primo periodo (le prime poche frasi) di un articolo, diciamo fino ad arrivare al link “Leggi il resto. . . ”;
- Linkare la rimanente parte dell’articolo all’originale su http://robitex.wordpress.com.

Grazie per la collaborazione.

7.2 Colophon

Questo documento è stato composto con LaTeX attraverso uno script in Lua chiamato wp2pdf che elabora il file originale *html* del post pubblicato sul blog in WordPress. Si tratta di una versione migliorata del codice pubblicato sul blog stesso. Per saperne di più contattatemi via posta elettronica all’indirizzo nel titolo del documento, o lasciate un commento sul blog.